Computing and exploiting tree-decompositions for (Max-)CSP

Philippe Jégou, Samba Ndojh Ndiaye and Cyril Terrioux
LSIS - Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ.u-3mrs.fr

Abstract

Methods exploiting the tree-decomposition notion seem to provide the best approach for solving constraint networks w.r.t. the theoretical time complexity. Nevertheless, they have not shown a real practical interest yet. So, in this paper, we first study several methods for computing an approximate optimal tree-decomposition before assessing their relevance for solving CSPs. Then, we propose and compare several strategies to achieve the best depth-first traversal of the associated cluster tree w.r.t. CSP solving. These strategies concern the choice of the root cluster (i.e. the first visited cluster) and the order according to which we visit the sons of a given cluster. Finally, we propose a new decomposition strategy and heuristics which both rely on probabilistic criteria and which significantly improve the runtime.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. Modeling a problem as a CSP consists in defining a set X of variables $x_1, x_2, \ldots x_n$, which must be assigned in their respective finite domain, by satisfying a set C of constraints which express restrictions between the different possible assignments. A solution is an assignment of every variable which satisfies all the constraints. Determining if a solution exists is a NP-complete problem. In this article, we also consider the Valued CSP extension (VCSP [1]) which allows us to express and solve over-constrained CSPs (i.e. CSPs without any solution). In such a case, the problem consists in finding a complete assignment which optimizes a given criterion on the constraint satisfaction. Of course, it is harder than the decision problem.

The usual method for solving CSPs is based on backtracking search, which, in order to be efficient, must use both filtering techniques and heuristics for choosing the next variable or value. This approach, often efficient in practice, has an exponential theoretical time complexity in $O(e.d^n)$ for an instance having n variables and e constraints and whose largest domain has d values. Several works have been developed, in order to provide better theoretical complexity bounds according to particular features of the instance. The best known complexity bounds are given by the "tree-width" of a CSP (often denoted w). This parameter is related to some topological properties of the constraint graph which represents the interactions between variables via the constraints. It leads to a time complexity in $O(n.d^{w+1})$. Different methods have been proposed to reach this bound like *Tree-Clustering* [2] (see [3] for a survey and a theoretical comparison of these methods). They rely on the notion of tree-decomposition of the constraint graph. They aim to cluster variables such that the cluster arrangement is a tree. Depending on the instances, we can expect a significant gain w.r.t. enumerative approaches. Yet, the space complexity, often linear for enumerative methods, may make such an approach unusable in practice. It can be reduced to $O(n.s.d^s)$ with s the size of the largest minimal separators of the graph [4]. These theoretical results can be extended to VCSP [5]. Several works based on this approach have been performed. Most of them only present theoretical results. Except [6, 7, 8], no practical results have been provided.

This paper deals with two different ways of making efficient these methods in practice. On the one hand, we are interested in computing a good tree-decomposition. As finding an optimal tree-decomposition is NP-Hard, approximate optimal tree-decompositions are often exploited. However, although this choice is first induced by runtime reasons, it seems sensible from a practical viewpoint. Indeed, we will show that, in practice, these rough tree-decompositions are generally better than optimal ones for CSP solving. On the other hand, given a tree-decomposition, we propose and compare several strategies to achieve the best depth-first traversal of the associated

cluster tree w.r.t. CSP solving. These strategies concern the choice of the root cluster (i.e. the first visited cluster) and the order according to which we visit the sons of a given cluster. They rely on topological or semantic features of the considered instance. Then, we show how heuristics based on a probabilistic assessment of the number of partial solutions of a decomposed network can significantly improve the runtime. In particular, we introduce a new algorithm which computes a triangulation by ordering the vertices in order to minimize the mathematical expectation of the solution number in each cluster associated to a decomposition. Note that we perform this work by using the BTD method [6] which seems to be one of few structural methods which have been implemented and used successfully for practical CSP solving.

This paper is organized as follows. Section 2 provides the basic notions about CSPs and tree-decompositions. Section 3 analyzes several methods for computing tree-decompositions. In section 4, we propose some heuristic methods for guiding the exploration of the cluster tree and we assess their practical interests. In section 5, we show how these heuristics can be improved thanks to a probabilistic assessment of the solution number. In section 6, we outline some future works.

2 Preliminaries

A *constraint satisfaction problem* (CSP) is defined by a tuple (X, D, C). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. The variables are subject to the constraints from C. Given an instance (X, D, C), the CSP problem consists in determining if there is an assignment of each variable which satisfies each constraint. This problem is NP-complete. In this paper, without loss of generality, we only consider binary constraints (i.e. constraints which involve two variables). So, the structure of a CSP can be represented by the graph (X, C), called the *constraint graph*. The vertices of this graph are the variables of X and an edge joins two vertices if the corresponding variables share a constraint.

Tree-Clustering [2] is the reference method for solving CSPs thanks to the structure of its constraint graph. It is based on the notion of tree-decomposition of graphs [9]. Given a graph $G = (X, C)$, a *tree-decomposition* of G is a pair $(E, \mathcal{T})$ with $\mathcal{T} = (I, F)$ a tree and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster) E_i is a node of $\mathcal{T}$ and verifies: (1) $\cup_{i \in I} E_i = X$, (2) for each edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$, and (3) for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $E_i \cap E_j \subseteq E_k$. The width of a tree-decomposition $(E, \mathcal{T})$ is equal to $max_{i \in I}|E_i| - 1$. The *tree-width* w of G is the minimal width over all the tree-decompositions of G.

The time complexity of Tree-Clustering is $O(n.d^{w+1})$. Unfortunately, computing an optimal tree-decomposition (i.e. a tree-decomposition with width w) is NP-Hard [10]. So, many works deal with this problem. They often exploit an algorithmic approach related to *triangulated* graphs (see [11] for an introduction to triangulated graphs). A graph is said triangulated if it has a perfect elimination order, i.e. a vertex order $\sigma = (x_1, \ldots, x_n)$ such that, for any vertex x_i, the vertices in the neighborhood of x_i which follow x_i in σ form a clique. The link between triangulated graphs and tree-decompositions is obvious. Indeed, given a triangulated graph, the set of maximal cliques $E = \{E_1, E_2, \ldots, E_k\}$ of (X, C) corresponds to the family of subsets associated with a tree-decomposition. As any graph G is not necessarily triangulated, we can obtain a tree-decomposition by triangulating G. We call *triangulation* the addition to G of a set C' of edges such that $G' = (X, C \cup C')$ is triangulated. The width of G' is equal to the maximal size of cliques minus one in graph G'. The tree-width of G is then equal to the minimal width over all triangulations. Generally, when we exploit tree-decompositions for solving CSPs, we only consider approximations of optimal triangulations. Hence, the time complexity is then $O(n.d^{w^+ + 1})$ with $w^+ + 1$ the size of the largest cluster ($w + 1 \leq w^+ + 1 \leq n$). Likewise, the space complexity is $O(n.s.d^s)$ with s the size of the largest minimal *separator*, i.e. the size of the largest intersection between two clusters ($s \leq w^+$). So, to make structural methods efficient, we must a priori minimize the values of bounds w^+ and s.

The graph G in figure 1(a) is already triangulated. The maximum size of cliques is 4 and the tree-width of G is 3. Figure 1(b) presents a tree whose nodes correspond to maximal cliques. It is a possible tree-decomposition for G. So, we get $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_4, x_5, x_6\}$, and $E_4 = \{x_3, x_7, x_8\}$.

In the literature, the BTD method [6] seems provide empirical results among the best ones obtained by structural methods. Surprisingly, the results presented in [6, 7], have been obtained without exploiting any heuristic when computing a tree-decomposition. By heuristic, we mean on the one hand the way of computing a tree-

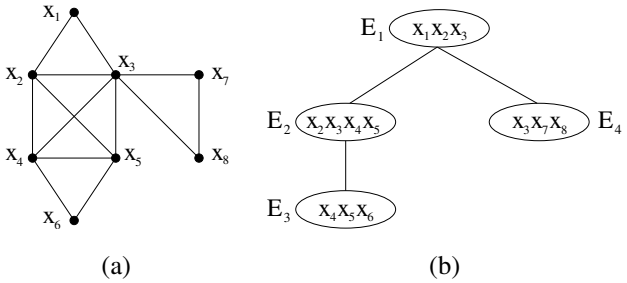

Figure 1: A constraint graph for 8 variables (a) and a possible tree-decomposition (b).

decomposition which minimizes the parameter w^+, and on the other hand, the strategies to achieve a good depth-first traversal of the cluster tree w.r.t. CSP solving. These strategies concern the choice of the root cluster (i.e. the first visited cluster) and the ordering according to which we visit the sons of a given cluster. These choices are essential since they induce a particular variable order when solving the problem and we know the importance of the variable order for the efficiency of any solving method. Moreover, such a study has never been performed (likely because, before BTD, few structural methods have been implemented and used successfully).

3 Computing a tree-decomposition

3.1 Methods based on graphic criteria

In this section, we raise the problem of computing "good" tree-decompositions by exploiting "good" triangulations. Several approaches and algorithms have been proposed for triangulations. In any case, they aim to minimize either the number of added edges, or the size of the cliques in the triangulated graph. We can distinguish four classes of approaches:

1. **Optimal triangulations.** As the problem is NP-hard, no polynomial algorithm is known yet. Hence, the proposed algorithms have an exponential time complexity. Unfortunately, their implementations do not have much interest from a practical viewpoint. For instance, the algorithm described in [12], whose time complexity is $O(n^4.(1.9601^n))$, has never been implemented due to the weak expected interest in practice [13]. Moreover, a recent work [14] has shown that the algorithm proposed in [15] cannot solve small graphs (50 vertices and 100 edges). Finally, the approach of [14] which only computes the tree-width, thanks to a branch and bound algorithm, seems promising for computing optimal triangulations. So we try to exploit it, but, unfortunately, we cannot provide any empirical results due to its practical inefficiency.

2. **Approximation algorithms.** These algorithms approximate the optimum by a constant factor. Their complexity is often polynomial in the tree-width [16]. However, this approach seems unusable. Indeed, the last algorithm proposed by Amir has a time complexity in $O(n^3.\log^4(n).k^5.\log(k))$ with an hidden constant greater than 850 [17]. Moreover, according to [18], it requires a runtime between 6 minutes and 6 days depending on the considered instance while a naive heuristic triangulation obtains better results (w.r.t. the tree-width) in at most two minutes.

3. **Minimal triangulation.** A minimal triangulation computes a set C' such that, for every subset $C" \subset C'$, the graph $G' = (X, C \cup C")$ is not triangulated. Note that a minimal triangulation is not necessarily optimal (minimum). The main interest of this approach is related to the existence of polynomial algorithms. For instance, the algorithms LEX-M [19] and LB [20] have a time complexity in $O(ne')$ with e' the number of edges in the triangulated graph.

4. **Heuristic triangulation.** These methods generally build a perfect elimination order by adding some edges to the initial graph. They often achieve this work in polynomial time (generally between $O(n + e')$ and

Instance	n	e	n-LEX-M		n-LB		n-min-fill		n-MCS	
			w^+	time	w^+	time	w^+	time	w^+	time
CELAR02	100	311	10	0.42	10	0.36	10	0.53	10	0.33
CELAR03	200	721	17	4.71	17	3.71	14	5.78	17	4.32
CELAR06	100	350	11	0.42	11	0.37	11	0.58	11	0.37
CELAR07	200	817	19	4.42	18	3.80	16	6.44	18	4.20
CELAR08	458	1655	20	55.85	19	82.73	16	73.57	19	51.74
CELAR09	340	1130	18	39.96	18	38.89	16	31.43	19	36.36
GRAPH05	100	416	28	1.00	26	0.68	25	1.34	31	0.97
GRAPH06	200	843	58	15.56	53	7.92	54	19.64	58	15.65
GRAPH11	340	1425	106	146.16	90	39.63	91	162.90	104	150.13
GRAPH12	340	1256	99	140.09	85	45.19	85	148.28	96	142.62
GRAPH13	458	1877	146	558.38	120	115.43	126	710.06	131	640.62

Table 1: Tree-width obtained after triangulation and triangulation time (in s) for graphs from "CALMA" archive.

n	p	n-LEX-M		n-LB		n-min-fill		n-MCS	
		w^+	time	w^+	time	w^+	time	w^+	time
50	20%	10.60	0.02	10.20	0.03	10.02	0.14	13.96	0.01
50	40%	11.70	0.02	10.28	0.03	10.00	0.11	15.00	0.01
50	60%	12.52	0.02	10.72	0.02	9.96	0.06	14.68	0.01
100	20%	11.86	0.18	10.24	0.34	10.22	2.15	13.92	0.06
100	40%	13.52	0.22	10.44	0.32	10.30	1.47	16.06	0.06
100	60%	15.34	0.19	10.30	0.28	10.08	0.79	17.44	0.05
200	20%	13.60	1.33	10.66	2.17	10.62	36.99	14.64	0.32
200	40%	17.78	2.20	10.68	2.90	10.72	29.61	17.48	0.42
200	60%	22.98	1.86	10.66	2.80	10.48	12.09	19.48	0.36

Table 2: Tree-width obtained after triangulation and triangulation time (in s) for random partial k-trees.

$O(n(n+e'))$) but they do not provide any minimality warranty. Nonetheless, in practice, they can be easily implemented and their interest seems justified [21]. In effect, Kjærulff has observed that these heuristics compute triangulations reasonably close to the optimum. In the following, we consider two heuristics: MCS and min-fill. MCS relies on the order computed by the algorithm of [22] which recognizes the triangulated graphs. Min-fill orders the vertices from 1 to n by choosing as next vertex one which leads to add a minimum of edges when completing the subgraph induced by its unnumbered neighbors.

3.2 Experimental study of triangulation

According to the experimentations presented in the literature, the two first approaches do not appear very interesting as a first step of a CSP solving method. Indeed, their runtime seems too expensive w.r.t. the improvement we could expect for the value w^+. Hence, we prefer to focus our attention on the two other approaches. To assess their interest, we experiment them on two kinds of benchmarks: graphs from real-world problems and random graphs which are subgraphs of graphs with suitable features w.r.t. tree decomposition. We discard classical random graphs since structural solving methods must only be applied on problems with suitable topological properties. Moreover, their tree-width is often important. For instance, for $n = 50$ and $e = 300$, the tree-width observed in [14] is about 27.

Table 1 presents empirical results[1] for some graphs of the CALMA archive[2] (real-world frequency assignment

[1] All the experimentations are performed on a Linux-based PC with a Pentium IV 2.4 GHz and 512 MB of memory. Except in table 1, the results we present are the mean results on 30 instances.

[2] For more details, see http://fap.zib.de/problems/CALMA

problems). We compare four triangulation algorithms, namely n-LEX-M, n-LB, n-min-fill and n-MCS, defined respectively from LEX-M, LB, min-fill and MCS. Precisely, each algorithm n-X fixes the choice of the first vertex and then uses the method X to order the remaining vertices. It repeats this process by choosing each vertex as the first vertex. We observe that n-LB and n-min-fill obtain the best results. These results appear generally better than ones obtained by the MSVS heuristic [23] (based on network flow techniques instead of triangulation).
Then, we apply n-LEX-M, n-LB, n-min-fill and n-MCS on random partial k-trees. These partial k-trees are produced by removing $p\%$ edges ($p = 20$, 40 or 60) from random k-trees with n vertices (i.e. triangulated graphs whose each maximal clique has $k + 1$ vertices). We exploit here a model close to one proposed in [14]. We note that the most interesting results w.r.t. tree-width (see table 2) are performed by n-min-fill, while n-LB offers a promising trade-off between the runtime and the quality of w^+.
As an indication, a random choice of the first vertex leads, of course, to worse results. However, these results are often very close to the previous ones. They are obtained in a time divided by n w.r.t. the times provided in tables 1 and 2. For instance, for CALMA problems, the time does not exceed 2 s.
Finally, note that, here, we only assess the quality of a decomposition w.r.t. the value of the parameter w^+. Nonetheless, from the viewpoint of (V)CSP solving, the most relevant criterion is related to the solving efficiency obtained thanks to the computed tree-decomposition. Of course, this computation must be achieved in reasonable time. The next section deals with this question and the way of exploring the trees associated to tree-decomposition.

4 Strategies for computing and exploring a tree-decomposition

This section aim to compare the decomposition techniques based on triangulation w.r.t. the efficiency of (V)CSP solving whereas, usually, such works only rely on the comparisons of graphical criteria like w^+. Hence, in order to fairly compare the runtime achieved from each decomposition techniques, we must determine the more relevant heuristics for exploring the cluster tree since the quality of a decomposition also depends on the way it is visited. Indeed, without suitable variable heuristics for guiding the search, we cannot expect backtracking algorithms like FC or MAC to solve efficiently any problem. It is the same for structural methods. For them, the variable order is induced by the used tree-decomposition and so the variable heuristics correspond to heuristics for achieving an interesting depth-first traversal of the cluster tree w.r.t. CSP solving. They consist in choosing the first visited cluster (called the root cluster) and ordering the sons of each cluster.

4.1 Choosing a root cluster and ordering the son clusters

At least, two kinds of criterion can be considered for choosing a root cluster. On the one hand, the *local criteria* only assess the relevance of a candidate cluster without taking into account the interactions with the other clusters. For instance, a possible local criterion is the size of the cluster (noted Size). On the other hand, the global criteria assess the relevance w.r.t. the location of the cluster in the tree. For such criteria, we use the notion of distance, noted $dist(x, y)$, between two vertices x and y of a graph G, which is defined by the length of a shortest path between x and y.
Let $G = (X, C)$ be a graph. The *centre* (CTR) of G is a vertex x s.t. x minimizes $max\{dist(x, y) : y \in X\}$. The *barycentre* (BARY) of G is a vertex x s.t. x minimizes $\Sigma_{y\in X} dist(x, y)$. A vertex x is said *peripheral* (PERI) in G if x maximizes $max\{dist(x, y) : y \in X\}$. A vertex x is said *strongly peripheral* (SPERI) in G if x maximizes $\Sigma_{y\in X} dist(x, y)$. Note that, for a tree, finding these different vertices can be achieved in $O(n^2)$. Applied to the tree associated with a tree-decomposition, these definitions must be extended. A natural extension consists in weighting the distance $dist(x, y)$ by the size of the cluster E_y. So, in the previous definitions, we replace $dist(x, y)$ by $dist(x, y).|E_y|$. We denote X_w the weighted version of criterion X.
After having chosen a root cluster, we now study the order used for visiting the sons of a given cluster. On the one hand, we define two criteria based on the size of the clusters: we first choose the largest cluster (LC) or the smallest one (SC). On the other hand, we order the clusters according to the increasing size of their separator with their parent (SEP). The strategy NO consists in exploiting no heuristic for ordering the sons.
We have experimented the different criteria on random structured (V)CSPs whose graph is a clique tree (with a limited size for cliques and separators). Each random structured instances of class (n, d, r, t, s, ns) is built from the

Class	Runtime in seconds			
(n, d, r, t, s, ns)	Size	BARY	PERI	SPERI
(75,15,10,98,2,10)	25.34	23.06	48.77	70.09
(100,10,15,30,3,15)	40.98	63.42	523.72	454.75
(125,10,15,30,3,20)	72.78	86.03	460.77	448.17
(150,10,15,29,3,25)	73.26	139.36	280.19	278.63

Table 3: [VCSP] Choosing a root cluster (results obtained without ordering sons).

Class	Runtime in seconds								
(n, d, r, t, s, ns)	Size	BARY	BARY$_w$	CTR	CTR$_w$	PERI	PERI$_w$	SPERI	SPERI$_w$
(150,25,15,200,5,15)	6.95	7.57	7.62	6.91	6.75	9.22	10.56	10.93	12.22
(150,30,15,300,5,15)	18.15	19.29	19.30	18.21	18.00	21.91	23.08	23.97	28.14
(150,25,20,171,5,15)	26.87	92.88	87.97	88.36	79.19	103.67	161.91	147.63	148.58
(200,25,15,200,5,20)	10.64	14.01	12.79	12.85	12.90	10.79	12.60	12.62	14.50
(200,25,15,204,5,25)	6.65	12.51	11.47	11.15	11.63	14.14	18.09	12.78	17.78
(225,25,15,205,5,30)	6.60	11.26	11.32	10.89	11.95	14.68	19.60	24.34	26.44

Table 4: [CSP] Choosing a root cluster (results obtained without ordering sons).

model described in [6]. It consists of n variables having d values in their domain. Its constraint graph is a clique tree with ns cliques whose largest size is r and whose separator size does not exceed s (so it is triangulated and its tree-width is r). Each constraint forbids t tuples. For VCSPs, we consider, in practice, Max-CSPs. The empirical results of tables 3-5 have been obtained from the triangulation method MCS. Of course, as the graphs are already triangulated, we have observed similar trends for the other triangulation methods, that is the best results are always performed by the same heuristics for choosing the root and ordering the sons.

Tables 3 and 4 present the results obtained for different choices of a root cluster for VCSPs and CSPs respectively. In both cases, we observe that a (strongly) peripheral root must be avoided. In contrast, the strategies BARY and CTR appear interesting even if the Size strategy seems be the best one. Regarding the weighted versions, the weighting increases the effect of the root location. Indeed, choosing a weighted (strongly) peripheral root leads to worse results than PERI and SPERI's ones while by choosing a weighted (bary)centre we obtain similar or better results than BARY and CTR's ones. So we can choose as root cluster either a (bary)centre of the tree, or the largest cluster, even if the latter seems to provide the most promising results. Here, the intuition seem to agree with the observation. Indeed, in the both cases, the cluster is chosen such that we first visit the most constrained part of the problem. In other words, these heuristics satisfy the famous first-fail principle.

In table 5, we compare the son ordering heuristics on random structured CSPs. The strategy SC appear to be a relevant criterion. Indeed, unlike the choice of a root cluster, it seems more interesting to choose first the smallest cluster. We can also note that the results obtained with the strategy SEP are close to ones of SC, what is mostly explained by the fact that, in practice, small clusters often have small separators.

Class	Runtime in seconds			
(n, d, r, t, s, ns)	NO	LC	SC	SEP
(150,25,15,200,5,15)	6.95	6.73	5.08	5.11
(150,30,15,300,5,15)	18.15	17.90	12.79	12.82
(150,25,20,171,5,15)	26.87	27.17	21.47	21.42
(200,25,15,200,5,20)	10.64	10.59	6.26	6.22
(200,25,15,204,5,25)	6.65	6.70	5.60	5.59
(225,25,15,205,5,30)	6.60	6.67	6.23	6.10

Table 5: [CSP] Ordering the cluster sons (with the Size heuristic).

Instance	d	t	Time for unlimited s				Time for s limited to 10			
			LEX-M	LB	min-fill	MCS	LEX-M	LB	min-fill	MCS
CELAR02	50	1216	2.72	2.81	2.73	2.80	2.74	2.82	2.72	2.80
CELAR03	30	373	2.22	57.71	M	1.95	2.23	2.60	1.45	1.51
CELAR06	50	1155	3.40	3.52	3.41	3.50	3.43	3.53	3.41	3.48
CELAR07	25	209	12.79	M	13.23	4.66	4.92	4.83	4.86	4.69
CELAR09	25	209	12.47	T	11.82	6.72	4.69	4.88	4.66	4.76

Table 6: [CSP] Runtime (in s) for solving CSPs respectively for an unlimited separator size s and for a size s limited to 10. T and M indicate that some instances cannot be solved either for time reason or for a lack of memory.

p	t	LEX-M		LB		min-fill		MCS		min-exp	
		w^+	time	w^+	time	w^+	time	w^+	time	w^+	time
10%	215	18.50	5.53	14.00	3.95	15.97	10.66	14.03	4.06	23.50	4.34
20%	237	22.00	4.07	14.00	4.37	16.33	6.74	14.00	3.53	23.03	3.81
30%	257	23.30	82.79	14.00	2.85	17.20	5.49	15.03	3.81	20.37	1.20
40%	285	24.90	78.22	14.00	1.11	15.33	1.21	15.33	5.88	17.10	1.57

Table 7: [CSP] Runtime (in s) and value of w^+ for class $(150, 25, 15, t, 5, 15)$ after removing $p\%$ edges (with s limited to 5).

According to these results, we exploit, in subsection 4.2, the heuristics Size for the root choice and SC for ordering sons. By so doing, we expect to fairly compare the considered decomposition methods.

4.2 Experimental study

In the frame of CSP solving, the quality of a decomposition mostly depends on the practical efficiency we obtain by exploiting it. So we experiment and compare LEX-M, LB, min-fill and MCS w.r.t. CSP solving. We consider random CSPs whose graph is one of some CALMA instances (cf. table 6). Surprisingly, the most interesting decompositions are computed by MCS. Furthermore, when we modify the decompositions by limiting the maximal size of separators, the gap between the triangulations significantly decreases. Note that, for efficiency reasons, it is our interest to reduce the value of s, by aggregating the clusters which share a large intersection.

Then, we study the triangulation interest when solving partial random structured (V)CSPs. For each instance, we randomly produce a random structured (V)CSP and then we remove $p\%$ edges like for partial k-trees. For the decision problem (see table 7), the least promising method, namely MCS, obtains interesting results. Only LB obtains similar or better results w.r.t. the value of w^+ or the CSP solving. On VCSP instances (see table 8), MCS clearly outperforms the other methods including LB although LB and MCS compute tree-decompositions with close value for w^+. Therefore, on the whole, MCS seems the most robust heuristic since it often provides the best approximation of w^+ while offering a limited value of s and solving efficiently (V)CSP.

In conclusion, we can think that the study about finding a good decomposition w.r.t. (V)CSP solving must be

p	t	LEX-M		LB		min-fill		MCS	
		w^+	time	w^+	time	w^+	time	w^+	time
10%	103	10.70	69.08	9.70	40.83	10.23	22.50	9.00	5.54
20%	110	11.27	150.41	9.60	4.24	11.87	140.33	9.00	4.34
30%	119	13.00	203.40	9.76	22.45	10.93	81.28	9.00	10.31

Table 8: [VCSP] Runtime (in s) and value of w^+ for class $(75, 15, 10, t, 2, 10)$ after removing $p\%$ edges (with s limited to 5).

carried on. Our results suggest several orientations. First, for solving CSPs, the heuristic triangulations in polynomial time might be sufficient to produce a suitable decomposition. Indeed, the optimal triangulations appear too expensive in time w.r.t. the improvement we can expect for the solving. In contrast, for VCSP solving, as the problem is significantly harder, a better decomposition might allow us to significantly reduce the runtime of the solving method. Hence, some works need to be done to propose new methods which would compute a better tree-decomposition for solving VCSPs. Finally, we have observed that the value of s is an important criterion for the practical solving efficiency. Table 6 illustrates this fact since LB or min-fill cannot success in solving efficiently some classes when they exploit an unlimited value for s. However, when s is bounded, LB and min-fill may obtain results close to ones of MCS. Likewise, bounding the value of s significantly increases the results obtained by TM.
A finer observation of our results shows that the behavior of the decomposition algorithms mostly depends on the size and the density of the produced clusters. Hence, we note that MCS obtains small clusters whose density is important, what allows to well approximate the famous first-fail principle. This principle consists in doing the choices which lead to failures as quickly as possible. By so doing, one can hope reducing the cost of failures. As this principle has significantly improved the backtracking algorithms, it could be interesting to fully introduce it in the computation of a tree-decomposition at the triangulation step or when we choose the way of exploring the cluster tree. With this aim in view, we exploit, in the next section, the mathematical expectation of the number of solution of an instance [24].

5 Heuristics based on the expected number of partial solutions

The assessment of the number of solutions by computing its mathematical expectation allows us to define new heuristics for guiding the choices done when we compute a tree-decomposition. Indeed, this criterion takes into account the problem density, the domain size and the constraint tightness. Hence, it allows us to make some choices at the triangulation step in order to produce clusters generally easier to solve and then to traverse the cluster tree by choosing first the clusters which have a minimum number of solutions. By so doing, we can expect the failures to occur earlier. In this section, we introduce this notion for proposing new triangulation and traversal heuristics.

5.1 Estimating the number of partial solutions

The mathematical expectation of the number of solutions of a CSP has been introduced in [24]. Let us consider a CSP $P = (X, D, C)$ and, for $1 \leq i < j \leq n$, the tightness t_{ij} of the constraint between x_i and x_j. In the following formula, the tightness t_{ij} is defined as the ratio between the number of forbidden tuples and the number of possible tuples, if the constraint exist, otherwise $t_{ij} = 0$. The mathematical expectation of the number of solutions of P is $E(P) = d^n \prod_{1 \leq i < j \leq n}(1 - t_{ij})$. It represents a probabilistic number of solutions. If we consider a complete assignment $a = (a_1, \ldots, a_n)$, a is a solution of P if $1 \leq i < j \leq n$, a_i and a_j are compatible w.r.t. the constraint c_{ij}. The probability of this event is $(1 - t_{ij})$. So the probability that a is a solution is $\prod_{1 \leq i < j \leq n}(1 - t_{ij})$. As the total number of assignments is d^n, the expected number of solutions is $E(P) = d^n \prod_{1 \leq i < j \leq n}(1 - t_{ij})$. This assessment provides a better estimation of the difficulty of the problem and the runtime than the density of the constraint graph or the number of constraints.
In our case, we are not interested in the expectation of the whole problem, but only in ones of the produced clusters. So, we consider the solutions of these subproblems, what explains the term of partial solutions. This assessment of the number of partial solutions allows us to improve the comparison of the decomposition heuristics.

5.2 Integration in the comparisons of tree-decomposition heuristics

The expected number of solutions is an additional criterion to fairly compare the different heuristics. Regarding the triangulation, we exploit the product of the expected number of partial solutions of each cluster as a comparison criterion. Table 9 presents the results obtained on the partial random structured CSPs of table 7. This criterion seems to be the more relevant since each consistent assignment on a cluster is a partial solution of the whole problem and, in the solving step, we try to extend the partial assignments to produce a global consistent assignment.

Note that this product considers several times the variables of each separator. However, in practice, the values it obtains corroborate the runtimes observed in table 7. Indeed, the behavior of the different heuristics for this product is close to one observed for the runtime, except for LEX-M. LEX-M produces fewer clusters but these clusters are larger. So, it is clear that the imprecisions due to separators are less important than for the other methods.

p	t	$LEX-M$	LB	$min-fill$	MCS	$min-exp$
10%	215	1.84×10^{90}	7.59×10^{94}	1.06×10^{103}	7.64×10^{90}	6.96×10^{105}
20%	237	1.28×10^{105}	1.30×10^{124}	8.04×10^{120}	6.20×10^{114}	1.73×10^{129}
30%	237	4.24×10^{157}	1.69×10^{160}	3.17×10^{166}	3.03×10^{138}	1.39×10^{158}
40%	285	5.37×10^{195}	2.99×10^{209}	8.79×10^{221}	1.01×10^{202}	1.68×10^{218}

Table 9: [CSP] Product of the expected number of partial solutions of each cluster.

Likewise, the expected number of solutions can be exploited to improved the traversal of the cluster tree. According to the first-fail principle, it seems better to start the exploration with the clusters with the smallest number of solutions. So, as this new criterion explains the behavior of the decomposition heuristics, it seems quite natural to use it to propose a more efficient decomposition heuristic.

5.3 A decomposition heuristic based on the expected number of solutions

We now define a new triangulation heuristic, denoted *min-exp* which rely on the expected number of solutions. This heuristic aims to produce cliques with a reduced number of solutions. It proceeds by ordering dynamically the vertices from 1 to n. At each step, the selected vertex is one which produces a clique whose expected number of solutions is minimum. If two vertices have the same expected number of solutions, we choose one which add the smallest number of edges. We must note that the clique produced from a vertex consists of this vertex and all its unnumbered neighbors, like for min-fill. Finally, note that the time-complexity of our approach is close to one of min-fill.

Furthermore, we have also defined a new heuristic for exploring the cluster tree. This heuristic is based both on the expectation and on the size of clusters. Indeed, our experiments have shown that it is always better to choose as root cluster a cluster with a large size while taking into account its density. Precisely, our new heuristic (denoted *MRC*) choose as root cluster one which minimizes the ratio between the expected number of solutions and the size of the cluster. Likewise, we order the sons according to the increasing value of this ratio. We denote *MRS* this heuristic. These heuristics allow to start the exploration with large clusters having few solutions.

We assess the practical interest of these new heuristics on the partial random structured CSPs of table 7. The last column of table 7 shows that min-exp is always close the best result (w.r.t. the CSP solving), despite of a larger value of w^+. Using the expected number of solutions during the triangulation step leads to add more edges and so to increase the size of clusters and separators. Furthermore, in table 7, we exploit the Size heuristic which is not the best one for this triangulation. Moreover, the columns (a) (respectively (b)) of table 10 present the runtime of each decomposition method by using the MRC heuristic for choosing the root and the SC heuristic (resp. MRS heuristic) for ordering the sons. In the both cases, we note a significant improvement for each method w.r.t. the results presented in table 7. However, MCS and min-exp remain the most efficient.

We note that the heuristics based on the expected number of solutions allow us to significantly improve the solving efficiency. Moreover, the quality of the min-exp triangulation could be improved thanks to more sensible choices in order to reduce the value of the parameter w^+.

p	t	LEX-M		LB		min-fill		MCS		min-exp	
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
10%	215	4.72	4.31	4.04	3.46	11.04	6.55	3.23	2.85	3.78	3.34
20%	237	9.97	10.29	2.65	2.57	5.13	4.50	2.91	1.87	3.35	3.33
30%	237	18.26	18.38	18.22	17.80	2.62	3.15	1.74	1.70	1.12	1.02
40%	285	1.10	0.87	1.16	0.99	1.98	2.01	0.81	0.52	0.95	0.66

Table 10: [CSP] Runtime for solving partial random structured CSP with (a) MRC and SC heuristics, (b) MRC and MRS heuristics.

6 Discussion and Conclusion

In this article, we have studied several heuristics with a view to improve the efficiency of CSP solving methods based on a tree-decomposition of the constraint network. This study, which could not be achieved previously, takes now on importance for solving hard instances with suitable structural properties. For example, the instances we have used have a suitable structure and are seldom solved by backtracking methods like FC or MAC.

First, we have considered several approaches for computing a tree-decomposition by triangulating the constraint graph. The methods which compute an optimal decomposition or approximate it by a constant factor turn to be unusable as a first step for CSP solving due to runtime reasons or implementation difficulties. In contrast, some algorithms with a polynomial time complexity (easily implemented and often relied on heuristics) compute suitable tree-decompositions w.r.t. the structural criteria and the CSP solving. However, we have noted that limiting the value of a structural parameter, namely the size s of the largest minimal separator of the used tree-decomposition, allows us to improve the solving runtime. This fact contradicts the theory (i.e. the time complexity) which requires to minimize w^+ rather than s.

Then, we have proposed and compared several strategies to achieve the best depth-first traversal of the associated cluster tree w.r.t. CSP solving. These strategies concerned the choice of the root cluster (i.e. the first visited cluster) and the order according to which we visit the sons of a given cluster. Our conclusions strengthens the famous "first-fail" principle. Indeed, the best choice for the root cluster consists in choosing the largest cluster or a (bary)centre of the cluster tree (this notion considers the (weighted) location of a given cluster in the tree). Regarding the order used for visiting the sons of a given cluster, two heuristics (namely choosing the smallest cluster first and choosing first the cluster with the smallest intersection with its parent) provide the more interesting results. In practice, the gains are about 30% on the considered instances.

Finally, we have exploited the notion of expected number of partial solutions in order to propose better heuristics for computing a tree-decomposition and for guiding the traversal of the cluster tree during the solving. This expectation approximates, in fact, the number of solutions of a subproblem and takes into account the size of domains, the number of constraints and their tightness. Hence, we have a more relevant criterion for exploiting the first-fail principle. Its use has allowed us to define a new triangulation algorithm (namely min-exp) which is more efficient in practice w.r.t. CSP solving. Likewise, we have proposed two new heuristics for choosing the root cluster and ordering the sons. These two heuristics have led to significant improvements of every triangulation heuristic while confirming the great efficiency of MCS and min-exp.

The promising gains we have observed and the limits of our study let us think that this work must be extended in at least two ways. The first way consists in improving the computation of tree-decompositions. There exist many works related to this computation. Unfortunately, most of them aim to produce a tree-decomposition with the value w^+ as small as possible. As our experiments have shown that this criterion is not relevant enough for CSP solving, new methods for computing tree-decompositions well-adapted for CSP solving, like min-exp, should be proposed. On the other hand, one can improve the efficiency of structural methods like BTD by freeing them from the variable order induced by the tree-decomposition. Indeed, the correctness of BTD requires the use of partial variable order induced by the traversal of the cluster tree. Hence, freeing structural methods from the induced order allows us to better exploit the notion of heuristic for choosing variables. Such an approach is not isolated since a recent work [25] has led to the definition of an efficient structural method for SAT.

Bibliography

1. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of IJCAI*, pages 631–637, 1995.
2. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
3. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
4. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
5. C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of CP*, pages 709–723, 2003.
6. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
7. P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proceedings of ECAI*, pages 196–200, 2004.
8. G. Gottlob, M. Hutle, and F. Wotawa. Combining hypertree, bicomp and hinge decomposition. In *Proceedings of ECAI*, pages 161–165, 2002.
9. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.
10. S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.
11. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press. New-York, 1980.
12. F. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of ICALP*, pages 568–580, 2004.
13. I. Todinca. Private communication. 2005.
14. V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of UAI*, pages 201–208, 2004.
15. K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulation. In *Proceedings of AAAI*, pages 185–190, 1997.
16. E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of UAI*, pages 7–15, 2001.
17. V. Bouchitté, D. Kratsch, H. Muller, and I. Todinca. On treewidth approximations. *Discrete Appl. Math.*, 136(2-3):183–196, 2004.
18. E. Amir. Approximation algorithms for treewidth, 2002. http://reason.cs.uiuc.edu/eyal/paper.html.
19. D. Rose, R. Tarjan, and G. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on computing*, 5:266–283, 1976.
20. A. Berry. A Wide-Range Efficient Algorithm for Minimal Triangulation. In *Proceedings of SODA*, january 1999.
21. U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark, 1990.
22. R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.
23. A. M. C. A. Koster, H. L. Bodlaender, and C. P. M. van Hoesel. Treewidth: Computational Experiments. Technical Report 01–38, Berlin, Germany, 2001.
24. B. Smith. The Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI*, pages 100–104, 1994.
25. W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proceedings of ICTAI*, pages 542–548, 2004.