Ant Colony Optimization for Tree Decompositions

Thomas Hammerl and Nysret Musliu

Institute of Information Systems, Vienna University of Technology, Austria
thomas.hammerl@gmail.com, musliu@dbai.tuwien.ac.at

Abstract. Instances of constraint satisfaction problems can be solved efficiently if they are representable as a tree decomposition of small width. Unfortunately, the task of finding a decomposition of minimum width is NP-complete itself. Therefore, several heuristic and metaheuristic methods have been developed for this problem. In this paper we investigate the application of different variants of Ant Colony Optimization algorithms for the generation of tree decompositions. Furthermore, we extend these implementations with two local search methods and we compare two heuristics that guide the ACO algorithms. Our computational results for selected instances of the DIMACS graph coloring library show that the ACO metaheuristic gives results comparable to those of other decomposition methods such as branch and bound and tabu search for many problem instances. One of the proposed algorithms was even able to improve the best known upper bound for one problem instance. Nonetheless, for some larger problems the best existing methods outperform our algorithms.

1 Introduction

Many constraint satisfaction problems (CSPs) can be solved efficiently if they have a tree decomposition of small width. Each tree decomposition has a characteristic called *width* and each CSP problem can be transformed to many different valid decompositions. The smaller a decomposition's width the faster the solution to the problem can be computed.

To illustrate the application of tree decomposition for solving CSP problems suppose that we have to find solutions for the the graph coloring problem (GCP), which is a well known CSP in the literature. For this problem we have to find a coloring of vertices of a given graph in such a way that no two vertices connected by an edge share the same color. An instance of the GCP is shown on the left side of Figure 1. The task is now to find a valid coloring just using the colors red, green, and blue.

One naive approach to solve this problem might be to try out all possible combinations of variable assignments and see which ones are valid. There are d^n possible combinations in general where d is the number of available colors and n is the number of vertices.

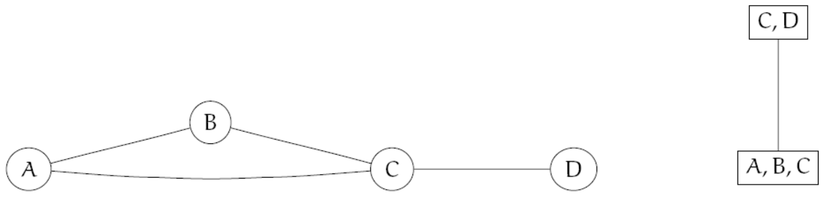


Fig. 1. Instance of the graph coloring problem and one possible tree decomposition

To solve this problem by tree decomposition, we should first generate the tree decompositon of the corrosponding problem graph. The concept of tree decomposition has been introduced first by Robertson and Seymour [14]:

Definition 1. *(see [14], [10]) Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair (T, χ), where $T = (I, F)$ is a tree with node set I and edge set F, and $\chi = \{\chi_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

1. $\bigcup_{i \in I} \chi_i = V$,
2. *for every edge $(v, w) \in E$, there is an $i \in I$ with $v \in \chi_i$ and $w \in \chi_i$, and*
3. *for all $i, j, k \in I$, if j is on the path from i to k in T, then $\chi_i \cap \chi_k \subseteq \chi_j$.*

The width of a tree decomposition is $max_{i \in I} |\chi_i| - 1$. The treewidth of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G.

One possible tree decomposition for our graph coloring problem is shown on the right side of Figure 1. This tree decomposition fulfills all conditions of the previous definition. If we want to solve the graph coloring problem based on this tree decomposition, we can start out by solving the subproblems given by each vertex in the tree decomposition. Using our naive approach of trying out all possible combinations of variable assignments we generate 3^3 (27) different solution candidates for the vertex containing A, B, and C. Because of the constraints $A \neq B$, $A \neq C$, and $B \neq C$ only six of them are valid. For the subproblem containing the vertices C and D we generate 3^2 (9) solution candidates and rule out three of them because of the constraint $C \neq D$. We can now get all solutions to the whole problem by joining the subproblem solutions. Therefore, we will take a look at the variables both subproblems have in common. In this case, that is the variable C. Each solution for the subproblem A, B, C is joined with the solutions for the subproblem C, D sharing the same color for the vertex C. By using the tree decomposition we had to generate 36 combinations of variable assignments in order to determine all solutions compared to the 81 combinations we had to generate without the tree decomposition. This difference increases very quickly with the size of the graph coloring problem and constraint satisfaction problems in general. The smaller the subproblems in the tree decomposition the more efficient we can solve a particular problem. This is why we are interested in finding tree decompositions of small *width*.

Tree decompositions can be generated from a given graph by successive elimination of graph vertices. Each time a vertex is eliminated a new tree node is created that contains the eliminated vertex and its neighbors. Additionally, the neighbors of the eliminated node are connected in the remaining graph. It is guaranteed that there is a so-called *optimal elimination ordering* that yields the tree decomposition with the minimum width of all valid tree decompositions for the given constraint graph. Therefore, one way to generate a tree decomposition of small width is to search for a good ordering of the vertices of the graph. Unfortunately, there are $n!$ different elimination orderings. For that reason not only exact methods but also many approximation algorithms have been applied to the problem of finding tree decompositions of small width.

A complete algorithm for tree decompositions is proposed by Gogate and Dechter [7]. This algorithm applies different pruning techniques, and provides anytime solutions, which are good upper bounds for tree decompositions. Heuristic techniques for the generation of tree decompositions with small width are mainly based on searching for a good elimination ordering of graph nodes. Several heuristics that run in polynomial time have been proposed for finding a good elimination ordering of nodes. These heuristics select the ordering of nodes based on different criteria, such as the degree of the nodes (min degree heuristic), the number of edges to be added to make the node simplicial (min-fill heuristic), connectivity with the vertices previously selected in the elimination ordering (Maximum Cardinality Search (MCS) [17]) etc. For other types of heuristics based on the elimination ordering of nodes see [10].

Metaheuristic approaches have also been used for tree decompositions. Simulated annealing was applied by Kjaerulff [9]. Applications of genetic algorithms for tree decompositions are presented in [11] and [13]. A tabu search approach for generation of the tree decompositions has been proposed by Clautiaux et al [2]. The authors reported good results for DIMACS vertex coloring instances. Their approach improved the previous results in literature for 53% of instances. Some of the results in [2] have been further improved by Gogate and Dechter [7]. Upper bounds for several other problems have been improved by iterated local search algorithm [12].

Ant Colony Optimization (ACO) has not been applied yet for tree decompositions. In this paper we investigate the following variants of ACO algorithms for finding tree decompositions of small width: Simple Ant System ([3], [6]), Elitist Ant System ([3], [6]), Rank-Based Ant System [1], Max-Min Ant System ([15], [16]), and Ant Colony System [4]. We propose two different pheromone update strategies and implement two stagnation measures that indicate the degree of diversity of the solutions constructed by the ants. Furthermore, we implement two constructive heuristics (Min-Degree, Min-Fill) that can be incorporated alternatively into every ACO variant as a guiding function and investigated the combination of ACO with two existing local search methods: Hill Climbing and Iterated Local Search [12]. Finally we compare the results achieved by Ant Colony System for 62 DIMACS graph coloring instances with the results of other state of the art heuristic and exact algorithms.

2 Generation of Tree Decompositions by Ant Colony Optimization

Ant Colony Optimization (*ACO*) is a population-based metaheuristic introduced by Marco Dorigo [3] in 1992. As the name suggests the technique was inspired by the behaviour of "real" ants. Ant colonies are able to find the shortest path between their nest and a food source just by depositing and reacting to pheromones while they are exploring their environment. The basic principles driving this system can also be applied to many combinatorial optimization problems. For a detailed description of different ACO algorithms and their applications the reader is referred to the book *"Ant Colony Optimization"* [5] written by Dorigo and Stützle.

In this section we propose the application of ACO to the problem of finding tree decompositions of small width. We have implemented different ACO variants and combined these variants with different guiding heuristics, local search methods and pheromone update strategies that we will discuss after giving an explanation of the basic structure of the algorithm.

A simple constraint graph and the corresponding ACO construction tree are shown in Figure 2. The construction tree can be obtained from the constraint graph as follows: (1) Create a root node s that will be the starting point of every ant in the colony; (2) For every vertex of the constraint graph append a child node to the root node s; (3) To every leaf node append a child node for every vertex of the constraint graph that is neither represented by the leaf node itself nor by an ancestor of this node; (4) Repeat step 3 until there are no nodes left to append.

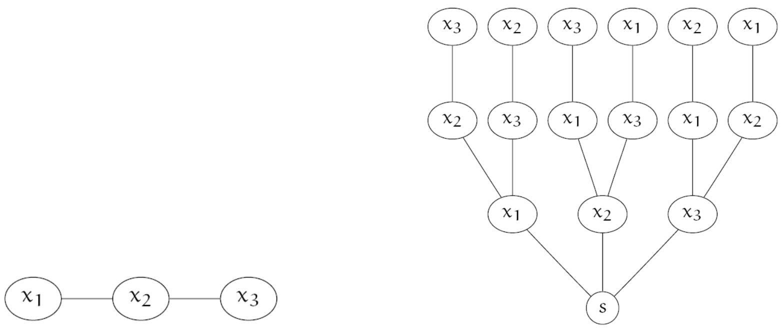


Fig. 2. Constraint graph $\mathcal{G}$ and the ACO construction tree.

All possible elimination orderings for the constraint graph can now be represented as a path from the root node s to one of the leaf nodes in the construction tree. Therefore each of the ants finds such a path and at each node on its way the ant decides where to move next probabilistically based on the pheromone trails and a heuristic value both associated with the outgoing edges.

2.1 Pheromone Trails

A pheromone trail constitutes the desirability to eliminate a certain vertex x after another vertex y. The more pheromone is located on a trail the more likely the corresponding vertex will be chosen by the ant. A way to represent the pheromone trails of our construction tree is the matrix as shown below:

$$\mathcal{T} = \begin{pmatrix} \tau_{x_1x_1} & \tau_{x_1x_2} & \tau_{x_1x_3} \\ \tau_{x_2x_1} & \tau_{x_2x_2} & \tau_{x_2x_3} \\ \tau_{x_3x_1} & \tau_{x_3x_2} & \tau_{x_3x_3} \\ \tau_{sx_1} & \tau_{sx_2} & \tau_{sx_3} \end{pmatrix} \tag{1}$$

Each row contains the amounts of pheromone located on the trails connecting a certain node with all the other nodes. For example, the first row contains the pheromone levels related to the node x_1 describing the desirability of eliminating x_2 ($\tau_{x_1x_2}$) respectively x_3 ($\tau_{x_1x_3}$) immediately after x_1. The last row is dedicated to the root node s that is the starting point for every ant.

All pheromone trails are initialized to the same value in the beginning of the algorithm that is computed according to the following equation:

$$\tau_{ij} = \frac{m}{W_\eta} \qquad \forall \tau_{ij} \in \mathcal{T} \tag{2}$$

W_η is the width of the decomposition obtained using the guiding heuristic (min-degree or min-fill) while m is the size of the ant colony.

2.2 Heuristic Information

The ants make their decision which vertex to eliminate next not solely based on the pheromone matrix but also consider a guiding heuristic. We have implemented two different heuristics. In order to compute both of these heuristic values we need to maintain a separate graph in addition to the construction tree. We will call this graph the *elimination graph* because this graph is obtained from the original constraint graph by successively eliminating the vertices traversed by the ant in the construction tree. Further, we will denote this graph as $E(\mathcal{G}, \sigma)$ where $\mathcal{G}$ is the original constraint graph and σ is a partial elimination ordering.

Min-Degree The value for the min-degree heuristic is computed according to this equation:

$$\eta_{ij} = \frac{1}{d(j, E(\mathcal{G}, \sigma)) + 1} \tag{3}$$

The expression $d(j, E(\mathcal{G}, \sigma)$ represents the degree of vertex j in the elimination graph $E(\mathcal{G}, \sigma)$.

Min-Fill The value for the min-fill heuristic is computed according to this equation:

$$\eta_{ij} = \frac{1}{f(j, E(\mathcal{G}, \sigma)) + 1} \tag{4}$$

The expression $f(j, E(\mathcal{G}, \sigma)$ represents the number of edges that would be added to the elimination graph due to the elimination of vertex j.

2.3 Probabilistic Vertex Elimination

We will now take a more detailed look on how exactly the ants move from node to node on the construction tree. All of the ACO variants with the exception of Ant Colony System use Equation 5 alone to compute the probability p_{ij} of moving from a node i to another node j where α and β are parameters that can be passed to the algorithm in order to weight the pheromone trails and the heuristic values.

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} [\eta_{ij}]^{\beta}}{\sum\limits_{l \in E(\mathcal{G}, \sigma)} [\tau_{il}]^{\alpha} [\eta_{il}]^{\beta}}, \qquad \text{if } j \in E(\mathcal{G}, \sigma) \tag{5}$$

This probability is computed for each vertex left in the elimination graph. According to these probabilities the ant decides which vertex to eliminate next.

Ant Colony System introduces an additional parameter q_0 that constitutes the probability that the ant moves to the "best" node instead of making a probabilistic decision:

$$j = \begin{cases} \arg\max_{l \in E(\mathcal{G}, \sigma)} \{[\tau_{il}]^{\alpha} [\eta_{il}]^{\beta}\}, \text{ if } q \leq q_0; \\ \text{Equation 5, otherwise;} \end{cases} \tag{6}$$

If a randomly generated number q in the interval of $[0, 1]$ is less or equal q_0 then the ant moves to the node that otherwise would have the highest probability to be chosen. Ties are broken randomly.

Ant Colony System also introduces a so-called local pheromone update. After an ant has constructed its solution it removes pheromone from the trails belonging to its solution according to the following equation whereas ξ is a variant-specific parameter and τ_0 is initial amount of pheromone:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{7}$$

The motivation for this is to diversify the search so that subsequent ants will more likely choose other branches of the construction tree.

2.4 Pheromone Update

After each of the ants has constructed an elimination ordering (that optionally has been improved by a local search thereafter) the values in the pheromone matrix are updated reflecting the quality of the constructed solutions which will enable the subsequent ants in the following iteration to make decisions in a more informed manner. Moreover, pheromone is removed from the pheromone trails so poor solutions can be forgotten that might have been the best known solutions in earlier iterations of the algorithm.

Pheromone Deposition Given an elimination ordering σ_k that was constructed by an ant k we need to determine for each subsequent elimination (i, j) in σ_k the amount of pheromone that will be deposited on the corresponding pheromone trail τ_{ij}. We implemented an edge-independent and an edge-specific pheromone update strategy. The first adds the same amount of pheromone to all trails belonging to σ_k while the latter adds more or less pheromone to individual trails depending on the quality of a certain elimination.

The edge-independent pheromone update strategy adds the reciprocal value of the tree decomposition's width to all pheromone trails that are part of σ_k:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{W(\sigma_k)}, \text{ if } (i, j) \text{ belongs to } \sigma_k; \\ 0, \text{ otherwise;} \end{cases} \tag{8}$$

In contrast to the edge-independent update strategy the edge-specific update strategy deposits different amounts of pheromone onto the trails belonging to the same elimination ordering:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{d(j, E(\mathcal{G}, \sigma_{kj}))/|E(\mathcal{G}, \sigma_{kj})|} \cdot \frac{1}{W(\sigma_k)}, \text{ if } (i, j) \text{ belongs to } \sigma_k; \\ 0, \text{ otherwise;} \end{cases} \tag{9}$$

This amount depends on the ratio between the degree of the vertex j when it was eliminated $d(j, E(\mathcal{G}, \sigma_{kj}))$ and the number of vertices left in the elimination graph $|E(\mathcal{G}, \sigma_{kj})|$ at that time.[1]

Which ants are allowed to deposit pheromone and how this pheromone is weighted varies between the different ACO variants. The reader is referred to [5] for description of these variants.

Pheromone Evaporation After the pheromone has been added to the trails a certain amount of pheromone is removed. This amount is determined based on the pheromone evaporation rate ρ:

$$\tau_{ij} = (1 - \rho)\tau_{ij} \qquad \forall \tau_{ij} \in \mathcal{T} \tag{10}$$

While all the other ACO variants remove pheromone from every pheromone trail Ant Colony System only removes pheromone from the trails belonging to the best known elimination ordering σ_{bs}:

$$\tau_{ij} = (1 - \rho)\tau_{ij} \qquad \forall (i, j) \in \sigma_{bs} \tag{11}$$

[1] σ_{kj} is the partial elimination ordering that is obtained from σ_k by omitting j and all vertices that are eliminated after j.

2.5 Local Search

All ACO variants can optionally be extended with one of two local search methods we implemented for tree decompositions. Both of these algorithms try to improve the quality of the solutions that were constructed by the ant colony by changing the positions of certain vertices in the elimination orderings. We used two local search techniques in this paper: an hill climbing algorithm and an iterated local search similar to the algorithm proposed by Musliu [12]. Both of these algorithms are discussed in detail in Section 5.4 of the master's thesis [8] this paper is based on.

2.6 Stagnation Measures

If the distribution of the pheromone on the trails becomes too unbalanced due to the pheromone depositions, the ants will generate very similar solutions causing the search to stagnate. In order to enable the algorithm to detect such situations we have implemented two stagnation measures (Variation Coefficient and $\overline{\lambda}$ Branching Factor) proposed by Dorigo and Stützle [5] that indicate how explorative the search behaviour of the ants is. A detailed description of stagnation measures is given in [8] (page 67).

3 Computational Results

In order to evaluate and compare the performance of the different ACO algorithms, a series of experiments were performed. All of these experiments were performed for the ten representative instances of the DIMACS Graph Coloring Challenge.

We experimented with variant-independent parameters and parameters of each ACO variant. After setting of all parameters to values that were obtained through trial and error the following experiments were performed (time-limit of 200 seconds was set for each run):

- $[(\alpha, \beta), \ldots] = [(1, 10), (2, 20), (3, 30), (5, 40), (2, 50)]$: the combination of $\alpha = 2$ and $\beta = 50$ outperformed the other combinations (considering best average width over five runs) in 27 of 50 experiments.
- Guiding Heuristics: we compared min-fill and min-degree heuristics. The results clearly indicated that the min-fill heuristic gives better results. Nonetheless, the min-degree heuristic is much more time-efficient. For instance, the ACO algorithms were only able to complete one iteration within 200 seconds for the problem instance le450_5a using the min-fill heuristic while 144 iterations could be performed using the min-degree heuristic. This is why we decided to use the min-degree heuristic for all remaining experiments since we would otherwise be unable to investigate the impact of the pheromone trails on the search behaviour of the ants due to the small number of iterations.

– Number of ants: 5, 10, 20, 50, 100. Best results were obtained by using ant colonies consisting of 100 ants for Simple and Elitist Ant System (SES, EAS), 50 ants for Rank-Based Ant System(RAS), 20 ants for Max-Min Ant System (MAS) and five ants for Ant Colony System (ACS).
– Weight e for Elitist Ant System: 2, 4, 6 and 10. Elitist weight of 10 gave best results.
– Number of ants w to deposit pheromone in every iteration in Rank-Based Ant System: 3, 5, 7 and 10. Best results were obtained with $w = 10$.
– Max-Min Ant System: $[(a, f), \ldots] = [(10, 2), (3, 5), (10, 5), (3, 2)]$. $(3, 5)$ parameter combination gave best results.
– Ant Colony System: $[(q_0, \xi), \ldots] = [(0.1, 0.3), (0.5, 0.05), (0.1, 0.05), (0.5, 0.3)]$. $(0.5, 0.3)$ gives the best results among all of these combinations.

Note that we initialize τ_0 the pheromone trails to m/W_η because according to [5] it is a good heuristic to initialize the pheromone trails to a value that is slightly higher than the expected amount of pheromone deposited by the ants in one iteration. Additionally we set the pheromone evaporation rate ρ to 0.1 for our experiments because that also worked well for the travelling salesman problem according to [5].

After we had found good parameter settings for each ACO variant we were now ready to compare them. Therefore, five runs were performed by each variant for every instance of the experimental set with $\alpha = 2$, $\beta = 50$, $\rho = 0.1$ and a time-limit of 500 seconds. Min-degree was used as the guiding heuristic. Additionally, the parameter settings for each variant were applied based on the results of the prior experiments.

Instance	Minimum Width					Average Width				
	SAS	EAS	RAS	MAS	ACS	SAS	EAS	RAS	MAS	ACS
DSJC125.1	65	65	65	64	**63**	65.6	65.4	65.4	64	**63.8**
games120	37	38	38	37	37	38.8	38.8	38.6	37.4	**37**
le450_5a	311	312	**303**	308	309	313.6	314	**310.2**	312.8	311.4
le450_5b	313	314	311	**307**	312	313.6	315.8	314.8	**313.2**	313.4
miles500	25	25	25	25	25	25.4	25.2	25.8	**25**	25.2
myciel6	35	35	35	35	35	35	35	35	35	35
myciel7	68	68	69	68	69	68.8	68.8	69	68.8	69
queen12_12	114	113	112	112	**111**	114.6	114	114.4	113.2	**112**
queen8_8	48	48	48	47	47	48	48	48.2	47	47
school1	237	231	232	**228**	232	238	235.2	235.4	233.4	**233.2**

Table 1. Comparison of minimum and average widths achieved by all ACO variants over 5 runs. Given in bold are those values that represent single-best solutions among all variants.

Table 3 lists the minimum and the average width achieved by each ACO variant for each problem instance. According to these results Max-Min Ant System and Ant Colony System performed slightly better than the other variants. Only once, for the problem instance le450_5a, Rank-Based Ant System was able to achieve better results than Max-Min Ant System and Ant Colony System. For all other problem instances one of these two variants delivered the best minimum and average width. Since Ant Colony System more often gave the single best solution among all variants, we decided to focus our remaining investigations on this ACO variant.

In Section 2.4 we presented two different pheromone update strategies. In order to compare them we have applied Ant Colony System with each of them. The edge-specific pheromone update strategy gave slightly better results than the edge-independent strategy.

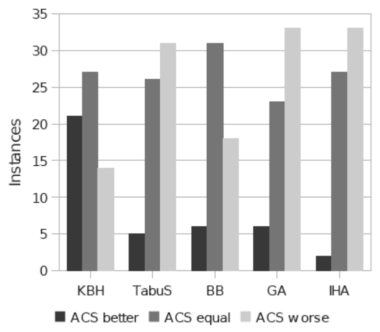
Our final experiments dealt with the combination of Ant Colony System with the iterated local search and the hill climbing algorithm. Ant Colony System in combination with the iterated local search clearly outperformed the hybrid algorithm consisting of Ant Colony System and the hill climbing local search. ACS+HC was only able to give better results than ACS+ILS for two out of the ten problem instances (see page 91 in [8]).

4 Comparison with the results in the literature

Based on the results of our experiments we compared our Ant Colony System (ACS) variants with the results from the literature for 62 well known DIMACS graph coloring instances. All results reported in this paper have been obtained on a machine equipped with 48GB of memory and two Intel(R) Xeon(R) CPUs (E5345) having a clock rate of 2.33GHz. We performed 10 runs for each example with our ACS and ACS+ILS algorithms. Each run was performed with a time-limit of 1000 seconds. The best results from a set of algorithms proposed by Koster, Bodlaender and Hoesel in [10] (KBH) were obtained with a Pentium 3 800MHz processor. Results of Tabu Search (TabuS) algorithm proposed in [2] were obtained with a Pentium 3 1 GHz processor. Experiments for the branch and bound (BB) algorithm presented by Gogate and Dechter in [7] were performed on a Pentium 4 2.4 GHz processor using 2 GB of memory. The results with the genetic algorithm (GA) in [13] were obtained in a Intel(R) Xeon(TM) 3.2 GHz processor and 4 GB of memory. Results of iterative heuristic algorithm (IHA) [12] were obtained with a Pentium 4 processor 3 GHz and 1 GB of RAM.

Figure 3 visualizes for how many problem instances ACS respectively ACS+ILS gave a better, equal or worse minimum width compared (regarding the best found solution) with each of the other decomposition methods . As can be seen, both algorithms outperformed KBH on more instances than vice versa but only ACS+ILS also managed to outperform BB.

Algorithms GA, IHA and TabuS outperform our algorithms considering the number of found better upper bounds. However, the time limit of our algorithm was set to 1000 seconds, whereas other algorithms were executed for much longer time. Unfortunately, due to the space limitation of this paper we can not present these results here, but the reader is referred to [12] (this paper presents the execution times of KBH, BB, TabuS, GA and IHA). Based on these results it is clear that for large examples the execution time of these algorithms was much longer compare to our algorithms. Note that our ACS+ILS was able to find an improved upper bound of 30 for the problem instance *homer.col*. By applying the ACS+ILS algorithm with the min-fill heuristic we could further



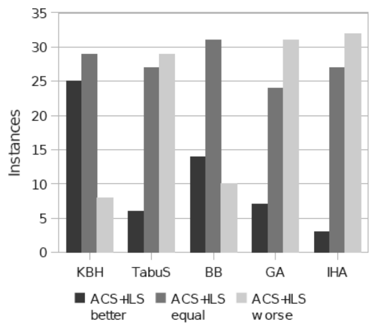


Fig. 3. Comparison of ACO algorithms with other decomposition methods.

improve the upper bound for this instance to 29. Considering comparison of ACS and ACS+ILS, for 25 problem instances ACS+ILS gave a better minimum width than ACS on its own. ACS was never able to outperform ACS+ILS with the exception of the problem instances *inithx.i.2* and *inithx.i.3* for which ACS achieved a better average width than ACS+ILS.

5 Conclusions

In this paper we have applied the ant colony optimization metaheuristic to the problem of finding tree decomposition of small width. Our experiments suggested that the ACO variants Max-Min Ant System and Ant Colony System give the best results for tree decompositions. We have applied Ant Colony System with and without the iterated local search to 62 benchmark graphs. The hybrid algorithm turned out to give better results than Ant Colony System on its own. It could improve the best known upper bound of the problem instance *homer.col* from 31 to 29. For 28 instances the algorithm was able to return a width equal to the best known upper bound. Nevertheless, especially for more complex problem instances both algorithms gave worse results than the best methods in literature. However, the time limit of our algorithm was set to 1000 seconds, whereas other algorithms were executed for longer time.

Subject of future research is the investigation of self-adaptive parameter settings. The algorithm could make use of the stagnation measures in order to adjust parameters such as the evaporation rate ρ autonomously. Another viable extension worth of further investigation is the application of ant colonies consisting of a number of ants proportional to the number of vertices in the constraint graph. That possibly could help to improve the quality of the pheromone updates and therefore could also improve the convergence behaviour of the algorithm.

Acknowledgments: The research herein is partially conducted within the competence network Softnet Austria (`http://www.soft-net.at/`) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische

Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT). Additionally, this work was partially supported by the Austrian Science Fund (FWF), project P20704-N18.

References

1. B. Bullnheimer, R. F. Hartl, and C. Strauss. A New Rank Based Version of the Ant System: A Computational Study. *Central European Journal for Operations Research and Economics*, 7(1):25-38, 1999.
2. F. Clautiaux, A. Moukrim, S. Négre, and J. Carlier. Heuristic and meta-heurisistic methods for computing graph treewidth. *RAIRO Oper. Res.*, 38:13–26, 2004.
3. M. Dorigo. *Optimization, Learning and Natural Algorithms [in Italian]*. PhD thesis, Dipartimento die Elettronica, Politecnico di Milano, Milan, 1992.
4. 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, April 1997.
5. M. Dorigo and T. Stützle. *Ant Colony Optimization.* Bradford Book, 2004. ISBN 0262042193.
6. M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.
7. Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. *In Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence, UAI-04*, pages 201–208, 2004.
8. Thomas Hammerl. Ant Colony Optimization for Tree and Hypertree Decompositions *Master's Thesis, Vienna University of Technology*, 2009.
9. U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 1:2–17, 1992.
10. A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics 8, Elsevier Science Publishers*, 2001.
11. P. Larranaga, C.M.H Kujipers, M. Poza, and R.H. Murga. Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing (UK)*, 7(1):1997, 1991.
12. N. Musliu. An iterative heuristic algorithm for tree decomposition. In *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, Volume 153, pages 133-150, 2008. Carlos Cotta, Jano van Hemert (Eds.).
13. N. Musliu and W. Schafhauser. Genetic algorithms for generalised hypertree decompositions. *European Journal of Industrial Engineering*, 1(3):317–340, January 2007.
14. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of treewidth. *Journal Algorithms*, 7:309–322, 1986.
15. T. Stützle and H. Hoos. Max-min Ant System and local search for the traveling salesman problem. In *IEEE International Conference on Evolutionary Computation*, pages 309–314, 1997.
16. T. Stützle and H. Hoos. Max-min Ant System. *Future Gener. Comput. Syst.*, 16 (9):889–914, 2000.
17. R.E. Tarjan and M. Yannakakis. Simple linear-time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.