Artificial Intelligence Research and Development
M. Villaret et al. (Eds.)

doi:10.3233/FAIA210116

Choosing the Root of the Tree Decomposition When Solving WCSPs: Preliminary Results

Aleksandra PETROVA [a,1], Javier LARROSA [a] and Emma ROLLON [a]
[a] *Dept. of Computer Science, UPC, Barcelona, Spain*
(petrova, larrosa, erollon)@cs.upc.edu

Abstract. In this paper we analyze the effect of selecting the root in a tree decomposition when using decomposition-based backtracking algorithms. We focus on optimization tasks for Graphical Models using the BTD algorithm. We show that the choice of the root typically has a dramatic effect in the solving performance. Then we investigate different simple measures to predict near optimal roots. Our study shows that correlations are often low, so the automatic selection of a near optimal root will require more sophisticated techniques.

Keywords. Weighted CSPs, Tree Decomposition, Discrete Optimization

1. Introduction

Many combinatorial optimization tasks on graphs can be computed in linear time when the graph is acyclic. It is well-known that in many cases we can benefit from this observation even when the graph of interest is cyclic by using a so-called *tree decomposition* which maps the graph into a tree [1]. Roughly speaking, the nodes in the decomposition contain the cyclic parts of the graph and the edges capture the acyclic interaction between the cyclic parts. Then, the tree decomposition can be used to guide a dynamic programming algorithm that proceeds bottom-up in the decomposition solving the respective subproblems. Typically such algorithms have time and space complexity exponential in the size of the largest cyclic part, which is called the (tree decomposition) *width* [2].

This approach has been widely studied in the context of *Graphical Models* [3], which is an umbrella term that covers a broad number of problems such as *Bayesian Networks*, *Markov Networks* [4], *Weighted CSPs* [5,3], etc. If we can find a small tree width for the instance of interest, then we can apply directly a dynamic programming algorithm (e.g. Bucket Elimination [6,7]) and solve it very efficiency. However, if we can only find decompositions with large width, then dynamic programming is useless (typically for its space complexity) and we have to rely on heuristic search methods such as Depth-first or Best-first. Heuristic search algorithms are unfeasible in terms of worst-case complexity because they are exponential in the problems size. However, the Graphical Models com-

[1]Projects RTI2018-094403-B-C33, funded by: FEDER/Ministerio de Ciencia e Innovacin Agencia Estatal de Investigacin,Spain

munity has enhanced them with intelligent pruning techniques and they can solve many large instances in pretty reasonable time [8].

A very fruitful line of research is the exploitation of decomposition trees beyond small widths by adapting heuristic search to the problem decomposition. The basic idea is simple and has been known for decades as AND/OR search [9]. In the context of Graphical Models corresponds to moving from a tabular to a memoization implementation of DP. However, the real challenge is to make it work in practice, that is, to add all the advantageous pruning techniques developed for standard heuristic search [10,11,12]. Such algorithms, that we call decomposition-based backtracking algorithms, have a proven time and space worst-case complexity exponential in the tree width, but their actual performance is much better than that due to the pruning techniques.

The definition of decomposition tree is not rooted, but decomposition-based algorithms must pick one. For standard implementations of dynamic programming algorithms such choice is not very important, since worst-case complexity is a tight bound of average-case complexity. However, some authors have reported that in decomposition-based backtracking algorithms choosing the root has an impact in performance [13].

In this paper we want to dig into this issue. We focus on the *backtracking tree-decomposition* algorithm BTD [10] inside toulbar2 [2] which is arguably the best example of this line of work.

We report an experiment in which we solve a carefully selected set of instances with all clusters as root. The experiment confirms that the impact of choosing a good root is high. We also observe that such impact changes very much from one problem to another. In some instances the best root hardly halves the time of the worst root. However, in other instances the best root makes the algorithm thousands of times faster than the expected time of making a random choice. The results from the experiment raise the natural question of how to identify a good root from the structure of the decomposition. To answer that question we have measured the correlation between performance and some simple yet reasonable cluster parameters. We observed that such simple parameters are often uncorrelated to performance, so they cannot be used as a heuristic way to identify a near optimal root.

To the best of our knowledge, this is the first systematic study on the impact of root selection in decomposition-based backtracking algorithms for Graphical Model optimization tasks, and the first attempt to identify a predictor of near optimal roots. Since we have not found simple single measurements correlated to algorithmic performance, we hypothesize that more sophisticated techniques such as machine learning are needed.

2. Preliminaries

2.1. Graphical Models

A *Graphical Model* is a tuple $P = \{X, D, F\}$ where $X = \{x_1, \ldots, x_n\}$ is the set of *variables*, $D = \{d_1, \ldots, d_n\}$ is the set of *domains* (d_i is the domain of x_i), F is the set of *cost functions*. Each cost function $f_S \in F$ has associated a subset of variables $S \subseteq X$, called *scope*, and the function assigns a cost to each possible assignment of these variables. A Graphical Model implicitly specifies an objective function

[2]https://miat.inrae.fr/toulbar2/

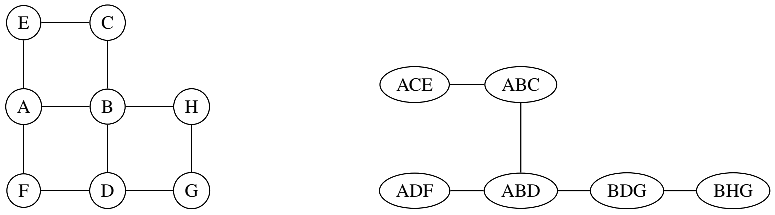

Figure 1. Interaction graph of a graphical model with 8 variables (one per node) and arity-2 cost functions (one per edge) (left) and one of its possible tree-decompositions (right).

$$F(X) = \sum_{f_i \in F} f_S(S)$$

and the goal that we are considering here is to minimize it. In probabilistic problems (i.e, the objective function has a probabilistic interpretation) this task is called *most probable explanation*. In non-probabilistic problems it is usually referred as *Weighted CSP*.

Solving a Graphical model is an NP hard problem, which means that the existence of polynomial algorithms is unlikely to exist. The design of effective algorithms for this problem has attracted a lot of research interest in the last decade [3,4].

2.2. Interaction Graph

The term Graphical Model comes from the existence of an underlying graph that captures important structural properties. Given a Graphical Model $P = \{X, D, F\}$, its *Interaction graph*, noted $G_P = (V, E)$, is an undirected graph with vertices V and edges E. There is one vertex $i \in V$ associated to each variable $x_i \in X$, and there is an edge $(i, j) \in E$ if and only if there some cost function $f_S \in F$ with $\{i, j\} \subseteq S$. Thus, the interaction graph tells pairs of variables that are linked (or connected) via cost functions. Figure 1 (left) shows the interaction graph of a graphical model with 8 variables (one per node) and 10 arity-2 cost functions (one per edge).

It is well-known that graphical models whose interaction graph is acyclic can be solved efficiently. For graphical models with a cyclic interaction graph, we can identify its acyclic sub-components through a tree-decomposition.

2.3. Tree Decomposition

A **tree-decomposition** of a graphical model $P = \{X, D, F\}$ is a tree $T = (V, A)$. For every vertex $e \in V$ there is a cluster $C_e \subseteq X$. The set of clusters must cover all the variables (i.e, $\cup_{e \in V} C_e = X$) and all the cost functions (for every $f_S \in F$ there is some cluster C_e such that $S \subseteq C_e$). Furthermore, if a variable x_i appears in two clusters C_e and C_k, it must also appear in all the clusters on the unique path from e to k (this is called the running intersection property).

The **tree-width** of a tree decomposition, noted w, is $\max_{e \in V}\{|C_e|\} - 1$. The tree-width of G is the minimum tree-width of all tree decompositions of G.

Figure 1 (right) shows a width 2 tree-decomposition of the graphical model whose cyclic interaction graph is depicted on the left.

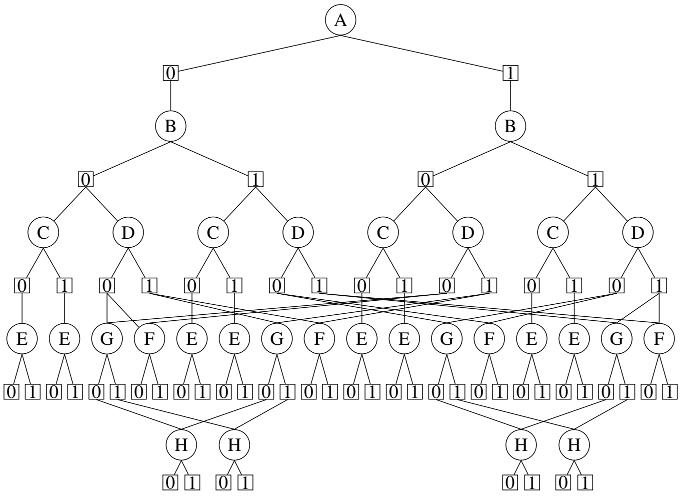

Figure 2. AND/OR graph search space.

2.4. Decomposition-based Backtracking Algorithms

Let $T = (V,A)$ be a tree decomposition of graphical model P. If we chose a node $r \in V$ as the tree root, then each tree node $e \in V$ has a parent $pa(e)$ and a set of children $ch(e)$. The separator of C_e is the set of common variables with its parent $S_e = C_e \cap pa(C_e)$. The set of proper variables of C_e is $V_e = C_e - S_e$. Note that proper variables of clusters define a partition of the variables. We denote $[i]$ the index of the cluster where x_i is a proper variable. Each cost function f_S is associated to the higher cluster that contains S and we denote $[S]$ the index of such cluster.

Decomposition-based backtracking algorithms assign variables in a top-down order with respect the decomposition. In other words, if variable x_i is a proper variable of cluster C_e (that is $[i] = e$), it cannot be assigned until all the variables of its parent $pa(C_e)$ have been assigned. Therefore, we can think of a partial assignment t as tree-structured. Corresponding to a top-down partial labeling of the variables in the decomposition.

We denote P_e the problem with all the variables of C_e and its descendent clusters and all the cost functions having in their domain at least one proper variable of these clusters. Consider a partial assignment t of the variables in P that includes the variables in C_e but does not include any proper variable of any child of C_e. Then for every C_k child of C_e, we denote $P_k(t)$ to P_k conditioned by t. Each one of these conditioned subproblems can be solved independently. Consequently, the search space traversed by this algorithms is the space of tree-structured partial assignments. To fully mimic the dynamic programming approach and inherit its good worst-case complexity, these algorithms may record the optimum of each solved subproblem, so they need not to solve it more than once. In terms of search space, this corresponds to merging nodes that correspond to identical subproblems. Figures 2 and 3 show two different search spaces for the tree decomposition of Figure 1 assuming binary domains. The two search spaces correspond to choosing two different roots for the tree-decomposition. Choosing the root producing the most cost-effective search space is the topic addressed in this paper.

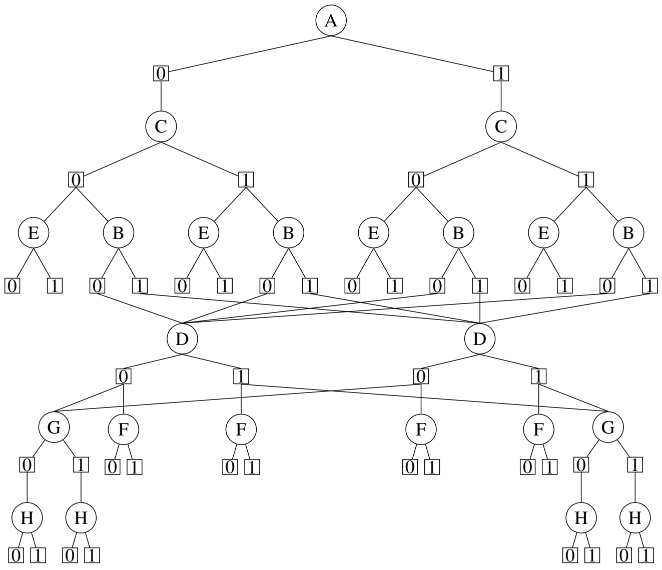

Figure 3. Another AND/OR graph search space for the same problem.

3. Experimental Design

In order to conduct our study we needed a diverse set of instances that were neither too easy (in order to make comparisons meaningful) nor too hard (so that executions would not exceed our computation budget). For that purpose we used two repositories:

- The evalgram repository (http://genoweb.toulouse.inra.fr/ degivry/evalgm/)
- The Cost Function Library (https://forgemia.inra.fr/thomas.schiex/cost-function-library/-/tree/master/)

Together they contain more than 16500 instances coming from different application domains such as *Bioinformatics*, *Boolean discrete* problems, B*ayesian netorkws*, *Airplane landing*, *Satelite observations*, *Graph coloring*, *Tractability-preserving Transformations of Global Cost Functions*, *Warehouse location problems* and many others.

We made a massive execution of all the instances using *toulbar2* default and selected instances whose execution time range from 10 to 30 minutes [3]. For each problem having selected instances we selected two of them. Then we computed their tree decomposition using the *minfill* heuristic [14] and eliminated hard instances having more than 100 clusters and instances having large tree widths (not suitable for decomposition-based algorithms). We further eliminated instances that were problematic for different reasons such as having several connected components or having soft global constraints. As a result we obtained 19 diverse and challenging instances.

[3]For obvious reasons (*i.e*, $16500 \times 0.5 = 344$ days-cpu), in this experiment we used all the machines that we had available. Although they were roughly similar, they were not identical, so cpu times between instances are not comparable

Instance Name	var.	dom.	const.	arity	costs	optimum	width	separ.	clust.
autocorr_bern50-13	50	5	4120	4	4171	747152	12	12	38
carseqtern_13_37	285	$2-13$	1046	3	1332	57	13	13	272
graph06_r	198	$6-44$	841	2	1040	4123	58	50	136
scen06-18reduc	82	$4-26$	327	2	409	3263	11	8	44
composed	83	$8-10$	624	2	707	2	46	46	34
rus_50_100_3_2	135	2	3407	2	3543	1340579	34	34	101
hole10	110	2	561	10	671	1	71	52	19
geo	50	20	466	2	516	1	22	22	24
sanr400_0.5.clq	400	2	39816	2	40217	387	381	381	19
aim	200	2	427	3	628	105	68	63	122
packupweighted1	707	2	2659	9	3367	186592	69	55	388
packupweighted2	613	2	2463	9	3077	93956	51	48	343
wellparhardtern9_9	163	$1-3$	1192	3	1355	44	61	59	102
wellpartern5_79	254	$2-13$	1046	3	1332	4035	22	16	197
parity_learn_48_24_6.2	459	$2-19$	4120	4	4171	5	23	19	436
or_chain_244.fg	750	2	1604	3	2355	397147782	84	64	591
cnf2.80.1000.266842	80	2	802	2	883	146	55	54	25
max_cut_50_500_1.asc	50	2	500	2	550	188	36	36	14
cnf3	150	2	710	3	861	64	101	98	48

Table 1. Benchmark Description. Composed, geo, aim, packupweighted1, packupweighted2 and cnf3 full names are composed-75-1-2-9_ext_100, geo50-20-d4-75-86_ext_1000, aim-200-1_6-yes1-4.cnf.mo, 978532fa-c730-11df-b070-00163e3d3b7c_l1, e69a0e36-9ef1-11df-9d4a-00163e46d37a_l1, _weightedregular, cnf2.80.1000.266842, respectively.

Table 1 contains information about the instances. It can be seen that the guarantee of benchmark diversity does not come only from the different origin of instance, but also from their syntactical structure. The number of variables (column 2) ranges from 50 to 750, the domain size (column 3) ranges from 2 to 44, the number of cost functions (column 4), the maximum arity of cost functions (column 5) ranges from 2 to 10. The number of different costs appearing over the cost functions (column 6) ranges from 409 to 40217. The optimal value (column 7) ranges from 1 to 397147782. In terms of decomposability, using the minfill heuristic the tree-width (column 8) ranges from 11 to 381 and the separator size (column 9) from 8 to 381. Finally, the number of clusters (column 10) ranges from 14 to 591.

Toulbar2 makes a sophisticated pre-process before starting the solving process. This pre-process may change slightly the problem structure (e.g. eliminating variables) and in turn the tree decomposition. To avoid that these changes could obfuscate our results, we transformed each instance to its VAC (that is, *virtual arc-consistent* [5]) pre-processed version (e.g. "`toulbar2 instance.wcsp -A -z=2`") and computed a minfill tree decomposition again (e.g. "`toulbar2 instanceVAC.wcsp -B=1 -hbfs: -O=-3 -Z=1`").

Then we solved each instance *instanceVAC.wcsp* with each node *r* of the tree decomposition as root using the BTD algorith [10] (e.g."`toulbar2 instanceVAC.wcsp -O=-3 -B=1 -hbfs: -R=r`"). In order to make the comparison of cpu time meaningful in this experiment all executions were done using identical machines (*Fujitsu Primergy CX250 S1 with Intel Xeon E5-2660 @ 2,2 Ghz and 128G of RAM*). Jobs were managed

Instance Name	**worst**	**mean**	**best**	**worst / best**	**mean / best**
autocorr_bern50-13	3059	1479	669	4.57	2.21
carseqtern_13_37	$*(17)$	1849	498	211.76	108.8
graph06_r	$*(106)$	2861	0	5077.57	4035.69
scen06-18reduc	$*(39)$	3340	24	148.95	138.21
composed	$*(19)$	2072	0	50000	28780.4
rus_50_100_3_2	630	504	375	1.68	1.34
hole10	$*(15)$	3468	1417	2.54	2.32
geo	$*(10)$	2374	720	4.99	3.16
sanr400_0.5.clq	746	627	534	1.4	1.17
aim	$*(117)$	3513	329	30.77	29.79
packupweighted1	$*(242)$	2722	327	14.88	11.22
packupweighted2	2417	1033	134	18.01	7.7
wellparhardtern9_9	$*(23)$	1596	157	156.52	68.76
wellpartern5_79	$*(17)$	1031	92	211.76	60.35
parity_learn_48_24_6.2	1874	1132	491	3.82	2.3
or_chain_244.fg	43	6	0	60.37	9.01
cnf2.80.1000.266842	$*(20)$	3356	1860	180	161.17
max_cut_50_500_1.asc	$*(1)$	2431	1325	3600	2257.49
cfn3	$*(27)$	2614	235	133.33	94.84

Table 2. Results solving each instance with every tree decomposition cluster as root. Times (**worst**, **mean** and **best** columns) are given in seconds.

with SLURM queue system with each job requesting exclusive use of 1 core and 8Gb of RAM. To bound the duration of the experiment a time out was set to 3600 seconds [4]

Results from this experiment are reported in Table 2. The second and fourth columns report execution times for the worst and best roots. In the second column, an asterisk indicates that the time out has been reached as many times as the number within parenthesis. The third column reports the average time over all the roots or, equivalently, the expected time by choosing a root randomly. However, note that in most of the instances the average is underestimated, since the timeout is often reached for several roots. Columns 5 and 6 report ratios. Note again that ratios are underestimations in all the instances where the timeout is reached.

The main observation from the ratios is that most of the times choosing the right root has a dramatic effect in the algorithm's performance. In a couple of instances, the best root makes the algorithm thousands of times faster than the worst root and even than the expected time from a random root selection. In most instances the best root makes the algorithm around an order of magnitude faster. There is only one instance where the best root is more than half the time of the worst.

[4]Note that if every execution reached the timeout a total of 2954 hours (123 days) of cpu would be required

Instance Name	$S(T,\cdot)$	$Sd(T,\cdot)$	$CD(T,\cdot)$	$H(T,\cdot)$
autocorr_bern50-13	-	-	0.95	0.95
carseqtern_13_37	0.24	0.25	0.12	0.12
graph06_r	0.35	0.36	−0.29	−0.25
scen06-18reduc	−0.8	−0.79	−0.75	−0.84
composed	0.98	0.98	−0.89	−0.89
rus_50_100_3_2	0.07	.0.07	−0.02	−0.02
hole10	−0.86	−0.86	0.99	0.99
geo	−0.72	−0.72	0.71	0.62
sanr400_0.5.clq	−0.77	−0.77	0.74	0.61
aim	0.7	0.7	−0.27	0.38
packupweighted1	-	-	0.31	0.21
packupweighted2	-	-	-	−0.34
wellparhardtern9_9	−0.15	−0.15	−0.47	−0.45
wellpartern5_79	0.04	0.03	0.11	0.05
parity_learn_48_24_6.2	−0.06	−0.05	0.05	0.08
or_chain_244.fg	0.15	0.15	−0.48	−0.08
cnf2.80.1000.266842	0.7	0.71	−0.81	−0.8
max_cut_50_500_1.asc	−0.9	−0.9	0.74	0.74
cfn3	−0.63	−0.63	0.31	0.38

Table 3. Results solving each instance with every tree decomposition cluster as root. Correlation with four different measures.

The previous results confirm our initial hypothesis of the interest of finding automatic mechanisms to identify near optimal roots. Aiming at that, we looked for a simple predictor. In particular we considered four options:

- **Cluster Size:** The size of cluster $e \in V$, noted $S(T,e)$, is the number of variables that it contains $S(T,e) = |C_e|$. It seems reasonable choosing as root r the node with the largest size $S(T,r)$, because it means that the solving process will start with a large cluster of highly connected variables. This idea follows the well-known fail-fist principle which, in our context, means that the propagation effect of local consistency properties will become apparent as early as possible. In other words, selecting the node with the largest cluster represents a greedy way to produce good lower bounds quickly which in turn will produce good pruning.
- **Domain Aware Cluster Size:** The previous definition takes all the variables as equivalent. If we want to use cluster size as a proxy of search space size a better approach is to take into account domain sizes. Accordingly, we define,

$$Sd(T,e) = \sum_{x_i \in C_e} \log |d_i|$$

  It seems reasonable choosing the root r with the largest $Sd(T,r)$ as a refinement of cluster size. Note that for instances in which all the variables have the same domain size, $Sd(T,r)$ and $S(T,r)$ are equivalent.
- **Cluster Decomposition Size:** The cluster decomposition size with root $r \in V$, noted $CD(T,r)$, is the size of the largest sub-problem after the assignment of the

variables in C_e. Formally, $CD(T,r)$ is the maximum $|D(T,e)|$ over all $e \in ch(r)$ defined as,

$$D(T,e) = V_e \cup \cup_{k \in ch(e)} D(T,k)$$

where $ch(e)$ is implicit by the choice of r as the root. It seems reasonable choosing as root the node with the smallest decomposition because it is the root that breaks the problem into sub-problems whose largest one is minimal. In other words, the assignment of its variables produces the minimal largest sub-problem. Note that this measure also favors rooting with large clusters since a large root leaves less variables for the sub-problems

- **Cluster Height:** The previous definition only considers the decomposition after the root assignment. It does not account for subsequent deeper decompositions. To incorporate that we define the height of a tree decomposition rooted with $r \in V$ as the size of the longest path starting from r. Formally, the height of root r is $H(T,r)$ with,

$$H(T,e) = |V_e| + \max_{k \in ch(e} H(T,k)$$

where $ch(e)$ is implicit by the choice of r as the root. It seems reasonable choosing as root the node with the lowest height, $\min_r H(T,r)$ because it means that backtracking will search on tree-structured assignments of minimum height (i.e, they will be the widest and most shallow).

Table 3 reports, for each instance and its given tree-decomposition T, the correlation between cpu time and the four measurements: cluster size $S(T,\cdot)$, domain-aware cluster size $Sd(T,\cdot)$, cluster decomposition size $CD(T,\cdot)$ and cluster height $H(T,\cdot)$. In the *autocorr* and *packupweighted* instances some measurements did not change over roots, so correlation could not be computed. We were expecting to find negative correlations between cpu and (domain aware) cluster size, and positive correlations between cpu and decomposition and height. Some instances (e.g. hole10, geo, max-cut, cnf) behaved as we expected and the correlations were high, which means that our four measures predict the quality of the root. In some other instances (e.g. rus, parity,...) correlations are very low, so none of our measures do not capture the quality of the root. Surprisingly, in some other instances (e.g. graph, composed,...) correlations were high but with the sign opposed to our conjecture, so our measures worked totally counter intuitively. In summary, simple synctactic measurements of the tree decomposition do not seem to capture the quality of the clusters as roots.

4. Conclusions and Future Work

In this paper we report our preliminary results in the quest of identifying near optimal roots of tree decompositions to be used in decomposition-based backtracking algorithms for Graphical Models. We proposed 4 different simple criteria based on synctactical measures of the tree-decomposition.

We created a small benchmark selecting from two well-known repositories 19 highly diverse challenging instances. We performed a systematic experiment solving each instance with every root and checked for correlations. The first lesson to be extracted from the experiment is the confirmation of the impact of the root on the algorithm performance, which validates the importance of our work. The second lesson is about the significance of the proposed measures: *i*) none of them showed significant correlation with all the instances, and *ii*) some instances correlated as expected, but others correlated counter intuitively. Therefore, we conclude that although choosing the root is important, simple measures based on syntactical features of the tree decomposition do not seem to predict well good clusters. Accordingly, in our future work we will search for more informed data (e.g. taking into account cost functions) and more sophisticated techniques to combine it (e.g. machine learning).

References

[1] Bodlaender HL, Grigoriev A, Koster AMCA. Treewidth Lower Bounds with Brambles. Algorithmica. 2008;51(1):81-98.

[2] Robertson N, Seymour PD. Graph minors. III. Planar tree-width. J Comb Theory, Ser B. 1984;36(1):49-64.

[3] Dechter R. Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms, Second Edition. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers; 2019.

[4] Darwiche A. Modeling and Reasoning with Bayesian Networks. Cambridge University Press; 2009.

[5] Cooper MC, de Givry S, Sánchez-Fibla M, Schiex T, Zytnicki M, Werner T. Soft arc consistency revisited. Artif Intell. 2010;174(7-8):449-78.

[6] Dechter R. Bucket Elimination: A Unifying Framework for Reasoning. Artif Intell. 1999;113(1-2):41-85.

[7] Bertelè U, Brioschi F. On Non-serial Dynamic Programming. J Comb Theory, Ser A. 1973;14(2):137-48.

[8] Cooper MC, de Givry S, Schiex T. Graphical Models: Queries, Complexity, Algorithms (Tutorial). In: Paul C, Bläser M, editors. 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France. vol. 154 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2020. p. 4:1-4:22.

[9] Nils N. Artificial Intelligence: A New Synthesis. Morgan Kaufmann; 1998.

[10] Terrioux C, Jégou P. Bounded Backtracking for the Valued Constraint Satisfaction Problems. In: Rossi F, editor. Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings. vol. 2833 of Lecture Notes in Computer Science. Springer; 2003. p. 709-23.

[11] Marinescu R, Dechter R. AND/OR Branch-and-Bound for Graphical Models. In: Kaelbling LP, Saffiotti A, editors. IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005. Professional Book Center; 2005. p. 224-9.

[12] Allouche D, de Givry S, Katsirelos G, Schiex T, Zytnicki M. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In: Pesant G, editor. Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. vol. 9255 of Lecture Notes in Computer Science. Springer; 2015. p. 12-29.

[13] Jégou P, Terrioux C. Combining restarts, nogoods and bag-connected decompositions for solving CSPs. Constraints An Int J. 2017;22(2):191-229.

[14] Dechter R. Constraint processing. Elsevier Morgan Kaufmann; 2003.