Generation of Tree Decompositions by Iterated Local Search

Nysret Musliu

Vienna University of Technology, Karlsplatz 13, 1040 Vienna, Austria

Abstract. Many instances of NP-hard problems can be solved efficiently if the treewidth of their corresponding graph is small. Finding the optimal tree decompositions is an NP-hard problem and different algorithms have been proposed in the literature for generation of tree decompositions of small width. In this paper we propose a novel iterated local search algorithm to find good upper bounds for treewidth of an undirected graph. We propose two heuristics, and their combination for generation of the solutions in the construction phase. The iterated local search algorithm further includes the mechanism for perturbation of solution, and the mechanism for accepting solutions for the next iteration. The proposed algorithm iteratively applies the heuristic for finding good elimination ordering, the acceptance criteria, and the perturbation of solution. We proposed and evaluated different perturbation mechanisms and acceptance criteria. The proposed algorithms are tested on DIMACS instances for vertex coloring, and they are compared with the existing approaches in literature. Our algorithms have a good time performance and for 17 instances improve the best existing upper bounds for the treewidth.

1 Introduction

The concept of tree decompositions is very important due to the fact that many instances of constraint satisfaction problems and in general NP-hard problems can be solved in polynomial time if their treewidth is bounded by a constant. The process of solving problems with bounded treewidth includes two phases. In the first phase the tree decompositions with small upper bound for treewidth are generated. The second phase includes solving a problem (based on the generated tree decomposition) with a particular algorithm such as for example dynamic programming. The efficiency of solving of problem based on its tree decompositions depends from the width of tree decompositions. Thus it is of high interest to generate tree decompositions with small width.

In this paper we investigate the generation of tree decompositions of undirected graphs. The concept of tree decompositions has been first introduced by Robertson and Seymour ([11]):

Definition 1. *(see [11], [9]) 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 1 shows a graph G (19 vertices) and a possible tree decomposition of G. The width of shown tree decomposition is 5.

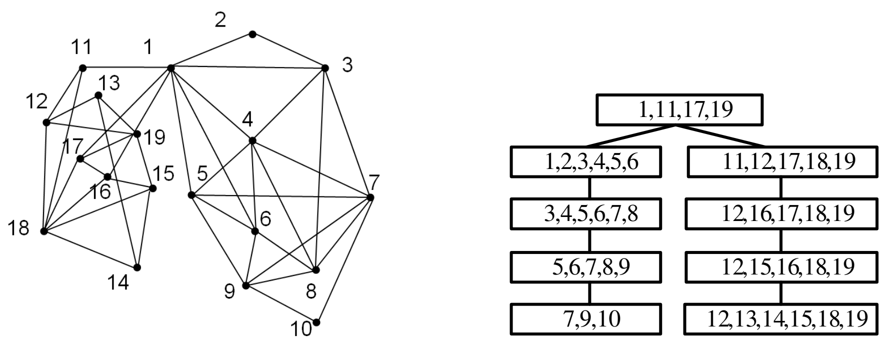

Fig. 1. A graph G (left) and a tree decomposition of G (right)

For the given graph G the treewidth can be found from its triangulation. Further 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 with an edge $e \in E$. The neighbourhood of 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 exist 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 any $i \in \{1, 2, \ldots, n\}$, $\sigma(i)$ is a simplicial vertex in $G[\sigma(i), \ldots, \sigma(n)]$ [3]. In [4] 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. The process of elimination of nodes from the given graph G is illustrated in Figure 2. Suppose that we have given the following elimination ordering: $10, 9, 8, \ldots$. The vertex 10 is first eliminated from G. When this vertex is eliminated no new edges are added in the graph G and H (graph H is not shown in the figure), as all neighbours of node 10 are connected. Further 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 $(5, 8)$ 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 [9].

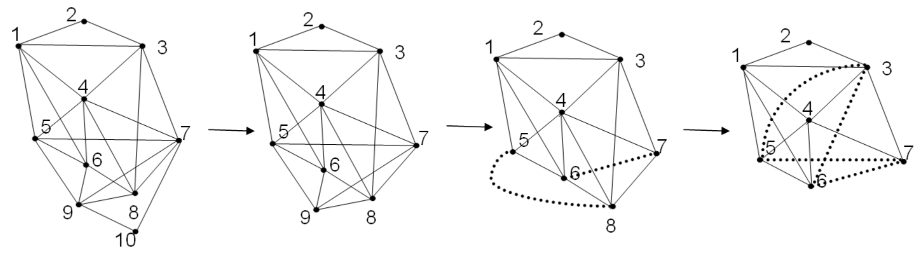

Fig. 2. Illustration of the elimination of nodes 10, 9, and 8 in process of constructing of triangulated graph

The treewidth of a triangulated graph is equal to the largest clique of triangulated graph minus 1 ([5]). Calculation of the largest clique for the triangulated graphs has complexity $O(|V| + |E|)$ ([5]). For every graph $G = (V, E)$, there exists a triangulation of G, $\overline{G} = (V, E \bigcup E_t)$, with $tw(\overline{G}) = tw(G)$. Finding the treewidth of a graph G is equivalent to finding a triangulation $\overline{G}$ of G with minimum clique size. The last two lemmas can be found in [9].

1.1 Algorithms for tree decompositions

For the given graph and integer k, deciding whether the graph has a tree decomposition with a treewidth at most k is an NP-hard problem [1]. To solve this problem different complete and heuristic algorithms have been proposed in the literature. Examples of complete algorithms for tree decompositions are [12] and [6]. Gogate and Dechter [6] reported good results for tree decompositions by using the branch and bound algorithm. They showed that their algorithm is superior compared to the algorithm proposed in [12]. The branch and bound algorithm proposed in [6] applies different pruning techniques, and provides anytime solutions, which are good upper bounds for tree decompositions.

Heuristic techniques for generation of tree decompositions with small width are mainly based on searching for a good perfect 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, the number of edges to be added to make the node simplicial (the node which neighbours are fully connected), etc. Maximum Cardinality Search (MCS) proposed by Tarjan and Yannakakis ([13] constructs the ordering of nodes iteratively by picking the next node which has the largest number of neighbors in the ordering (the ties are broken randomly). The min-fill heuristics picks iteratively the node which adds the smallest number of edges when eliminated. Min-degree heuristic picks the next vertex to be eliminated based on its degree. The next node to be eliminated is chosen based on the smallest degree. According to [6] the min-fill heuristic performs better than MCS and min-degree heuristic. Min-degree heuristic has been improved by Clautiaux et al ([3] by adding a new criterion based on the lower bound of the treewidth for the graph obtained when the node is eliminated. For other types of heuristics based on the elimination ordering of nodes see [9].

Metaheuristic approaches have also been used for tree decompositions. Simulated annealing was used by Kjaerulff ([8]). Application of genetic algorithm for tree decompositions is presented in [10]. A tabu search approach for generation of the tree decompositions has been proposed by Clautiaux et al [3]. The authors reported good results for DIMACS vertex coloring instances ([7]). Their approach improved the previous results in literature for 53% of instances. Some of the results in [3] have been further improved by Gogate and Dechter [6]. The reader is referred to [2] for other approximation algorithms, and the information for lower bounds algorithms.

In this paper we propose new heuristic algorithms with the aim of improving existing upper bounds for tree decompositions and reducing the running time of algorithms for different problems. Two simple heuristics for searching in the elimination ordering of nodes are proposed. These local heuristics are based on changing of positions of nodes in ordering, which cause the largest clique when eliminated. The proposed heuristics are exploited by a new iterated local search algorithm in the construction phase. The proposed iterative local search algorithm applies iteratively the construction heuristic and additionally includes the perturbation mechanism and the acceptance criteria. These algorithms have been applied in 62 DIMACS instances for vertex coloring. For several problems we report new upper bounds for the treewidth, and for most of problems the tree decomposition is generated in a reasonable amount of time. Our results have been compared with the results reported in [9],[6], and [3], which to our best knowledge report the best results known yet in literature considering the width of tree decompositions for these instances. For up to date information for the best upper and lower bounds for treewidth for different instances the reader is referred to TreewidthLIB:http://www.cs.uu.nl/ hansb/treewidthlib/.

2 An Iterative local search algorithm

As described in the previous section, the generation of tree decomposition with small width can be done by finding an appropriate elimination ordering which produces a triangulated graph with smallest maximum clique size. In this section we present an algorithm which searches among the possible ordering of nodes to find a small treewidth for the given graph. The algorithm contains a local search heuristic for constructing a good ordering, and the iterative process, during which the algorithm calls the local search techniques with the initial solution that is produced in previous iteration. The algorithm includes also a mechanism for acceptance of a candidate solution for the next iteration. Although the constructing phase is very important, choosing the appropriate perturbation in each iteration as well as the mechanism for acceptance of solution are also very important to obtain good results using an iterative local search algorithm. The proposed algorithm is presented in Algorithm 1.

Algorithm 1 Iterative heuristic algorithm - IHA

```
Generate initial solution $S1$

while Number of Iterations < MAXIterations do
  Get solution $S2$ from the execution of one of local search techniques proposed
  in the next section. The local search technique uses the solution $S1$ as an initial
  solution

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

  Apply perturbation in solution $S1$

end while
```

As an initial solution we use 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 fast a solution produced by these heuristics, our algorithm starts with very poor initial solution.

2.1 Local search techniques

We propose two local search methods for generation of a good solution which will be 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 those vertices in the ordering, which cause the largest clique during the elimination process. The motivation for using this method is the reduction of the number of solutions that should be evaluated. The first proposed technique (LS1) is presented in Algorithm 2.

As we see above, the proposed algorithm applies a very simple heuristic. A vertex is chosen randomly among the vertices that have the same number of neighbourhood vertices as the largest clique obtained during the elimination

Algorithm 2 Local Search Algorithm 1 - LS1 (InputSolution)

```
while NrNotImprovments < MAXNotImprovments do
    Select a vertex in the elimination ordering which causes the largest clique (ties
    are broken randomly if there are several vertices which cause the cliques with the
    same size)

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

end while
```

process. We experimented with two types of moves. 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 heuristic will stop if the solution is not improved after a certain number of iterations. Although this is a very simple heuristic, using it alone does not produce good results for the tree decompositions. Whereas combination with the iterative method (see Algorithm 1) it generates good results.

The second proposed heuristic (LS2) is presented in Algorithm 3. This technique is similar to algorithm LS1. However, in this technique in each iteration we apply the same procedure as in the LS1 with some probability p, whereas with probability $1 - p$, the best solution is selected (ties are broken randomly) from the neighbourhood of solution. The neighbourhood of a solution is obtained by generation of all solutions which are obtained by swapping of selected vertex with all its neighbour vertices in the graph.

2.2 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 or 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. We propose three perturbation mechanisms for the solution:

- 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.

Determining the number of nodes N that will be moved is complex and may be dependent on the problem. To avoid this problem we propose an adaptive

Algorithm 3 Local Search Algorithm 2 - LS2 (InputSolution)

```
while NrNotImprovments < MAXNotImprovments do
  With probability $p$:

  Select a vertex in the elimination ordering which causes the largest clique (ties
  are broken randomly)

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

  With probability $1 - p$:
  Select a vertex in the elimination ordering which causes the largest clique (ties
  are broken randomly)

  Generate neighbourhood of the solution by swapping the selected vertex with its
  neighbours, i.e. all solutions are generated by swapping the selected vertex with
  its neighbours

  Select the best solution from the generated neighbourhood

end while
```

perturbation mechanism that takes into consideration the feedback from the search process. The number of nodes N varies from 2 to 10, and the algorithm begins with small perturbation with $N = 2$. 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. We applied each perturbation mechanism separately, and also considered combination of two perturbations, so that one perturbation is applied for the certain number and another perturbation is applied for the certain next number of iterations.

2.3 Acceptance of solution in iterated algorithm

Different techniques can be applied for acceptance of the solution obtained by the local search technique. If the solution is accepted it will be perturbed and will serve as an initial solution for the next call of one of the local search techniques. We experimented with the following variants for acceptance of solution for the next iteration (see Algorithm 1):

- Solution S2 is accepted only if it has a better width than the solution S1.
- Solution S2 is always accepted.
- Solution S2 is accepted if its treewidth is not larger than the treewidth of the best yet found solution minus x, where x is an integer.

2.4 Setting of parameters

Using our algorithm we experimented with two proposed local search techniques for construction phase, different perturbation, different acceptance criteria, swap neighbourhood, and different termination criteria for the local search procedures. For algorithm LS2 we experimented with probability $p = 10, 30, 50$. Considering the acceptance of solution in iterated local search we experimented with three variants described in Section 2.3. For the third variant we experimented with $x = 2$ and $x = 3$. We did experiments with three types of perturbations: RandPert, MaxCliquePer, and DestroyPartPer. Additionally, we experimented with combination of RandPert and MaxCliquePer. The current best results presented in this paper are obtained with the iterative heuristic algorithm (IHA) and these parameters: LS1 algorithm (see Algorithm 2) is used in the construction phase and this algorithm stops if the solution does not improve for 10 iterations ($MAXNotImprovments = 10$). In the perturbation phase are used both RandPert and MaxCliquePer perturbations. Initially RandPert with $N = 2-10$ is applied. Further the algorithm switches alternatively between two perturbations RandPert and MaxCliquePer, when IHA runs for 100 iterations without improvement of a solution. For accepting of solution in IHA the third variant is used. The solution produced in construction phase is accepted if its width is not more than the width of the best current solution plus 3.

3 Computational Results

In this section we report on computational results obtained with the current implementation of methods described in this paper. The results for 62 DIMACS vertex coloring instances are given. These instances have been used for testing of several methods for tree decompositions proposed in the literature (see [9], [3], and [6]). Our algorithms have been implemented in C++ and the current experiments were performed with a Intel Pentium 4 CPU 3GHz, 1GB RAM.

We compare our results with the results reported in [9], [3], and [6]. The results reported in [9] are obtained in Pentium 3.8GHz processor. Results reported in [3] are obtained with Pentium 3 1GHz processor, and the results reported in [6] are obtained with Pentium-4 2.4 Ghz, 2GB RAM machine. To our best knowledge these papers present the best existing upper bounds for treewidth for these 62 instances.

In Table 1 the results for the treewidth for DIMACS graph coloring instances are presented. First and second columns of the table 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 [9]. The TabuS column presents the results reported in [3], while the column BB shows the results obtained with the branch and bound algorithm proposed in [6]. The last two columns present results obtained by our algorithm proposed in this paper. In our algorithm are executed three runs for each instance. In column IHA-best is given the best width obtained in three runs for each instance, and the column IHA-AVG gives the average of treewidth over 3 runs.

In Table 2 for each instance is given the time (in seconds) needed to produce the treewidth presented in Table 1 for all algorithms. The time results given in [6] present the time in which the best solutions are found. The results given in [3] present the time of the overall run of the algorithm in one instance (number of iterations is 20000 and the algorithm stops after 10000 non-improving solutions). For our algorithm are given the time results for finding of best solutions (IHA-best(AVG)) and the time of the overall run of algorithm (IHA-total (AVG)) in each instance (AVG indicates that the average over three runs is taken). IHA algorithm stops for easy instances after 10 seconds of non improvement of solution, for middle instances after 100 seconds, and for harder instances after 3000 seconds of non improvement of solution. The maximal running time of algorithm for each instance is set to be 5000 seconds.

Based on the results given in Tables 1 and 2 we conclude that our algorithm gives better results for 35 instances compared to [9] for the upper bound of treewidth, whereas algorithm in [9] gives better results than our algorithm for 1 problem. Compared to the algorithm proposed in [3] our approach gives better upper bounds for 17 instances, whereas algorithm in [3] gives better upper bounds for 5 instances. Further, compared to branch and bound algorithm proposed in [6] our algorithm gives better upper bounds for treewidth for 24 instances, whereas the branch and bound algorithm gives better results compared to our algorithm for 4 instances. Considering the time, a direct comparison of algorithms can not be done, as the algorithms are executed in computers with different processors and memory. However, as we can see based on the results in Table 2 our algorithm gives good time performance and for some instances it decreases significantly the time needed for generation of tree decompositions. Based on our experiments the efficiency of our algorithm is due to applying of LS1 algorithm in the construction phase of IHA. In LS1 only one solution is evaluated during each iteration. When using LS2 the number of solutions to be evaluated during most of iterations is much larger.

4 Conclusions

In this paper, we proposed a new heuristic algorithm for finding an upper bound of tree decompositions for a given undirected graph. The proposed algorithm has been applied in different DIMACS vertex coloring instances. The results show that our algorithm achieves very good results for the upper bound of treewidth for different instances. In particular the algorithm improves the best existing treewidth upper bounds for 17 instances, and it has a good time performance.

For the future work we are considering further improvement of proposed algorithm by automatic adaptation of different parameters such as the acceptance criteria, perturbation mechanism, and other parameters in the local search procedure. Additionally we plan to extend the existing algorithm for generation of hypertree decompositions.

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

Instance	$\|V\| / \|E\|$	KBH	TabuS	BB	IHA-best	IHA-AVG
anna	138 / 986	12	12	12	12	12
david	87 / 812	13	13	13	13	13
huck	74 / 602	10	10	10	10	10
homer	561 / 3258	31	31	31	31	31
jean	80 / 508	9	9	9	9	9
games120	120 / 638	37	33	-	**32**	32
queen5_5	25 / 160	18	18	18	18	18
queen6_6	36 / 290	26	25	25	25	25
queen7_7	49 / 476	35	35	35	35	35
queen8_8	64 / 728	46	46	46	**45**	45.3
queen9_9	81 / 1056	59	58	59	58	58
queen10_10	100 / 1470	73	72	72	72	73
queen11_11	121 / 1980	89	88	89	88	88.7
queen12_12	144 / 2596	106	**104**	110	105	106.3
queen13_13	169 / 3328	125	**122**	125	123	124
queen14_14	196 / 4186	145	141	143	141	142.7
queen15_15	225 / 5180	167	**163**	167	164	166.3
queen16_16	256 / 6320	191	186	205	186	187.7
fpsol2.i.1	269 / 11654	66	66	66	66	66
fpsol2.i.2	363 / 8691	31	31	31	31	31
fpsol2.i.3	363 / 8688	31	31	31	31	31
inithx.i.1	519 / 18707	56	56	56	56	56
inithx.i.2	558 / 13979	35	35	**31**	35	35
inithx.i.3	559 / 13969	35	35	**31**	35	35.3
miles1000	128 / 3216	49	49	49	49	49
miles1500	128 / 5198	77	77	77	77	77
miles250	125 / 387	9	9	9	9	9
miles500	128 / 1170	22	22	22	23	24.3
miles750	128 / 2113	37	36	37	36	37
mulsol.i.1	138 / 3925	50	50	50	50	50
mulsol.i.2	173 / 3885	32	32	32	32	32
mulsol.i.3	174 / 3916	32	32	32	32	32
mulsol.i.4	175 / 3946	32	32	32	32	32
mulsol.i.5	176 / 3973	31	31	31	31	31
myciel3	11 / 20	5	5	5	5	5
myciel4	23 / 71	11	10	10	10	10
myciel5	47 / 236	20	19	19	19	19
myciel6	95 / 755	35	35	35	35	35.7
myciel7	191 / 2360	74	66	**54**	66	67.7
school1	385 / 19095	244	188	-	**184**	203.3
school1_nsh	352 / 14612	192	162	-	**155**	158.7
zeroin.i.1	126 / 4100	50	50	-	50	50
zeroin.i.2	157 / 3541	33	32	-	32	32.3
zeroin.i.3	157 / 3540	33	32	-	32	32.7
le450_5a	450 / 5714	310	256	307	**253**	254.7
le450_5b	450 / 5734	313	254	309	**248**	250
le450_5c	450 / 9803	340	272	315	272	274
le450_5d	450 / 9757	326	278	303	**267**	271.3
le450_15a	450 / 8168	296	272	-	**264**	267.7
le450_15b	450 / 8169	296	**270**	289	271	273.7
le450_15c	450 / 16680	376	359	372	**357**	359.7
le450_15d	450 / 16750	375	360	371	**354**	356
le450_25a	450 / 8260	255	234	255	**221**	227.7
le450_25b	450 / 8263	251	233	251	**228**	229
le450_25c	450 / 17343	355	327	349	327	328.7
le450_25d	450 / 17425	356	336	349	**330**	333.7
dsjc125.1	125 / 736	67	65	64	**60**	60.7
dsjc125.5	125 / 3891	110	109	109	**108**	108.3
dsjc125.9	125 / 6961	119	119	119	119	119
dsjc250.1	250 / 3218	179	173	176	**169**	170.3
dsjc250.5	250 / 15668	233	232	231	**230**	230.3
dsjc250.9	250 / 27897	243	243	243	243	243

Table 2. Algorithms comparison regarding time needed for generation of tree decompositions

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	IHA-best(AVG)	IHA-total(AVG)
anna	138 / 986	1.24	2776.93	1.64	0.1	11.0
david	87 / 812	0.56	796.81	77.6538	0.1	11.0
huck	74 / 602	0.24	488.76	0.041	0.1	11.0
homer	561 / 3258	556.82	157716.56	10800	105.7	206.7
jean	80 / 508	0.29	513.76	0.05	0.1	11.0
games120	120 / 638	5.2	2372.71	-	123.3	224.3
queen5_5	25 / 160	0.04	100.36	5.409	0.1	11.0
queen6_6	36 / 290	0.16	225.55	81.32	0.1	11.0
queen7_7	49 / 476	0.51	322.4	543.3	0.1	11.0
queen8_8	64 / 728	1.49	617.57	10800	17.7	118.7
queen9_9	81 / 1056	3.91	1527.13	10800	1.0	102.0
queen10_10	100 / 1470	9.97	3532.78	10800	5.3	106.3
queen11_11	121 / 1980	23.36	5395.74	10800	11.0	112.0
queen12_12	144 / 2596	49.93	10345.14	10800	18.3	119.3
queen13_13	169 / 3328	107.62	16769.58	10800	30.7	131.7
queen14_14	196 / 4186	215.36	29479.91	10800	834.9	3835.0
queen15_15	225 / 5180	416.25	47856.25	10800	249.3	3250.0
queen16_16	256 / 6320	773.09	73373.12	10800	182.2	3183.0
fpsol2.i.1	269 / 11654	319.34	63050.58	0.587076	6.7	17.7
fpsol2.i.2	363 / 8691	8068.88	78770.05	0.510367	11.0	22.0
fpsol2.i.3	363 / 8688	8131.78	79132.7	0.492061	6.7	17.7
inithx.i.1	519 / 18707	37455.1	101007.52	26.3043	10.7	21.7
inithx.i.2	558 / 13979	37437.2	121353.69	0.05661	12.7	23.7
inithx.i.3	559 / 13969	36566.8	119080.85	0.02734	10.7	21.7
miles1000	128 / 3216	14.39	5696.73	10800	29.3	130.3
miles1500	128 / 5198	29.12	6290.44	6.759	1.0	12.0
miles250	125 / 387	10.62	1898.29	1.788	5.7	16.7
miles500	128 / 1170	4.37	4659.31	1704.62	771.8	3772.0
miles750	128 / 2113	8.13	3585.68	10800	9.7	110.7
mulsol.i.1	138 / 3925	240.24	3226.77	1.407	0.1	11.0
mulsol.i.2	173 / 3885	508.71	12310.37	3.583	0.3	11.3
mulsol.i.3	174 / 3916	527.89	9201.45	3.541	0.7	11.7
mulsol.i.4	175 / 3946	535.72	8040.28	3.622	1.0	12.0
mulsol.i.5	176 / 3973	549.55	13014.81	3.651	1.0	12.0
myciel3	11 / 20	0	72.5	0.059279	0.1	11.0
myciel4	23 / 71	0.02	84.31	0.205	0.1	11.0
myciel5	47 / 236	2	211.73	112.12	0.1	11.0
myciel6	95 / 755	29.83	1992.42	10800	0.3	11.3
myciel7	191 / 2360	634.32	19924.58	10800	11.0	22.0
school1	385 / 19095	41141.1	137966.73	-	2105.4	4794.2
school1_nsh	352 / 14612	2059.52	180300.1	-	3006.3	4885.8
zeroin.i.1	126 / 4100	17.78	2595.92	-	0.1	11.0
zeroin.i.2	157 / 3541	448.74	4825.51	-	42.7	143.7
zeroin.i.3	157 / 3540	437.06	8898.8	-	3.3	104.3
le450_5a	450 / 5714	7836.99	130096.77	10800	2336.3	4789.3
le450_5b	450 / 5734	7909.11	187405.33	10800	3641.7	5001.0
le450_5c	450 / 9803	103637.17	182102.37	10800	1057.3	3947.0
le450_5d	450 / 9757	96227.4	182275.69	10800	735.3	3736.3
le450_15a	450 / 8168	6887.15	117042.59	-	3235.0	4942.0
le450_15b	450 / 8169	6886.84	197527.14	10800	4073.0	5001.0
le450_15c	450 / 16680	122069	143451.73	10800	2446.3	4599.7
le450_15d	450 / 16750	127602	117990.3	10800	3359.3	5001.0
le450_25a	450 / 8260	4478.3	143963.41	10800	2629.7	4739.3
le450_25b	450 / 8263	4869.97	184165.21	10800	3039.3	4555.3
le450_25c	450 / 17343	10998.68	151719.58	10800	3737.3	5001.0
le450_25d	450 / 17425	11376.02	189175.4	10800	2911.0	5001.0
dsjc125.1	125 / 736	171.54	1532.93	10800	696.7	3697.7
dsjc125.5	125 / 3891	38.07	2509.97	10800	1.3	12.3
dsjc125.9	125 / 6961	55.6	1623.44	260.879	0.1	11.0
dsjc250.1	250 / 3218	5507.86	28606.12	10800	1554.3	4115.7
dsjc250.5	250 / 15668	1111.66	14743.35	10800	351.7	3352.7
dsjc250.9	250 / 27897	1414.58	30167.7	10800	0.3	11.3

Acknowledgments: This paper was supported by the Austrian Science Fund (FWF) project: *Nr. P17222-N04, Complementary Approaches to Constraint Satisfaction*

References

1. *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.*
2. *H. L. Bodlaender. Discovering treewidth.* technical report UU-CS-2005-018, Utrecht University*, 2005.*
3. *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.*
4. *D. R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs.* Pacific Journal of Mathematics*, 15:835–855, 1965.*
5. *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.*
6. *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.*
7. *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.*
8. *U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing.* Statistics and Computing*, 1:2–17, 1992.*
9. *A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments.* Electronic Notes in Discrete Mathematics 8, Elsevier Science Publishers*, 2001.*
10. *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.*
11. *N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of treewidth.* Journal Algorithms*, 7:309–322, 1986.*
12. *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.*
13. *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.*