Ant Colony Optimization for the Edge-Weighted k-Cardinality Tree Problem

Christian Blum
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium
e-mail: cblum@ulb.ac.be
tel: (32) 2 650 3168, fax: (32) 2 650 2715

Abstract

In this paper we deal with an *NP*-hard combinatorial optimization problem, the k-cardinality tree problem in edge-weighted graphs. This problem has several applications in practice, which justify the need for efficient methods to obtain good solutions. Metaheuristic applications have already been shown to be successful in tackling the k-cardinality tree problem in the past. In this paper we propose an ACO algorithm for the edge-weighted k-cardinality tree problem based on the Hyper-Cube Framework for Ant Colony Optimization. We investigate the usefulness of a higher order pheromone representation in contrast to the standard first order pheromone representation and compare our algorithms to a multi–start local search and a heuristic developed to tackle the problem.

1 Introduction

The k-cardinality tree problem is a combinatorial optimization problem which generalizes the well known minimum weight spanning tree problem. It consists in finding in a node- or edge-weighted graph $G = (V, E)$ a subtree with exactly k edges, such that the sum of the weights is minimal. Due to various applications, e.g. in oil-field leasing [23], facility layout [18], quorum-cast routing [9] and telecommunications [21] it has gained considerable interest in recent years. In this paper we will deal with the k-cardinality tree problem in edge-weighted graphs. The problem can be formally defined as follows. Let $G = (V, E)$ be a graph with a weight function $w : E \rightarrow I\!N$ on the edges. We denote by $\mathcal{T}_k$ the set of all k-cardinality trees in G. Then the edge-weighted problem (G, w, k) is to find a k-cardinality tree $T_k \in \mathcal{T}_k$ which minimizes

$$w(T_k) = \textstyle\sum_{e \in E(T_k)} w(e). \qquad (1)$$

Several authors have proved independently that the edge-weighted k-cardinality tree problem (1) is *NP*-hard, see [17, 26]. In [26] it has been shown that it is still *NP*-hard if $w(e) \in \{1, 2, 3\}$ for all edges e and $G = K_n$, but polynomially solvable if there are only two distinct weights. Several authors have considered special types of graphs. One of the results is that the problem is polynomially solvable if G is a tree (see [25]). The edge-weighted problem is *NP*-complete for planar graphs and for points in the plane, when edge weights correspond to distances between the points (see [26]). In the same paper polynomial algorithms for decomposable graphs and graphs with bounded tree-width have been given. There is also a polynomial algorithm for the case when all points lie on the boundary of a convex region. In [14], the authors have focused on properties of the distance matrix. They have assumed that $G = K_n$ and have proved several results (both *NP*-completeness and polynomial time solvability) on the complexity of the problem with graded distance matrices.

Concerning methodology, both exact and heuristic algorithms have been developed, with a general focus on approximation algorithms. We first note that integer programming formulations have been presented in [17] and later in [20]. Based on detailed studies of the associated polyhedron in the former paper a Branch and Cut algorithm has been developed and implemented in [19]. The code and also implementations of most of the heuristics in [16] are documented in [15]. A Branch and Bound method is described in [9]. The heuristics mentioned are based on greedy and dual greedy strategies and also make use of dynamic programming approaches. Other constructive heuristics have been presented in [9].

More recently, authors successfully applied metaheuristic methods to the k-cardinality tree problem (see Tab. 1 for an overview). Metaheuristics[1] include but are not restricted to Simulated Annealing (SA), Evolutionary Computation (EC) with its most famous representative the Genetic Algorithm (GA),

[1]See [5] for an overview on metaheuristics.

Table 1: An overview of metaheuristic approaches to tackle the k-cardinality tree problem.

Publication	Problem-type	Metaheuristic
M.J. Blesa and F. Xhafa [2], 2000	edge-weighted	TS
M.J. Blesa, P. Moscato and F. Xhafa [1], 2001	edge-weighted	Memetic Algorithm
C. Blum [3], 1998 C. Blum and M. Ehrgott [4], 2001	node-weighted	TS and EC
F. Catanas [8], 1997	node-weighted + edge-weighted	TS and EC
K. Jornsten and A. Lokketangen [24], 1997	edge-weighted	TS
N. Mladenovic [27], 2001	edge-weighted	VNS

Tabu Search (TS), explorative search methods such as Iterated Local Search (ILS), and Ant Colony Optimization (ACO). Among the metaheuristics applied to the k-cardinality tree problem are Evolutionary Computation, Tabu Search, and Variable Neighborhood Search (VNS) (see Tab. 1). The aim of this paper is to show how Ant Colony Optimization can be successfully applied to the edge-weighted k-cardinality tree problem. We investigate the usefulness of a higher order pheromone representation in contrast to the standard first order representation and compare the results obtained by our algorithms to a multi–start local search and a heuristic developed to tackle the problem. The remainder of the paper is organized as follows. In Sec. 2 we briefly outline the concepts of Ant Colony Optimization and a particular way of implementing ACO algorithms, called the Hyper-Cube Framework. In Sec. 3 we present the framework and the components of the ACO algorithm to tackle the edge-weighted k-cardinality tree problem. In Sec. 4 we present results and finally in Sec. 5 we draw some conclusions and give an outlook to future work.

2 Ant Colony Optimization

Ant Colony Optimization (ACO) [10, 13, 11] is a recently proposed metaheuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the foraging behavior of real ants. This behavior enables them to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called *pheromone* on the ground. When they decide about a direction to go they choose, in probability, paths marked by strong pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.
In ACO algorithms, an artificial ant incrementally constructs a solution by adding opportunely defined solution components to a partial solution under consideration[2]. For doing that, artificial ants perform randomized walks on a completely connected graph $\mathcal{G}_c = (\mathcal{C}, \mathcal{L})$ whose vertices are the solution components $\mathcal{C}$ and the set $\mathcal{L}$ are the connections. This graph is commonly called *construction graph*. The problem constraints Ω are built into the ants' constructive procedure in a way such that in every step of the construction process only feasible solution components are permitted to be added to the current partial solution. In ACO algorithms we work with a set of *pheromone values* τ and also with a set of *heuristic values* η. These values are used by the ants' heuristic rule to make probabilistic decisions on how to move on the construction graph. The probabilities involved in moving on the construction graph are commonly called *transition probabilities*.
The first ACO algorithm proposed was Ant System (AS) [13]. Although AS is important, because it was the first ACO algorithm proposed, in the last few years some changes and extensions of AS have been proposed, e.g. Ant Colony System (ACS) [12] and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [29]. In general, ACO algorithms have been proven to be a very effective – for some problems like the QAP even the state-of-the-art – metaheuristic method for combinatorial optimization problem solving.

2.1 The Hyper-Cube Framework

The Hyper-Cube Framework – recently proposed by Blum et al. [6] – is a certain way of implementing ACO algorithms. This way of implementing ACO algorithms comes with several benefits. Maybe the most important one is the property of scaling objective function values.
To a set of pheromone values $\tau = \{\tau_1, ..., \tau_n\}$ in ACO algorithms usually a pheromone updating rule of the following kind is applied.

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \textstyle\sum_{j=1}^{n_s} \Delta^j \tau_i \tag{2}$$

where

$$\Delta^j \tau_i = \begin{cases} f(s^j) & \text{if } s^j \text{contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} . \tag{3}$$

[2]Therefore, the ACO metaheuristic can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined.

$\Delta^j \tau_i$ is the contribution of a solution s^j to the update for pheromone value τ_i (n_s is the number of solutions used for updating the pheromone values), ρ is the evaporation rate (a small positive constant), and f is a function which is monotone in the quality of the solution (for minimization problems it usually maps the quality of a solution to its inverse). In the Hyper-Cube Framework a normalization of the contribution of every solution used for updating the pheromone values is done in the following way.

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \rho \cdot \textstyle\sum_{j=1}^{n_s} \Delta'^j \tau_i \tag{4}$$

where

$$\Delta'^j \tau_i = \begin{cases} \frac{f(s^j)}{\sum_{l=1}^{n_s} f(s^l)} & \text{if } s^j \text{contributes to } \tau_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where we multiply the sum of normalized contributions with the evaporation rate ρ. This formula can be reformulated as:

$$\tau_i \leftarrow \tau_i + \frac{\rho}{\sum_{l=1}^{n_s} f(s^l)} \left(\textstyle\sum_{j=1}^{k} f(s^j) \cdot \delta(s^j, \tau_i) - \tau_i \right) \tag{6}$$

where

$$\delta(s^j, \tau_i) = \begin{cases} 1.0 & \text{if } s^j \text{contributes to } \tau_i \\ 0.0 & \text{otherwise} \end{cases} \tag{7}$$

This leads to a scaling of the objective function values and the pheromone values are implicitly limited to the interval $[0,1]$ (see [6] for a more detailed description).

3 ACO for the k-cardinality tree problem

In this section we outline the framework of our ACO algorithm for the edge-weighted k-cardinality tree problem. The basic framework of our algorithm is shown in Alg. 1. In Alg. 1, $\tau = \{\tau_1, ..., \tau_n\}$ is a set of pheromone values, n_a is the number of ants used in every iteration, ${T_k}^j$ are solutions to the problem, cf is a numerical value which we called the convergence factor, ${T_k}^{ib}$ is the iteration best solution, ${T_k}^{rb}$ is the restart best solution and ${T_k}^{gb}$ is the best solution found from the start of the algorithm.

InitializePheromoneValues(τ): In every version of our algorithm we initialize all the pheromone values to 0.5.

ConstructSolution(τ): To tackle the k-cardinality tree problem with an ACO algorithm we have to define the constructive heuristic to be used in a probabilistic manner to construct solutions to the problem. In ACO algorithms artificial ants construct a solution by building a path on a *construction graph* $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ where the elements of the set $\mathcal{C}$ (called *components*) and the elements of the set $\mathcal{L}$ (called *links*) are given for the k-cardinality tree problem as follows:

Algorithm 1 ACO for the k-cardinality tree problem

```
input: a problem instance $(G, w, k)$
${T_k}^{gb} \leftarrow$ NULL
${T_k}^{rb} \leftarrow$ NULL
$cf \leftarrow 0$
InitializePheromoneValues($\tau$)
while termination conditions not met do
  for $j = 1$ to $n_a$ do
    ${T_k}^j \leftarrow$ ConstructSolution($\tau$)
    LocalSearch(${T_k}^j$)
  end for
  ${T_k}^{ib} \leftarrow$ argmin($w({T_k}^1), ..., w({T_k}^{n_a})$)
  ApplyPheromoneUpdate($cf$,$\tau$,${T_k}^{ib}$,${T_k}^{rb}$,${T_k}^{gb}$)
  Update(${T_k}^{ib}$,${T_k}^{gb}$,${T_k}^{rb}$)
  $cf \leftarrow$ ComputeConvergenceFactor($\tau$,${T_k}^{ib}$)
  if algorithm converged then
    ResetPheromoneValues($\tau$)
    ${T_k}^{rb}$ = NULL
  end if
end while
output: ${T_k}^{gb}$
```

$$\begin{aligned} \mathcal{C} &= E(G) \cup \{c_{source}, c_{sink}\} \\ \mathcal{L} &= \{(e_i, e_j) \mid e_i, e_j \in E(G), e_i \neq e_j\} \\ &\cup \;\; \{(c_{source}, e) \mid e \in E(G)\} \\ &\cup \;\; \{(e, c_{sink}) \mid e \in E(G)\} \end{aligned}$$

Note that all links in $\mathcal{L}$ are directed. This graph $\mathcal{G}$ is fully connecting the edges of G (which are the components of $\mathcal{G}$) plus a source component c_{source} (and arcs from the source component to every component of $\mathcal{G}$) and a sink component c_{sink} (and arcs from every component in $\mathcal{G}$ to the sink component).
To build a solution an ant starts from the source component c_{source} of the construction graph and does k construction steps as shown in Alg. 2. In every step of the ant construction phase we can only add an

Algorithm 2 Ant construction phase

```
Ant is placed on $c_{source}$
$J_1 = \{e = [v_r, v_s] \mid e \in \mathcal{C}\}$
for $t = 1$ to $k$ do
  Choose $e^\star = [v_i, v_j] \in J_t$ to probability $p(e^\star|T_t)$
  Ant moves to the component associated with $e^\star$
  $E(T_t) = E(T_t) \cup e^\star$
  $V(T_t) = V(T_t) \cup \{v_i, v_j\}$
  $J_{t+1} = \{e = [v_r, v_s] \mid e \notin E(T_t)$, either $v_r \in V(T_t)$ or $v_s \in V(T_t)\}$
end for
Ant moves to $c_{sink}$
```

edge $e = [v_i, v_j]$ to the partial k-cardinality tree T_t $(t \in \{1, k-1\})$ if exactly one of the two nodes incident with this edge (v_i, or v_j) is already in the node set $V(T_t)$ of T_t. The generation of the transition probabilities $p(e|T_t)$ for all $e \in J_t$ is dependent on the pheromone representation to be explained in the following.

There are a number of design decisions to be made when developing an ACO algorithm to tackle a combinatorial optimization problem. One of the most crucial decisions is the choice of a pheromone model. For the TSP for example it is a fairly obvious choice to put a pheromone value on every link between a pair of cities. For other combinatorial optimization problems the choice is not as obvious as for the TSP (see [28] for MAX-SAT, or [7] for FOP Shop scheduling). Often this problem can be stated as the problem of assigning pheromone values to the decision variables themselves (first order pheromone values) or to subsets of decision variables (higher order pheromone values). In the following we present two different pheromone representations (models) for the edge-weighted k-cardinality tree problem.

1) Pheromone values on decision variables: This first pheromone model (called $\mathsf{PH}_{\mathsf{1storder}}$ further on) is the most simple choice of a pheromone representation for the edge-weighted k-cardinality tree problem. To every edge $e_i \in E(G)$[3] we have associated a pheromone value τ_{e_i}. Therefore, if $|E(G)| = m$ we have m pheromone values. The probabilities $p(e_i|T_t)$ for the edges in set J_t of the ant construction phase to be chosen by the ant (called transition probabilities) are determined as follows. If $t = 1$ the transition probabilities are

$$p(e_i|T_1) = \begin{cases} \frac{\tau_{e_i}}{\sum_{e_l \in J_1} \tau_{e_l}} & : \text{ if } e_i \in J_1 \\ 0 & : \text{ otherwise} \end{cases} \tag{8}$$

where J_1 is the set of operations allowed to be scheduled next (see Alg. 2). As a good edge (an edge with a low weight) is not necessarily a good starting point for building a low weight k-cardinality tree, we decided not to use any heuristic information in this formula. This is different for the next $k-1$ construction steps. For $t > 1$ the transition probabilities are

$$p(e_i|T_t) = \begin{cases} \frac{\tau_{e_i} \cdot \frac{1}{w(e_i)}}{\sum_{e_l \in J_t} \tau_{e_l} \cdot \frac{1}{w(e_l)}} & : \text{ if } e_i \in J_t \\ 0 & : \text{ otherwise} \end{cases} \tag{9}$$

This means that for the second and consecutive steps the distribution given by the pheromone values is influenced by the weights of the edges. Low edge weights result in a higher probability to be chosen by the ants and the other way around. With this pheromone representation the algorithm tries to learn for every edge the desirability of having it in a solution. This pheromone model doesn't take into account any dependencies between decision variables.

2) Pheromone values on pairs of decision variables: This pheromone model (called $\mathsf{PH}_{\mathsf{2ndorder}}$ further on) takes into account dependencies between decision variables. To every pair $< e_i, e_j >$ (where $e_i \neq e_j$) of edges in $E(G)$ we have associated a pheromone value $\tau_{<e_i,e_j>}$ (where $\tau_{<e_i,e_j>}$ and $\tau_{<e_j,e_i>}$ are the same). We also use the pheromone values of the pheromone model $\mathsf{PH}_{\mathsf{1storder}}$ for the first construction step (when $t = 1$). Therefore, in this model we have $m + (m^2 - m)$ pheromone values. If $t = 1$ the transition probabilities $p(e_i|T_1)$ are generated as shown in equation (8). If $t > 1$ the transition probabilities $p(e_i|T_t, t)$ are generated as follows:

$$\begin{cases} \frac{\left(\sum_{e_j \in E(T_t)} \tau_{<e_j,e_i>}\right) \cdot \frac{1}{w(e_i)}}{\sum_{e_l \in J_t} \left(\sum_{e_j \in E(T_t)} \tau_{<e_j,e_l>}\right) \cdot \frac{1}{w(e_l)}} & : \text{ if } e_i \in J_t \\ 0 & : \text{ otherwise} \end{cases} \tag{10}$$

where J_t is as described above. With this pheromone representation the algorithm tries to learn for every pair of edges the desirability of having them together in a solution. As we have pheromones on pairs of edges, this pheromone model takes into account all first order dependencies between decision variables.

$\mathsf{LocalSearch}({T_k}^j)$: The most important ingredient of a local search method is the neighborhood function. Let T_k be a k-cardinality tree. The neighborhood $\mathcal{N}_{Swap}(T_k)$ of a k-cardinality tree T_k consists of all k-cardinality trees which can be generated from T_k by cutting off one of the leaf edges e from T_k and adding one edge from the neighborhood of $T_k \setminus e$. This neighborhood function has the advantage to be easy to compute, but it is probably coming with the disadvantage of quite a few low quality local minima. However we decided to use this simple neighborhood function in a steepest descent local search (best improvement) in order not to spend a too high percentage of the computation time on the local search.

$\mathsf{ApplyPheromoneUpdate}(cf, \tau, {T_k}^{ib}, {T_k}^{rb}, {T_k}^{gb})$: For updating the pheromone values we are using a so-called *online delayed pheromone update rule*. We always use 3 different solutions for updating the pheromone values[4], the best solution found in the current iteration ${T_k}^{ib}$, the restart best solution ${T_k}^{rb}$ and the best solution found since the start of the algorithm ${T_k}^{gb}$. In contrast to the usual updating rule of the Hyper-Cube Framework as shown in equation (6), in our updating rule the influence of each on of these 3 solutions is dependent one the state of convergence of the algorithm

[3]We consider the edges of graph G to be the decision variables of the problem.

[4]Note that a similar scheme was used in [29].

(given by the convergence factor cf) rather than by the quality of the solutions themselves. First we compute an update value ξ_e for every edge $e \in E(G)$ in the following way.

$$\xi_e \leftarrow \kappa_{ib}\delta({T_k}^{ib}, e) + \kappa_{rb}\delta({T_k}^{rb}, e) + \kappa_{gb}\delta({T_k}^{gb}, e) \quad (11)$$

where κ_{ib} is the influence weight of ${T_k}^{ib}$, κ_{rb} the influence weight of ${T_k}^{rb}$, κ_{gb} the influence weight of ${T_k}^{gb}$ and $\kappa_{ib} + \kappa_{rb} + \kappa_{gb} = 1.0$. The δ-function is defined as follows.

$$\delta(T_k, e_i) \quad = \quad \begin{cases} 1.0 & \text{if } e_i \in E(T_k) \\ 0.0 & \text{otherwise} \end{cases} \quad (12)$$

To the pheromone values τ_{e_i} of pheromone model $\mathsf{PH}_{\mathsf{1storder}}$ we then apply the following update rule.

$$\tau_{e_i} \quad \leftarrow \quad \tau_{e_i} + \rho \cdot (\xi_{e_i} - \tau_{e_i}) \quad (13)$$

To the pheromone values $\tau_{<e_i,e_j>}$ of the pheromone model $\mathsf{PH}_{\mathsf{2ndorder}}$ we apply basically the same pheromone update rule. We compute for every ordered pair of edges $< e_i, e_j >$ the value $\xi_{<e_i,e_j>}$ by using in equation (11) the following δ-function.

$$\delta(T_k, \tau_{<e_i,e_j>}) \quad = \quad \begin{cases} 1.0 & \text{if } e_i, e_j \in E(T_k) \\ 0.0 & \text{otherwise} \end{cases} \quad (14)$$

Then for updating the pheromone values we use the following rule.

$$\tau_{<e_i,e_j>} \leftarrow \tau_{<e_i,e_j>} + \rho \cdot \left(\xi_{<e_i,e_j>} - \tau_{<e_i,e_j>}\right) \quad (15)$$

Depending on the convergence factor cf the influence of every one of these 3 solutions on the pheromone update is determined. The convergence factor cf is a value providing an estimate about the state of convergence of the system. The convergence factor is computed in the following way.

$$cf \; = \; \frac{\sum_{e \in E(G)} \delta({T_k}^{ib}, e) \cdot (1.0 - \tau_e)}{k} \quad (16)$$

where we use the δ-function defined in equation (12). As by using the Hyper-Cube Framework for updating the pheromone values, the pheromone values can only assume values between 0.0 and 1.0 (see [6]) it obviously holds that cf also only can assume values between 0.0 and 1.0. It is also clear that if cf is close to 0.0 the system is in a state where the probability to produce solution ${T_k}^{ib}$ is close to 1 and therefore the probability to produce a solution different to ${T_k}^{ib}$ is close to 0. This is what we informally call the state of convergence for our system.
From experience gathered with the algorithm we chose the schedule of settings for values ρ,κ_{ib}, κ_{rb} and κ_{gb} as shown in Tab. 2. In the following we give an interpretation of the choice of parameters shown in Tab. 2. At the beginning of the search process the evaporation rate (which is in the Hyper-Cube Framework more appropriately called learning rate) is set to the value 0.15, because at the beginning of the search there is no need to be very careful. The algorithm should rather drift around in the search space to get a kind of global perspective. Also the influence of the best solution found in an iteration is quite high, which also supports the algorithm drifting through the search space. Once the algorithm starts converging (cf falls below 0.3) we decrease the learning rate (in order to perform a more careful search) and increase the influence of the best solution found since the restart of the algorithm. Once the algorithm is near to the state of convergence only the restart best solution is used to update the pheromone values and we decrease the learning rate even more in the hope to find a better solution near the restart best solution. Before the algorithm is completely converged we use the best solution found since the start of the algorithm to update the pheromone values. This action basically results in a shift of the probability distribution given by the pheromone values toward the best solution found. The reason behind that is the hope to find a better solution in-between two good solutions which are the restart best and the overall best solution in this case. This idea is very similar to ideas we can find in Path Relinking [22] for example.

$\mathsf{Update}({T_k}^{ib}, {T_k}^{rb}, {T_k}^{gb})$: In this procedure we replace the old solution ${T_k}^{rb}$ with ${T_k}^{ib}$ if $w({T_k}^{ib}) < w({T_k}^{rb})$. We do the same for ${T_k}^{gb}$.

$\mathsf{ComputeConvergenceFactor}(\tau, {T_k}^{ib})$: The convergence factor cf is re-computed in every iteration according to equation (16).

$\mathsf{ResetPheromoneValues}(\tau)$: In this procedure we reset all pheromone values τ_e to the start value 0.5.

4 Test results

We chose three different problem instances for a preliminary testing of our algorithms. Two of them are complete grid graphs[5] with 10 rows and 10 columns which sums up to 100 nodes and 180 edges. These graphs are called 10x10_1.gg and 10x10_2.gg in the following. The weights of the nodes were randomly generated using a uniform distribution on the integers between 1 and 100. There are two motivations for choosing grid graphs for testing our algorithms. Problems in practice are often modeled as grid graphs (e.g., the oil-field leasing problem in [23]). Also, it was observed in earlier publications (see [3]) that the problem is considerably harder to solve in grid graphs compared to unstructured graphs. Additionally we chose one of the

[5]No edges or nodes are missing in the grid.

Table 2: The schedule used for values ρ,κ_{ib}, κ_{rb} and κ_{gb} depending on the value of the convergence factor cf.

	$cf > 0.3$	$cf \leq 0.3$, $cf > 0.05$	$cf \leq 0.05$, $cf > 0.025$	$cf \leq 0.025$
ρ	0.15	0.1	0.05	0.1
κ_{ib}	2/3	1/3	0	0
κ_{rb}	1/3	2/3	1	0
κ_{gb}	0	0	0	1

graphs with 400 nodes and 800 edges used by Jornsten and Lokketangen in [24] to test their algorithm. This graph is called g400-4-01.dat. We applied the following four different algorithms to these three graphs:

- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{1storder}}$, further on called ACO1.
- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{2ndorder}}$, further on called ACO2.
- Alg. 1 using pheromone model $\mathsf{PH}_{\mathsf{1storder}}$, without updating the pheromone values. This is resulting basically in a multi–start local search, further on called MSLS.
- A heuristic based on greedy strategies developed in [16], which is called KCP.

The results are shown in Tables 3–5 (best results in bold). We started the algorithms for a range of cardinalities between 2 and $|V(G)|$. On each graph, each algorithm was applied 20 times for each cardinality. The results are reported for every algorithm (except for KCP) on every problem instance in four columns. The first column (titled "Obj.") contains the average of the best found solutions out of 20 runs. The second column (titled $\sqrt{\sigma}$) contains the standard deviation of these best found solutions. The third column (titled "time") contains the average of the times (in seconds) when the best solution of a run was found. Finally the fourth column (titled $\sqrt{\sigma}$) contains the standard deviation of these times. The stopping criterion for all the algorithms (except KCP) was a maximum amount of running time. We allowed the same amount of running time to all the algorithms. This amount of running time is dependent on the cardinality, and given in seconds by $1 + \frac{k \cdot |V(G)|}{100}$.
From the results in Tables 3–5 we can draw several conclusions. ACO1 is among the tested algorithms clearly the best one. Except for very high cardinalities – where the heuristic KCP is likely to produce the optimal solution – it clearly beats the other algorithms in average quality, in standard deviation of the quality, and in average time the best solution was found. This superiority is especially obvious for cardinalities in the middle of the cardinality range where the problem is harder to solve than at the beginning or the end of the cardinality range. The difference between ACO1 and MSLS points out, that the usage of pheromone values for the k-cardinality tree problem seems very fruitful. At first sight it seems surprising that ACO2 doesn't reach the quality of ACO1, because ACO2 is taking into account first order dependencies between decision variables compared to no dependencies in ACO1. However, if we consider the quadratic increase in complexity of the algorithm[6], the outcome of the experimental results become understandable. Due to the considerably increased complexity, ACO2 needs much more time to find good solutions. This is getting more obvious with growing graph size. Therefore the "use of more information" seems not to be very promising in ACO algorithms for the k-cardinality tree problem.

5 Conclusions and outlook to the future

In this work we presented an ACO algorithm for the edge-weighted k-cardinality tree problem. We presented two different pheromone models, the first of them not taking into account any dependencies between decision variables, the second one taking into account first order dependencies between decision variables. It turned out, that for the k-cardinality tree problem it doesn't seem beneficial to take into account dependencies between decision variables, because the increased complexity slows the algorithm considerably down. In the future we plan to improve the efficiency of our algorithm in order to compare it to state-of-the-art metaheuristics for the k-cardinality tree problem. We also plan to investigate the usefulness of diversification schemes for our algorithm.

Acknowledgments: This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] M.J. Blesa, P. Moscato, and F. Xhafa. A Memetic Algorithm for the Minimum Weighted k-Cardinality

[6]The number of pheromone values is quadratic in the number of edges of the graph.

Table 3: Results for grid graph 10x10_1.gg (100 nodes, 180 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**5**	0	0.008	0.007	**5**	0	0.065	0.106	**5**	0	0.002	0.005	**5**
5	**52**	0	0.020	0.009	**52**	0	0.186	0.251	**52**	0	0.011	0.017	**52**
10	**161**	0	0.042	0.038	**161**	0	0.175	0.163	**161**	0	0.017	0.022	**161**
15	**254**	0	0.23	0.140	**254**	0	0.998	0.579	**254**	0	0.414	0.454	258
20	**368**	0	0.525	0.363	**368**	0	3.101	3.342	**368**	0	2.999	2.171	370
25	**465**	0	4.662	3.896	469.05	6.378	8.457	6.607	477	6.950	12.79	7.294	505
30	**579**	0	7.296	5.378	582.75	3.809	15.023	8.814	597.95	6.747	15.725	8.521	627
35	**689.45**	0.944	15.094	10.585	691.7	2.002	24.568	8.196	721.65	9.3148	13.049	9.478	752
40	**804.55**	1.394	17.435	12.332	806.1	1.618	19.529	11.490	853.75	13.396	17.758	9.352	878
45	**912**	3.641	24.067	14.795	918.55	3.605	24.704	10.933	973.3	14.179	24.387	14.124	988
50	**1023.45**	1.145	16.132	13.863	1027.25	4.865	29.549	12.344	1103.75	22.318	25.173	14.431	1106
55	**1139.3**	3.812	22.04	15.270	1144.75	6.248	34.9	8.576	1253	17.474	29.934	15.324	1280
60	**1257.15**	0.366	35.317	16.970	1263.7	4.932	37.319	9.840	1396.55	22.497	32.940	17.351	1404
65	**1400.1**	0.307	32.226	15.456	1406.2	4.741	39.4555	9.147	1556.6	24.489	25.657	17.176	1565
70	**1538.25**	0.550	29.426	13.584	1542.8	4.396	52.42	8.685	1709.85	37.307	29.738	17.042	1696
75	**1689.5**	1.192	42.561	21.223	1694.75	8.801	59.657	9.304	1872	31.522	35.631	25.631	1840
80	**1867.65**	1.755	44.51	17.818	1877.35	7.922	68.369	8.701	2065.1	21.875	33.666	25.342	2005
85	**2066.9**	2.381	48.310	20.819	2081.1	7.362	71.026	9.454	2263.4	27.933	42.992	24.616	2128
90	**2299.05**	1.431	51.622	23.100	2311.05	6.082	80.548	10.948	2530.2	29.881	48.499	20.828	2301
95	**2594.5**	1.877	64.463	26.011	2610.25	8.058	81.702	8.712	2872.9	22.083	49.320	27.162	2599
98	2795.15	2.906	71.549	23.218	2812.6	11.408	90.541	6.419	3124.6	35.858	68.147	24.817	**2791**

Table 4: Results for grid graph 10x10_2.gg (100 nodes, 180 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**4**	0	0.012	0.010	**4**	0	0.065	0.068	**4**	0	0.003	0.005	**4**
5	**24**	0	0.013	0.006	**24**	0	0.054	0.045	**24**	0	0.004	0.007	**24**
10	**65**	0	0.027	0.015	**65**	0	0.138	0.147	**65**	0	0.018	0.017	**65**
15	**162**	0	0.116	0.051	**162**	0	0.561	0.229	**162**	0	0.230	0.139	**162**
20	**255**	0	0.202	0.163	**255**	0	1.025	1.558	**255**	0	0.197	0.256	257
25	**309**	0	0.453	0.180	**309**	0	1.675	0.831	**309**	0	2.505	1.835	329
30	**422**	0	1.034	0.348	**422**	0	3.447	1.201	425.3	3.357	14.779	9.166	425
35	**520**	0	3.331	3.425	520.1	0.307	7.463	6.599	535.1	9.425	17.029	10.449	570
40	**602**	0	5.14	5.285	602.75	1.831	10.09	2.613	663.9	19.185	19.774	10.753	658
45	**693.25**	1.118	15.246	10.150	695.3	2.617	25.856	12.139	759.4	19.146	23.635	14.133	780
50	**792**	0	3.916	0.693	**792**	0	13.740	2.510	873.35	30.475	24.791	13.466	857
55	**905**	0	11.800	6.686	906.6	1.391	21.907	6.549	1012.3	23.299	26.930	18.319	973
60	**1019**	0	6.544	1.136	**1019**	0	24.003	5.312	1152.2	16.577	29.297	19.926	1080
65	**1148.8**	0.410	18.610	17.345	1153.45	6.581	47.362	10.014	1279.55	20.309	31.098	19.773	1189
70	**1277**	0	16.936	10.631	1285.8	8.983	55.328	9.765	1439.35	24.381	38.059	16.969	1344
75	**1432.6**	1.142	28.197	16.183	1435.05	4.370	62.882	7.456	1599.4	29.077	34.194	23.764	1465
80	**1607.7**	3.130	29.585	20.031	1615.2	8.489	60.664	12.852	1794.1	25.708	39.664	20.211	1634
85	**1853**	0	30.097	18.776	1860.8	10.211	69.03	14.123	2065.9	30.371	45.102	26.022	1863
90	**2118**	2.051	35.263	21.807	2123.4	6.029	73.458	11.994	2375.05	31.091	36.224	22.214	2132
95	2429.7	1.592	49.771	21.225	2439.55	7.279	75.830	10.991	2701.5	31.103	50.298	33.037	**2429**
98	2631.35	0.988	58.222	30.827	2640.65	9.027	81.049	14.101	2934.05	30.5829	50.318	27.746	**2631**

Table 5: Results for graph g400-4-01.dat from Jornsten and Lokketangen (400 nodes, 800 edges)

	ACO1				ACO2				MSLS				KCP
k	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.	$\sqrt{\sigma}$	time	$\sqrt{\sigma}$	Obj.
2	**8**	0	0.132	0.116	8.85	2.621	2.354	1.959	**8**	0	0.088	0.091	**8**
20	**253**	0	5.388	5.916	254.1	2.291	36.806	17.368	**253**	0	16.270	11.070	272
40	**563**	0	35.785	20.175	566.95	3.691	100.601	30.748	608.45	16.519	94.069	45.563	604
60	**919.25**	0.716	102.26	59.518	923.1	3.291	207.44	20.860	1013.2	27.532	96.098	74.733	1022
80	**1306.75**	2.048	192.245	73.594	1334.15	10.095	295.5	32.245	1483.35	24.615	135.729	98.816	1408
100	**1731.1**	5.280	271.189	86.696	1817.75	12.961	367.046	36.947	1971.15	33.750	186.865	130.132	1926
120	**2180**	10.031	338.813	71.064	2328.95	26.727	452.901	41.416	2512.3	35.797	231.954	159.193	2350
140	**2641.05**	17.425	408.078	149.159	2875.35	34.392	523.885	55.076	3048.25	43.770	326.553	138.83	2822
160	**3129.8**	14.028	461.798	132.681	3477.55	62.543	617.647	80.852	3653.8	56.113	313.43	195.834	3220
180	**3641.15**	16.480	445.803	50.486	4139.45	50.625	687.47	104.207	4254.95	43.192	381.348	205.387	3758
200	**4180.6**	21.685	573.88	65.199	4756	76.595	824.476	110.62	4874.1	45.750	416.265	212.639	4262
220	**4753.95**	16.452	702.897	69.385	5465.75	62.905	837.577	171.348	5546	57.945	493.855	255.638	4806
240	**5342.85**	18.765	861.051	84.423	6176.7	83.805	1009.41	105.39	6227	78.016	673.384	233.757	5429
260	**5960.35**	19.505	1005.04	38.325	6938.05	62.001	945.168	216.461	6943.55	70.770	529.437	275.012	6040
280	**6618.45**	28.010	1083.75	29.585	7669.05	74.339	1139.98	199.618	7683.1	59.158	547.638	309.312	6715
300	**7312.75**	16.476	1161.57	38.844	8493.9	102.496	1044.78	252.145	8418.2	92.126	659.805	333.841	7386
320	**8029.15**	20.625	1232.46	48.928	9298.25	56.904	1283.54	257.799	9275.2	56.083	768.435	361.237	8063
340	8808.1	19.325	1318.16	41.868	10168	90.851	1359.32	271.292	10132.4	64.570	678.346	405.055	**8805**
360	9641.65	23.351	1403.69	33.868	11126.6	114.205	1497.57	293.392	11125.3	75.764	652.205	373.956	**9554**
380	10585.5	20.914	1466.83	36.829	12336.1	95.999	1555.28	427.12	12256.3	74.524	746.958	405.787	**10451**
398	11702.6	28.593	1441.75	82.004	14327	113.37	1461.42	385.02	14216.5	74.663	729.97	448.678	**11433**

Tree Subgraph Problem. In *Proceedings of the Metaheuristics International Conference MIC'2001*, volume 1, pages 85–90, Porto, Portugal, July 2001.

[2] M.J. Blesa and F. Xhafa. A C++ Implementation of Tabu Search for k-Cardinality Tree Problem based on Generic Programming and Component Reuse. In c/o tranSIT GmbH, editor, *Net.ObjectDsays 2000 Tagungsband*, pages 648–652, Erfurt, Germany, 2000. Net.ObjectDays-Forum.

[3] C. Blum. Optimality criteria and local search methods for node-weighted k-cardinality tree and subgraph problems. Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, 1998. In German.

[4] C. Blum and M. Ehrgott. Local search algorithms for the k-cardinality tree problem. Technical Report TR/IRIDIA/2001-12, IRIDIA, Université Libre de Bruxelles, Belgium, 2001. submitted to Discrete Applied Mathematics.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Technical Report TR/IRIDIA/2001-13, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.

[6] C. Blum, A. Roli, and M. Dorigo. HC-ACO: The hyper-cube framework for ant colony optimization. In *Proceedings of Metaheuristics International Conference MIC'2001*, pages 399–403, Porto, Portugal, July 2001.

[7] C. Blum and M. Sampels. Ant Colony Optimization for FOP Shop scheduling: a case study on different pheromone representations. Accepted for presentation at the Congress on Evolutionary Computation, CEC 2002, May 2002.

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

[9] S.Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM'94*, Los Alamitos, USA, 1994. IEEE Society Press.

[10] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, DEI, Politecnico di Milano,Italy, 1992. pp. 140.

[11] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[12] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[13] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.

[14] T. Dudas, B. Klinz, and G.J. Woeginger. The computational complexity of the k-minimum spanning tree problem in graded matrices. *Computers and Mathematics with Applications*, 36:61–67, 1998.

[15] M. Ehrgott and J. Freitag. K_TREE / K_SUBGRAPH: A program package for minimal weighted k-cardinality-trees and -subgraphs. *European Journal of Operational Research*, 1(93):214–225, 1996. code available at http://www.mathematik.uni-kl.de/~wwwwi/WWWWI/ORSEP/contents.html.

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

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

[18] 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.

[19] J. Freitag. Minimal k-cardinality trees. Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, 1993. in german.

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

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

[22] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

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

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

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

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

[27] N. Mladenovic. Variable neighborhood search for the k-cardinality tree problem. In *Proceedings of the Metaheuristics International Conference MIC'2001*, 2001.

[28] A. Roli, C. Blum, and M. Dorigo. ACO for maximal constraint satisfaction problems. In *Proceedings of Metaheuristics International Conference MIC'2001*, pages 187–191, Porto, Portugal, July 2001.

[29] T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.