68. Ant Colony Optimization for the Minimum-Weight Rooted Arborescence Problem

Christian Blum, Sergi Mateo Bellido

The minimum-weight rooted arborescence problem is an *NP*-hard combinatorial optimization problem which has important applications, for example, in computer vision. An example of such an application is the automated reconstruction of consistent tree structures from noisy images. In this chapter, we present an ant colony optimization approach to tackle this problem. Ant colony optimization is a metaheuristic which is inspired by the foraging behavior of ant colonies. By means of an extensive computational evaluation, we show that the proposed approach has advantages over an existing heuristic from the literature, especially for what concerns rather dense graphs.

68.1 Introductiory Remarks

Solving combinatorial optimization problems with approaches from the swarm intelligence field has already a considerably long tradition. Examples of such approaches include particle swarm optimization (PSO) [68.1] and artificial bee colony (ABC) optimization [68.2]. The oldest – and most widely used – algorithm from this field, however, is ant colony optimization (ACO) [68.3]. In general, the ACO metaheuristic attempts to solve a combinatorial optimization problem by iterating the following steps: (1) Solutions to the problem at hand are constructed using a pheromone model, that is, a parameterized probability distribution over the space of all valid solutions, and (2) (some of) these solutions are used to change the pheromone values in a way being aimed at biasing subsequent sampling toward areas of the search space containing high quality solutions. In particular, the reinforcement of solution components depending on the quality of the solutions in which they appear is an important aspect of ACO algorithms. It is implicitly assumed that good solutions consist of good solution components. To learn which components contribute to good solutions most often helps assembling them into better solutions.

In this chapter, ACO is applied to solve the minimum-weight rooted arborescence (MWRA) problem, which has applications in computer vision such as, for example, the automated reconstruction of consistent tree structures from noisy images [68.4]. The structure of this chapter is as follows. Section 68.2 provides a detailed description of the problem to be tackled. Then, in Sect. 68.3 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 (DP) for finding the best solution within this constructed arborescence. The second contribution is to be found in the application of ACO [68.3] to the MWRA problem. This algorithm is described in Sect. 68.4. Finally, in Sect. 68.5 an exhaustive experimental evaluation of both algorithms in comparison with an existing heuristic from the literature [68.5] is presented. The chapter is concluded in Sect. 68.6.

68.2 The Minimum-Weight Rooted Arborescence Problem

As mentioned before, in this work we consider the MWRA problem, which is a generalization of the problem proposed by *Venkata Rao* and *Sridharan* in [68.5, 6]. The MWRA problem 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 vertex*. 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 [68.7]). Moreover, note that $\mathcal{A}$ contains all arborescences, not only those with maximal size. The objective function value (that is, the weight) $f(T)$ of an arboresence $T \in \mathcal{A}$ is defined as follows:

$$f(T) := \sum_{a \in T} w(a) \,. \tag{68.1}$$

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 Fig. 68.1.

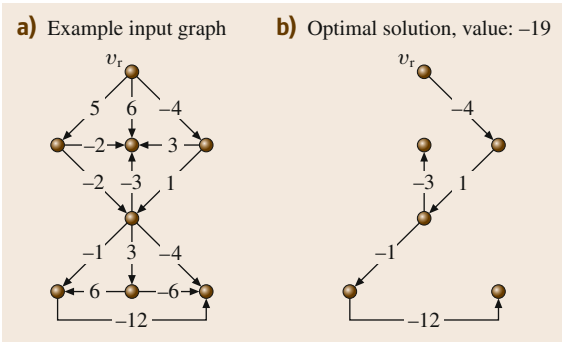

Fig. 68.1a,b **(a)** An input DAG with eight vertices and 14 arcs. The uppermost vertex is the root vertex v_r. **(b)** The optimal solution, that is, the arborescence rooted in v_r which has the minimum weight among all arborescence rooted in v_r that can be found in the input graph

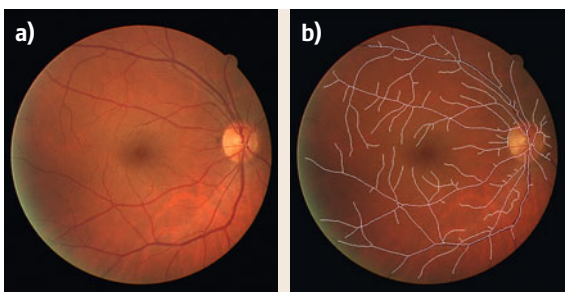

Fig. 68.2a,b **(a)** A 2D image of the retina of a human eye. The problem consists in the automatic reconstruction (or delineation) of the vascular structure. **(b)** The reconstruction of the vascular structure as produced by the algorithm proposed in [68.4]

The differences to the problem proposed in [68.5] are as follows. The authors of [68.5] require the root vertex 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 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 [68.5], the MWRA problem is NP-hard. Concerning the existing work, the literature only offers the heuristic proposed in [68.5], which can also be applied to the more general MWRA problem.

The definition of the MWRA problem as previously outlined is inspired by a novel method which was recently proposed in [68.4] 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 [68.4] 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 (Fig. 68.2, for 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.

68.3 DP-Heur: A Heuristic Approach to the MWRA Problem

In this section, we propose a new heuristic approach for solving the MWRA problem. First, starting from the root vertex v_r, a spanning arborescence T' in G is constructed as outlined in lines 2–9 of Algorithm 68.1. Second, a 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 [68.8] 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 $|V| - 1$ arcs (where V is the vertex set of the input graph G), the DP algorithm is applied with $|V| - 1$. Then, among all the minimum-weight arborescences rooted in v_r for $l \leq |V| - 1$, 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 labeled DP-Heur. As a final remark, let us mention that for the description of this heuristic, it was assumed that the input graph is connected. Appropriate changes have to be applied to the description of the heuristic if this is not the case.

Algorithm 68.1 Heuristic DP-Heur for the MWRA problem

1: **input:** a DAG $G = (V, A)$, and a root node v_r
2: $T'_0 := (V'_0 = \{v_r\}, A'_0 = \emptyset)$
3: $A_{\text{pos}} := \{a = (v_q, v_l) \in A \mid v_q \in V'_0, v_l \notin V'_0\}$
4: **for** $i = 1, \ldots, |V| - 1$ **do**
5: $\quad a^* = (v_q, v_l) := \text{argmin}\{w(a) \mid a \in A_{\text{pos}}\}$
6: $\quad A'_i := A'_{i-1} \cup \{a^*\}$
7: $\quad V'_i := V'_{i-1} \cup \{v_l\}$
8: $\quad T'_i := (V'_i, A'_i)$
9: $\quad A_{\text{pos}} := \{a = (v_q, v_l) \in A \mid v_q \in V'_i, v_l \notin V'_i\}$
10: **end for**
11: $T :=$ Dynamic_Programming$(T'_{|V|-1}, k = |V| - 1)$
12: **output:** arborescence T

68.4 Ant Colony Optimization for the MWRA Problem

The ACO approach for the MWRA problem which is described in the following is a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System (*MM*AS) [68.9] implemented in the hyper-cube framework (HCF) [68.10]. The algorithm, whose pseudocode can be found in Algorithm 68.2, works roughly as follows. At each iteration, a number of n_a solutions to the problem is probabilistically constructed based on both pheromone and heuristic information. 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.

Algorithm 68.2 Ant Colony Optimization for the MWRA Problem

1: **input:** a DAG $G = (V, A)$, and a root node v_r
2: $T^{\text{bs}} := (\{v_r\}, \emptyset)$, $T^{\text{rb}} := (\{v_r\}, \emptyset)$, $cf := 0$, *bs_update* := **false**
3: $\tau_a := 0.5$ for all $a \in A$
4: **while** termination conditions not met **do**
5: $\quad S := \emptyset$
6: $\quad$ **for** $i = 1, \ldots, n_a$ **do**
7: $\quad\quad T :=$ Construct_Solution(G, v_r)
8: $\quad\quad S := S \cup \{T_i\}$
9: $\quad$ **end for**
10: $\quad T^{\text{ib}} := \text{argmin}\{f(T) \mid T \in S\}$
11: $\quad$ **if** $T^{\text{ib}} < T^{\text{rb}}$ **then** $T^{\text{rb}} := T^{\text{ib}}$
12: $\quad$ **if** $T^{\text{ib}} < T^{\text{bs}}$ **then** $T^{\text{bs}} := T^{\text{ib}}$

13: ApplyPheromoneUpdate (cf,bs_update,$\mathcal{T}$,T^{ib},T^{rb},T^{bs})
14: $cf := \text{ComputeConvergenceFactor}(\mathcal{T})$
15: **if** $cf > 0.99$ **then**
16: **if** $bs\_update =$ **true then**
17: $\tau_a := 0.5$ for all $a \in A$
18: $T^{\text{rb}} := (\{v_r\}, \emptyset)$
19: $bs\_update :=$ **false**
20: **else**
21: $bs\_update :=$ **true**
22: **end if**
23: **end if**
24: **end while**
25: **output:** T^{bs}, the best solution found by the algorithm

Construct_Solution(G, v_r): This function, first, constructs a spanning arborescence T' in the way which is shown in lines 2–9 of Algorithm 68.1. However, the choice of the next arc to be added to the current arborescence at each step (see line 5 of Algorithm 68.1) is done in a different way. Instead of deterministically choosing from A_{pos}, the arc which has the smallest weight value, the choice is done probabilistically, 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

$$w_{\max} := \max\{w(a) \mid a \in A\} . \tag{68.2}$$

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

$$\eta(a) := w_{\max} + 1 - w(a) . \tag{68.3}$$

In this way, the heuristic information of all arcs is a positive number. Moreover, the arc with minimal weight will have the highest value concerning the heuristic information. Given an arborescence T'_i (obtained after the ith construction step), and the nonempty set of arcs A_{pos} that may be used for extending T'_i, the probability for choosing arc $a \in A_{\text{pos}}$ is defined as follows

$$\mathbf{p}(a \mid T'_i) := \frac{\tau_a \cdot \eta(a)}{\sum_{\hat{a} \in A_{\text{pos}}} \tau_{\hat{a}} \cdot \eta(\hat{a})} . \tag{68.4}$$

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 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

$$a^* := \text{argmax}\{\mathbf{p}(a \mid T'_i) \mid a \in A_{\text{pos}}\} . \tag{68.5}$$

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

The output T of the function Construct_Solution(G, v_r) is chosen to be the minimum-weight arborescence which is encountered during the process of constructing T', that is,

$$T := \text{argmin}\{f(T'_i) \mid i = 0, \ldots, |V| - 1\} .$$

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:

$$\tau_a := \tau_a + \rho \cdot (\xi_a - \tau_a) , \tag{68.6}$$

where

$$\xi_a := \kappa_{\text{ib}} \cdot \Delta(T^{\text{ib}}, a) + \kappa_{\text{rb}} \cdot \Delta(T^{\text{rb}}, a) + \kappa_{\text{bs}} \cdot \Delta(T^{\text{bs}}, a) , \tag{68.7}$$

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 $\kappa_{\text{ib}} + \kappa_{\text{rb}} + \kappa_{\text{bs}} = 1$. After the application

Table 68.1 Setting of κ_{ib}, κ_{rb}, and κ_{bs} depending on the convergence factor cf and the Boolean control variable *bs_update*

	bs_update = FALSE			*bs_update*
	cf < 0.7	cf ∈ [0.7, 0.9)	cf ≥ 0.9	TRUE
κ_{ib}	2/3	1/3	0	0
κ_{rb}	1/3	2/3	1	0
κ_{bs}	0	0	0	1

of (68.6), pheromone values that exceed $\tau_{max} = 0.99$ are set back to τ_{max}, and pheromone values that have fallen below $\tau_{min} = 0.01$ are set back to τ_{min}. This prevents the algorithm from reaching a state of complete convergence. Finally, note that the exact values of the weights depend on the convergence factor *cf* and on the value of the Boolean control variable *bs_update*. The standard schedule as shown in Table 68.1 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 τ_{min} or τ_{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 labeled ACO.

68.5 Experimental Evaluation

The algorithms proposed in this chapter – that is, DP-HEUR and ACO – were implemented in ANSI C++ using GCC 4.4 for compiling the software. Moreover, the heuristic proposed in [68.5] was reimplemented. As mentioned before, this heuristic – henceforth labeled 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 Gb of memory. In the following, we first describe the set of benchmark instances that have been used to test the three algorithms. Afterward, the algorithm tuning and the experimental results are described in detail.

68.5.1 Benchmark Instances

A diverse set of benchmark instances was generated in the following way. Three parameters are necessary for the generation of a benchmark instance $G = (V, A)$. Hereby, n and m indicate, respectively, the number of vertices and the number of arcs of G, while $q \in [0, 1]$ indicates the probability for the weight of any arc to be positive (rather than negative). The process of the generation of an instance starts by constructing a random arborescence T with n vertices. The root vertex of T is called v_r. Each 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. The weight of each arc was chosen by, first, deciding with probability q if the weight is to be positive (or nonpositive). In the case of a positive weight, the weight value was chosen uniformly at random from $[1, 100]$, while in the case of a nonpositive weight, the weight value was chosen uniformly at random from $[-100, 0]$.

In order to generate a diverse set of benchmark instances, the following values for n, m, and q were considered:

- $n \in \{20, 50, 100, 500, 1000, 5000\}$;
- $m \in \{2n, 4n, 6n\}$;
- $q \in \{0.25, 0.5, 0.75\}$.

For each combination of n, m, and q, a total of 10 problem instances were generated. This resulted in a total of 540 problem instances, that is, 180 instances for each value of q.

68.5.2 Algorithm Tuning

The proposed ACO algorithm has several parameters that require appropriate values. The following parameters, which are crucial for the working of the algorithm, were chosen for tuning:

- $n_a \in \{3, 5, 10, 20\}$: the number of ants (solution constructions) per iteration;
- $\rho \in \{0.05, 0.1, 0.2\}$: the learning rate;
- $\delta \in \{0.0, 0.4, 0.7, 0.9\}$: the determinism rate.

We chose the first problem instance (out of 10 problem instances) for each combination of n, m, and q for tuning. A full factorial design was utilized. This means that ACO was applied (exactly once) to each of the problem instances chosen for tuning. The stopping criterion was fixed to 20 000 solution evaluations for each application of ACO. For analyzing the results, we used

a rank-based analysis. However, as the set of problem instances is quite diverse, this rank-based analysis was performed separately for six subsets of instances. For defining these subsets, we refer to the instances with $n \in \{20, 50, 100\}$ as *small instances*, and the remaining ones as *large instances*. With this definition, each of the three subsets of instances concerning the three different values for q, was further separated into two subsets concerning the instance size. For each of these six subsets, we used the parameter setting with which ACO achieved the best average rank for the corresponding tuning instances. These parameter settings are given in Table 68.2.

68.5.3 Results

The three algorithms considered for the comparison were applied exactly once to each of the 540 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. As in the case of the tuning experiments, the stopping criterion for ACO was fixed to 20 000 solution evaluations. Tables 68.3–68.5 present the results averaged – for each algorithm – 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 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 (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 $G = (V, A)$). Finally, the fourth column (with heading *time (s)*) contains the average computation 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.

Table 68.2 Parameter setting (concerning ACO) used for the final experiments

	$q = 0.25$	$q = 0.5$	$q = 0.75$
Small instances	$n_a = 20$ $\rho = 0.2$ $\delta = 0.7$	$n_a = 20$ $\rho = 0.2$ $\delta = 0.7$	$n_a = 5$ $\rho = 0.05$ $\delta = 0.4$
Large instances	$n_a = 20$ $\rho = 0.2$ $\delta = 0.9$	$n_a = 20$ $\rho = 0.2$ $\delta = 0.9$	$n_a = 20$ $\rho = 0.2$ $\delta = 0.9$

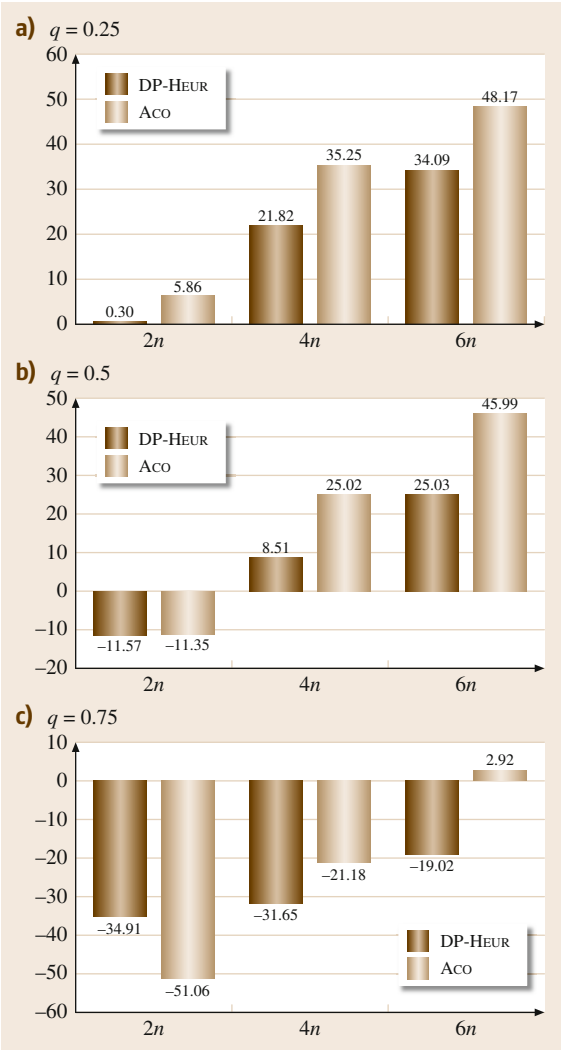

Fig. 68.3a–c Average improvement (in %) of ACO and DP-HEUR over VENSRI. Positive values correspond to an improvement, while negative values indicate that the respective algorithm is inferior to VENSRI. The improvement is shown for the three different arc-densities that are considered in the benchmark set, that is, $m = 2n$, $m = 4n$, and $m = 6n$

Table 68.3 Experimental results for the 180 instances with $q = 0.25$. ACO is compared to the heuristic proposed in this work (DP-HEUR), and the algorithm from [68.5] (VENSRI)

n	m	DP-HEUR				VENSRI				ACO				
		Value	Std	Size	Time (s)	Value	Std	Size	Time (s)	Value	Std	Size	Evals	Time (s)
20	$2n$	−855.40	(133.68)	17.00	< 0.01	−845.60	(150.48)	17.40	< 0.01	**−962.70**	(147.81)	17.70	2543.70	0.62
		−1093.10	(155.68)	18.10	< 0.01	−999.20	(85.56)	18.30	< 0.01	**−1248.10**	(113.66)	18.10	4435.00	0.74
	$6n$	−1138.20	(201.72)	17.70	< 0.01	−1024.00	(131.08)	18.10	< 0.01	**−1321.00**	(139.52)	18.30	5904.00	0.80
50	$2n$	−2089.30	(250.78)	42.10	< 0.01	−2066.70	(284.37)	44.40	< 0.01	**−2379.10**	(237.14)	44.00	6522.60	1.21
	$4n$	−3022.80	(258.93)	47.00	< 0.01	−2560.10	(266.46)	47.60	< 0.01	**−3314.40**	(188.43)	47.60	9299.60	1.41
	$6n$	−3281.40	(221.73)	48.20	< 0.01	−2522.60	(215.18)	48.60	< 0.01	**−3620.30**	(131.79)	48.50	13 101.90	1.58
100	$2n$	−4260.00	(402.24)	87.70	< 0.01	−4325.40	(326.54)	91.20	0.01	**−4944.00**	(436.59)	90.20	10 949.00	2.19
	$4n$	−5386.50	(364.51)	92.70	< 0.01	−4598.90	(209.84)	95.70	0.01	**−6161.10**	(291.93)	94.80	16 187.30	2.99
	$6n$	−6222.80	(315.85)	95.30	< 0.01	−5015.80	(304.61)	96.40	0.01	**−7048.60**	(229.84)	96.20	17 028.70	3.66
500	$2n$	−20 899.10	(1169.37)	438.70	0.06	−21 040.80	(625.26)	461.50	1.79	**−23 501.40**	(796.48)	454.70	17 159.70	30.52
	$4n$	−27 856.50	(1275.43)	473.30	0.07	−22 648.40	(696.45)	484.70	1.78	**−30 853.00**	(908.66)	480.00	18 153.90	48.36
	$6n$	−30 372.50	(889.92)	482.00	0.06	−23 487.20	(717.32)	490.00	1.83	**−33 892.20**	(793.20)	485.10	16 940.30	56.65
1000	$2n$	−42 022.20	(2058.38)	876.60	0.24	−41 817.70	(1178.70)	922.60	14.21	**−45 498.70**	(1950.29)	916.20	17 749.30	110.68
	$4n$	−53 728.80	(1987.03)	946.90	0.26	−44 435.50	(1431.45)	971.80	14.64	**−60 574.80**	(960.59)	960.10	16 568.40	172.62
	$6n$	−61 829.30	(1621.11)	965.70	0.25	−45 723.40	(884.88)	981.90	14.44	**−67 974.40**	(1054.47)	973.30	16 376.90	242.84
5000	$2n$	−206 085.80	(2554.80)	4365.00	4.99	−205 287.40	(2691.58)	4591.30	1939.72	**−214 243.30**	(2850.45)	4576.90	15 441.40	2715.48
	$4n$	−266 109.50	(8892.83)	4728.30	6.51	−217 967.60	(1604.04)	4857.60	2009.05	**−294 401.20**	(4810.59)	4818.90	14 802.70	4518.43
	$6n$	−297 185.10	(7903.69)	4815.00	6.41	−220 559.40	(3876.02)	4909.20	2015.12	**−328 173.90**	(4492.29)	4874.30	16 287.20	5752.23

Table 68.4 Experimental results for the 180 instances with $q = 0.5$. ACO is compared to the heuristic proposed in this work (DP-HEUR), and the algorithm from [68.5] (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	**−631.30**	(171.28)	15.00	4766.50	0.62
	$4n$	−831.60	(230.68)	15.90	< 0.01	−806.30	(108.14)	17.40	< 0.01	**−1009.90**	(149.29)	17.30	4812.50	0.72
	$6n$	−1031.10	(197.50)	17.70	< 0.01	−947.10	(151.05)	17.80	< 0.01	**−1210.70**	(152.26)	17.90	3920.30	0.83
50	$2n$	−1246.30	(273.88)	33.60	< 0.01	−1476.70	(295.11)	38.50	< 0.01	**−1584.70**	(294.18)	39.20	9133.00	1.28
	$4n$	−1912.30	(432.79)	39.70	< 0.01	−1812.30	(208.43)	43.80	< 0.01	**−2466.90**	(278.53)	43.50	11 435.80	1.40
	$6n$	−2372.70	(368.03)	43.60	< 0.01	−2166.10	(307.75)	45.70	< 0.01	**−2923.40**	(265.90)	44.90	10 244.00	1.62
100	$2n$	−2523.10	(442.91)	67.10	< 0.01	−2828.70	(409.73)	76.20	0.01	**−3187.60**	(436.84)	74.90	12 870.70	2.33
	$4n$	−3903.00	(659.69)	82.30	< 0.01	−3871.70	(305.29)	89.90	0.02	**−5031.30**	(375.58)	89.20	14 445.10	3.08
	$6n$	−4819.40	(582.18)	87.30	< 0.01	−4059.70	(374.22)	93.10	0.02	**−5867.70**	(423.46)	91.20	16744.20	3.64
500	$2n$	−12 404.50	(1308.74)	348.90	0.06	−14 085.50	(608.59)	398.70	2.12	**−14 381.90**	(750.38)	408.60	18 360.90	31.48
	$4n$	−18 321.80	(2222.19)	402.00	0.06	−17 256.00	(703.46)	449.20	2.28	**−22 194.00**	(1901.21)	447.80	18 880.40	40.71
	$6n$	−22 386.60	(2202.23)	434.90	0.06	−18 896.40	(739.65)	471.60	2.38	**−27 130.80**	(688.55)	458.80	16 207.20	53.55
1000	$2n$	−24 493.80	(1577.30)	671.60	0.23	**−26 995.80**	(995.40)	770.10	17.40	−26 046.20	(1336.03)	786.30	17 380.20	115.79
	$4n$	−37715.40	(3030.59)	811.80	0.23	−34317.50	(1461.89)	905.10	18.69	**−44 061.00**	(1602.97)	893.20	17 454.10	160.78
	$6n$	−45 280.10	(2376.76)	875.00	0.27	−36 790.50	(846.78)	941.40	19.41	**−53 867.00**	(1368.28)	920.20	16 219.20	212.90
5000	$2n$	−119 122.90	(4980.74)	3371.60	5.23	**−135 333.80**	(2296.56)	3921.10	2440.70	−114 887.00	(4657.51)	3733.50	15 223.20	2674.64
	$4n$	−177 605.60	(7388.53)	4045.10	6.42	−163 385.60	(2153.92)	4550.00	2585.65	**−202 096.40**	(7870.45)	4399.70	14 275.70	3982.99
	$6n$	−217 112.00	(12667.37)	4325.60	7.29	−171 483.70	(2839.81)	4707.20	2679.99	**−251 128.00**	(4891.47)	4568.70	16 443.00	4905.29

Table 68.5 Experimental results for the 180 instances with $q = 0.75$. ACO is compared to the heuristic proposed in this work (DP-HEUR), and the algorithm from [68.5] (VENSRI)

n	m	DP-HEUR				VENSRI				ACO				
		Value	Std	Size	Time (s)	Value	Std	Size	Time (s)	Value	Std	Size	Evals	Time (s)
20	$2n$	−186.50	(134.06)	7.80	< 0.01	−229.70	(157.66)	10.00	< 0.01	**−242.20**	(164.32)	9.90	1431.00	1.27
	$4n$	−391.50	(133.13)	9.90	< 0.01	−479.30	(107.17)	12.80	< 0.01	**−560.70**	(106.44)	13.60	6451.30	1.44
	$6n$	−549.10	(212.91)	12.60	< 0.01	−698.10	(174.68)	15.70	< 0.01	**−838.30**	(147.90)	15.70	5398.70	1.51
50	$2n$	−488.00	(170.61)	19.70	< 0.01	−580.80	(167.41)	24.70	< 0.01	**−642.70**	(202.15)	23.80	7511.10	1.74
	$4n$	−861.10	(287.25)	26.60	< 0.01	−1125.50	(160.68)	35.50	< 0.01	**−1336.60**	(146.24)	34.00	10 451.90	1.89
	$6n$	−1231.40	(344.87)	31.20	< 0.01	−1449.90	(110.20)	41.10	< 0.01	**−1817.80**	(239.20)	38.60	11 572.90	2.08
100	$2n$	−907.90	(342.95)	34.10	< 0.01	−1182.90	(384.00)	45.50	0.02	**−1224.50**	(449.23)	45.30	12 586.30	2.66
	$4n$	−1766.80	(226.75)	53.90	< 0.01	−2216.20	(272.32)	70.90	0.02	**−2605.40**	(311.13)	68.20	15 318.80	3.30
	$6n$	−2787.80	(527.96)	67.00	< 0.01	−2938.20	(294.57)	80.50	0.03	**−3811.80**	(416.14)	80.80	16809.60	3.92
500	$2n$	−4647.40	(804.52)	198.00	0.06	**−6495.10**	(828.27)	263.40	2.85	−5183.40	(900.98)	268.40	17 705.80	31.59
	$4n$	−7723.40	(1393.23)	263.30	0.06	**−10 393.40**	(599.96)	367.50	3.39	−10 386.10	(1215.50)	340.20	17 027.00	41.39
	$6n$	−11 642.30	(1255.86)	311.20	0.07	−12 691.90	(608.40)	403.50	3.72	**−15 203.50**	(849.56)	396.80	17 841.20	53.84
1000	$2n$	−8323.40	(1555.82)	367.30	0.23	**−11 908.10**	(1371.03)	483.60	24.04	−8009.90	(1106.18)	473.70	18 558.30	116.37
	$4n$	−14 434.90	(1298.71)	518.90	0.28	**−20 508.10**	(1527.07)	725.20	28.28	−18 301.00	(1484.42)	693.30	16070.90	158.25
	$6n$	−22 564.10	(1500.62)	621.80	0.29	−25 288.30	(629.85)	822.50	30.79	**−28 023.90**	(1994.36)	755.10	16 971.50	194.97
5000	$2n$	−36 776.60	(2336.68)	1831.10	5.43	**−58 465.20**	(1821.57)	2473.00	3222.83	−23 289.00	(1874.45)	1896.70	16 259.30	2710.57
	$4n$	−68 235.80	(9123.95)	2563.50	6.80	**−101 955.80**	(1607.50)	3663.00	3683.96	−74 538.30	(5125.67)	3172.50	16 524.40	3656.49
	$6n$	−96 284.50	(7465.28)	2994.70	4.25	**−123 714.50**	(1310.01)	4120.80	4192.95	−121 960.40	(4233.52)	3707.60	14 190.90	4459.02

Concerning the 180 instances with $q = 0.25$, the results allow us 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 29.8% over VENSRI. Figure 68.3a 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 over VENSRI for all combinations of n and m, apart from $(n = 100, m = 2n)$ and $(n = 500, m = 2n)$. This seems to indicate that, also for DP-HEUR, the sparse instances pose more of a challenge than the dense instances. Averaged over all problem instances, DP-HEUR obtains an improvement of 18.6% 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 Fig. 68.3a. 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.

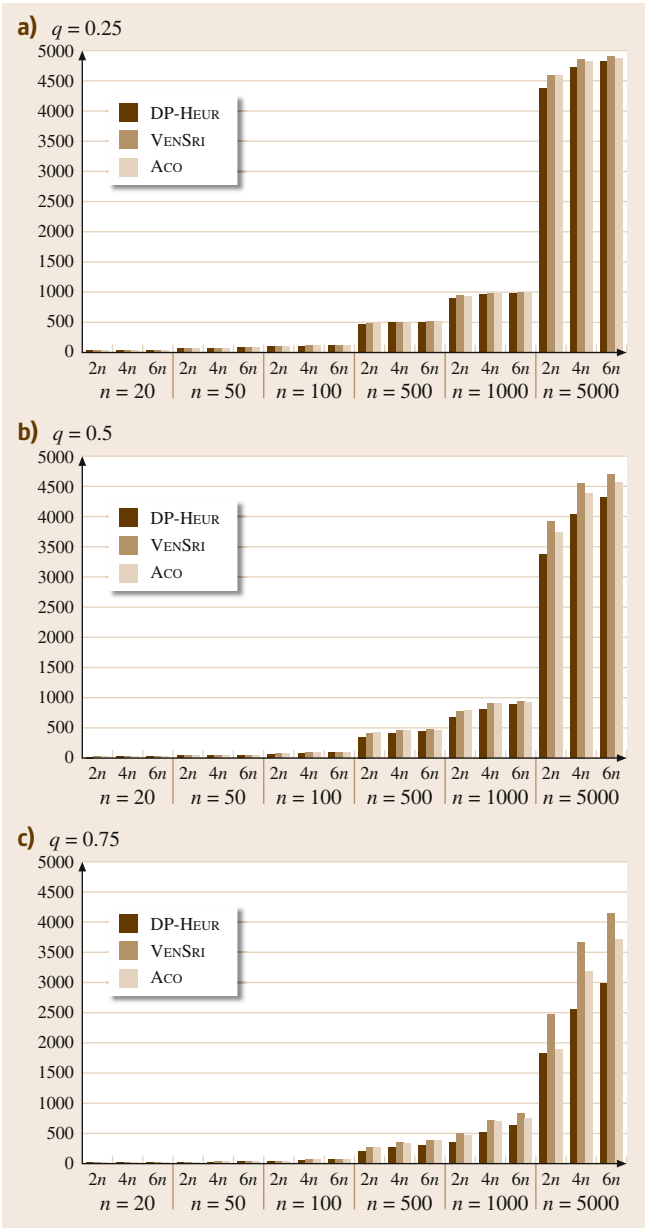

Fig. 68.4 These graphics show, for each combination of n and m, information about the average size – in terms of the number of arcs – of the solutions produced by DP-HEUR, ACO, and VENSRI

Concerning the remaining 360 instances ($q = 0.5$ and $q = 0.75$), we can make the following additional observations. First, both ACO and DP-HEUR seem to experience a downgrade in performance (in comparison to the performance of VENSRI) when q increases. This holds especially for rather large and rather sparse graphs. While both algorithms still obtain an average improvement over VENSRI in the case of $q = 0.5$ – that is, 19.9% improvement in the case of ACO and 7.3% in the case of DP-HEUR – both algorithms are on average inferior to VENSRI in the case of $q = 0.75$.

Finally, Fig. 68.4 presents the information which is contained in column *size* of Tables 68.3–68.5 in graphical form. 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 in between these two extremes. Moreover, with growing q the difference in solution size as produced by the three algorithms seems to be more pronounced. We currently have no explanation for this aspect, which certainly deserves further examination.

68.6 Conclusions and Future Work

In this work, we have proposed a heuristic and an ACO approach for the minimum-weight rooted arboresence problem. The heuristic makes use of dynamic programming as a subordinate procedure. Therefore, it may be regarded as a hybrid algorithm. In contrast, the proposed ACO algorithm is a pure metaheuristic approach. The experimental results show that both approaches are superior to an existing heuristic from the literature in those cases in which the number of arcs with positive weights is not too high and in the case of rather dense graphs. However, as far as sparse graphs with a rather large fraction of positive weights are concerned, the existing heuristic from the literature seems to have advantages over the algorithms proposed in this chapter.

Concerning future work, we plan to develop a hybrid ACO approach which makes use of dynamic programming as a subordinate procedure, in a way similar to the proposed heuristic. Moreover, we plan to implement an integer programming model for the tackled problem – in the line of the model proposed in [68.11] for a related problem – and to solve the model with an efficient integer programming solver.

References

68.1 R. Poli, J. Kennedy, T. Blackwell: Particle swarm optimization – an overview, Swarm Intell. **1**(1), 33–57 (2007)

68.2 M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, A.H.-L. Chen: A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, Inf. Sci. **181**(16), 3459–3475 (2011)

68.3 M. Dorigo, T. Stützle: *Ant Colony Optimization* (MIT, Cambridge 2004)

68.4 E. Türetken, G. González, C. Blum, P. Fua: Automated reconstruction of dendritic and axonal trees by global optimization with geometric priors, Neuroinformatics **9**(2/3), 279–302 (2011)

68.5 V. Venkata Rao, R. Sridharan: Minimum-weight rooted not-necessarily-spanning arborescence problem, Networks **39**(2), 77–87 (2002)

68.6 V. Venkata Rao, R. Sridharan: The minimum weight rooted arborescence problem: Weights on arcs case, IIMA Working Papers WP1992-05-01_01106 (Indian Institute of Management Ahmedabad, Research and Publication Department, Ahmedabad 1992)

68.7 W.T. Tutte: *Graph Theory* (Cambridge Univ. Press, Cambridge 2001)

68.8 C. Blum: Revisiting dynamic programming for finding optimal subtrees in trees, Eur. J. Oper. Res. **177**(1), 102–114 (2007)

68.9 T. Stützle, H.H. Hoos: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system, Future Gener. Comput. Syst. **16**(8), 889–914 (2000)

68.10 C. Blum, M. Dorigo: The hyper-cube framework for ant colony optimization, IEEE Trans. Syst. Man Cybern. B **34**(2), 1161–1172 (2004)

68.11 C. Duhamel, L. Gouveia, P. Moura, M. Souza: Models and heuristics for a minimum arborescence problem, Networks **51**(1), 34–47 (2008)