INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

An Iterative Heuristic Algorithm for Tree Decomposition

DBAI-TR-2007-56

Nysret Musliu

DBAI TECHNICAL REPORT

2007

DBAI TECHNICAL REPORT

DBAI TECHNICAL REPORT DBAI-TR-2007-56, 2007

An Iterative Heuristic Algorithm for Tree Decomposition

Nysret Musliu,[1]

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 is presented new iterated local search algorithm to find good upper bounds for treewidth of an undirected graph. The iterated local search algorithm consist from construction phase, and includes the mechanism for perturbation of solution, and the mechanism for accepting of solution for the next iteration. In the construction phase the solutions are generated by the heuristics which searches for good elimination ordering of nodes of graph, based on moving of only vertices that produce the largest clique in the elimination process. 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. The described algorithms have a good time performance and for several instances improve the best existing upper bounds for the treewidth.

[1]Technische Universität Wien mailto: musliu@dbai.tuwien.ac.at

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 decomposition with small upper bound for treewidth is 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.

Tree decomposition has been used 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 [15]. Koster et al [13] propose the application of tree decompositions for frequency assignment problem. Tree decomposition has also been applied for problem of vertex cover on planar graphs [1]. Further, the solving of partial constraint satisfaction problems (e.g. MAX-SAT) with tree decomposition based method has been investigated in [12]. In computational biology tree decompositions has been used for protein structure prediction [20] etc.

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 [16]:

Definition 1 *(see [16], [11]) 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.

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.

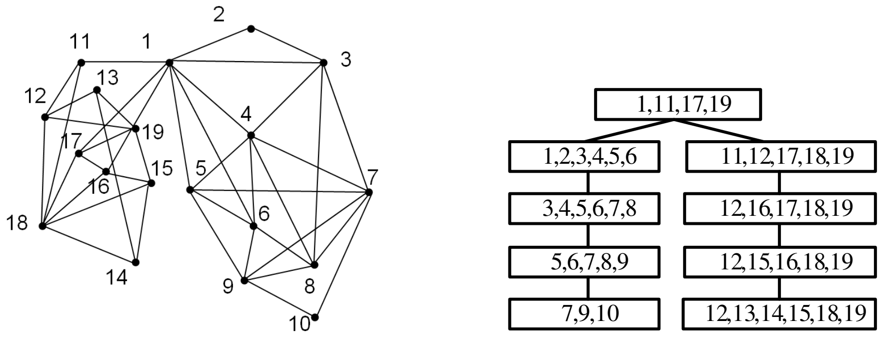

Figure 1: A graph G (left) and a tree decomposition of G (right)

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 $G[\sigma(i), \ldots, \sigma(n)]$ [5]. In [6] 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 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 [11].

For generation of tree decomposition during the vertex elimination process, first the nodes of tree decomposition are created. This is illustrated in Figure 2. When vertex 10 is eliminated a new tree decomposition node is created. This 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 $\{5, 6, 7, 8, 9\}$ 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 the 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 $\{10, 9, 8\}$ that is created when vertex 10 is eliminated, is connected with the tree decomposition node which will be created when the next vertex in the ordering which appear in $\{10, 9, 8\}$ is eliminated. In this case the node $\{10, 9, 8\}$ 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 $\{10, 9, 8\}$. This rule is further applied for connection of

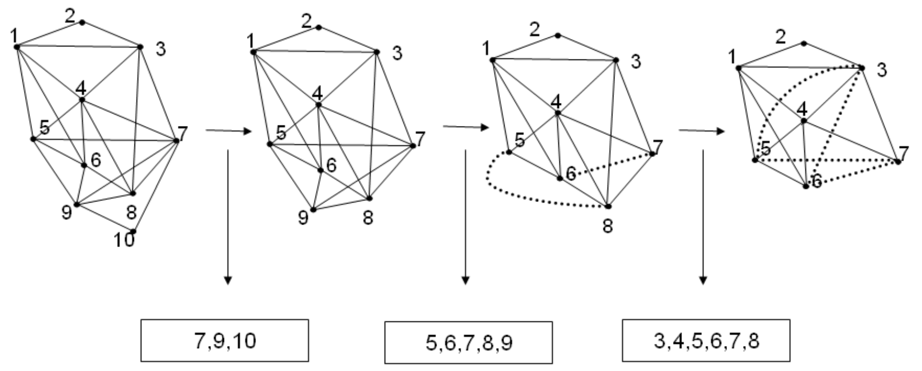

Figure 2: Illustration of the elimination of nodes 10, 9, and 8, and generation of tree decomposition nodes during the construction of triangulated graph

other tree decomposition nodes, and from the graph a part of tree decomposition in Figure 1 will be constructed.

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 [7]. Moreover, the largest clique of trinagulated graph can be calculated in polynomial time. The complexity of calculation of the largest clique for the triangulated graphs is $O(|V| + |E|)$ [7]. 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 [11]).

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 [2]. To solve this problem different complete and heuristic algorithms have been proposed in the literature. Examples of complete algorithms for tree decompositions are [18], [8], and [3]. Gogate and Dechter [8] 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 [18]. The branch and bound algorithm proposed in [8] 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 used successfully for small graphs.

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 etc.

Maximum Cardinality Search (MCS) proposed by Tarjan and Yannakakis [19], initially selects a random vertex of the graph to be the first vertex in the elimination ordering. The next vertex will be picked such that it has the highest connectivity with the vertices previously selected in the elimination ordering. The 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 (the ties are broken randomly). The selected vertex is made simplicial 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.

Minimum degree heuristic picks first the vertex with the minimum degree. 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 decomposition in a reasonable amount of time. According to [8] the min-fill heuristic performs better than MCS and min-degree heuristic. Although these heuristics give sometimes good upper bounds for tree decompositions, with more advanced techniques, usually better upper bounds can be found for most problems. Min-degree heuristic has been improved by Clautiaux et al [5] 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 [11].

Metaheuristic approaches have also been used for tree decomposition. Simulated annealing was used by Kjaerulff [10] for similar problem to tree decomposition. Application of genetic algorithm for tree decompositions is presented in [14]. The algorithm proposed by [14] with some changes in fitness function has been tested on different problems for tree decompositions in [17]. A tabu search approach for generation of the tree decompositions has been proposed by Clautiaux et al [5]. The authors reported good results for DIMACS vertex coloring instances [9]. Their approach improved the previous results in literature for 53% of instances. Some of the results in [5] have been further improved by Gogate and Dechter [8]. The reader is referred to [4] for other approximation algorithms, and the information for lower bounds algorithms.

In this chapter we propose new heuristic algorithms with the aim of improving existing upper bounds for tree decomposition 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 iterative local search algorithm applies iteratively the construction heuristic and additionally includes the perturbation mechanism and the solution 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 [11],[8], [5], and [17] 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 $S1$

$BestSolution = S1$

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 $BestSolution$
```

The proposed 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 fast a solution produced by these heuristics, our algorithm starts with very poor initial solution.

After construction of the initial solution the iterative phase starts. In this phase iteratively the construction method is called, then the solution produced in the construction phase is tested if it fulfils the acceptance criteria, and the perturbation mechanism is applied. The construction phase includes the local search procedure which is used to improve the input solution. We propose two different local search techniques that can be used in the construction phase. These techniques are described in Section 2.1. The solution returned from the construction phase will be accepted for the next iteration if it fulfils the specific criteria determined by the solution acceptance mechanism. We experimented with different possibilities for the acceptance of the solution returned from the construction phase. These variants are described in Section 2.3. If the solution does not fulfil the acceptance criteria this solution will be discarded and the best current found solution is selected. In the selected solution the perturbation mechanism is applied. Different possibilities are used for perturbation. The types of perturbation we used for experimentation are described in Section 2.2. 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. For the termination criteria the time limit is used.

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.

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. 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 swap move was shown to give better results. The heuristic will stop if the solution is not improved after a certain number of iterations. We experimented with different $MAXNotImprovments$. LS1 alone is a simple heuristic and usually can not produce good results for tree decomposition. However, by using this heuristic as a construction heuristic in the iterated local search algorithm (see Algorithm 1) good results for tree decomposition are obtained.

The second proposed heuristic (LS2) is presented in Algorithm 3. This technique is similar to algorithm LS1. However, this technique differs from LS1 considering the exploration of the neighbourhood. In LS2 in some of iterations the neighbourhood of solution consist from only one solution which is generated with swapping of vertex in the elimination ordering which causes the largest clique, with another vertex located in the randomly chosen position. The use of this

Algorithm 2 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**
    $NrNotImprovments + +$
  **end if**

**end while**

RETURN $BestLSSolution$
```

neighbourhood in particular iteration will depend from the parameter p, which determines the probability of applying this neighbourhood in each iteration. We experimented with different values for parameter p. With probability $1 - p$, other type of neighbourhood will be explored. The neighbourhood of current solution in this case consists from all solutions which can be obtained by swapping of a vertex in the elimination ordering which causes the largest clique, with the vertices in the elimination ordering, which are 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 thus the number of solution 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.

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 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

Algorithm 3 Local Search Algorithm 2 - LS2 (InputSolution)

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

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

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

end while

RETURN $BestLSSolution$
```

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.

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 is not to different from the previous solution. MaxCliquePer concentrates on moving of 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 considering for moving only the vertices that makes the width of tree decomposition large. 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 we propose an adaptive perturbation mechanism 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 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 additionally considered combination of two perturbations. The mixed perturbation applies (in Algorithm 1) two perturbations: RandPert, and MaxCliquePer. The algorithm starts with RandPert, an switchs alternatively between two perturbations if the solution is not improved for a determined number of iterations. Note that we experimented with different sizes of perturbation sizes for each type of perturbation. The experimental evaluation of different perturbations is presented in Section 3.

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 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 acceptance of solution is very restrictive. In this variant the solution from the construction phase will be 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 second variant of acceptance of solution, 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.

3 Evaluation of algorithms

The algorithms described in Section 2 are experimentally tested in DIMACS vertex coloring instances. Using our algorithm we experimented with two proposed local search techniques for construction phase, different perturbation, different acceptance criteria, swap move, and different termination criteria for the local search procedures. For algorithm LS2 we experimented with different probabilities for p. 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. For each type of perturbation mechanism we experimented with different perturbation sizes.

In Table 1 results for different perturbations mechanisms for 20 DIMACS problems are presented. These problems are selected among the hardest problems in each class of DIMACS problems. The results for all problems and the comparison with the results in the literature are presented in the next section. The perturbation mechanisms shown in Table 1 are the following:

- (P1) RandPert with size of perturbation 3
- (P2) RandPert with the size of perturbation 8
- (P3) MaxCliquePer
- (P4) DestroyPartPer with perturbation size 5
- (P5) DestroyPartPer with perturbation size 10
- (P6) Mixed perturbation (RandPert+MaxCliquePer) with the size of perturbation 3
- (P7) Mixed perturbation with perturbation size 8
- (P8) Mixed perturbation with the adaptive perturbation (with size 2-11)

In the Mixed perturbation are used both RandPert and MaxCliquePer perturbations. Initially RandPert with $N = 2 - 11$ is applied. Further the algorithm switches alternatively between two perturbations RandPert and MaxCliquePer, when IHA runs for 100 iterations without improvement of best existing solution.

Table 1: Comparision of different perturbation mechanisms

Instance	P1	P2	P3	P4	P5	P6	P7	P8
games120.col	33	33.4	32.8	33	33.2	32.2	32.4	32.4
queen14_14	145	146	146.6	143.6	144.6	142.6	145.6	143.2
queen15_15	168.4	168.4	168	168.4	169.2	167.2	167	165.4
queen16_16	189.2	193.4	194.2	191.4	193.2	191.4	191	190.8
inithx.i.3.col	35	35	35	35	35	35	35	35
miles500.col	24.2	25.2	23.2	24.6	25.4	24.2	23.8	24.2
myciel7.col	67.2	66	67.2	67.2	66.6	69	66	66
school1.col	189	195.2	226.4	189.8	196.2	193.4	199.4	187.4
school1_nsh	186.2	174	181.6	165.6	169.2	173	165.8	170.6
zeroin.i.3.col	32.8	32.8	32.6	32.8	33	32.6	33	32.6
le450_5a.col	263.6	280.6	301.4	271.4	290.2	272	278.8	279.4
le450_15b.col	278.4	284.6	287.4	279.6	290.6	278.4	282	279.6
le450_25a.col	240	245.4	253	244.4	251.8	234.8	240.8	240.2
le450_25b.col	237.4	244.6	248.2	241.6	253.2	235.4	243.2	235.8
le450_25c.col	331.2	339.8	353.6	336.8	339.8	340	336.4	334
le450_25d.col	337.4	346	351.6	341	343.6	337	339	337.2
DSJC125.1.col	61.6	62.2	61.2	61.2	63.2	63	62	61.8
DSJC125.5.col	108.2	108.4	108	108.4	108.4	108.2	108.2	108
DSJC250.1.col	171.2	172.6	176	172	176.2	171.6	171.6	171.6
DSJC250.5.col	230.6	231.2	231	230.2	231	230.6	231	230.2

The results in Table 1 presents the average width of tree decompositions over 5 runs for each example. Maximal run time of each run is 500 seconds, and the algorithm stops after 200 seconds of non improvement of the best solution. Based on the results give in the Table 1 we can conclude that considering the tree width the best results are obtained with the perturbation P8. Similar results are obtained with perturbations P1 and P6. Perturbation which includes only moving of nodes with largest cliques gives the worse results and in general for other perturbations, if the size of perturbation is large the results are worse.

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 $N = 2 - 11$ 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 smaller than the width of the best current solution plus 3.

3.1 Comparision with results in literature

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 [11], [5], and [8]). 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 [11], [5], and [8]. Additionally we include the recent results obtained by Genetic Algorithm [17]. The results reported in [11] are obtained in Pentium 3, 800 Mhz processor. Results reported in [5] are obtained with Pentium 3, 1GHz processor, and the results reported in [8] are obtained with Pentium-4, 2.4 Ghz, 2GB RAM machine. Genetic algorithm [17] has been evaluated in a Intel(R) Xeon(TM) 3.20 GHz, 4 GB RAM. To our best knowledge these papers present the best existing upper bounds for treewidth for these 62 instances.

In Tables 2, and 3 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 [11]. The TabuS column presents the results reported in [5], while the column BB shows the results obtained with the branch and bound algorithm proposed in [8]. Columns GA-best and GA-AVG represents results obtained with Genetic Algorithm [17]. Column GA-best presents the best width obtained in 10 runs, and the column GA-AVG gives the average of treewidth over 10 runs. The last two columns present results obtained by our algorithm proposed in this paper with the settings which were given in the previous section. In our algorithm are executed 10 runs for each instance. In column IHA-best is given the best width obtained in 10 runs for each instance, and the column IHA-AVG gives the average of treewidth over 10 runs.

In Tables 4, and 5 for each instance is given the time (in seconds) needed to produce the treewidth presented in Tables 2, and 3 for all algorithms. The time results given in [8] present the time in which the best solutions are found. The results given in [5] 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). The running time of GA [17] is presented in column GA. For our algorithm are given the average time in which the best solution is found (IHA-best) and the time of the overall run of algorithm (IHA-total) in each instance (average over ten runs is taken). IHA algorithm stops for easy instances after 10 seconds of non improvement of solution, for middle instances after 200 seconds, and for harder instances after 10000 seconds of non improvement of solution. The maximal running time of algorithm for each instance is set to be 10000 seconds.

Based on the results given in Tables Tables 2, 3, 4, and 5 we conclude that considering the best result over 10 runs, our algorithm gives better results for 35 instances compared to [11] for the upper bound of treewidth, whereas algorithm in [11] gives better results than our algorithm for no problem. Comparing KBH to IHA average over 10 runs, KBH gives better results for 7 instances, and IHA-AVG for 35 instances. Compared to the algorithm proposed in [5] our approach gives better upper bounds for 25 instances, whereas algorithm in [5] gives no better upper bounds than our algorithm. Comparing TabuS to our average, TabuS give better results for 21 instances, whereas IHA-AVG gives better results for 18 instances. Further, compared to branch and bound algorithm proposed in [8] 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 3 instances. Comparing this algorithm to our average, this algorithm gives better results for 11 examples, whereas IHA-AVG is better for 24 instances. Considering comparison of GA and IHA, for the best width over 10 runs, our algorithm gives better results for 20 problems, whereas GA gives better results for 5 problems. For the average width in 10 runs, IHA-AVG is better than GA-AVG in 29 examples, whereas GA-AVG is better than IHA-AVG in 12 examples. Overall our algorithm is very good compared to other algorithms considering the width, and it gives new upper bounds for 14 instances (cells of table in bold).

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 Tables 4, and 5 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 presented a new heuristic algorithm for finding an upper bound of tree decompositions for a given undirected graph. The proposed algorithm has a structure of iterated local search algorithm and it includes different perturbation mechanisms and different variants for acceptance of solution for the next iteration. For the construction phase two simple local search based heuristics are proposed. Although the proposed constructive heuristics are simple, the whole iterated local search algorithm that uses these heuristics in a construction phase gives good results for tree decomposition. In particular using of construction method which includes only moving of the nodes that produce the largest clique in the elimination ordering has been shown to be more efficient.

The proposed algorithm has been applied in 62 DIMACS vertex coloring instances. Additionally the results of our algorithm have been compared with the best existing upper bounds for width of tree decomposition for these instances. The results show that our algorithm achieves good results for the upper bound of treewidth for different size of instances. In particular the algorithm improves the best existing treewidth upper bounds for many instances, and it has a good time performance.

For the future work we are considering the hybridization of our algorithm with the genetic algorithm for generation of tree decomposition. Furthermore, the algorithms described in this chapter can be used for generation of generalized hypertree decomposition (GHD). Generalized hypertree decomposition is a concept than includes additional condition and it is applied directly into the hypergraph. Good tree decomposition usually produce good generalized hypertree decomposition and the methods used for tree decomposition can be extended to generate GHD.

Table 2: Algorithms comparison regarding treewidth for DIMACS graph coloring instances

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	GA-best	GA-AVG	IHA-best	IHA-AVG
anna	138 / 986	12	12	12	12	12	12	12
david	87 / 812	13	13	13	13	13	13	13
huck	74 / 602	10	10	10	10	10	10	10
homer	561 / 3258	31	31	31	31	31	31	31.2
jean	80 / 508	9	9	9	9	9	9	9
games120	120 / 638	37	33	-	32	32	32	32.2
queen5_5	25 / 160	18	18	18	18	18	18	18
queen6_6	36 / 290	26	25	25	26	26	25	25
queen7_7	49 / 476	35	35	35	35	35.2	35	35
queen8_8	64 / 728	46	46	46	45	46	45	45.3
queen9_9	81 / 1056	59	58	59	58	58.5	58	58.1
queen10_10	100 / 1470	73	72	72	72	72.4	72	72.3
queen11_11	121 / 1980	89	88	89	87	88.2	87	87.7
queen12_12	144 / 2596	106	104	110	104	105.7	**103**	104.4
queen13_13	169 / 3328	125	122	125	121	123.1	121	122.2
queen14_14	196 / 4186	145	141	143	141	144	**140**	142.6
queen15_15	225 / 5180	167	163	167	162	164.8	162	166.3
queen16_16	256 / 6320	191	186	205	186	188.5	186	188.2
fpsol2.i.1	269 / 11654	66	66	66	66	66	66	66
fpsol2.i.2	363 / 8691	31	31	31	32	32.6	31	31.1
fpsol2.i.3	363 / 8688	31	31	31	31	32.3	31	31.2
inithx.i.1	519 / 18707	56	56	56	56	56	56	56
inithx.i.2	558 / 13979	35	35	**31**	35	35	35	35
inithx.i.3	559 / 13969	35	35	**31**	35	35	35	35
miles1000	128 / 3216	49	49	49	50	50	49	49.2
miles1500	128 / 5198	77	77	77	77	77	77	77
miles250	125 / 387	9	9	9	10	10	9	9.3
miles500	128 / 1170	22	22	22	24	24.1	22	23.5
miles750	128 / 2113	37	36	37	37	37	36	36.9
mulsol.i.1	138 / 3925	50	50	50	50	50	50	50
mulsol.i.2	173 / 3885	32	32	32	32	32	32	32
mulsol.i.3	174 / 3916	32	32	32	32	32	32	32
mulsol.i.4	175 / 3946	32	32	32	32	32	32	32
mulsol.i.5	176 / 3973	31	31	31	31	31	31	31
myciel3	11 / 20	5	5	5	5	5	5	5
myciel4	23 / 71	11	10	10	10	10	10	10
myciel5	47 / 236	20	19	19	19	19	19	19
myciel6	95 / 755	35	35	35	35	35	35	35.4
myciel7	191 / 2360	74	66	**54**	66	66	66	67.2
school1	385 / 19095	244	188	-	185	192.5	**178**	190.5
school1_nsh	352 / 14612	192	162	-	157	163.1	**152**	156.4

Table 3: Algorithms comparison regarding treewidth for DIMACS graph coloring instances

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	GA-best	GA-AVG	IHA-best	IHA-AVG
zeroin.i.1	126 / 4100	50	50	-	50	50	50	50
zeroin.i.2	157 / 3541	33	32	-	32	32.7	32	32.7
zeroin.i.3	157 / 3540	33	32	-	32	32.9	32	32.6
le450_5a	450 / 5714	310	256	307	**243**	248.3	244	250
le450_5b	450 / 5734	313	254	309	248	249.9	**246**	249.3
le450_5c	450 / 9803	340	272	315	**265**	267.1	266	273
le450_5d	450 / 9757	326	278	303	265	265.6	265	267.2
le450_15a	450 / 8168	296	272	-	265	268.7	**262**	267.9
le450_15b	450 / 8169	296	270	289	265	269	**258**	266.7
le450_15c	450 / 16680	376	359	372	351	352.8	**350**	355.4
le450_15d	450 / 16750	375	360	371	**353**	356.9	355	357.5
le450_25a	450 / 8260	255	234	255	225	228.2	**216**	222.6
le450_25b	450 / 8263	251	233	251	227	234.5	**219**	227.2
le450_25c	450 / 17343	355	327	349	**320**	327.1	322	327.4
le450_25d	450 / 17425	356	336	349	**327**	330.1	328	332.3
dsjc125.1	125 / 736	67	65	64	61	61.9	**60**	61.1
dsjc125.5	125 / 3891	110	109	109	109	109.2	**108**	108
dsjc125.9	125 / 6961	119	119	119	119	119	119	119
dsjc250.1	250 / 3218	179	173	176	169	169.7	**167**	168.6
dsjc250.5	250 / 15668	233	232	231	230	231.4	**229**	230.1
dsjc250.9	250 / 27897	243	243	243	243	243.1	243	243

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

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.

Table 4: Algorithms comparison regarding time needed for generation of tree decompositions

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	GA	IHA-best	IHA-total
anna	138 / 986	1.24	2776.93	1.64	213	0.1	11
david	87 / 812	0.56	796.81	77.6538	154	0.1	11
huck	74 / 602	0.24	488.76	0.041	120	0.1	11
homer	561 / 3258	556.82	157716.56	10800	1118	127	327.8
jean	80 / 508	0.29	513.76	0.05	120	0	11
games120	120 / 638	5.2	2372.71	-	462	145.8	346.8
queen5_5	25 / 160	0.04	100.36	5.409	33	0.1	11
queen6_6	36 / 290	0.16	225.55	81.32	51	0.1	11.1
queen7_7	49 / 476	0.51	322.4	543.3	92	0.1	11
queen8_8	64 / 728	1.49	617.57	10800	167	28.8	229.8
queen9_9	81 / 1056	3.91	1527.13	10800	230	5.2	206.2
queen10_10	100 / 1470	9.97	3532.78	10800	339	28.3	229.3
queen11_11	121 / 1980	23.36	5395.74	10800	497	29.6	230.6
queen12_12	144 / 2596	49.93	10345.14	10800	633	106.7	304.2
queen13_13	169 / 3328	107.62	16769.58	10800	906	3266.12	10001
queen14_14	196 / 4186	215.36	29479.91	10800	1181	5282.2	10001
queen15_15	225 / 5180	416.25	47856.25	10800	1544	3029.51	10001
queen16_16	256 / 6320	773.09	73373.12	10800	2093	7764.57	10001
fpsol2.i.1	269 / 11654	319.34	63050.58	0.587076	1982	4.8	15.8
fpsol2.i.2	363 / 8691	8068.88	78770.05	0.510367	1445	8.4	19.4
fpsol2.i.3	363 / 8688	8131.78	79132.7	0.492061	1462	8.7	19.7
inithx.i.1	519 / 18707	37455.1	101007.52	26.3043	3378	10.2	21.2
inithx.i.2	558 / 13979	37437.2	121353.69	0.05661	2317	11.7	22.7
inithx.i.3	559 / 13969	36566.8	119080.85	0.02734	2261	10.6	21.6
miles1000	128 / 3216	14.39	5696.73	10800	559	54.2	255.2
miles1500	128 / 5198	29.12	6290.44	6.759	457	0.7	11.7
miles250	125 / 387	10.62	1898.29	1.788	242	2.9	13.9
miles500	128 / 1170	4.37	4659.31	1704.62	442	81	282
miles750	128 / 2113	8.13	3585.68	10800	536	112.2	313.2
mulsol.i.1	138 / 3925	240.24	3226.77	1.407	671	0.1	11
mulsol.i.2	173 / 3885	508.71	12310.37	3.583	584	0.8	11.8
mulsol.i.3	174 / 3916	527.89	9201.45	3.541	579	0.5	11.5
mulsol.i.4	175 / 3946	535.72	8040.28	3.622	578	0.9	11.9
mulsol.i.5	176 / 3973	549.55	13014.81	3.651	584	1.1	12.1
myciel3	11 / 20	0	72.5	0.059279	14	0.1	11
myciel4	23 / 71	0.02	84.31	0.205	34	0.1	11
myciel5	47 / 236	2	211.73	112.12	80	0.1	11
myciel6	95 / 755	29.83	1992.42	10800	232	0.4	11.4
myciel7	191 / 2360	634.32	19924.58	10800	757	18.2	219.2
school1	385 / 19095	41141.1	137966.73	-	4684	4688.3	10001
school1_nsh	352 / 14612	2059.52	180300.1	-	4239	4971.8	10001

Table 5: Algorithms comparison regarding time needed for generation of tree decompositions

Instance	$\|V\|/\|E\|$	KBH	TabuS	BB	GA	IHA-best	IHA-total
zeroin.i.1	126 / 4100	17.78	2595.92	-	641	0.1	11.1
zeroin.i.2	157 / 3541	448.74	4825.51	-	594	43	244
zeroin.i.3	157 / 3540	437.06	8898.8	-	585	22	223
le450_5a	450 / 5714	7836.99	130096.77	10800	6433	7110.3	10001
le450_5b	450 / 5734	7909.11	187405.33	10800	6732	5989.9	10001
le450_5c	450 / 9803	103637.17	182102.37	10800	5917	4934.8	10001
le450_5d	450 / 9757	96227.4	182275.69	10800	5402	4033.8	10001
le450_15a	450 / 8168	6887.15	117042.59	-	6876	6191	10001
le450_15b	450 / 8169	6886.84	197527.14	10800	6423	6385.7	10001
le450_15c	450 / 16680	122069	143451.73	10800	4997	4368.9	10001
le450_15d	450 / 16750	127602	117990.3	10800	4864	3441.8	10001
le450_25a	450 / 8260	4478.3	143963.41	10800	6025	7377.9	10001
le450_25b	450 / 8263	4869.97	184165.21	10800	6045	6905.8	10001
le450_25c	450 / 17343	10998.68	151719.58	10800	6189	5345.9	10001
le450_25d	450 / 17425	11376.02	189175.4	10800	6712	4118.9	10001
dsjc125.1	125 / 736	171.54	1532.93	10800	501	334.95	10001
dsjc125.5	125 / 3891	38.07	2509.97	10800	261	66.0	267.0
dsjc125.9	125 / 6961	55.6	1623.44	260.879	110	0.1	11.0
dsjc250.1	250 / 3218	5507.86	28606.12	10800	1878	4162.4	10001
dsjc250.5	250 / 15668	1111.66	14743.35	10800	648	753.5	10001
dsjc250.9	250 / 27897	1414.58	30167.7	10800	238	0.5	11.3

[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. Discovering treewidth. *technical report UU-CS-2005-018, Utrecht University*, 2005.

[5] 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.

[6] D. R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[7] 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.

[8] 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.

[9] D. S. Johnson and M. A. Trick. Clique, coloring, and satisfiability: Second DIMACS implementation challenge. *Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society*, 26, 1993.

[10] U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2(1):2–17, 1992.

[11] A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics 8, Elsevier Science Publishers*, 2001.

[12] A. Koster, S. van Hoesel, and A. Kolen. Solving partial constraint satisfaction problems with tree-decomposition. *Networks*, 40(3):170–180, 2002.

[13] Arie M.C.A. Koster, Stan P.M. van Hoesel, and Antoon W.J. 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.

[14] 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*, 7 (1):19–34, 1997.

[15] 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.

[16] N. Robertson and P. D. Seymour. Graph minors. II: algorithmic aspects of tree-width. *Journal Algorithms*, 7:309–322, 1986.

[17] Werner Schafhauser. New heuristic methods for tree decompositions and generalized hypertree decompositions. Master's thesis, Vienna University of Technology, 2006.

[18] 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.

[19] 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.

[20] Jinbo Xu, Feng Jiao, and Bonnie Berger. A tree-decomposition approach to protein structure prediction. *IEEE Computational Systems Bioinformatics Conference*, 2005.