Metaheuristic Algorithms and Tree Decomposition

Thomas Hammerl, Nysret Musliu, and Werner Schafhauser

Vienna University of Technology, Database and Artificial Intelligence Group
{hammerl, musliu, schafhauser}@dbai.tuwien.ac.at

This chapter deals with the application of evolutionary approaches and other metaheuristic techniques for generating tree decompositions. Tree decomposition is a concept introduced by Robertson and Seymour [34] and it is used to characterize the difficulty of constraint satisfaction and NP-hard problems that can be represented as a graph. Although in general no polynomial algorithms have been found for such problems, particular instances can be solved in polynomial time if the treewidth of their corresponding graph is bounded by a constant. The process of solving problems based on tree decomposition comprises two phases. First, a decomposition with small width is generated. Basically in this phase the problem is divided into several sub-problems, each included in one of the nodes of the tree decomposition. The second phase includes solving a problem (based on the generated tree decomposition) with a particular algorithm such as dynamic programming. The main idea is that by decomposing a problem into sub-problems of limited size, the whole problem can be solved more efficiently. The time for solving the problem based on its tree decomposition usually depends on the width of the tree decomposition. Thus it is of high interest to generate tree decompositions having small widths.

Finding the treewidth of a graph is an NP-hard problem [2]. In order to solve this problem different algorithms have been proposed in the literature. Exact methods such as branch and bound techniques can be used only for small graphs. Therefore, metaheuristic algorithms based on genetic algorithms [18] , simulated annealing [22], tabu search [14], iterated local search [29] , and ant-colony optimization ([7], [9]) have been proposed in the literature to generate good upper bounds for larger graphs. Such techniques have been applied very successfully and they are able to find the best existing upper bounds for many benchmark problems in the literature.

In this chapter we will first introduce the concept of tree decomposition, and then give a survey on metaheuristic techniques used to generate tree decompositions. Three approaches based on genetic algorithms, iterated local search and ant-colony optimization that were proposed in the literature will be described in detail. Finally, we will also mention briefly two recent approaches that exploit tree decompositions within metaheuristic search.

1 Tree Decompositions

We start with an informal description of tree decomposition. Suppose that we have to find solutions for the graph colouring problem (*GCP*), which is a well known constraint satisfaction problem (CSP) in the literature. For this problem we have to find a colouring 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-hand side of Figure 1. The task is now to find a valid colouring just using the colours red, green, and blue.

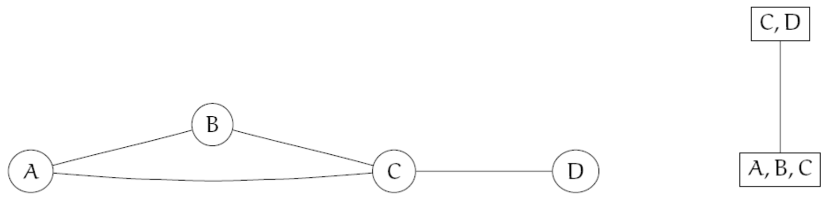

Fig. 1. Instance of the graph coloring problem and a possible tree decomposition

A naive approach to solve this problem might be to try out all possible combinations of variable assignments and see which ones are valid. In general there are d^n possible combinations, where d is the number of available colours and n is the number of vertices.

To solve this problem by tree decomposition, first we generate the tree decomposition of the corresponding problem graph. Informally, a tree decomposition is a tree containing a group of graph vertices where each tree node fulfils the following conditions: each vertex of the graph appears in one of the nodes of the tree; if two vertices are connected in the graph, they must appear together in some of the tree nodes; connectedness condition must be fulfilled, i.e. if a vertex appears in two different nodes of the tree, it must appear also in other nodes between these two nodes. The formal definition of tree decomposition is given in the next section.

If we want to solve the graph colouring problem in Figure 1 based on its tree decomposition, we can start out by solving the subproblems given by each node in the tree decomposition. Using a naive approach of trying out all possible combinations of variable assignments one has to 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 that 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 colour for the vertex C. By using the tree decomposition we have to generate 36 combinations of variable assignments in order to determine all solutions compared to the 81 combinations we would have to generate without the tree decomposition. This difference increases very quickly with the size of the graph colouring problem and constraint satisfaction problems in general. The smaller the subproblems in the tree decomposition the more efficiently we can solve a particular problem. This motivates our interest in finding tree decompositions of small *width*.

Note that tree decompositions have been applied for several applications, like combinatorial optimization problems, expert systems, computational biology etc. The use of tree decomposition for inference problems in probabilistic networks is shown in [28]. Koster et al [26] propose the application of tree decompositions for frequency assignment problem. Tree decomposition has also been applied for the vertex cover problem on planar graphs [1]. Furthermore, solving partial constraint satisfaction problems (e.g. MAX-SAT) with tree decomposition based method has been investigated in [25]. In computational biology tree decompositions has been used for protein structure prediction [39]. Recently, the application of tree decomposition in Answer-Set Programming has been investigated in [30].

1.1 Formal definitions

The concept of tree decompositions has been first introduced by Robertson and Seymour [34]. The formal definition of tree decomposition is given below (see [34], [24]).

Definition 1. *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.

Figure 2 shows a graph G and a possible tree decomposition of G. The width of shown tree decomposition is 3.

For the given graph G the treewidth can be found from its triangulation. In the following we will give basic definitions, explain how the triangulation of graph can be constructed, and give lemmas which give relation between the treewidth and the triangulated graph.

Two vertices u and v of graph $G(V, E)$ are neighbours, if they are connected by an edge $e \in E$. The neighbourhood of a vertex v is defined as: $N(v) := \{w | w \in V, (v, w) \in E\}$. A set of vertices is clique if they are fully connected. An edge connecting two non-adjacent vertices in the cycle is called chord. The graph is triangulated if there exists a chord in every cycle of length larger than 3.

A vertex of a graph is simplicial if its neighbours form a clique. An ordering of nodes $\sigma(1, 2, \ldots, n)$ of V is called a perfect elimination ordering for G if for

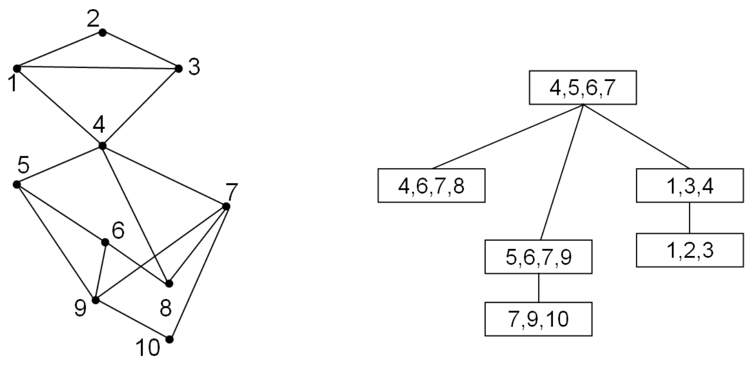

Fig. 2. A graph G (left) and a tree decomposition of G (right)

any $i \in \{1, 2, \ldots, n\}$, $\sigma(i)$ is a simplicial vertex in $G[\sigma(i), \ldots, \sigma(n)]$ [6]. In [12] it is proved that the graph G is triangulated if and only if it has a perfect elimination ordering. Given an elimination ordering of nodes the triangulation H of graph G can be constructed as following. Initially $H = G$, then in the process of elimination of vertices, the next vertex in order to be eliminated is made simplicial vertex by adding of new edges to connect all its neighbours in current G and H. The vertex is then eliminated from G. This process is repeated for all vertices in the ordering.

The treewidth of a triangulated graph can be calculated based on its cliques. For the given triangulated graph the treewidth is equal to its largest clique minus 1 [13]. Moreover, the largest clique of a triangulated graph can be calculated in polynomial time. The complexity of calculating the largest clique for the triangulated graphs is $O(|V| + |E|)$ [13]. For every graph $G = (V, E)$, there exists a triangulation of G, $\overline{G} = (V, E \bigcup E_t)$, with $tw(\overline{G}) = tw(G)$. Thus, finding the treewidth of a graph G is equivalent to finding a triangulation $\overline{G}$ of G with minimum clique size (for more information see [24]).

The process of elimination of nodes from the given graph G is illustrated in Figure 3. Suppose that we have given the following elimination ordering: $10, 9, 8, 7, 2, 3, 6, 1, 5, 4$. The vertex 10 is first eliminated from G. When this vertex is eliminated no new edges are added to the graph G and H (graph H is not shown in the figure), as all neighbours of node 10 are connected. From the remained graph G the vertex 9 is eliminated. To connect all neighbours of vertex 9, two new edges are added in G and H (edges $(5, 7)$ and $(6, 7)$). The process of elimination continues until the triangulation H is obtained. A more detailed description of the algorithm for constructing a graph's triangulation for a given elimination ordering is found in [24].

For generating the tree decomposition during the vertex elimination process, first the nodes of the tree decomposition are created. This is illustrated in Figure 3. When vertex 10 is eliminated a new tree decomposition node is created. This

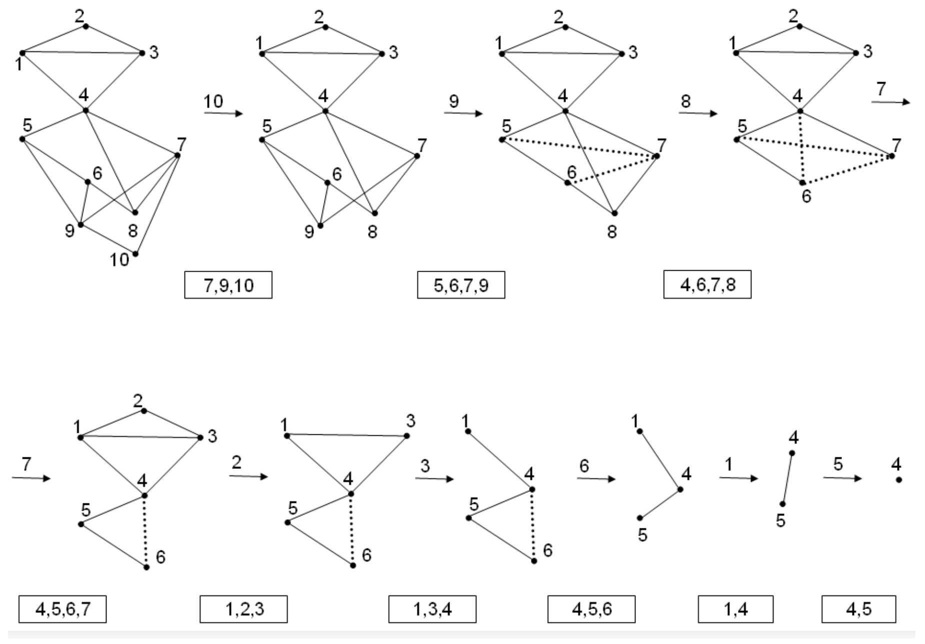

Fig. 3. Elimination of vertices 10, 9, 8, 7, 2, 3, 6, 1, 5, 4. When a vertex is eliminated a tree node containing eliminated vertex and its neighbours is created.

node contains the vertex 10 and all other vertices which are connected with this vertex in current graph G. Further the next tree node with vertices $\{5, 6, 7, 9\}$ is created when the vertex 9 is eliminated. To the end of elimination process all tree decomposition nodes will be created. The created tree nodes should be connected, such that the connectedness condition for vertices is fulfilled. This is the third condition in the tree decomposition definition. To fulfil this condition the tree decomposition nodes are connected as following. The tree decomposition node with vertices $\{7, 9, 10\}$ that is created when vertex 10 is eliminated, is connected with the tree decomposition node which will be created when the next vertex which appears in $\{7, 9, 10\}$ is eliminated. In this case the node $\{7, 9, 10\}$ should be connected with the node created when vertex 9 is eliminated, because this is the next vertex in the ordering that is contained in $\{7, 9, 10\}$. This rule is further applied for connection of other tree decomposition nodes, and from the graph the tree decomposition in Figure 2 will be constructed. Note that some of tree nodes that are created in the elimination process are not presented in the tree decomposition, because they are contained in larger tree nodes. For example the node $\{4, 5, 6\}$ which is created by eliminating vertex 6 is already contained in the node $\{4, 5, 6, 7\}$ which is created by eliminating vertex 7. Moreover, tree nodes which are created by eliminating vertices $1, 5, 4$ are also contained in other larger tree nodes.

2 Generating tree decompositions by genetic algorithms and other metaheuristic techniques

As described in the previous section the width of the tree decomposition depends on the elimination ordering of vertices. Therefore, the task of finding tree decomposition with minimal width consists of finding the best permutation of graph vertices. This problem is similar to the travelling salesman problem, but with a different objective function.

In the past two decades researchers have been proposing different techniques to find tree decompositions for different benchmark examples. This includes the exact techniques based on tree search and branch and bound, the simple greedy techniques and metaheuristic techniques. In this chapter we focus on metaheuristic techniques. At the end of this section we will also shortly describe other approaches used for tree decompositions.

The metaheuristic techniques applied for tree decomposition can be divided in two groups: population based/nature inspired techniques, and local search techniques. Regarding nature inspired techniques the application of genetic algorithms has been investigated in [27] and [33], and ant colony optimization has been used in [17]. Examples of local search techniques for tree decompositions are [23], [6] and [32].

2.1 Genetic Algorithms for Tree Decomposition

Application of genetic algorithm for tree decompositions has been first investigated in [27]. This algorithm tried to minimize a weight associated with the decompositions of Bayesian networks which is not exactly the same as the width of the tree decomposition. In [33] this algorithm has been extended for generating hypertree decompositions and with some changes in fitness function (the width of tree decompositions has been used as a objective function) has been tested on different problems from the literature. The following description of genetic algorithm for tree decomposition is based on our previous work in [33].

Genetic algorithms (GAs) were developed by [18]. They try to find a good solution for an optimization problem by imitating the principle of evolution. Genetic algorithms alter and select individuals from a population of solutions for the optimization problem. In the following we describe frequently used terms within the field of genetic algorithms:

population	... set of candidate solutions
individual	... a single candidate solution
chromosome	... set of parameters determining the properties of a solution
gene	... single parameter

A genetic algorithm tends to optimize the value of an objective function of an optimization problem, in terms of genetic algorithms also called *fitness function*. At the beginning a genetic algorithm creates an initial population containing randomly or heuristically created individuals. These individuals are evaluated and assigned a *fitness* value, which is the value of the fitness function for the solution represented by the individual. The population is evolved over a number of generations until a halting criterion is satisfied. At each generation the population undergoes *selection* and *recombination*, also called *crossover* and *mutation*.

During the selection process the genetic algorithm decides which individuals from the current population are allowed to enter the next population. This decision is based on the fitness value of the individuals and individuals of better fitness should enter the next population with higher probability than individuals of lower fitness. Not selected individuals are discarded and will not be evolved further.

The recombination process or crossover combines different properties of several parent solutions within one or more children solutions, also denoted as *offsprings*. Crossover exchanges properties between the individuals with the aim of increasing the average quality of the population.

During the mutation process individuals are slightly altered. Mutation is used to explore new regions of the search space and to avoid early convergence to local optima.

In practice parameters are used in order to control the behaviour of a genetic algorithm. Typical *control parameters* are mutation rate, crossover rate, population size and parameters for selection techniques. The choice of the control parameters has a crucial effect on the quality of the best solution found by a genetic algorithm.

The genetic algorithm for tree decomposition presented below is named *GA-tw* and was implemented in [33]. Algorithm 2.1 presents algorithm *GA-tw* in pseudo code notation.

The algorithm takes as input a graph and several control parameters. Individual solutions are vertex orderings. Each individual is assigned the width of the tree decomposition returned from the corresponding vertex ordering as its fitness value.

Initially *GA-tw* generates a population consisting of randomly created individuals. Tournament selection was chosen as the selection technique. Tournament selection selects an individual by randomly choosing a group of several individuals from the former population. The individual of highest fitness (smallest width) within this group is selected to join the next population. This process is applied until enough individuals have entered the next population. Finally, after a certain number of generations, algorithm *GA-tw* will return the best fitness (smallest width) of an individual found during the search process.

Crossover and mutation operators

Within the genetic algorithms in [33] nearly all types of crossover operators and all mutation operators were implemented. The same operators were also applied in [27] for decomposing the moral graph of Bayesian networks.

Algorithm 1 Genetic algorithm for tree decompositions - *GA-tw*

```
$t = 0$
initialize $(population(t), n)$
evaluate $population(t)$

while $t < max\_gen$ do
  $t = t + 1$
  $population(t)$ = tournament_selection($population(t-1), s$)
  $(population(t), p_c)$
  $(population(t), p_m)$
  $population(t)$
end while

return the smallest width found during the search
```

Crossover operators:

– partially-mapped crossover (PMX)
– cycle crossover (CX)
– order crossover (OX1)
– order-based crossover (OX2)

- position-based crossover (POS)
- alternating-position crossover (AP)

Mutation operators:

- displacement mutation operator (DM)
- exchange mutation operator (EM)
- insertion mutation operator (ISM)
- simple-inversion mutation operator (SIM)
- inversion mutation operator (IVM)
- scramble mutation operator (SM)

We will describe the crossover and mutation operators which returned the best results of algorithm *GA-tw* in more detail:

Order Crossover (OX1)

The order crossover operator determines a crossover area within the parents by randomly selecting two positions within the ordering. The elements in the crossover area of the first parent are copied to the offspring. Starting at the end of the crossover area all elements outside the area are inserted in the same order in which they occur in the second parent.

Order-Based Crossover (OX2)

The order-based crossover operator selects at random several positions in the parent orderings by tossing a coin for each position. The elements of the first parent at these positions are deleted in the second parent. Afterwards they are reinserted in the order of the second parent.

Position-Based Crossover (POS)

The position-based crossover operator also starts with selecting a random set of positions in the parent strings by tossing a coin for each position. The elements at the selected positions are exchanged between the parents in order to create the offsprings. The elements missing after the exchange are reinserted in the order of the second parent.

Exchange Mutation Operator (EM)

The exchange mutation operator randomly selects two elements in the solution and exchanges them.

Insertion Mutation Operator (ISM)

The insertion mutation operator randomly chooses an element in the solution and moves it to a randomly selected position.

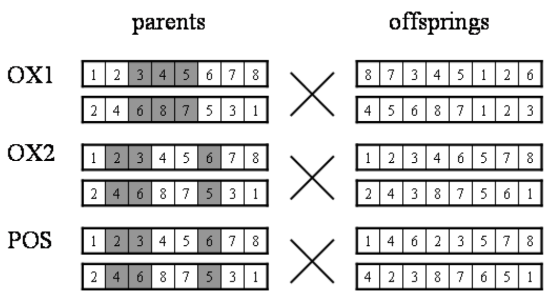

Fig. 4. Selected crossover operators for vertex orderings.

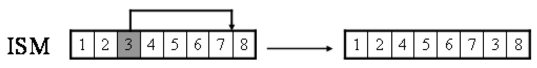

Fig. 5. Selected mutation operators for vertex orderings.

The genetic algorithm implemented in [27] was applied to two artificial graphs. This genetic approach returned competitive results when compared to results obtained by simulated annealing [23]. The algorithm implemented in [33] was evaluated on 62 graphs of the Second DIMACS graph colouring challenge ([19]). Different experiments were performed to find the best parameter values for parameters of the genetic algorithm and it turned out that the position based crossover operator (POS) and the insertion mutation operator (ISM) were best suited for finding tree decompositions of small width. Existing upper bounds for treewidth for several DIMACS instances could be improved.

2.2 Ant Colony Optimization for Tree Decompositions

Ant Colony Optimization (ACO) has been applied for tree decompositions in [17] and [16]. The current section is based on [17] and describes different ant colony optimization variants applied for tree decomposition.

Ant Colony Optimization is a population-based metaheuristic introduced by Marco Dorigo et al [7], [9]. 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"* [10].

The following variants of ACO algorithms for finding good upper bounds for tree decompositions were investigated in [17] and [16]: Simple Ant System ([7], [9]), Elitist Ant System ([7], [9]), Rank-Based Ant System [5], Max-Min Ant System ([36], [37]), and Ant Colony System [8]. Two different pheromone update strategies were proposed and two stagnation measures were implemented that indicate the degree of diversity of the solutions constructed by the ants. Furthermore, two constructive heuristics (Min-Degree, Min-Fill) were implement and incorporated alternatively into every ACO variant as a guiding function, and the combination of ACO with two existing local search methods: Hill Climbing and Iterated Local Search [32] was investigated.

A simple constraint graph and the corresponding ACO construction tree are shown in Figure 6. 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.

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

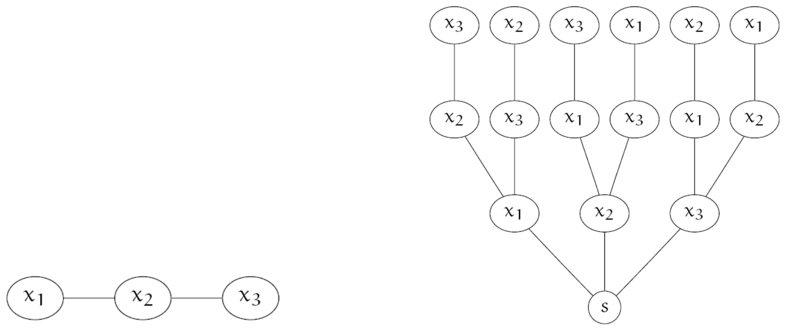

Fig. 6. Constraint graph $\mathcal{G}$ and the ACO construction tree.

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.

Pheromone Trails A pheromone trail gives information how favourable it is 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 construction tree in Figure 6 is the matrix as shown below:

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

In this matrix 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 related 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.

Heuristic Information The ants make their decision about which vertex to eliminate next not solely based on the pheromone matrix but also consider a guiding heuristic. Two different heuristics have been implemented. In order to compute them, a separate graph in addition to the construction tree is maintained. This graph is called the *elimination graph* because it is obtained from the original constraint graph by successively eliminating the vertices traversed by the ant in the construction tree. Further, this graph is denoted 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 node i represents the last eliminated node, whereas j is a node which is not eliminated yet. 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.

Probabilistic Vertex Elimination In the following is shown 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 makes a greedy move 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 is to diversify the search so that subsequent ants will more likely choose other branches of the construction tree.

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 gradually removed from the pheromone trails so that solutions that might have been the best known solutions in earlier iterations of the algorithm can be forgotten.

Pheromone Deposition In this step for an elimination ordering σ_k that was constructed by an ant k the amount of pheromone that will be deposited for each (i, j) in σ_k is determined. An edge-independent and an edge-specific pheromone update strategy were considered. 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. (σ_{kj} is the partial elimination ordering that is obtained from σ_k by omitting j and all vertices that are eliminated after j.)

The selection of ants that deposit pheromone and the weighting of this pheromone varies between the different ACO variants. The reader is referred to [10] 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:

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

Ant Colony System only removes pheromone from the trails belonging to the best known elimination ordering σ_{bs}:

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

Hybridization with Local Search All ACO variants were extended with two local search methods 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. Two local search techniques were used: an hill climbing algorithm and an iterated local search similar to the algorithm proposed in [32].

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 two stagnation measures were implemented (Variation Coefficient and $\overline{\lambda}$ Branching Factor) proposed by Dorigo and Stützle [10] that indicate how explorative the search behaviour of the ants is. A detailed description of stagnation measures is given in [16] (page 67).

All described Ant Colony Optimization variants in [17] were evaluated experimentally with DIMACS Graph Coloring Challenge instances. Max-Min Ant System and Ant Colony System performed slightly better than the other variants. Although the Ant Colony Optimization in general could not compete with iterated local search and genetic algorithms, it could improve the upper bound for one of problems.

2.3 Iterated Local Search for Tree Decomposition

The application of iterated local search for generating tree decompositions has been investigated in [31], [32]. In this section we give the description of this algorithm based on these references.

The algorithm is based on the iterated local search framework and it includes a simple local search heuristic to generate good orderings, and an iterative process in which the algorithm calls a local search technique with the initial solution produced in the previous iteration. The algorithm also includes a mechanism for acceptance of a candidate solution for the next iteration. Although the constructing phase is very important, choosing the appropriate perturbation at each iteration as well as the mechanism for acceptance of solution are also crucial to obtain good results for an iterative local search algorithm. The iterated local search algorithm for tree decomposition is presented below.

The algorithm starts with an initial solution which takes an order of nodes as they appear in the input. Better initial solutions can also be constructed by using other heuristics which run in polynomial time, such as Maximum Cardinality Search, min-fill heuristic, etc. However, as the proposed method usually finds a

Algorithm 2 Iterative heuristic algorithm - IHA

```
Generate initial solution $S1$

$BestSolution = S1$

while Termination Criteria is not fulfilled do
    $S2 = ConstructionPhase(S1)$

    if Solution $S2$ fulfils the acceptance criteria then
        $S1 = S2$
    else
        $S1 = BestSolution$
    end if

    Apply perturbation in solution $S1$

    Update $BestSolution$ if solution $S2$ has better (or equal) width than the current
    best solution

end while

RETURN $BestSolution$
```

solution produced by these heuristics in a very short time, the algorithm starts with an ordering of nodes given in the input.

After constructing the initial solution the iterative phase starts. In this phase the local search method is called iteratively, and then the selected solution is perturbed. Two different local search techniques that can be used in the construction phase were proposed. The solution returned from the construction phase is accepted for the next iteration if it fulfils the specific criteria determined by the solution acceptance mechanism. Experiments with different possibilities for the acceptance of the solution returned from the construction phase were performed. If the solution does not fulfil the acceptance criteria this solution is discarded and the currently best solution is selected. In the selected solution the perturbation mechanism is applied. Different possibilities are used for perturbation. The perturbed solution is given as an input solution in the next call of the construction phase. This process continues until the termination criterion is fulfilled.

Two local search methods were proposed for generating a good solution which is used as an initial solution with some perturbation in the next call of the same local search algorithm. Both techniques are based on the idea of moving only vertices in the ordering which cause the largest clique during the elimination process. The motivation for using this method is to reduce the number of solutions that should be evaluated. The first proposed technique named LS1 is presented below.

Algorithm 3 Local Search Algorithm 1 - LS1 (InputSolution)

```
$BestLSSolution = InputSolution$
$NrNotImprovments = 0$

while $NrNotImprovments < MAXNotImprovments$ do
    In current solution ($InputSolution$) select a vertex in the elimination ordering
    which causes the largest clique when eliminated - ties are broken randomly if
    there are several vertices which cause the clique equal with the largest clique

    Swap this vertex with another vertex located in a randomly chosen position

    if the current solution is better than $BestLSSolution$ then
        $BestLSSolution = InputSolution$
        $NrNotImprovments = 0$
    else
        $NrNotImprovments = NrNotImprovements + 1$
    end if

end while

RETURN $BestLSSolution$
```

The proposed algorithm applies a simple heuristic. In the current solution a vertex is chosen randomly among the vertices that produce the largest clique in the elimination process. Then the selected vertex is moved from its position. Two types of moves were used. In the first variant the vertex is inserted in a random position in the elimination ordering, while in the second variant the vertex is swapped with another vertex located in a randomly selected position, i.e. the two chosen vertices change their position in the elimination ordering. The swap move was shown to give better results. The heuristic stops if the solution does not improve for a certain number of iterations. Experiments with different $MAXNotImprovments$ were performed. LS1 alone is a simple heuristic and usually can not produce good results for tree decompositions. However, by using this heuristic as a local search heuristic in the iterated local search algorithm good results for tree decompositions are obtained.

The second proposed heuristic (LS2) is similar to algorithm LS1. However, this technique differs from LS1 regarding the exploration of the neighbourhood. In LS2 in some of iterations the neighbourhood of solution consists of only one solution which is generated by swapping a vertex (that causes the largest clique) in the elimination ordering with another vertex located in the randomly chosen position. This neighbourhood is used in a particular iteration with probability p. Experiments with different values for parameter p were performed. With probability $1-p$, the other type of neighbourhood will be explored. The neighbourhood of current solution in this case consists of all solutions which can be obtained by swapping of a vertex (which causes the largest clique) in the elimination ordering with its neighbours. The best solution from the generated neighbourhood is selected for the next iteration in the LS2. Note that in this technique the number of solutions that have to be evaluated is much larger than in LS1. In particular in the first phase of search the node which causes the largest clique usually has many neighbours and therefore the number of solutions to be evaluated when the second type of neighbourhood is used is equal to the size of the largest clique produced during the elimination process.

Perturbation During the perturbation phase the solution obtained by local search procedure is perturbed and the newly obtained solution is used as an initial solution for the new call of the local search technique. The main idea is to avoid the random restart. Instead of random restart the solution is perturbed with a bigger move(s) as those applied in the local search technique. This enables some diversification that helps to escape from the local optimum, but avoids beginning from scratch (as in case of random restart), which is very time consuming. Three perturbation mechanisms were proposed:

– RandPert: N vertices are chosen randomly and they are moved into new random positions in the ordering.
– MaxCliquePer: All nodes that produce the maximal clique in the elimination ordering are inserted in a new randomly chosen positions in the ordering.
– DestroyPartPert: All nodes between two positions (selected randomly) in the ordering are inserted in the new randomly chosen positions in the ordering.

The perturbation RandPert just perturbs the solution with a larger random move and would be kind or random restart if N is very large. Keeping N smaller avoids restarting from completely new solution, and the perturbed solution does not differ much from the previous solution. MaxCliquePer concentrates on moving only vertices which produce maximal clique in the elimination ordering. The basic idea for this perturbation is to apply a technique similar to min-conflict heuristic, by moving only the vertices that cause large treewidth. DestroyPartPert is similar to RandPert, except that the selected nodes to be moved are located near each other in the elimination ordering.

Determining the number of nodes N that will be moved is complex and may be dependent on the problem. To avoid this problem an adaptive perturbation mechanism was proposed that takes into consideration the feedback from the search process. The number of nodes N varies from 2 to some number y (determined experimentally), and the algorithm begins with small perturbation ($N = 2$). If during the iterative process (for a determined number of iterations) the local search technique produces solutions with same tree width for more than 20% of cases, the size of perturbation is increased by 1, otherwise the size of N will be decreased by 1. This enables an automatic change of perturbation size based on the repetition of solutions with the same width.

The combination of two perturbations was considered. The mixed perturbation applies two perturbations: RandPert, and MaxCliquePer. The algorithm starts with RandPert, and switches alternatively between two perturbations if the solution is not improved for a determined number of iterations. Experiments with different sizes of perturbation sizes for each type of perturbation were performed.

Acceptance criterion Different techniques can be applied for accepting the solution obtained by the local search technique. Following variants for acceptance of solution for the next iteration were used:

- Solution returned from the construction phase is accepted only if it has a better width than the best current existing solution.
- Solution returned from the construction phase is always accepted.
- Solution is accepted if its treewidth is smaller than the treewidth of the best yet found solution plus x, where x is an integer.

The first variant for accepting a solution is very restrictive. In this variant the solution from the construction phase is accepted only if it improves the best existing solution. Otherwise, the best existing solution is perturbed and it is used as input solution for next call of the construction phase. In the second variant, the iterated local search applies the perturbation in a solution returned from the construction phase, independently from the quality of produced solution. The third variant is between the first and the second variant, and in this case the solution which does not improve the best existing solution can be accepted for the next iteration, if its width is smaller than the best found width plus some bound.

2.4 Other techniques for Tree decomposition

This section gives a short overview on other approaches applied for tree decomposition. Examples of complete algorithms for tree decompositions are [35], [15], and [3]. Gogate and Dechter [15] reported good results for tree decompositions by using branch and bound algorithms. They showed that their algorithm is superior compared to the algorithm proposed in [35]. The branch and bound algorithm proposed in [15] applies different pruning techniques, and provides anytime solutions, which are good upper bounds for tree decompositions. The algorithm proposed in [3] includes several other pruning and reduction rules and is successful on small graphs. The complete techniques described above have exponential running time in the worst case and can only be used to find the optimal width for not too large graphs.

To generate good upper bounds (which can be sufficient for many applications) for treewidth several greedy heuristic techniques that run in polynomial time have been proposed. These heuristics select the ordering of nodes step by step based on different criteria, such as the degree of the nodes, the number of edges to be added to make the node simplicial etc. Most popular techniques are Maximum Cardinality Search (MCS), Min-fill heuristic and Minimum Degree heuristic.

Maximum Cardinality Search (MCS) [38] initially selects a random vertex of the graph to be the first vertex in the elimination ordering (the elimination ordering is constructed from right to left). The next vertex will be picked such that it has the highest connectivity with the vertices previously selected in the elimination ordering. Ties are broken randomly. MCS repeats this process iteratively until all vertices are selected.

The min-fill heuristic first picks the vertex which adds the smallest number of edges when eliminated (ties are broken randomly). The selected vertex is made simplicial (a vertex of a graph is simplicial if its neighbours form a clique) and it is eliminated from the graph. The next vertex in the ordering will be any vertex that adds the minimum number of edges when eliminated from the graph. This process is repeated iteratively until the whole elimination ordering is constructed.

The minimum degree heuristic picks first the vertex with the minimum degree. The selected vertex is made simplicial and it is removed from the graph. Further, the vertex that has the minimum number of unselected neighbours will be chosen as the next node in the elimination ordering. This process is repeated iteratively.

MCS, min-fill, and min-degree heuristics run in polynomial time and usually produce tree decompositions in a reasonable amount of time. According to [15] the min-fill heuristic performs better than MCS and min-degree heuristic. Although these heuristics sometimes give good upper bounds for tree decompositions, more advanced techniques usually provide better upper bounds for most problems. Min-degree heuristic has been improved by Clautiaux et al [6] by adding a new criterion based on the lower bound of the treewidth for the graph obtained when the node is eliminated. Recently, Kask et al [20] proposed an iterative greedy variable ordering algorithm to improve the greedy heuristics above. We refer to [24] and [4] for a survey of different upper bounds algorithms.

2.5 Comparison of algorithms for tree decomposition

In this section we compare results obtained with between metaheuristic aproaches described in this paper and other existing methods in the literature. The results of these methods for 62 DIMACS vertex colouring instances are given. These instances have been used for testing several methods for tree decompositions proposed in the literature. The compared methods have been executed in different computers and we give here only results regarding the width of the tree decomposition. The reader is referred to [24], [6], [15], [33], [16], and [32], for the information about the computers used and the time needed to generate solutions.

In Tables 1 and 2 the results for DIMACS graph colouring instances are presented. First and second columns of the tables present the instances and the number of nodes and edges for each instance. In column KBH are shown the best results obtained by algorithms in [24]. The TabuS column presents the results reported in [6], and the column BB shows the results obtained with the branch and bound algorithm proposed in [15]. Finally, columns GA, IHA, and ACO represent respectively results obtained with a genetic algorithm [33], iterated local search [32], and ant colony optimization [17], [16] .

Based on the results given in Tables 1 and 2 we conclude that regarding the width of tree decomposition, the metaheuristic techniques described in this paper give very good results and for many instances the best existing upper bounds for the treewidth .

2.6 Application of Tree Decomposition in Metaheuristic Techniques

Traditionally, tree decompositions have been applied used to solve constraint satisfaction problems exactly by dynamic programming algorithms. Recently, researchers have been investigating the incorporation of tree decomposition within metaheuristics techniques. The work in this direction is just in the starting phase and to the best of our knowledge only two papers investigated yet the application of tree decomposition in metaheuristic search.

In [21] tree decomposition based heuristics have been developed for the two-dimensional bin packing problem with conflicts. The aim is to find a conflict-free packing of given items by using minimal number of bins. Tree decomposition is applied to decompose a problem instance into subproblems which can be solved independently. First a tree decomposition is obtained, and then each item is assigned to a specific cluster (this phase is called cluster-seperation). Then these clusters are considered as subproblems which are solved iteratively. Finally, the partial solutions from subproblems are merged to obtain solutions for the whole problem.

Another application of tree decomposition includes the approach introduced by Fontaine et al [11] where tree decomposition is used to guide the exploration for the search space. Authors propose a method called Decomposition Guided VNS that exploits the graph of clusters to build neighbourhood structures. By using clusters better intensification and diversification is achieved. For example the moves are favoured in regions that are closely linked and the search is diversified by selecting new clusters and therefore exploring new regions of the search space.

3 Conclusion

Several metaheuristic approaches based on nature inspired strategies and local search have been used successfully in the literature for generating tree decompositions. Among these approaches, genetic algorithms and iterated local search based algorithms provide best upper bounds for many benchmark instances.

Although metaheuristic techniques currently provide state-of-the-art upper bounds for most problems, the runtime of such algorithms for large graphs is still high. Greedy heuristic approaches generate slightly worse upper bounds, but are more efficient. Therefore, developing more efficient metaheuristics for tree decompositions is still a challenging task. Moreover, for many problems the treewidth is still not known, and the question is if the current metaheuristics can

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	GA	IHA	ACO
anna	138 / 986	12	12	12	12	12	12
david	87 / 812	13	13	13	13	13	13
huck	74 / 602	10	10	10	10	10	10
homer	561 / 3258	31	31	31	31	31	30
jean	80 / 508	9	9	9	9	9	9
games120	120 / 638	37	33	-	32	32	37
queen5_5	25 / 160	18	18	18	18	18	18
queen6_6	36 / 290	26	25	25	26	25	25
queen7_7	49 / 476	35	35	35	35	35	35
queen8_8	64 / 728	46	46	46	45	45	46
queen9_9	81 / 1056	59	58	59	58	58	59
queen10_10	100 / 1470	73	72	72	72	72	73
queen11_11	121 / 1980	89	88	89	87	87	89
queen12_12	144 / 2596	106	104	110	104	103	109
queen13_13	169 / 3328	125	122	125	121	121	128
queen14_14	196 / 4186	145	141	143	141	140	150
queen15_15	225 / 5180	167	163	167	162	162	174
queen16_16	256 / 6320	191	186	205	186	186	201
fpsol2.i.1	269 / 11654	66	66	66	66	66	66
fpsol2.i.2	363 / 8691	31	31	31	32	31	31
fpsol2.i.3	363 / 8688	31	31	31	31	31	31
inithx.i.1	519 / 18707	56	56	56	56	56	56
inithx.i.2	558 / 13979	35	35	31	35	35	31
inithx.i.3	559 / 13969	35	35	31	35	35	31
miles1000	128 / 3216	49	49	49	50	49	50
miles1500	128 / 5198	77	77	77	77	77	77
miles250	125 / 387	9	9	9	10	9	9
miles500	128 / 1170	22	22	22	24	22	25
miles750	128 / 2113	37	36	37	37	36	38
mulsol.i.1	138 / 3925	50	50	50	50	50	50
mulsol.i.2	173 / 3885	32	32	32	32	32	32
mulsol.i.3	174 / 3916	32	32	32	32	32	32
mulsol.i.4	175 / 3946	32	32	32	32	32	32
mulsol.i.5	176 / 3973	31	31	31	31	31	31
myciel3	11 / 20	5	5	5	5	5	5
myciel4	23 / 71	11	10	10	10	10	10
myciel5	47 / 236	20	19	19	19	19	19
myciel6	95 / 755	35	35	35	35	35	35
myciel7	191 / 2360	74	66	54	66	66	66

Table 1. Algorithms comparison regarding treewidth for DIMACS graph colouring instances.

Instance	$\lvert V\rvert/\lvert E\rvert$	KBH	TabuS	BB	GA	IHA	ACO
school1	385 / 19095	244	188	-	185	178	228
school1_nsh	352 / 14612	192	162	-	157	152	185
zeroin.i.1	126 / 4100	50	50	-	50	50	50
zeroin.i.2	157 / 3541	33	32	-	32	32	33
zeroin.i.3	157 / 3540	33	32	-	32	32	33
le450_5a	450 / 5714	310	256	307	243	244	304
le450_5b	450 / 5734	313	254	309	248	246	308
le450_5c	450 / 9803	340	272	315	265	266	309
le450_5d	450 / 9757	326	278	303	265	265	290
le450_15a	450 / 8168	296	272	-	265	262	288
le450_15b	450 / 8169	296	270	289	265	258	292
le450_15c	450 / 16680	376	359	372	351	350	368
le450_15d	450 / 16750	375	360	371	353	355	371
le450_25a	450 / 8260	255	234	255	225	216	249
le450_25b	450 / 8263	251	233	251	227	219	245
le450_25c	450 / 17343	355	327	349	320	322	346
le450_25d	450 / 17425	356	336	349	327	328	355
dsjc125.1	125 / 736	67	65	64	61	60	63
dsjc125.5	125 / 3891	110	109	109	109	108	108
dsjc125.9	125 / 6961	119	119	119	119	119	119
dsjc250.1	250 / 3218	179	173	176	169	167	174
dsjc250.5	250 / 15668	233	232	231	230	229	231
dsjc250.9	250 / 27897	243	243	243	243	243	243

Table 2. Algorithms comparison regarding treewidth for DIMACS graph colouring instances.

still be improved to find new upper bounds for such problems. To obtain better upper bounds it would be interesting to investigate some other approaches like memetic algorithms, large neighbourhood search and other hybrid techniques. Furthermore, the iterative improvement of the initial generated tree decomposition (based on vertex ordering) is an interesting question.

Finally, in some applications the treewidth is not the only important parameter for solving problems based on tree decompositions efficiently. Therefore, the development of metaheuristics for generating tree decompositions which optimize other features of tree decomposition would be of interest in the future.

Acknowledgments. The work was supported by the Austrian Science Fund (FWF): P20704-N18. Moreover, the research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

References

1. J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics*, 145:210–219, 2004.
2. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.
3. E. Bachoore and H. Bodlaender. A branch and bound algorithm for exact, upper, and lower bounds on treewidth. *AAIM 2006, LNCS*, 4041:255–266, 2006.
4. H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.
5. 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.
6. 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.
7. M. Dorigo. Optimization, learning and natural algorithms (in Italian), phd thesis, dipartimento di elettronica, politecnico di milano, 1992.
8. M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation*, 1(1):53–66, 1997.
9. 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*, 1996.
10. M. Dorigo and T. Stützle. *Ant Colony Optimization.* Bradford Book, 2004.
11. M. Fontaine, S. Loudni, and P. Boizumault. Guiding vns with tree decomposition. In *ICTAI*, pages 505–512, 2011.
12. D. R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

13. F. Gavril. Algorithms for minimum coloring, maximum clique, minimum coloring cliques and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1:180–187, 1972.
14. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Res.*, 5:533–549, 1986.
15. V. Gogate and R. 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.
16. T. Hammerl. Ant colony optimization for tree and hypertree decompositions, master thesis, vienna university of technology, 2009.
17. T. Hammerl and N. Musliu. Ant colony optimization for tree decompositions. In *EvoCOP*, pages 95–106, 2010.
18. J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
19. D. S. Johnson and M. A. Trick. The second dimacs implementation challenge: NP-hard problems: Maximum clique, graph coloring, and satisfiability. *Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society*, 1993.
20. K. Kask, A. Gelfand, L. Otten, and R. Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *AAAI*, 2011.
21. A. Khanafer, F. Clautiaux, and E.-G. Talbi. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & OR*, 39(1):54–63, 2012.
22. S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.
23. U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2(1):2–17, 1992.
24. A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics 8, Elsevier Science Publishers*, 2001.
25. A. Koster, S. van Hoesel, and A. Kolen. Solving partial constraint satisfaction problems with tree-decomposition. *Networks*, 40(3):170–180, 2002.
26. A. M. Koster, S. P. van Hoesel, and A. W. Kolen. Optimal solutions for frequency assignment problems via tree decomposition. *Graph Theoretic Concepts in Computer Science: 25th International Workshop, WG'99, LNCS 1665, pp. 338-350*, 1999.
27. P. Larranaga, C. Kujipers, M. Poza, and R. Murga. Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing*, 7 (1):19–34, 1997.
28. S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.
29. H. Lourenço, O. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics*, volume 57, pages 320–353. 2003.
30. M. Morak, N. Musliu, R. Pichler, S. Rümmele, and S. Woltran. Evaluating tree-decomposition based algorithms for answer set programming. In *Proceedings of the Learning and Intelligent Optimization Conference (LION 6)*, 2012.
31. N. Musliu. Generation of tree decompositions by iterated local search. In *EvoCOP*, pages 130–141, 2007.

32. N. Musliu. An iterative heuristic algorithm for tree decomposition. *Studies in Computational Intelligence, Recent Advances in Evolutionary Computation for Combinatorial Optimization, Carlos Cotta and Jano I. van Hemert Ed.*, 153:133–150, 2008.
33. N. Musliu and W. Schafhauser. Genetic algorithms for generalized hypertree decompositions. *European Journal of Industrial Engineering*, 1(3):317–340, 2007.
34. N. Robertson and P. D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Journal Algorithms*, 7:309–322, 1986.
35. K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. *In Proc. of National Conference on Artificial Intelligence (AAAI'97)*, pages 185–190, 1997.
36. T. Stützle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. *IEEE International Conference on Evolutionary Computation*, pages 309–314, 1997.
37. T. Stützle and H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914, 2000.
38. R. 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.
39. J. Xu, F. Jiao, and B. Berger. A tree-decomposition approach to protein structure prediction. *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, pages 247–256, 2005.