Hybrid Algorithms for the Minimum-Weight Rooted Arborescence Problem

Sergi Mateo[1], Christian Blum[1], Pascal Fua[2], and Engin Türetgen[2]

[1] ALBCOM Research Group, Universitat Politécnica de Catalunya, Barcelona, Spain
sergi.mateo.bellido@est.fib.upc.edu, cblum@lsi.upc.edu
[2] Computer Vision Lab, Ecole Polytechnique Fédérale de Lausanne, Switzerland
{pascal.fua@,engin.turetken}@epfl.ch

Abstract. Minimum-weight arborescence problems have recently enjoyed an increased attention due to their relation to imporant problems in computer vision. A prominent example is the automated reconstruction of consistent tree structures from noisy images. In this paper, we first propose a heuristic for tackling the minimum-weight rooted arborescence problem. Moreover, we propose an ant colony optimization algorithm. Both approaches are strongly based on dynamic programming, and can therefore be regarded as hybrid techniques. An extensive experimental evaluation shows that both algorithms generally improve over an exisiting heuristic from the literature.

1 Introduction

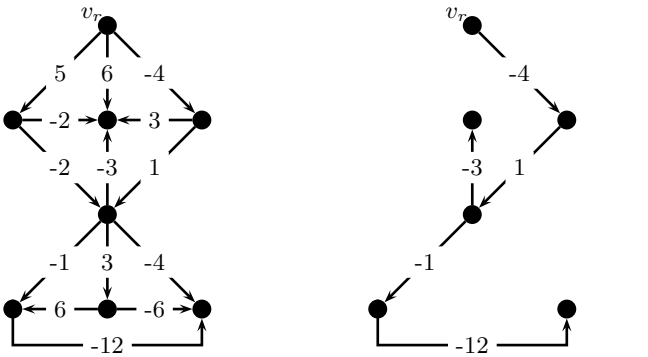
The minimum-weight rooted arborescence (MWRA) problem, which is considered in this work, is a generalization of the problem proposed by Venkata Rao and Sridharan in [10]. It can technically be described as follows. Given is a directed acyclic graph $G = (V, A)$ with integer weights on the arcs, that is, for each $a \in A$ exists a corresponding weight $w(a) \in \mathbb{Z}$. Moreover, a vertex $v_r \in V$ is designated as the *root node*. Let $\mathcal{A}$ be the set of all arborescences in G that are rooted in v_r. In this context, note that an arborescence is a directed, rooted tree in which all arcs point away from the root vertex (see also [9]). Moreover, note that $\mathcal{A}$ contains all arborescences, not only the ones with maximal size. The objective function value (that is, the weight) $f(T)$ of an arboresence $T \in \mathcal{A}$ is defined as follows:

$$f(T) := \sum_{a \in T} w(a) \ . \tag{1}$$

The goal of the MWRA problem is to find an arboresence $T^* \in \mathcal{A}$ such that the weight of T^* is smaller or equal to all other arborescences in $\mathcal{A}$. In other words, the goal is to minimize objective function $f(\cdot)$. An example of the MWRA problem is shown in Figure 1.

The differences to the problem proposed in [10] are as follows. The authors of [10] require the root v_r to have only one single outgoing arc. Moreover, numbering the vertices from 1 to $|V|$, the given acyclic graph G is restricted to contain

M. Dorigo et al. (Eds.): ANTS 2012, LNCS 7461, pp. 61–72, 2012.



(a) Example input graph (b) Optimal solution. Value: -19

Fig. 1. (a) shows an input DAG with eight vertices and 14 arcs. The uppermost vertex is the root vertex v_r. (b) shows the optimal solution, that is, the arborescence rooted in v_r which has the minimum weight among all arborescence rooted in v_r that can be found in the input graph.

only arcs $a_{i,j}$ such that $i < j$. These restrictions do not apply to the MWRA problem. Nevertheless, as a generalization of the problem proposed in [10], the MWRA problem is NP-hard. Concerning existing work, the literature only offers the heuristic proposed in [10], which can also be applied to the more general MWRA problem.

The definition of the MWRA problem as outlined above is inspired by a novel method which was recently proposed in [8] for the automated reconstruction of consistent tree structures from noisy images, which is an important problem, for example, in Neuroscience. Tree-like structures, such as dendritic, vascular, or bronchial networks, are pervasive in biological systems. Examples are 2D retinal fundus images and 3D optical micrographs of neurons. The approach proposed in [8] builds a set of candidate arborescences over many different subsets of points likely to belong to the optimal delineation and then chooses the best one according to a global objective function that combines image evidence with geometric priors (see Figure 2 for an example). The solution of the MWRA problem (with additional hard and soft constraints) plays an important role in this process. Therefore, developing better algorithms for the MWRA problem may help in composing better techniques for the problem of the automated reconstruction of consistent tree structures from noisy images.

The contribution of this work is as follows. First, a new heuristic for the MWRA problem is presented which is based on the deterministic construction of an arborescence of maximal size, and the subsequent application of dynamic programming for finding the best solution within this constructed arborescence. The second contribution is to be found in the application of ant colony optimization (ACO) [4] to the MWRA problem. As both the heuristic and the ACO approach are based on a sub-ordinate dynamic programming procedure, both

(a) Original 2D retinal image

(b) Reconstruction of the vascular structure

Fig. 2. (a) shows a 2D image of the retina of a human eye. The problem consists in the automatic reconstruction (or delineation) of the vascular structure. (b) shows the reconstruction of the vascular structure as produced by the algorithm proposed in [8].

algorithms can be seen as hybrid (meta-)heuristics [3]. An extensive experimental evalution of both algorithms shows their superiority to the only exisiting heuristic proposed in [10].

The outline of this paper is as follows. Section 2 is dedicated to the new heuristic proposed in this work. Furthermore, in Section 3 our ant colony optimization approach is outlined. Finally, an extensive experimental study is described in Section 4 and conclusions as well as an outlook to future work is given in Section 5.

2 A New Heuristic Approach

In the following we describe a new heuristic approach for solving the MWRA problem. First, starting from root vertex v_r, an arborescence T' of maximal size in G is constructed as outlined in lines 2–9 of Algorithm 1. Second, a dynamic programming (DP) algorithm is applied to T' in order to obtain the minimum-weight arborescence T that is contained in T' and rooted in v_r. The DP algorithm from [1] is used for this purpose. Given an undirected tree $T = (V_T, E_T)$ with vertex and/or edge weights, and any integer number $k \in [0, |V_T| - 1]$, this DP algorithm provides—among all trees with exactly k edges in T—the minimum-weight tree T^*. The first step of the DP algorithm consists in artificially converting the input tree T into a rooted arborescence. Therefore, the DP algorithm can directly be applied to arborescences. Morever, as a side product, the DP algorithm also provides the minimum-weight arborescences for all l with $0 \leq l \leq k$, as well as the minimum-weight arborescences rooted in v_r for all l with $0 \leq l \leq k$. Therefore, given an arborescence of maximal size T', which has $t \leq |V| - 1$ arcs (where V is the vertex set of the input graph G), the DP algorithm is applied with $k = t$. Then, among all the minimum-weight

Algorithm 1. Heuristic DP-HEUR for the MWRA problem

1: **input:** a DAG $G = (V, A)$, and a root node v_r
2: $T' := (V' = \{v_r\}, A' = \emptyset)$
3: $A_{\text{pos}} := \{a = (v_i, v_j) \in A \mid v_i \in V', v_j \notin V'\}$
4: **while** $A_{\text{pos}} \neq \emptyset$ **do**
5: $\quad a^* = (v_i, v_j) := \text{argmin}\{w(a) \mid a \in A_{\text{pos}}\}$
6: $\quad A' := A' \cup \{a^*\}$
7: $\quad V' := V' \cup \{v_j\}$
8: $\quad A_{\text{pos}} := \{a = (v_i, v_j) \in A \mid v_i \in V', v_j \notin V'\}$
9: **end while**
10: $T :=$ Dynamic_Programming$(T', k = |V| - 1)$
11: **output:** arborescence T

arborescences rooted in v_r for $l \leq t$, the one with minimum weight is chosen as the output of the DP algorithm. In this way, the DP algorithm is able to generate the minimum-weight arborescence T (rooted in v_r) which can be found in arborescence T'. The heuristic described above is henceforth labelled DP-HEUR.

3 Ant Colony Optimization for the MWRA Problem

The ant colony optimization (ACO) approach for the MWRA problem which is described in the following is a $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (MMAS) [6] implemented in the Hyper-Cube Framework (HCF) [2]. The algorithm, whose pseudocode can be found in Algorithm 2, works roughly as follows. At each iteration, a number of n_a solutions to the problem is probabilistically constructed based both on pheromone and heuristic information. Each solution construction consists of a first phase in which a rooted arborescence of maximal size T' in input graph G is probabilistically constructed, starting from the root vertex v_r. Moreover, in a second phase, the minimum-weight arborescence T rooted in v_r which exists in T' is obtained by dynamic programming. The second algorithmic component which is executed at each iteration is the pheromone update. Hereby, some of the constructed solutions—that is, the iteration-best solution T^{ib}, the restart-best solution T^{rb}, and the best-so-far solution T^{bs}—are used for a modification of the pheromone values. This is done with the goal of focusing the search over time on high-quality areas of the search space. Just like any other MMAS algorithm, our approach employs restarts consisting of a re-initialization of the pheromone values. Restarts are controlled by the so-called convergence factor (cf) and a Boolean control variable called *bs_update*. The main functions of our approach are outlined in detail in the following.

Construct_Arborescence_Of_Maximal_Size(G, v_r): This function contructs a solution in the way which is shown in lines 2–9 of Algorithm 1. The only difference is in the choice of the next arc to be added to the current arborescence T' at each step (line 5 of Algorithm 1). Instead of deterministically choosing from A_{pos} the arc which has the smallest weight value, the choice is done probabilistically,

Algorithm 2. Ant Colony Optimization for the MWRA Problem

input: a DAG $G = (V, A)$, and a root node v_r
$T^{bs} := (\{v_r\}, \emptyset)$, $T^{rb} := (\{v_r\}, \emptyset)$, $cf := 0$, $bs\_update :=$ **false**
$\tau_a := 0.5$ for all $a \in A$
while termination conditions not met **do**
  for $i = 1, \cdots, n_a$ **do**
    $T' :=$ Construct_Arborescence_Of_Maximal_Size(G, v_r)
    $T_i :=$ Dynamic_Programming$(T', k = |V| - 1)$
  end for
  $T^{ib} := \text{argmin}\{f(T_i) \mid T_1, \ldots, T_{n_a}\}$
  if $T^{ib} < T^{rb}$ **then** $T^{rb} := T^{ib}$
  if $T^{ib} < T^{bs}$ **then** $T^{bs} := T^{ib}$
  ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,T^{ib},T^{rb},T^{bs})
  $cf :=$ ComputeConvergenceFactor$(\mathcal{T})$
  if $cf > 0.99$ **then**
    if $bs\_update =$ **true then**
      $\tau_a := 0.5$ for all $a \in A$
      $T^{rb} := (\{v_r\}, \emptyset)$
      $bs\_update :=$ **false**
    else
      $bs\_update :=$ **true**
    end if
  end if
end while
output: T^{bs}, the best solution found by the algorithm

based on pheromone and heuristic information. The pheromone model $\mathcal{T}$ that is used for this purpose contains a pheromone value τ_a for each arc $a \in A$. The heuristic information $\eta(a)$ of an arc a is computed as follows. First, let

$$w_{\max} := \max\{w(a) \mid a \in A\}. \tag{2}$$

Based on this maximal weight of all arcs in G, the heuristic information is defined as follows:

$$\eta(a) := w_{\max} + 1 - w(a) \tag{3}$$

In this way, the heuristic information of all arcs is a positive number. Moreover, the arc with minimal weight will have the heighest value concerning the heuristic information. Given an arborescence T', and the non-empty set of arcs A_{pos} that may be used for extending T', the probability for choosing arc $a \in A_{\text{pos}}$ is defined as follows:

$$\mathbf{p}(a \mid T') := \frac{\tau_a \cdot \eta(a)}{\sum_{\hat{a} \in A_{\text{pos}}} \tau_{\hat{a}} \cdot \eta(\hat{a})} \tag{4}$$

However, instead of choosing an arc from A_{pos} always in a probabilistic way, the following scheme is applied at each construction step. First, a value $r \in [0, 1]$ is chosen uniformly at random. Second, r is compared to a so-called *determinism*

Table 1. Setting of κ_{ib}, κ_{rb}, κ_{bs}, and ρ depending on the convergence factor cf and the Boolean control variable *bs_update*

	bs_update = FALSE				*bs_update*
	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$	= TRUE
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1
ρ	0.1	0.1	0.1	0.1	0.1

rate $\delta \in [0, 1]$, which is a fixed parameter of the algorithm. If $r \leq \delta$, arc $a^* \in A_{\text{pos}}$ is chosen to be the one with the maximum probability, that is:

$$a^* := \text{argmax}\{\mathbf{p}(a \mid T') \mid a \in A_{\text{pos}}\} \tag{5}$$

Otherwise, that is, when $r > \delta$, arc $a^* \in A_{\text{pos}}$ is chosen probabilistically according to the probability values.

ApplyPheromoneUpdate(cf,*bs_update*,$\mathcal{T}$,T^{ib},T^{rb},T^{bs}): The pheromone update is performed in the same way as in all $\mathcal{MM}$AS algorithms implemented in the HCF. The three solutions T^{ib}, T^{rb}, and T^{bs} (as described at the beginning of this section) are used for the pheromone update. The influence of these three solutions on the pheromone update is determined by the current value of the convergence factor cf, which is defined later. Each pheromone value $\tau_a \in \mathcal{T}$ is updated as follows:

$$\tau_a := \tau_a + \rho \cdot (\xi_a - \tau_a) \ , \tag{6}$$

where

$$\xi_a := \kappa_{ib} \cdot \Delta(T^{ib}, a) + \kappa_{rb} \cdot \Delta(T^{rb}, a) + \kappa_{bs} \cdot \Delta(T^{bs}, a) \ , \tag{7}$$

where κ_{ib} is the weight of solution T^{ib}, κ_{rb} the one of solution T^{rb}, and κ_{bs} the one of solution T^{bs}. Moreover, $\Delta(T, a)$ evaluates to 1 if and only if arc a is a component of arborescence T. Otherwise, the function evaluates to 0. Note also that the three weights must be chosen such that $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. After the application of Equation 6, pheromone values that exceed $\tau_{\max} = 0.99$ are set back to $\tau_{\max}$, and pheromone values that have fallen below $\tau_{\min} = 0.01$ are set back to $\tau_{\min}$. This prevents the algorithm from reaching a state of complete convergence. Finally, note that the exact values of the weights depends on the convergence factor cf and on the value of the Boolean control variable *bs_update*. The standard schedule as shown in Table 1 has been adopted for our algorithm.

ComputeConvergenceFactor($\mathcal{T}$): The convergence factor cf is computed on the basis of the pheromone values:

$$cf := 2 \left(\left(\frac{\sum_{\tau_a \in \mathcal{T}} \max\{\tau_{\max} - \tau_a, \tau_a - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right)$$

This results in $cf = 0$ when all pheromone values are set to 0.5. On the other side, when all pheromone values have either value $\tau_{\min}$ or $\tau_{\max}$, then $cf = 1$. In all other cases, cf has a value in $(0, 1)$. This completes the description of all components of the proposed algorithm, which is henceforth labelled ACO.

4 Experimental Evaluation

The algorithms proposed in this work—that is, DP-HEUR and ACO—were implemented in ANSI C++ using GCC 4.4 for compiling the software. Moreover, we reimplemented the heuristic proposed in [10]. As mentioned in the introduction, this heuristic—henceforth labelled VENSRI—is the only existing algorithm which can directly be applied to the MWRA problem. All three algorithms were experimentally evaluated on a cluster of PCs equipped with Intel Xeon X3350 processors with 2667 MHz and 8 Gigabyte of memory. In the following, we first describe the set of benchmark instances that have been used to test the three algorithms. Afterwards, the experimental results are described in detail.

4.1 Benchmark Instances

Due to the lack of a publicly availabe set of benchmark instances, a benchmark set was generated. The construction of each DAG $G(V, A)$ from this benchmark set was based on a pre-defined number of vertices (n) and a pre-defined number of arcs (m). First, a random arborescence T with n vertices was generated. The root node of T is called v_r. Each one of the remaining $m-n+1$ arcs was generated by randomly choosing two vertices v_i and v_j, and adding the corresponding arc $a = (v_i, v_j)$ to T. In this context, $a = (v_i, v_j)$ may be added to T, if and only if by its addition no directed cycle is produced, and neither (v_i, v_j) nor (v_j, v_i) form already part of the graph. In order to generate a diverse set of benchmark instances we considered $n \in \{20, 50, 100, 500, 1000, 5000\}$ and $m \in \{2n, 4n, 6n\}$. A total of 10 problem instances was generated for each combination of n and m. This resulted in a total of 180 problem instances. The arc weights for all instances were chosen uniformly at random from $[-100, 100]$.

4.2 Results

The three algorithms considered for the comparison were applied exactly once to each of the 180 problem instances of the benchmark set. Although ACO is a stochastic search algorithm, this is a valid choice, because results are averaged over groups of instances that were generated with the same parameters. ACO was applied with $n_a = 10$—that is, 10 solution constructions per iteration—, with a determinism rate of $\delta = 0.9$, and with a stopping criterion of 10.000 solution evaluations per run. Table 2 presents the results of each algorithm averaged over the 10 instances for each combination of n and m (as indicated in the first two table columns). Four table columns are used for presenting the results of each algorithm. The column with heading **value** provides the average of the objective

Table 2. Experimental results. ACO is compared to the heuristic proposed in this work (DP-HEUR), and the algorithm from [10] (VENSRI).

n	m	DP-HEUR				VENSRI				ACO				
		value	**std**	**size**	**time (s)**	**value**	**std**	**size**	**time (s)**	**value**	**std**	**size**	**evals**	**time (s)**
20	$2n$	-524.50	(134.16)	12.60	< 0.01	-569.10	(156.69)	14.90	< 0.01	**-605.20**	(162.61)	14.30	394.80	1.10
	$4n$	-831.60	(230.68)	15.90	< 0.01	-806.30	(108.14)	17.40	< 0.01	**-996.60**	(153.12)	17.30	5243.80	1.13
	$6n$	-1031.10	(197.50)	17.70	< 0.01	-947.10	(151.05)	17.80	< 0.01	**-1196.50**	(151.63)	17.90	2666.20	1.26
50	$2n$	-1246.30	(273.88)	33.60	< 0.01	-1476.70	(295.11)	38.50	< 0.01	**-1571.00**	(288.52)	38.90	4635.00	4.41
	$4n$	-1912.30	(432.79)	39.70	< 0.01	-1812.30	(208.43)	43.80	< 0.01	**-2404.60**	(312.18)	43.40	7093.40	4.86
	$6n$	-2372.70	(368.03)	43.60	< 0.01	-2166.10	(307.75)	45.70	< 0.01	**-2884.90**	(251.08)	44.70	7474.40	5.00
100	$2n$	-2523.10	(442.91)	67.10	< 0.01	-2828.70	(409.73)	76.20	0.01	**-3130.40**	(445.62)	75.00	5714.70	19.21
	$4n$	-3903.00	(659.69)	82.30	< 0.01	-3871.70	(305.29)	89.90	0.02	**-4955.90**	(321.75)	88.90	8204.40	17.68
	$6n$	-4819.40	(582.18)	87.30	< 0.01	-4059.70	(374.22)	93.10	0.02	**-5782.70**	(391.22)	90.60	7642.70	18.14
500	$2n$	-12404.50	(1308.74)	348.90	0.06	-14085.50	(608.59)	398.70	2.12	**-15489.00**	(637.26)	378.00	8536.60	460.25
	$4n$	-18321.80	(2222.19)	402.00	0.06	-17256.00	(703.46)	449.20	2.28	**-22644.80**	(1537.49)	437.90	8902.20	675.37
	$6n$	-22386.60	(2202.23)	434.90	0.06	-18896.40	(739.65)	471.60	2.38	**-27279.50**	(446.92)	458.10	8620.70	688.30
1000	$2n$	-24493.80	(1577.30)	671.60	0.23	-26995.80	(995.40)	770.10	17.40	**-29915.40**	(1268.64)	742.90	9451.90	3016.34
	$4n$	-37715.40	(3030.59)	811.80	0.23	-34317.50	(1461.89)	905.10	18.69	**-45489.80**	(1463.91)	876.80	8332.50	4948.00
	$6n$	-45280.10	(2376.76)	875.00	0.27	-36790.50	(846.78)	941.40	19.41	**-54352.10**	(1001.77)	920.00	7409.30	4726.81
5000	$2n$	-119122.90	(4980.74)	3371.60	5.23	-135333.80	(2296.56)	3921.10	2440.70	**-146081.80**	(2377.79)	3758.50	8532.80	174329.90
	$4n$	-177605.60	(7388.53)	4045.10	6.42	-163385.60	(2153.92)	4550.00	2585.65	**-216564.20**	(4425.38)	4372.50	8592.40	321099.20
	$6n$	-217112.00	(12667.37)	4325.60	7.29	-171483.70	(2839.81)	4707.20	2679.99	**-258965.20**	(3947.91)	4566.70	8176.80	354718.90

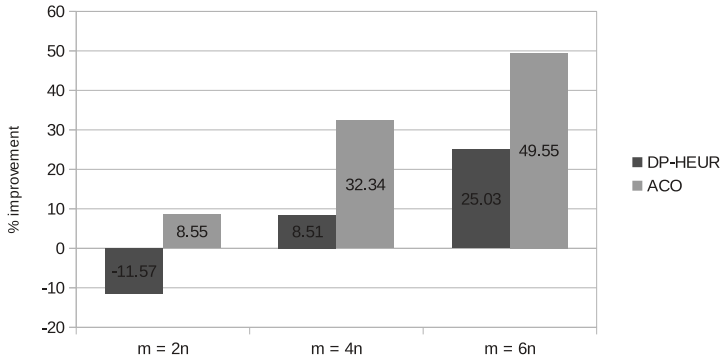


Fig. 3. Average improvement (in %) of ACO and DP-HEUR over VENSRI. Positive values correspond to an improvement, while negative values indicate that the respective algorithm is inferior to VENSRI. The improvement is shown for the three different arc-densities that are considered in the benchmark set, that is, $m = 2n$, $m = 4n$, and $m = 6n$.

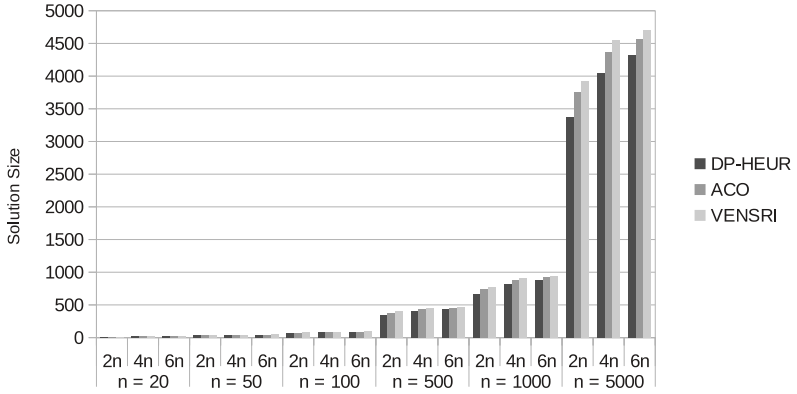
function values of the best solutions found by the respective algorithm for the 10 instances of each combination of n and m. The second column (with heading **std**) contains the corresponding standard deviation. The third column (with heading **size**) indicates the average size (in terms of the number or arcs) of the best solutions found by the respective algorithm.[1] Finally, the fourth column (with heading **time (s)**) contains the average compution time (in seconds). For all three algorithms, the computation time indicates the time of the algorithm termination. In the case of ACO, an additional table column (with heading **evals**) indicates at which solution evaluation, on average, the best solution of a run was found. Finally, for each combination of n and m, the result of the best-performing algorithm is indicated in bold font.

The results allow to make the following observations. First, ACO is for all combinations of n and m the best-performing algorithm. Averaged over all problem instances ACO obtains an improvement of 31.9% over VENSRI. Figure 3 shows the average improvement of ACO over VENSRI for three groups of input instances concerning the different arc-densities. It is interesting to observe that the advantage of ACO over VENSRI seems to grow when the arc-density increases. On the downside, these improvements are obtained at the cost of a significantly increased computation time. Concerning heuristic DP-HEUR, we can observe that it improves in all 12 combinations of n and m where $m \in \{4n, 6n\}$ over VENSRI. Interestingly, however, DP-HEUR is inferior to VENSRI for all combinations with $m = 2n$. In other words, DP-HEUR seems to be inferior to VENSRI when rather sparse input graphs are concerned, whereas the opposite is the case for more

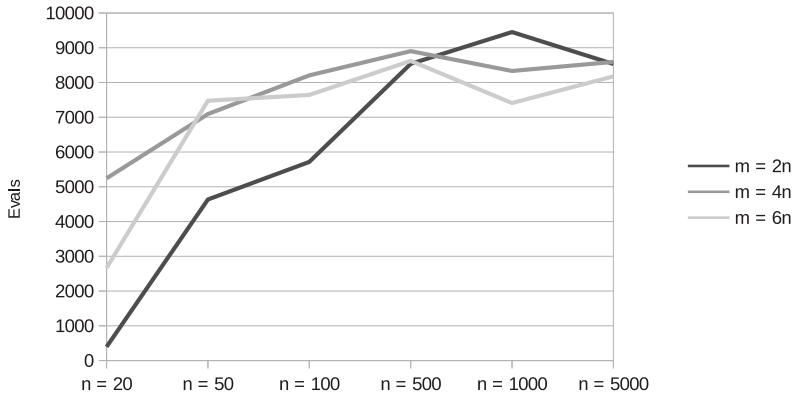
[1] Remember that solutions—that is, arborescences—may have any number of arcs between 0 and $|V| - 1$, where $|V|$ is the number of the input DAG $G = (V, A)$.

dense input graphs. Averaged over all problem instances, DP-Heur obtains an improvement of 8.87% over VenSri. The average improvement of DP-Heur over VenSri is shown for the three groups of input instances concerning the different arc-densities in Figure 3. Concerning a comparison of the computation times, we can state that DP-Heur has a clear advantage over VenSri especially for large-size problem instances.

Figure 4 presents the information which is contained in the columns of Table 2 that have headings **size** and **evals**. Concerning the average size of the solutions



(a) Average solution size



(b) Average number of solution evaluations at which the best solution of an Aco run is found

Fig. 4. (a) shows, for each combination of n and m, information about the average size—in terms of the number of arcs—of the solutions produced by DP-Heur, Aco, and VenSri. (b) shows for each combination of n and m the average number of solution evaluations at which the best solution of a run of Aco is found.

produced by the three algorithms (as shown in Figure 4(a)) it is interesting to observe that the solutions produced by DP-Heur consistently seem to be the smallest ones, while the solutions produced by VenSri seem generally to be the largest ones. The size of the solutions produced by Aco is generally inbetween these two extremes. We currently have no explanation for this aspect, which certainly deserves further examination.

Finally, Figure 4(b) presents the average number of solution evaluations at which the best solution of a run of Aco is found. Not surprisingly, when large graphs are concerned, significantly more solution evaluations are necessary for reaching the best solutions than when rather small graphs are tackled. Concerning a comparison between the groups of graphs characterized by different arc-densities, it can be observed that when rather small graphs are concerned Aco seems to faster in obtaining good solutions for sparse graphs. However, when the size of the input graph grows, this difference disappears.

5 Conclusions and Future Work

In this work we have proposed a heuristic and an ant colony optimization approach for the minimum-weight rooted arboresence problem. Both algorithms make use of dynamic programming as sub-ordinate procedure. Therefore, they may be regarded as hybrid algorithms. The experimental results show that both approaches improve (on average) over an existing heuristic from the literature. Interestingly, the advantage of the proposed algorithm over the existing heuristic grows with increasing arc-density of the input graph.

Concerning future work, we plan to apply both approaches to other types of problem instances. For example, we plan to generate problem instances in which the number of arcs with negative weights is significantly higher than the number of arcs with positive weights, or vice versa. Moreover, we plan to implement an integer programming model for the tackled problem—in the line of the model proposed in [5] for a related problem—and to solve the model with an efficient integer programming solver. In [7] we already proposed an extension of this model for the problem of reconstructing tree structures.

Acknowledgments. This work was supported by grant TIN2007-66523 (FORMALISM) of the Spanish government.

References

1. Blum, C.: Revisiting dynamic programming for finding optimal subtrees in trees. European Journal of Operational Research 177(1), 102–114 (2007)
2. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man and Cybernetics – Part B 34(2), 1161–1172 (2004)
3. Blum, C., Puchinger, J., Raidl, G., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11(6), 4135–4151 (2011)

4. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
5. Duhamel, C., Gouveia, L., Moura, P., Souza, M.: Models and heuristics for a minimum arborescence problem. Networks 51(1), 34–47 (2008)
6. Stützle, T., Hoos, H.H.: $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
7. Türetken, E., Benmansour, F., Fua, P.: Automated reconstruction of tree structures using path classifiers and mixed integer programming. In: Proceedings of CVPR 2012 – 25th IEEE Conference on Computer Vision and Pattern Recognition. IEEE Press (in press, 2012)
8. Türetken, E., González, G., Blum, C., Fua, P.: Automated reconstruction of dendritic and axonal trees by global optimization with geometric priors. Neuroinformatics 9(2-3), 279–302 (2011)
9. Tutte, W.T.: Graph Theory. Cambridge University Press, Cambridge (2001)
10. Venkata Rao, V., Sridharan, R.: Minimum-weight rooted not-necessarily-spanning arborescence problem. Networks 39(2), 77–87 (2002)