Practical Tractability of CSPs by Higher Level Consistency and Tree Decomposition

Shant Karakashian, Robert Woodward, and Berthe Y. Choueiry

University of Nebraska-Lincoln
{shantk|rwoodwar|choueiry}@cse.unl.edu

Abstract. One fundamental research result in the area of Constraint Processing (CP) is a condition that guarantees problem tractability by relating the consistency level of a Constraint Satisfaction Problem (CSP) to the structure of the problem. In our research, we propose to build effective problem-solving strategies that exploit the above-mentioned result in practice. To this end, our investigations target two fundamental mechanisms in CP: (1) Consistency properties and their algorithms and (2) Backtrack search in a tree decomposition. In particular, we propose a new consistency property whose level is controlled by a parameter, and present algorithms for enforcing it. Then, we investigate strategies for backtrack search that apply those algorithms in localized contexts defined by a tree decomposition of the constraint network.

1 Introduction

Research in Constraint Processing (CP) has identified various *islands of tractability* as classes of CSPs that are solvable in polynomial time in the size of the input. We single out the tractability condition specified by a relationship between the *level of a consistency* of a CSP and a structural parameter of the corresponding constraint network such as the *treewidth* or the *hypertree width*. The larger the width of the network is, the higher the level of consistency may need to be established in order to guarantee backtrack-free search. This approach is hindered in practice by two main difficulties: finding the treewidth or hypertree width of a constraint network is an $\mathcal{NP}$-hard task [1], and enforcing higher levels of consistency may require the addition of constraints to the CSP, thus modifying its structure and width parameters.

The question that we address in our research is: *How close can we approach in practice the tractability guaranteed by the relationship between the level of consistency in a CSP and the width of its constraint network?* We propose to achieve "practical tractability" by (1) proposing new local consistency properties whose level is controlled by a parameter and designing algorithms for enforcing them that do not modify the structure of the constraint network, (2) enforcing such consistency properties on the clusters of a tree decomposition of the CSP, and (3) adding redundant constraints in the separators between clusters to boost propagation and enhance communications between clusters.

The main algorithms that are used in practice for enforcing consistency consider combinations of at most three variables or two relations. The consistency properties proposed so far that apply to larger combinations of variables or constraints may in general require the addition of new constraints [2, 3]. Moreover, different problems require different levels of consistency. For this reason, it becomes important to explore new properties whose level of consistency can be controlled (i.e., parameterized consistency), but that do not modify the structure of the constraint network, and thus, do not increase its width.

The main techniques that exploit the structure of the constraint network for solving the CSP use a tree-decomposition embedding of the constraint network. Because finding the optimal decomposition is $\mathcal{NP}$-Hard, heuristics are used to find a 'good' decomposition such as join tree [4], hinge decomposition [5], and hypertree decomposition [6]. Moreover, synthesizing and storing a global constraint on the separators is necessary to guarantee backtrack-free search, but it is prohibitive in practice.

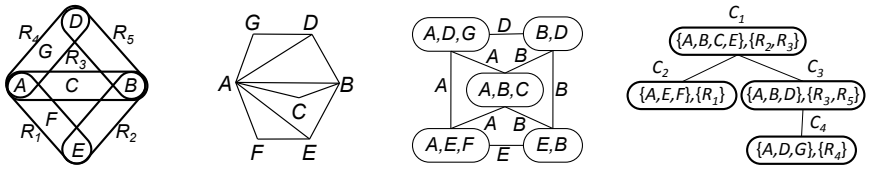
We propose to exploit the tree decomposition in the following problem-solving operations: (1) Localize the application of the consistency algorithms to the subproblems induced by the vertices of the tree decomposition; (2) Order the constraint-propagation process along the branches of the tree while favoring the most constrained paths; and (3) Enhance propagation by adding properly chosen redundant constraints between connected tree-vertices.

This paper is structured as follows. In Section 2, we give some necessary background information. In Section 3, we present the relational consistency property and outline two algorithms for enforcing it. In Section 4, we describe how we exploit the tree decomposition. Finally, in Section 5 we present some preliminary experimental results and conclude in Section 6. Preliminary results of parts of our work have already appeared in [7, 8].

2 Background

A constraint satisfaction problem (CSP) is defined by $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ is a set of domains, and $\mathcal{C}$ is a set of constraints. Each variable $A_i \in \mathcal{X}$ has a finite domain $D_i \in \mathcal{D}$, and is constrained by a subset of the constraints in $\mathcal{C}$. Each constraint $C_i \in \mathcal{C}$ is defined by a relation R_i specified over the *scope* of the constraint, *scope*(C_i), which are the variables to which the constraint applies, as a subset of the Cartesian product of the domains of those variables. The *arity* of a constraint is the cardinality of its scope. A tuple $t_i \in R_i$ is thus a combination of values for the variables in the scope of the constraint that is either allowed (i.e., support) or forbidden (i.e., conflict). In this paper, we consider only allowed tuples. A solution to the CSP is an assignment, to each variable, of a value taken from its domain such that all the constraints are satisfied. Solving a CSP consists in finding one or all solutions.

A CSP can be represented by several types of graphs: in the *hypergraph* of a CSP, as shown in Fig. 1a, the vertices represent the variables of the CSP and the hyperedges represent the scopes of the constraints. The *primal graph* of a CSP is



(a) (b) (c) (d)

Fig. 1: (a) Hypergraph, (b) Primal graph, (c) Dual graph, (d) Tree decomposition

a graph whose vertices represent the variables and the edges connect every two variables that appear in the scope of some constraint as shown in Fig. 1b. The *dual graph* of a CSP is a graph whose vertices represent the constraints of the CSP, and whose edges connect two vertices corresponding to constraints whose scopes overlap as in Fig. 1c. The dual CSP, $\mathcal{P}_D$, is thus a binary CSP where: (1) variables are the constraints of the original CSP; (2) the variables' domains are the tuples of the corresponding relations; and (3) the constraints enforce *equalities* over the shared variables. A tree decomposition of a CSP is a tree embedding of the constraint network of the CSP. The tree nodes are thus *clusters* of variables and constraints. A tree decomposition must satisfy two conditions: (1) each constraint appears in at least one cluster and the variables in its scope must appear in this cluster, and (2) for every variable, the clusters where the variable appears induce a connected subtree. Fig. 1d shows a tree decomposition of the CSP in Fig. 1a. A *separator* of two adjacent clusters is the set of variables in both clusters. A given tree decomposition is characterized by its *treewidth*, which is the maximum number of variables in a cluster minus one.

3 Relational Consistency Property and Algorithms

We introduce the property R($*$,m)C as a relational consistency property for non-binary CSPs [7, 8]. This property ensures that, given any set of m constraints, every tuple in the relation of one of those m constraints can be extended to all the variables in the union of the scopes of the constraints in an assignment that simultaneously satisfies all the constraints. We present two algorithms for enforcing this consistency property on a CSP; both algorithms are based on solving by backtrack search the dual CSP induced by the m relations:

- PerTupleSearch considers each tuple in each one of the m relations and ensures, by backtrack search, that it appears in a solution to the induced dual CSP. Thus, it solves, in the worst case, as many satisfiability problems as there are tuples in the m relations.
- AllSolSearch finds all the solutions of the induced dual CSP to determine which tuples must be kept. It executes a single backtrack search but may have to find, in the worst case, all the solutions of the induced CSP.

4 Relational Consistency on Tree Decomposition

We investigate the use of R$(*, m)$C in the context of a tree decomposition for the purpose of localizing the application of the consistency algorithms to the subproblems induced by the vertices of the clusters, ordering the constraint-propagation process along the branches of the tree, and enhancing propagation by adding redundant constraints to the separators.

Localizing Relational Consistency: Instead of computing the combinations of m constraints over the entire CSP, we propose to restrict ourselves to the combinations computed within each cluster, thus reducing the number of combinations to be considered.

Ordering the Propagation: We consider two strategies to order the clusters in which the consistency algorithm is applied. The first strategy follows the fixed order of clusters given by the MaxCliques algorithm [9]. The second strategy prioritizes the clusters by favoring those clusters whose children are most constrained. We evaluate the constraintedness of a cluster by the ratio of the removed tuples to the original tuples in its relations.

Redundancy at Separators: The application of R$(*, m)$C to a set of relations is always followed by a step where the filtered constraints are projected on the domain of the variables. When applying R$(*, m)$C individually to each cluster of a tree decomposition, the effects of filtering in one cluster are transferred to the adjacent cluster through the domains of the variables in the separator between the two clusters. Enforcing R$(*, m)$C does not require adding new constraints to the CSP. However, synthesizing a global constraint at each separator improves the 'communication' between clusters and guarantees backtrack-free search.

Synthesizing and storing those global constraints is typically prohibitive, especially in terms of space. For this reason, we propose to approximate the global constraints by adding redundant constraints. The strategy that we adopt here consists in adding the clusters' constraints to the separator after projecting them on the variables in the separator. In Section 6, we describe two other strategies.

5 Experimental Results

The experiments reported below evaluate some of the techniques described in this paper. We generate a tree decomposition that is an adaptation of the the tree-clustering technique of [4], by building the primal graph of the non-binary CSP, triangulating it, then using the join tree of the maximal cliques of the resulting triangulation. We add redundant constraints to the separators by projecting the constraints in the cluster on the variables in the separator. We use the PerTupleSearch algorithm localized to the clusters of a tree decomposition. The parameter m of R$(*, m)$C is set to the number of relations in a cluster, and, thus, the *consistency level enforced adapts locally to each cluster* in the tree

Table 1: Comparing R$(*, m)$C localized to the clusters with propagation orderings Priority and MaxClique against GAC and maxRPWC. Averages are computed over instances completed by all methods.

	#	Completed				#BT Free				Avg CPU in seconds				Avg #NV			
		Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC	Priority	MaxClique	GAC	maxRPWC
aim-100	24	20	**21**	15	16	15	**16**	1	1	18.66	**15.82**	509.02	479.67	**103.36**	**103.36**	10M	7M
aim-200	24	**9**	7	8	8	**8**	7	0	0	-	-	-	-	-	-	-	-
aim-50	24	24	24	24	24	**21**	**21**	1	3	1.01	**0.95**	1.51	1.31	**53.21**	**53.21**	43K	31K
comp-25	50	**45**	**45**	10	10	**44**	43	0	0	821.32	1,124.00	**719.61**	866.17	**45.29**	**45.29**	1M	1M
comp-75	40	**38**	37	3	3	**38**	37	0	0	8.19	4.54	**0.13**	0.17	**0.00**	**0.00**	1.00	1.00
dag-rand	25	**25**	**25**	5	0	**25**	**25**	0	0	-	-	-	-	-	-	-	-
ehi-85	100	**91**	10	74	55	**91**	10	0	0	417.65	450.23	**369.07**	663.08	**0.00**	**0.00**	86K	86K
ehi-90	100	**98**	7	55	44	**98**	7	0	0	383.21	539.02	**0.64**	0.91	**0.00**	**0.00**	10.00	10.00
modRenault	50	**50**	**50**	25	32	**50**	**50**	5	18	**5.91**	6.17	64.89	209.49	**81.39**	**81.39**	341K	1,213.30

decomposition. The propagation is ordered by the two strategies proposed in Section 4. Thus, we use two configurations for R$(*, m)$C and we refer to them as MaxClique and Priority.

We compare the two configurations of R$(*, m)$C against GAC2001 [10] and maxRPWC [11]. The four consistency algorithms are integrated as full lookahead strategies in a backtrack search using the domain/degree heuristic for dynamic variable ordering. The nodes in the search tree correspond to instantiations of the original variables of the CSP, and the count of node visits is the same for the four compared techniques. The experiments are conducted on benchmarks from the CSP Solver Competition[1] that are difficult to solve using GAC. We imposed a time limit of one hour per instance.

Table 1 gives the total number of instances in each benchmark, and the number of instances: solved within the time limit, and solved without backtracking. On the right side of the table, the average CPU time and number of nodes visited are given computed on the instances solved using the four algorithms.

We observe that R$(*, m)$C is able to solve most instances in those benchmarks without backtracking, thus achieving the practical tractability that we are aiming at. The average numbers of node visits show that R$(*, m)$C visits orders of magnitude fewer nodes than GAC and maxRPWC. The impact of fewer backtracking achieved by R$(*, m)$C is reflected in the number of instances completed in each benchmark. Also, we notice that by using the Priority ordering we are able to solve more instances than using the maxClique ordering.

6 Conclusion and Future Work

We propose a new local consistency property R$(*, m)$C with algorithms for enforcing it by localizing the application of the algorithms to the clusters of a tree

[1] http://www.cril.univ-artois.fr/CPAI08/

decomposition of a CSP. We also propose to modify the structure of the CSP to enhance propagation without increasing the width of the constraint network. We presented the results of our preliminary experiments comparing R$(*, m)$C to GAC and maxRPWC. The results indicate that we can achieve tractability in practice on many instances by solving them almost without backtracking.

In the future, we will evaluate two other strategies for approximating the global constraints on the separators: (1) adding binary constraints to the separator by generating a constraint for every fill-in edge obtained by a triangulation of the primal graph of the separator and (2) adding non-binary constraints to the separator that cover the maximal cliques of a triangulation of the separator's primal graph. Finally, we will evaluate our approach to count the number of solutions of the CSP and compare it to the BTD [12].

Acknowledgments Experiments were conducted on the equipment of the Holland Computing Center at the University of Nebraska-Lincoln. This research is supported by NSF Grant No. RI-111795.

References

1. Arnborg, S.A., Corneil, D.G., Proskurowski, A.: Complexity of Finding Embeddings in a K-Tree. SIAM Journal on Algebraic Discrete Methods **8** (1987) 277–284
2. Freuder, E.C.: Synthesizing Constraint Expressions. Communications of the ACM **21 (11)** (1978) 958–966
3. Dechter, R., van Beek, P.: Local and Global Relational Consistency. Theoretical Computer Science **173**(1) (1997) 283–308
4. Dechter, R., Pearl, J.: Tree Clustering for Constraint Networks. Artificial Intelligence **38** (1989) 353–366
5. Cohen, D.A., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. Journal of Computer and System Sciences **74**(5) (2008) 721–743
6. Gottlob, G., Scarcello, F.: Hypertree decompositions: A survey. In: Proceedings of the 26^{th} International Symposium on Mathematical Foundations of Computer Science (MFCS 01). (2001) 37–57
7. Karakashian, S., Woodward, R., Reeson, C., Choueiry, B.Y., Bessiere, C.: A First Practical Algorithm for High Levels of Relational Consistency. In: Proceedings of the 24^{th} AAAI Conference on Artificial Intelligence (AAAI 10). (2010) 101–107
8. Karakashian, S., Woodward, R.J., Choueiry, B.Y., Bessiere, C.: Relational Consistency by Constraint Filtering. In: Proceedings of the 25^{th} ACM Symposium On Applied Computing (ACM SAC 10). (2010) 2073–2074
9. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press Inc., New York, NY (1980)
10. Bessiere, C., Régin, J.C., Yap, R.H., Zhang, Y.: An Optimal Coarse-Grained Arc Consistency Algorithm. Artificial Intelligence **165(2)** (2005) 165–185
11. Bessiere, C., Stergiou, K., Walsh, T.: Domain Filtering Consistencies for Non-Binary Constraints. Artificial Intelligence **172** (2008) 800–822
12. Jégou, P., Terrioux, C.: Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks. Artificial Intelligence **146** (2003) 43–75