

An Empirical Comparison of CSP Decomposition Methods

Sathiamoorthy Subbarayan

Computational Logic and Algorithms Group
IT University of Copenhagen
Denmark
sathi@itu.dk

Abstract. We present an empirical comparison of decomposition techniques for CSPs. We compare three popular heuristics for tree decomposition and an exact method for optimal tree decomposition, along with two methods for hypertree decomposition. We use a small sample of instances in the experiments. We find that the connected hypertree decomposition method finds more optimal width decompositions than the other methods. In the case, the width of the tree decomposition obtained is the criteria for comparison, then the heuristics are still the better option than the optimal decomposition methods. Especially, the min-fill heuristic quickly finds relatively very good decompositions.

1 Introduction

The CSPs from many problem domains have an underlying tree-like structure. The tree-likeness of a CSP can be captured by the *treewidth* [1] of the *constraint graph (or hypergraph)* [2] of the CSP. We have observed low treewidth in many instances from several problem domains, like *configuration*, *fault-trees*, *protein-side chain packing*, *Bayesian networks* and *digital circuits*.

When a CSP instance is known to have a low treewidth, it is often advantageous to exploit the low treewidth by obtaining a *tree decomposition* [1] of the CSP. The obtained tree decomposition can be used to speed-up constraint solving. A well-known technique for exploiting tree decomposition for CSP solving is the *tree-clustering* [2] method. The speed-up is often high in applications like exact-inference in *Bayesian networks* and *configuration*, where there is a need to compile all the solutions.

The usual method for obtaining a tree-decomposition is to use one of the several heuristics on the constraint graph of an instance. The popular heuristics are *maximum cardinality search* (mcs) [3], *minimum degree* (mindeg) [4] and *minimum fill-in* (minfill) [5]. Recently, a branch and bound method for finding *optimal* (smallest) width tree-decompositions, augmented with lower bounding techniques, called *QuickBB* [6] has been introduced. Also recently, the *hypertree decomposition* [7] method has been introduced and the corresponding notion of *hypertree width* has been shown to have better theoretical properties over the prior decomposition methods.

The several choices available now for decomposing CSPs leads to the question of whether anyone of the choices, a heuristic or an optimal tree decomposition method or an optimal hypertree decomposition method, is the natural choice to use. We think the answer to this question requires an empirical comparison of the different decomposition methods on several realistic instances. Such an empirical comparison is necessary, since theoretical worst-cases might not be observed in the instances occurring in practice. Also, an empirical comparison is required to highlight the developments in the tools implementing the different methods. This paper presents such an empirical comparison.

On the experiments comparing the different decomposition methods on a sample of instances, we find that the connected hypertree decomposition method finds more optimal decompositions than the other methods. In the case, the criteria for comparison of the methods is the width of the obtained tree decomposition, then the min-fill heuristic seems the best choice, which quickly gives relatively very good decompositions.

This paper is organized as follows. The next section gives the necessary background by defining both tree and hypertree decomposition, listing the heuristic and optimal methods for them. The Section 3 presents the empirical results we have obtained. The following section concludes the paper.

2 The Decomposition Techniques

In this section, after some preliminary definitions, we define the notion of tree decomposition, and brief the heuristics and an exact method for tree decomposition. We then present the notion of hypertree decomposition and connected hypertree decomposition.

A *CSP instance* is a triple (V, D, C) , where V is a set of variables, D is a set of domains, with one domain for each $v \in V$, and C is a set of constraints. Each constraint $c_i \in C$ is a pair (s_i, r_i) , where $s_i \subseteq V$ is the scope of the constraint and r_i is a relation over the variables in s_i . Given an assignment A with a value to all the variables in V , the assignment is a *solution* to the CSP instance, if for each constraint $c_i \in C$, the projection of A on the scope s_i belongs to the relation r_i .

The *constraint hypergraph* of a CSP instance (V, D, C) is a hypergraph of the form (V, S) . The set of nodes V denotes that the hypergraph has a node for each variable in the CSP. The set of hyperedges S is defined to be $S = \{s_i \mid (s_i, r_i) \in C\}$, i.e., the scope of each constraint forms an hyperedge in the constraint hypergraph.

Some decomposition methods work on a *constraint graph* instead of *constraint hypergraph*. The *constraint graph* of the CSP (V, D, C) is the graph (V, E) , where the set of edges E is defined by edges belonging to cliques over each $s \in S$. That is $E = \{(a, b) \mid s \in S \wedge a, b \in s\}$. Hence, each pair of variables occurring together in a constraint scope forms an edge in the constraint graph. This graph is also called *primal graph* [2].

In the rest of the paper, terms like graph and hypergraph might be used, where they actually denote a constraint graph or a constraint hypergraph of a CSP instance.

2.1 The Tree Decomposition

A *tree decomposition* [1, 2] of a constraint hypergraph (V, S) is a tree (N, A) , satisfying the following two conditions. For each node $n \in N$, let $vars(n) \subseteq V$ be the variables *contained* in the node n . The two conditions are: (1) for each $s \in S$, there should be at least one node n such that $s \subseteq vars(n)$, (2) for each variable $v \in V$, the subset of nodes in the tree containing v induce a *connected subtree*. The first condition requires that each scope needs to be *covered* by at least one tree node. The second condition states that for each variable $v \in V$, the set of nodes $\{n \mid n \in N \wedge v \in vars(n)\}$ induce a connected subtree in the tree (N, A) . Although some decomposition methods might work on constraint graphs instead of hypergraphs, they result in tree decompositions satisfying the above specified properties over the corresponding constraint hypergraph.

The *width* of a tree decomposition (N, A) is $\max_{n \in N} \{vars(n) - 1\}$. The *treewidth* of a hypergraph, is the width of a smallest width decomposition of the hypergraph.

The advantage of having a tree decomposition is that finding a solution to the CSP using a given tree decomposition of width w takes time at most *exponential* [2] in the width w . Hence, when a class of CSP instances is known to have low treewidth, it is advantageous to exploit the low treewidth for CSP solving by using tree decomposition methods.

The Heuristics The three popular heuristic methods for obtaining tree decompositions are: *maximum cardinality search* (mcs) [3], *minimum degree* (mindeg) [4] and *minimum fill-in* (minfill) [5].

The heuristics take a *constraint graph* (V, E) as input and give a tree decomposition (N, A) as output.

Let for a set of nodes K , $clique(K)$ denote the set of edges of the clique over K . Let $neighbors_G(v)$ denote the neighbors of the node v in the graph G .

All the three heuristics mainly create a *linear order* of the variables in V called *elimination order*. Given an elimination order, we can create a matching tree decomposition.

Let an elimination order of (V, E) be $v_1 < v_2 < \dots < v_{|V|}$. Then, we can obtain a sequence of graphs called *elimination graphs*: $G_0, G_1, G_2, \dots, G_{|V|}$, where $G_0 = (V, E)$. For $i > 0$, the graph G_i is defined as $G_i = (V_i, E_i)$, where $V_i = V_{i-1} \setminus \{v_i\}$ and $E_i = clique(V_i) \cap (E_{i-1} \cup clique(neighbors_{G_{i-1}}(v_i)))$.

Let $E_T = \cup_{i \in \{0, 1, \dots, |V|\}} E_i$. Then, the graph (V, E_T) is a *triangulated* graph. Now, heuristically find the *maximal cliques* of (V, E_T) . Let, N be a set of nodes with each node $n \in N$ associated with a maximal clique found. Let, for each $n \in N$, $vars(n)$ be the subset of V corresponding to the nodes in the maximal clique associated with n .

Now, form a *weighted undirected* graph $WG = (N, \text{clique}(N))$, with the weight of an edge (a, b) being the number of variables common to both the end nodes, i.e., $\text{weight}(a, b) = |\text{vars}(a) \cap \text{vars}(b)|$. Let, the set of edges A be the edges of the WG occurring in a maximum spanning tree of WG .

Then, (N, A) is the tree decomposition of the constraint graph (V, E) corresponding to the elimination order $v_1 < v_2 < \dots < v_{|V|}$.

Since each of the three heuristics essentially define an elimination order, we can obtain a corresponding tree decomposition.

The QuickBB: An Exact Method The QuickBB [6] is an exact method for finding an optimal width tree decomposition of a graph. It is a search procedure over all elimination orders. The search procedure is enhanced by augmenting efficient pruning rules and lower bounding techniques.

2.2 The Hypertree Decomposition

A *hypertree* [7] of a hypergraph (V, S) is a triple (T, χ, λ) , where $T = (N, A)$ is a *rooted tree*. Both, χ and λ are mappings such that for each $n \in N$, $\chi(n) \subseteq V$ and $\lambda(n) \subseteq S$. For any rooted subtree T' of T , where $T' = (N', A')$, $\chi(T') = \cup_{n \in N'} \chi(n)$. The *width* of a hypertree (T, χ, λ) is $\max\{|\lambda(n)| \mid n \in N\}$. Let T_n denote the subtree of T rooted at the node n .

For any subset $S' \subseteq S$, let $\text{vars}(S')$ denote $\cup_{e \in S'} e$, i.e., the set of all variables occurring in S' .

A *hypertree decomposition* [7] of a hypergraph (V, S) is a hypertree $HD = (T, \chi, \lambda)$, which has to satisfy three conditions: (1) $T = (N, A)$ is a tree decomposition of (V, S) , (2) for each $n \in N$, $\text{vars}(n) = \chi(n)$, $\chi(n) \subseteq \text{vars}(\lambda(n))$, and (3) for each $n \in N$, $\chi(T_n) \cap \text{vars}(\lambda(n)) \subseteq \chi(n)$. The *hypertree width* is the minimum width of all possible hypertree decompositions.

A *connected hypertree* [8] decomposition is a hypertree decomposition $CHTD = (T, \chi, \lambda)$, satisfying two additional conditions: (1) for the root node r of T , $|\lambda(r)| = 1$ and (2) for any non-root node $c \in N$, with node p being the parent node of c , $\forall e \in \lambda(c). \exists v \in \chi(p) \cap \chi(c). v \in e$. The connected hypertree width is the minimum width of all connected hypertree decompositions.

The first condition of the connected hypertree definition enforces that the $|\lambda(r)|$, the number of edges associated with the root node, is at most one. The second condition enforces that each edge belonging to a non-root node c should contain at least one variable occurring in both the node c and its parent node. The connected hypertree decomposition has been shown to be a practically important variant of hypertree decomposition by empirical results [8].

3 Experiments

We have tested implementations of the three heuristics, the QuickBB [6] method and the (connected) hypertree decomposition method [8]. We used a small sample of instances, exact those used in [8]. The sample of instances are from three problem domains: configuration, fault-trees, and SMT.

Name	Exact Methods				Heuristic Methods		
	CHTD	HTD	QBB-d	QBB-mf	MCS	MinDeg	MinFill
	T,Hw,W	T,Hw,W	T,W	T,W	T,W	T,W	T,W
1-32.1	54, 4*, 17	1951, 4*, 17	A, 11	A, 11	0, 13	0.004, 11	0, 12
fischer1-1	TO, 64, 83	TO, 82, 105	TO, 52	A, 28	0, 58	0, 35	0.012, 33
fischer1-6	TO, 208, 281	TO, 456, 526	TO, 151	A, 105	0.012, 298	0.032, 121	0.992, 117
baobab1	TO, 6, 28	TO, 6, 28	A, 15	A, 14	0, 19	0, 17	0, 17
baobab3	518, 6*, 17	TO, 6, 19	A, 11	A, 10	0.004, 21	0, 10	0, 10
complex	TO, 10, 47	TO, 12, 50	A, 34	A, 28	0.004, 54	0.004, 32	0.016, 28
das9201	3, 5*, 16	1046, 5*, 16	A, 10	TO, 8	0.004, 14	0, 9	0, 8
isp9601	0.3, 4*, 12	56, 4*, 12	TO, 9	TO, 8	0.004, 14	0, 10	0.004, 9
isp9603	173, 7*, 20	TO, 7, 23	TO, 20	A, 11	0.004, 20	0, 14	0.004, 12
large-part	5, 3*, 16	40, 3*, 18	A, 14	A, 14	0, 14	0.004, 14	0, 14
large2	18, 4*, 18	225, 4*, 17	A, 12	A, 11	0, 13	0, 11	0, 11
renault	0.3, 2*, 11	0.1, 2*, 11	0.01, 9*	0.01, 9*	0.004, 10	0.004, 9	0.004, 9
# optimal	8	6	1	1	n/a	n/a	n/a

Table 1. The Experimental results

CHTD: *Connected Hypertree Decomposition*, HTD: *Hypertree Decomposition*, QBB-d: *QuickBB with default initialization*, QBB-mf: *QuickBB with min-fill initialization*, T: *Time in seconds*, W: *Width of the obtained tree decomposition*, Hw: *Width of the obtained hypertree decomposition*, k*: *Optimal width*, TO: *3600 seconds time out*, A: *aborted for unknown reason*.

For the experiments using the heuristics and the hypertree decomposition methods, we used the *TreeD*¹ decomposition library, which has an implementation of them. The *TreeD* library was implemented by us. For *QuickBB*, we used an implementation (version 0.1) that was distributed online by the author².

The *TreeD* library package also contains all the instances we used in the experiments. The instances are also in the graph format used by the *QuickBB* tool.

Experiments using the *QuickBB* tool was done with two different initialization options in the tool. One was default using a random elimination order as initial order and another one using the min-fill order.

The Table 1 lists the results of our experiments.

Each tool was given at most 3600 seconds to run on each instance. In many cases, the *QuickBB* tool aborted for some unknown reasons. In the case an experiment on an instance using a method ran out of time or aborted, we list the best width found by the method before stopping. For hypertree decomposition methods, we list the width of both the hypertree decomposition (T, χ, λ) obtained and the width of the matching tree decomposition T .

¹ *TreeD* - A Tree Decomposition Library: <http://www.itu.dk/~sathi/treed/>

² Vibhav Gogate: <http://www.ics.uci.edu/~vgogate/>

As can be observed from the table, the CHTD procedure finds the most number of optimal width decompositions, followed by HTD and then QuickBB. The QuickBB was able to find the optimal width of only the smallest instance.

If the criteria for comparison is the width of the tree decomposition obtained, then QBB-mf procedure results in relatively smaller width decompositions than the rest. But, QBB-mf does not improve much the decomposition given by min-fill method even after consuming relatively a lot of time.

It seems that, on a typical instance occurring in practice the heuristics are still better than the optimal methods in quickly finding good decompositions.

4 Conclusion

We have done an empirical comparison of the different decomposition methods for CSPs. On a sample of instances we used in the experiments, the optimal connected hypertree decomposition method results in more optimal decompositions than the other methods. In the case, the criteria for comparison is the width of the obtained tree decomposition, then the min-fill heuristic seems a better choice than the rest. The min-fill heuristic finds relatively good decomposition in a very small time. The Quick-BB method although could find optimal decompositions, often results in a decomposition of almost the same width as that by the min-fill heuristic.

References

1. Robertson, N., Seymour, P.: Graph minors. II: Algorithmic aspects of tree-width. *Journal of algorithms* **7** (1986) 309–322
2. Dechter, R.: *Constraint Processing*. Morgan Kaufmann (2003)
3. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing* **13** (1984) 566–579
4. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: *Graph Theory and Computing*. Academic Press (1972) 183–217
5. Kjærulff, U.: *Triangulation of Graphs: Algorithms Giving Small Total State Space*. Technical report, University of Aalborg (1990)
6. Gogate, V., Dechter, R.: A complete anytime algorithm for treewidth. In: *UAI*. (2004) 201–208
7. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. *AIJ* **124** (2000) 243–282
8. Subbarayan, S., Andersen, H.R.: Backtracking procedures for hypertree, hyper-spread and connected hypertree decomposition of CSPs. In: *IJCAI*. (2007) 180–185