Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Improving the Performance of Consistency Algorithms by Localizing and Bolstering Propagation in a Tree Decomposition

Shant Karakashian and **Robert J. Woodward** and **Berthe Y. Choueiry**
Constraint Systems Laboratory, University of Nebraska-Lincoln, USA
$\{$*shantk*$|$*rwoodwar*$|$*choueiry*$\}$@*cse.unl.edu*

Abstract

The tractability of a Constraint Satisfaction Problem (CSP) is guaranteed by a direct relationship between its consistency level and a structural parameter of its constraint network such as the treewidth. This result is not widely exploited in practice because enforcing higher-level consistencies can be costly and can change the structure of the constraint network and increase its width. Recently, R($*$,m)C was proposed as a relational consistency property that does not modify the structure of the graph and, thus, does not affect its width. In this paper, we explore two main strategies, based on a tree decomposition of the CSP, for improving the performance of enforcing R($*$,m)C and getting closer to the above tractability condition. Those strategies are: *a*) *localizing* the application of the consistency algorithm to the clusters of the tree decomposition, and *b*) *bolstering constraint propagation* between clusters by adding redundant constraints at their separators, for which we propose three new schemes. We characterize the resulting consistency properties by comparing them, theoretically and empirically, to the original R($*$,m)C and the popular GAC and maxRPWC, and establish the benefits of our approach for solving difficult problems.

1 Introduction

One tractability condition of Constraint Satisfaction Problems (CSPs) relates the *level of consistency* of a CSP to a structural parameter of its constraint network such as the *treewidth* or the *hypertree width* (Freuder 1982; Dechter 2003). In this paper, we propose new consistency properties to approximate that tractability condition, and empirically show that, for difficult problems, they outperform the widely used GAC, maxRPWC (Bessiere, Stergiou, and Walsh 2008) and also R($*$,m)C (Karakashian et al. 2010). We state from the outset that GAC is cheap, easy to implement, and widely successful. We focus here on problems that are too hard to solve with GAC. We show that such problems exist, and that they benefit from our approach. Defining criteria to predict whether GAC is appropriate or too weak to use is an important open question but beyond the scope of this paper. *The merit of our approach is in exploring how to solve problems for which GAC is not powerful enough.*

The consistency properties we propose are based on the relational consistency property R($*$,m)C (Karakashian et al. 2010), also called m-wise consistency in Relational Databases (Gyssens 1986). The advantage of R($*$,m)C is that the algorithm for enforcing it *a*) is parameterized by the level of consistency m; and *b*) does not add new constraints to the constraint network or modify its width. We propose two main strategies for improving the performance of this algorithm in the context of a tree decomposition of the CSP:

1. We *localize* the application of the consistency algorithm to each cluster of the tree decomposition, which allows us to increase the value of m to the number of relations in the cluster and, thus, the level of consistency enforced.
2. We *bolster* constraint propagation along the tree by adding redundant constraints at the separators between clusters. A perfect 'communication' between clusters requires a *unique* constraint over the separator's variables, but materializing such a constraint is prohibitive in terms of space (Fattah and Dechter 1996; Kask et al. 2005). We propose three approximation schemes to this end.

The contributions of this paper are as follows:

1. New relational consistency properties resulting from structurally exploiting a tree decomposition of a CSP.
2. A theoretical characterization of those new properties.
3. An Empirical evaluation of our approach establishing its benefits on difficult benchmarks, solving many problems in a backtrack-free manner and, thus, approaching 'practical tractability.'

The paper is structured as follows. Section 2 provides background information. Sections 3 and 4 explain consistency localization and propagation bolstering. Section 5 theoretically characterizes the new consistency properties. Section 6 discusses related work, and Section 7 our experiments and empirical results. Finally, Section 8 concludes the paper.

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 in $\mathcal{X}$ has a finite domain 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: it is two for a binary constraint, and greater than two for a non-binary constraint. 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 of finding one or all solutions.

There exist several graphical representations of a CSP. In the *hypergraph*, the vertices represent the variables of the CSP, and the hyperedges represent the scopes of the constraints (see example in Figure 1). In the *dual graph*, the vertices represent the constraints of the CSP, and the edges connect vertices corresponding to constraints whose scopes overlap (see Figure 2). In the *primal graph*, the vertices represent the CSP variables, and the edges connect every two variables that appear in the scope of some constraint.

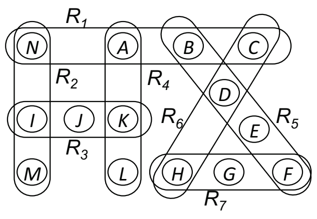

Figure 1: A hypergraph.

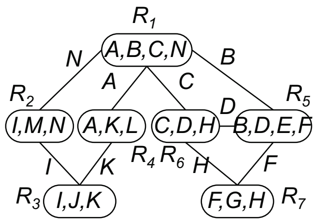

Figure 2: The dual graph.

A tree decomposition of a CSP is a tree embedding of its constraint network. The tree nodes are *clusters* of variables and constraints from the CSP. The set of variables of a cluster cl is denoted $\chi(cl) \subseteq \mathcal{X}$, and the set of constraints $\psi(cl) \subseteq \mathcal{C}$. A tree decomposition must satisfy two conditions: *a*) each constraint appears in at least one cluster and the variables in its scope must appear in this cluster; and *b*) for every variable, the clusters where the variable appears induce a connected subtree.

Many techniques for generating a tree decomposition of a CSP exist (Dechter and Pearl 1989; Jeavons, Cohen, and Gyssens 1994; Gottlob, Leone, and Scarcello 1999). We use an adaption for non-binary CSPs of the tree-clustering technique (Dechter and Pearl 1989). *First*, we triangulate the primal graph of the CSP using the min-fill heuristic(Kjærulff 1990). *Second*, we identify the maximal cliques in the resulting chordal graph using the MAXCLIQUES algorithm (Golumbic 1980), and use the identified maximal cliques to form the clusters of the tree decomposition. Figure 3 shows a triangulated primal graph of the example in Figure 1.

The dotted edges (B,H) and (A,I) in Figure 3 are fill-in edges generated by the triangulation algorithm. The ten maximal cliques of the triangulated graph are highlighted with 'blobs.' *Third*, we build the tree by connecting the clusters using the JOINTREE algorithm (Dechter 2003). While any cluster can be chosen as the root of the tree, we choose the cluster that minimizes the longest chain from the root to a leaf. Figure 4 shows the tree after connecting the max-

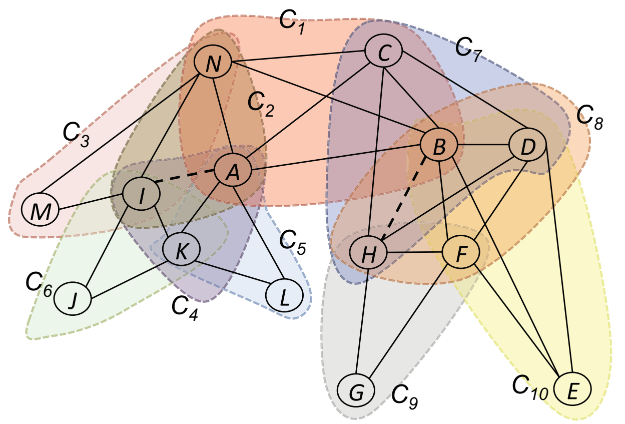

Figure 3: Triangulated primal graph and its maximal cliques.

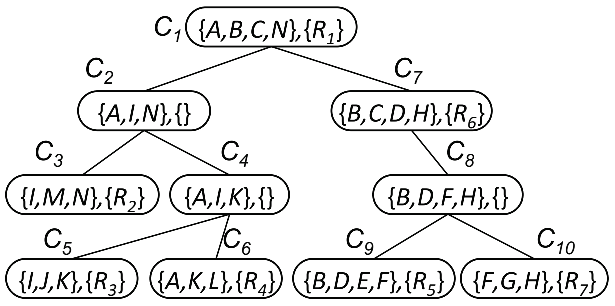

Figure 4: Tree decomposition.

imal cliques of Figure 3. *Finally*, we determine the variables and constraints of each cluster as follows: *a*) The variables of a cluster cl, $\chi(cl)$, are the variables in the maximal clique that yields the cluster; *b*) the clique in the primal graph formed by the vertices of a given constraint R of the CSP is, by construction, a subset of at least one maximal clique in the triangulated primal graph. Thus, *scope*(R) is a subset of the variables of at least one cluster; and *c*) The constraints of a cluster cl, $\psi(cl)$, are the constraints R_i, such that *scope*$(R_i) \subseteq \chi(cl)$. Figure 4 shows a tree decomposition produced by this process for the example of Figure 1. Note that we may end up with clusters with no constraints (e.g., C_2, C_4 and C_8), which are ignored during processing.

A *separator* of two adjacent clusters is the set of variables that are associated with both clusters. A given tree decomposition is characterized by its *treewidth*, which is the maximum number of variables in a cluster minus one.

3 Localizing Consistency to Clusters

We use the relational consistency property R($*$,m)C. This property guarantees that every combination of m relations is minimal, that is, every consistent assignment of variables appearing in the scope of a constraint can be extended to a consistent assignment of the variables in the scope of every $(m-1)$ other constraints. The algorithm for enforcing R($*$,m)C filters the existing relations without adding any new constraint to the problem (Karakashian et al. 2010).

The choice of the value of m was previously not addressed. Obviously, increasing the value of m increases the level of consistency enforced. However, the number of combinations of relations to consider increases exponentially with m: It is $\mathcal{O}(\binom{e}{m}) = \mathcal{O}(e^m)$ where e is the number of constraints in the problem. To address this problem, we advocate exploiting a tree decomposition of the CSP. By localizing R($*$,m)C to the clusters of the tree, we reduce e to the number of constraints in a cluster cl (i.e., $|\psi(cl)|$), and, consequently, decrease the total number of combinations that we have to store and handle for a given problem. In the extreme case, when $m = \psi(cl)$, two goals are achieved: *a*) We have to handle only one combination of constraints per cluster; and *b*) The value of m is directly determined by each cluster and 'adaptively' varies along the tree decomposition.

Moreover, thanks to localization, we control the order in which the clusters are processed. In our implementation, we follow the static ordering of the clusters given by the MAX-CLIQUES algorithm (Golumbic 1980), proceeding from the bottom to the top and back until quiescence.

Because localization prevents us from considering combinations of constraints across clusters, the consistency enforced is, as a result, weakened, as we discuss in Section 5.

4 Bolstering Propagation at Separators

We introduce the simple example of two adjacent clusters shown in Figure 5 to illustrate our approach. The variables in this example are $A, B, \ldots, F$ and the original constraints are $R_1, R_2, \ldots, R_7$. When a consistency algorithm is applied locally to a cluster, the effects of filtering relations in one cluster are propagated or transferred to a neighboring cluster only through the domains of the variables and those constraints common to both clusters (i.e., constraint R_4 in Figure 5). Thus, localization may compromise the effectiveness of constraint propagation across the entire problem. In this section, we explore ways to remedy this situation by adding redundant constraints at the separators to boost the transfer of information between clusters. In Sections 4.1, 4.2, and 4.3, we introduce three schemes to this end, explain how to build them, and discuss their implementation.

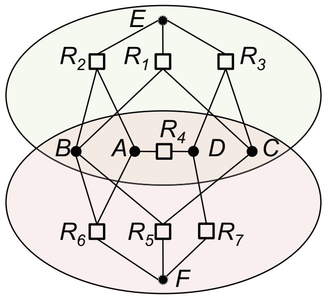

Figure 5: Two adjacent clusters.

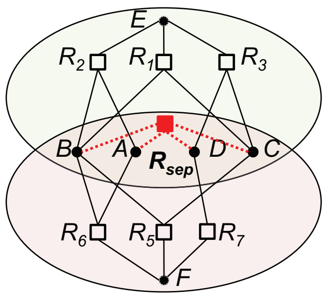

Figure 6: Unique constraint.

4.1 Three bolstering schemes

According to the Cluster-Tree Elimination algorithm, we can solve the CSP in a backtrack-free manner after adding unique constraints over the separators' variables and enforcing R($*$,$\psi(cl)$)C (i.e., minimal clusters) in a two-pass process from the leaves of the tree to its root and back (Dechter 2003; Kask et al. 2005). Thus, we would want to add, to every separator, a unique constraint over all the separator's variables (shown as R_{sep} in Figure 6). Unfortunately, the size of such a relation grows exponentially with the number of variables in the separator, which is prohibitive in practice. Thus, trading space for time becomes necessary (Fattah and Dechter 1996). Instead of generating one unique constraint per separator, we consider three schemes of increasing complexity and 'completeness:'

1. We add to each cluster the projection, on the variables of the cluster, of all the constraints (from outside the cluster), then we normalize the constraints in the cluster. That is, whenever the scope of a constraint is a subset of another constraint, we merge the two constraints (see Section 4.3). In our example, this process results in the new constraint R'_3 added to the lower cluster as shown in Figure 7.

2. In addition to those projected relations, we add all non-existing binary constraints that result from triangulating the subgraph induced by the separator's variables on the primal graph of the CSP. The subgraph induced by the separator on the primal graph of the CSP is shown in Figure 8, before and after triangulation (BD is a fill-in edge resulting from triangulation). In our example, the constraint R_a is added as shown in Figure 9.

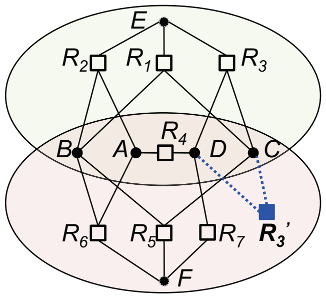

Figure 7: Constraint projections.

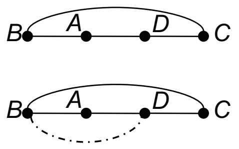

Figure 8: Induced primal graph.

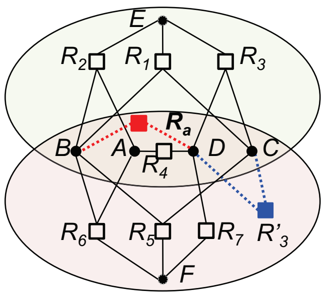

Figure 9: Binary constraints.

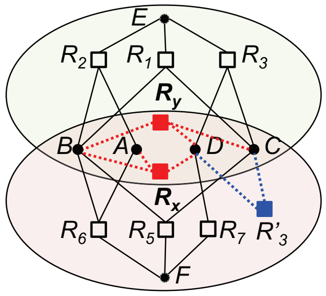

Figure 10: Clique constraints.

3. In addition to those projected relations, we add all non-existing non-binary constraints whose scopes are the maximal cliques of the triangulated subgraph induced by the separator's variables on the primal graph of the CSP. We call those constraints *clique constraints*. The relations R_x and R_y are added in our example in Figure 10.

4.2 Building the separators

Here we describe how we add the constraints to the separators for each of the three schemes introduced above. In all cases, we normalize the cluster.

Adding constraint projections. Every constraint is projected on the variables of every cluster and added to the cluster. Then, all the constraints in the cluster are normalized.

Adding binary constraints. First, we extract the subgraph induced by the separator's variables on the primal graph. Second, we triangulate the induced subgraph using the min-fill heuristic (Kjærulff 1990). Third, we generate a binary constraint for each fill-in edge generated by the min-fill heuristic. For example, the relation R_a is added to the separator in Figure 9, because after triangulating the primal graph induced by the variables in the separator in Figure 8, the fill-in edge BD is added.

Adding clique constraints. As in the case above, we first extract the subgraph induced by the separator's variables on the primal graph. Second, we triangulate the induced subgraph using the min-fill heuristic (Kjærulff 1990). Third, we identify the maximal cliques in the resulting chordal graph using the MAXCLIQUES algorithm (Golumbic 1980). Fourth, for each maximal clique, we generate a constraint over the variables in the maximal clique. For example, in Figure 10, we add to the separator the constraints R_x and R_y whose scopes are $\{A, B, D\}$ and $\{B, C, D\}$, respectively.

4.3 Transferring information between clusters

The information transferred from a cluster to its parent (or its child) transits via the domains of the separator's variables and the added redundant constraints. Because, in the above three bolstering schemes, we normalize the constraints to save on space and processing effort, we need a mechanism to ensure the fullest transfer of information between constraints of overlapping scopes in neighboring clusters. Assume that cluster cl_i is being processed after its neighbor cl_j was. For every relation R_j in cl_j, consider s the set of variables in the scope of R_j that are also in the separator between cl_i and cl_j, $s = \mathit{scope}(R_j) \cap \chi(cl_i) \cap \chi(cl_j)$. s must be a subset of some constraint R_i of cl_i (by construction of the projected constraints). Before processing cl_i, we make sure to filter R_i given R_j in a process akin to directional R($*$,2)C consistency. In the example of Figure 7, R_6, R_5, and R'_3 are used to filter R_2, R_1 and R_3, respectively.

5 Resulting Consistency Properties

We denote by cl-R($*$,m)C the consistency property corresponding the localized version of R($*$,m)C. We denote by cl+proj-R($*$,m)C, cl+bin-R($*$,m)C, and cl+clq-R($*$,m)C those resulting from combining localization and the addition of projected constraints, binary constraints, and clique constraints, respectively. Intuitively speaking, localization weakens R($*$,m)C because localization ignores combinations across clusters. In contrast, adding constraints increases the level of consistency. In Figure 11, we compare the new properties to GAC, maxRPWC and R($*$,m)C for $m = 2, 3, 4, |\psi(cl_i)|$. In this figure, the property at the source of an arrow is strictly weaker than the one at which the arrow points. The proofs are not included here for lack of space, however it is interesting to note that R($*$,2)C, cl+proj-R($*$,2)C, and cl+bin-R($*$,2)C are equivalent and so are R($*$,3)C and cl+proj-R($*$,3)C.

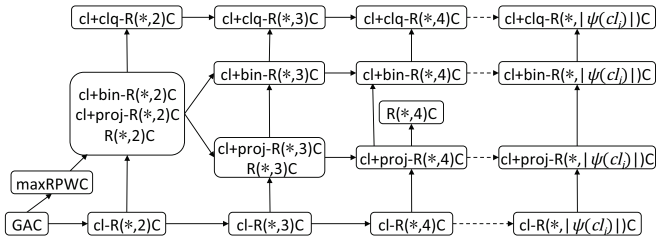

Figure 11: Comparing consistency properties.

When one considers the dual graph of a CSP, some edges in the dual graph may be redundant and could be removed without changing the set of solutions to the problem (Dechter 2003). Karakashian et al. (2010) recommended removing redundant edges using the algorithm of Janssen et al. (2010). This operation reduces the number of constraint combinations that R($*$,m)C must consider, and results in significant cost savings in time and space. The enforced consistency is strictly weaker than R($*$,m)C and denoted wR($*$,m)C. Given the advantageous cost of wR($*$,m)C, in our experiments, we do not implement the properties shown in Figure 11, but, instead, we test their weakened versions (i.e., cl-wR($*$,m)C, cl+proj-wR($*$,m)C, cl+bin-wR($*$,m)C, and cl+clq-wR($*$,m)C), obtained after removal of redundant edges in the dual graph. We do so for $m = 2, 3, 4$, however, for $m = |\psi(cl_i)|$, we do not remove the redundant edges because we consider a single combination per cluster. While the relationships shown in Figure 11 do not necessarily hold for the weakened versions of the consistency properties, in our experiments, they did.

6 Related Work

Our work extends the relational consistency property R($*$,m)C (Karakashian et al. 2010) by applying it in conceptually innovative ways that make it possible to increase the value of m and achieve higher-level consistencies. In general, this paper enforces stronger consistencies than wR($*$,m)C (Karakashian et al. 2010) while considering fewer combinations of constraints thanks to the localization. The empirical results in Section 7 demonstrate that our approach outperforms the original R($*$,m)C.

Our algorithm follows the spirit of the Cluster-Tree Elimination (CTE) algorithm (Kask et al. 2005). In our approach, similarly to the CTE, the scopes of the constraints over the separators' variables are defined before we compute the corresponding relations and do not change. The relations of those constraints change as an effect of message passing (for the CTE) or constraint propagation (for cl-R($*$,m)C). Because we do not add unique constraints at the separators, we cannot guarantee converging in two passes or solving the CSP. Thus, we use our algorithm as a full-lookahead schema in backtrack search. Nonetheless, experiments show that it yielded backtrack-free search on a large number of problems.

Fattah and Dechter (1996) study space-time tradeoffs of tree clustering by increasing the cluster sizes to reduce the separators sizes. In our approach, we reduce the space requirement by replacing the unique constraints at the separators with constraints of smaller scopes.

Based on the bucket elimination method (Dechter 1996; 1999), the mini-bucket elimination (MBE) algorithm generates relations from the mini-buckets (which are partitions of the relations in a cluster) and then projects them on the separators (Rollon and Dechter 2010). We generate only relations on the separators. Moreover, the sizes of the mini-buckets and those of the generated constraints are bounded by a fixed parameter z chosen by trial and error, while in our work, the sizes are automatically determined by the structure of the constraint graph at the separators.

The consistency property w-SC enforces inverse consistency (by domain filtering) of a relaxed CSP obtained by removing constraints in order to guarantee a tree decomposition of bounded width w (Jégou and Terrioux 2010). In our work, we use the decomposition to process the consistency locally and do not relax the CSP.

Algorithms for higher-order consistency by domain filtering that remain weaker than R($*$,m)C have been proposed (Bessiere, Stergiou, and Walsh 2008) and more recently improved upon (Paparrizou and Stergiou 2012). We compare our results to maxRPWC in the experiments.

Samaras and Stergiou (2005) improve the performance of an arc-consistency algorithm for the dual CSP (i.e., R($*$,2)C) by grouping tuples that have the same supports. While we target stronger consistencies, we should exploit their technique to improve the performance of our algorithms.

We exploit the tree decomposition only for enforcing and maintaining the consistency property. Unlike the BTD (Jégou and Terrioux 2003), the backtrack search used in Section 7 for solving a CSP instance does not follow the tree decomposition, but a heuristic for a dynamic variable ordering.

7 Empirical Evaluations

We compare the advantages of enforcing the properties listed in Table 1 to those of enforcing GAC (Bessiere et al. 2005), maxRPWC (Bessiere, Stergiou, and Walsh 2008), and wR($*$,m)C (Karakashian et al. 2010). All consistencies are enforced as full lookahead strategies in a backtrack search using the domain/degree heuristic for dynamic variable ordering. The benchmarks are selected from the CSP Solver Competition.[1] Because we target problems that require higher consistency levels than provided by GAC, we selected 32 benchmarks[2] that are not easily solved by GAC, but compared against GAC as a baseline for evaluation.

Type	$m = 2, 3, 4$	$m = \lvert\psi(cl_i)\rvert$
global	wR($*$,m)C	
local	cl-wR($*$,m)C	cl-R($*$,$\lvert\psi(cl_i)\rvert$)C
projection	cl+proj-wR($*$,m)C	cl+proj-R($*$,$\lvert\psi(cl_i)\rvert$)C
binary	cl+bin-wR($*$,m)C	cl+bin-R($*$,$\lvert\psi(cl_i)\rvert$)C
clique	cl+clq-wR($*$,m)C	cl+clq-R($*$,$\lvert\psi(cl_i)\rvert$)C

Table 1: Tested consistencies.

To evaluate the impact of localizing R($*$,m)C and bolstering propagation we distinguish two contexts: solvable and unsolvable CSPs. Indeed, we expect difficult, unsolvable CSPs to be more challenging for GAC and to require higher-level consistencies. In the selected benchmarks, 479 instances were unsatisfiable and 200 satisfiable. We set the maximum processing time per instance to two hours for two reasons: *a*) we targeted difficult instances; and *b*) we wanted to observe the effect of stronger consistencies (i.e., backtrack-free search, smaller trees) as opposed to measuring the effectiveness of our implementation. Table 7 reports statistics about the tree decompositions of the problem instances used in the experiments, giving the treewidth, the size of the largest separator, the largest number of constraints in a cluster, and the largest arity of a generated clique constraint. In Table 3, we report, for search using each consistency, the number of the tested instances that: *a*) Completed: search solved within the allocated time; *b*) BT-free: search solved in a backtrack-free manner; *c*) Min(#NV): search used the least number of nodes visited; and *d*) Fastest: solved the quickest by the corresponding algorithm (within 256 msec precision). Note that preprocessing time is included in all results. This time includes loading the instances, building the data structures, computing the tree decomposition, generating the separator constraints, and solving the problem. We ran our experiments on a large computer cluster with a heavy and variable load. Although the cluster's hardware is the same, the load varies and affects the precision of the clock time. For this reason, we measured the time in seconds computed from the instruction count instead of the clock time, after normalizing the instruction cost and the CPU speed across all runs (we compared the results in instruction counts and CPU time. Although they were qualitatively similar, the former was more reproducible and precise).

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

[2]aim-(50, 100, 200), composed-(25-10-20, 25-1-2, 25-1-25, 25-1-40, 25-1-80, 75-1-2, 75-1-25, 75-1-40, 75-1-80), dag-rand, dubois, graphColoring-(hosExtConvert, mug, register-mulsol, register-zeroin, sgb-book, sgb-games, sgb-miles, sgb-queen), hanoi, modifiedRenault, QCP-15, rand-(10-20-10, 8-20-5), rlfap(GraphsMod, Scens11, ScensMod), ssa, and tightness0.9

		Domain based		wR(∗,2)C					wR(∗,3)C					wR(∗,4)C					R(∗,$\vert\psi(cl_i)\vert$)C			
	#Instances	GAC	maxRPWC	global	local	projection	binary	clique	global	local	projection	binary	clique	global	local	projection	binary	clique	local	projection	binary	clique
Completed	UNSAT	167	142	170	167	172	169	162	191	232	237	232	218	190	225	230	226	223	285	**286**	282	271
	479	34.9%	29.6%	35.5%	34.9%	35.9%	35.3%	33.8%	39.9%	48.4%	49.5%	48.4%	45.5%	39.7%	47.0%	48.0%	47.2%	46.6%	59.5%	**59.7%**	58.9%	56.6%
	SAT	174	159	**179**	178	176	169	104	147	164	155	149	111	132	151	153	147	112	152	138	124	113
	200	87.0%	79.5%	**89.5%**	89.0%	88.0%	84.5%	52.0%	73.5%	82.0%	77.5%	74.5%	55.5%	66.0%	75.5%	76.5%	73.5%	56.0%	76.0%	69.0%	62.0%	56.5%
BT-free	UNSAT	0	30	70	39	70	70	74	97	104	139	139	132	141	104	142	142	149	187	**223**	**223**	213
	479	0.0%	6.3%	14.6%	8.1%	14.6%	14.6%	15.4%	20.3%	21.7%	29.0%	29.0%	27.6%	29.4%	21.7%	29.6%	29.6%	31.1%	39.0%	**46.6%**	**46.6%**	44.5%
	SAT	44	49	55	37	53	52	38	65	30	65	63	53	68	32	75	67	55	39	**77**	71	58
	200	22.0%	24.5%	27.5%	18.5%	26.5%	26.0%	19.0%	32.5%	15.0%	32.5%	31.5%	26.5%	34.0%	16.0%	37.5%	33.5%	27.5%	19.5%	**38.5%**	35.5%	29.0%
Min(#NV)	UNSAT	17	37	73	43	72	72	77	103	115	147	147	144	150	127	159	159	167	220	**249**	248	239
	479	3.5%	7.7%	15.2%	9.0%	15.0%	15.0%	16.1%	21.5%	24.0%	30.7%	30.7%	30.1%	31.3%	26.5%	33.2%	33.2%	34.9%	45.9%	**52.0%**	51.8%	49.9%
	SAT	47	51	64	37	62	61	39	69	38	76	70	61	78	63	108	94	73	83	**111**	100	79
	200	23.5%	25.5%	32.0%	18.5%	31.0%	30.5%	19.5%	34.5%	19.0%	38.0%	35.0%	30.5%	39.0%	31.5%	54.0%	47.0%	36.5%	41.5%	**55.5%**	50.0%	39.5%
Fastest	UNSAT	72	14	13	35	5	1	1	15	106	58	13	15	12	35	3	0	0	**176**	108	42	37
	479	15.0%	2.9%	2.7%	7.3%	1.0%	0.2%	0.2%	3.1%	22.1%	12.1%	2.7%	3.1%	2.5%	7.3%	0.6%	0.0%	0.0%	**36.7%**	22.5%	8.8%	7.7%
	SAT	**121**	31	45	47	23	14	12	26	30	27	13	11	7	26	14	9	10	34	18	13	12
	200	**60.5%**	15.5%	22.5%	23.5%	11.5%	7.0%	6.0%	13.0%	15.0%	13.5%	6.5%	5.5%	3.5%	13.0%	7.0%	4.5%	5.0%	17.0%	9.0%	6.5%	6.0%

Table 3: Number of instances for which search completed, those solved in a backtrack-free manner, with minimum NV, and fastest.

	max		median		mean	
	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT
treewidth	243	158	33	18	43.45	34.44
largest sep.	214	157	28	16.5	39.02	31.33
max($\vert\psi(cl_i)\vert$)						
local	1,243	211	16	8	109.54	18.35
projection	1,243	211	18	11	114.70	37.35
binary	1,243	653	24	12	199.50	80.65
clique	1,243	148	18	10	113.35	25.87
clique arity	48	26	7	4	7.40	5.97

Table 2: Characteristics of the benchmark data used in Table 3.

7.1 Aggregate results

First, we discuss $m = |\psi(cl_i)|$, which is a localized strategy and also our strongest consistency. It is clearly the overall winner. The highlighted cells in Table 3 indicate that $m = |\psi(cl_i)|$ outperforms all tested consistency levels in both SAT/UNSAT categories and on all four reported criteria with only two exceptions. Both exceptions are on SAT instances, and are related to time performance (on the criteria 'completed' and 'fastest'). This result strongly supports two of our claims: *a*) higher-level consistencies are useful for approaching tractability in practice; and *b*) localization by tree decomposition is a crucial facilitator to increasing the consistency level. (Indeed, very high-level consistencies are not possible without localization because of the number of constraint combinations that need to be stored and manipulated.) Regarding the two exceptions, we strongly suspect that the culprit is our implementation of our algorithm, which can benefit from various optimizations such as grouping tuples (Samaras and Stergiou 2005). We strongly suspect that a faster implementation would overcome this unique limitation of the algorithm. However, the number of nodes visited and the number instances solved backtrack free are solid indicators of the advantages of our approach, and establish the benefits of higher-level consistencies in tree decompositions.

Second, a little bolstering is great, but too much may be detrimental. For a given m value, we see that, grossly speaking, 'projection' yields the best results (versus global, localization, binary, and clique). Two reasons may explain why heavier bolstering (i.e., binary and clique) are not the winners that we expected: *a*) the heavier the bolstering, the more expensive the processing (indeed, the completion rate of clique degrades); and *b*) in most of the tested instances clusters seem to overlap heavily making the generation of redundant constraints an overkill. One may want to decide locally based on the overlap of the clusters which level of bolstering to apply.

The bolstering techniques of increasing 'sophistication' are designed to study the tradeoff between the effectiveness of constraint filtering, on the one hand, and the cost of generating and maintaining redundant constraints, on the other hand. Without bolstering, local properties are weak but fast. Bolstering by projection increases the effectiveness of constraint propagation but also increases the cost. Apparently, projection yields the right tradeoff between the strength of the enforced consistency and the overhead of processing the redundant constraints. The overhead of more aggressive bolstering via binary and clique constraints undermines the benefits of the increased constraint propagation, at least on the problem instances tested.

Finally, localization and projection always outperformed 'global' for all considered criteria, particularly when $m \geq 3$. For $m = 2$, localization and projection are equivalent to the global strategy confirming the theory of Section 5. Consequently, the additional processing is wasted.

7.2 A finer view

Figures 12 to 14 compare, pairwise the running time of the algorithms on individual instances for $m = 3$ with increasing sophistication (i.e., global, localization, projection, then clique). Figure 15 compares the performance of GAC, which is the fastest on SAT instances, to cl-R($*$,$|\psi(cl_i)|$)C, which is the fastest algorithm on UNSAT instances. Note the logarithmic scale. The diagonal line plotted indicates equal performance of both algorithms. The points above the diagonal indicate that the corresponding instances are solved faster by the algorithm on the horizontal axis, and vice versa. The points along the top edge indicate that the corresponding instances timed out for the algorithm on the vertical axes.

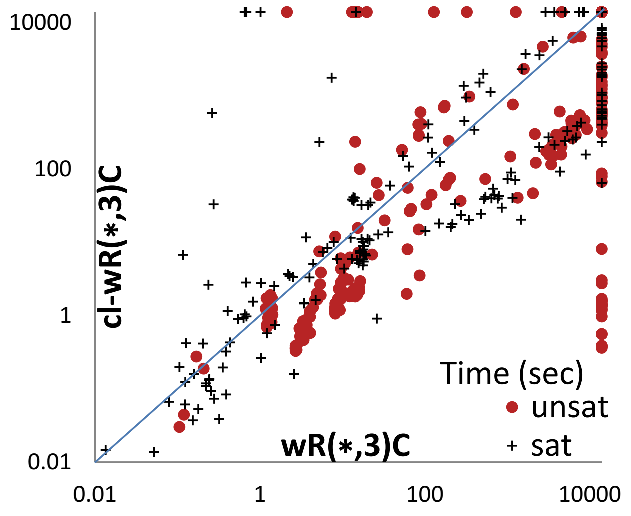

Figure 12: Impact of localization.

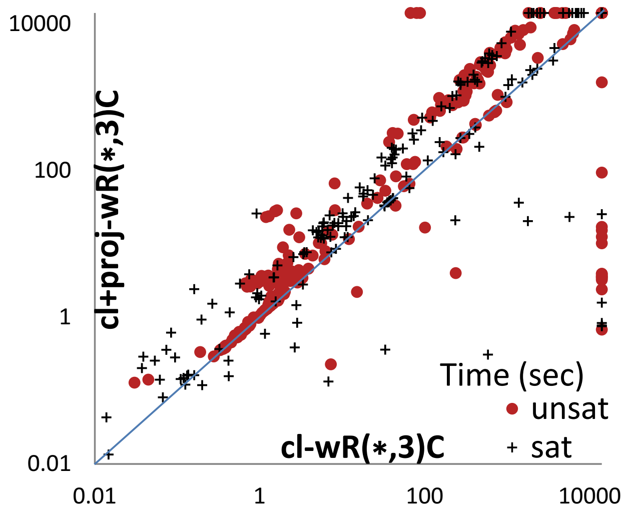

Figure 13: Bolstering by projection.

Figure 12 shows that localization (cl-wR($*$,3)C) outperforms the global version of the algorithm (wR($*$,3)C) by roughly an order of magnitude (points below the diagonal).

Figure 13 evaluates the cost of bolstering by projection (cl+proj-wR($*$,3)C). Most of the points are tightly clustered above the diagonal, reflecting the additional cost of processing the projected constraints. The cost of bolstering is compensated by a group of problem instances that are solved orders of magnitude faster with bolstering than without it (see points on the right edge).

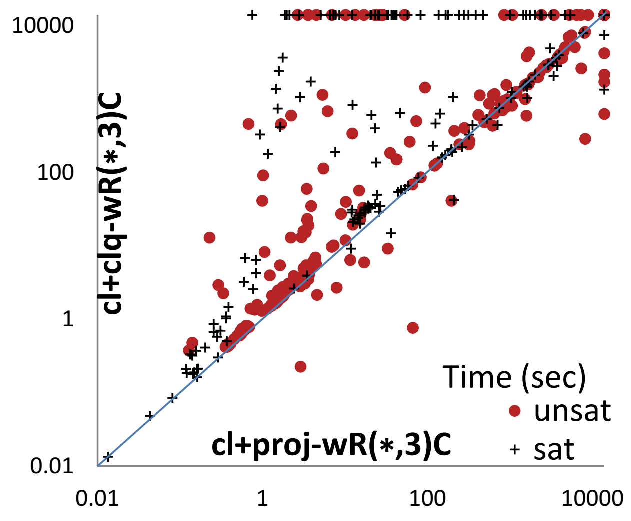

Figure 14: Bolstering by clique constraints.

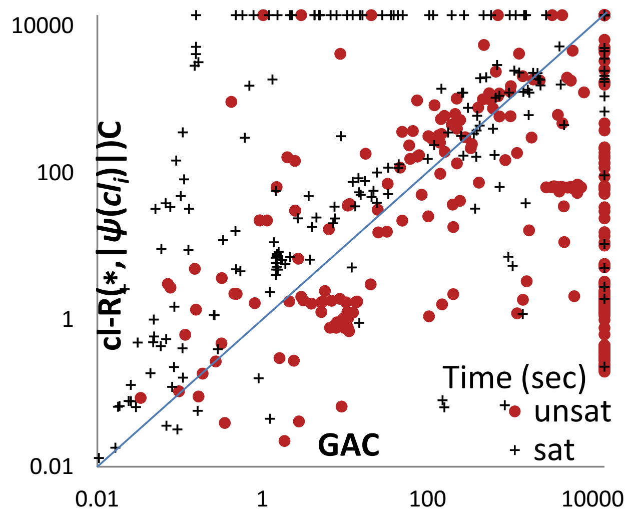

Figure 15: Compared to GAC.

Figure 14 compares projection and clique bolstering (cl+proj-R($*$,3)C versus cl+clq-wR($*$,3)C), and illustrates how bolstering can be an overkill. However, the points below the diagonal suggest that results can be improved when 'clique bolstering' is selectively applied to avoid situations where adjacent clusters significantly overlap.

Again, the best results were obtained with $m = |\psi(cl_i)|$. In Figure 15, we compare cl-R($*$,$|\psi(cl_i)|$)C to GAC. The large number of points clustered around the diagonal confirm that consistencies significantly stronger than GAC are worthwhile even when the running time is considered.

Note that in Figures 12 to 14 the time goes down to 0.01 seconds for a few trivial instances that we did not omit because we reported all the instances of the considered benchmarks.

8 Conclusion and Future Work

In this work we presented techniques to improve the performance of algorithms for higher-level consistency by localizing their application to the clusters of a tree decomposition and bolstering the propagation at the separators with redundant constraints. The empirical results demonstrated orders of magnitude time savings over GAC and R($*$,m)C on difficult CSP problems.

We explored new frontiers in high-level consistency, and showed excellent promise towards achieving 'practical tractability.' It also uncovers both the opportunity and the need to select consistency levels dynamically and locally depending on the structure of each cluster and its difficulty, opening the door for a 'fine grain/cluster level' portfolio-based method for consistency selection (Xu et al. 2008).

Acknowledgments

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

References

Bessiere, C.; Régin, J.-C.; Yap, R. H.; and Zhang, Y. 2005. An Optimal Coarse-Grained Arc Consistency Algorithm. *Artificial Intelligence* 165(2):165–185.

Bessiere, C.; Stergiou, K.; and Walsh, T. 2008. Domain Filtering Consistencies for Non-Binary Constraints. *Artificial Intelligence* 172:800–822.

Dechter, R., and Pearl, J. 1989. Tree Clustering for Constraint Networks. *Artificial Intelligence* 38:353–366.

Dechter, R. 1996. Bucket Elimination: A Unifying Framework for Probabilistic Inference Algorithms. In *Proc. of the Conference on Uncertainty in AI (UAI 96)*, 211–219.

Dechter, R. 1999. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence* 113(1-2):41–85.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Fattah, Y. E., and Dechter, R. 1996. An Evaluation of Structural Parameters for Probabilistic Reasoning: Results on Benchmark Circuits. In *Proc. of the Conference on Uncertainty in AI (UAI 96)*, 244–251.

Freuder, E. C. 1982. A Sufficient Condition for Backtrack-Free Search. *JACM* 29 (1):24–32.

Golumbic, M. C. 1980. *Algorithmic Graph Theory and Perfect Graphs*. New York, NY: Academic Press Inc.

Gottlob, G.; Leone, N.; and Scarcello, F. 1999. A Comparison of Structural CSP Decomposition Methods. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI 99)*, 394–399.

Gyssens, M. 1986. On The Complexity Of Join Dependencies. *ACM Transactions on Database Systems* 11 (1):81–108.

Janssen, P.; Jégou, P.; Nougier, B.; and Vilarem, M.-C. 1989. A Filtering Process for General Constraint-Satisfaction Problems: Achieving Pairwise-Consistency Using an Associated Binary Representation. In *Proceedings of the IEEE Workshop on Tools for Artificial Intelligence*, 420–427.

Jeavons, P. G.; Cohen, D. A.; and Gyssens, M. 1994. A Structural Decomposition for Hypergraphs. *Contemporary Mathematics* 178:161–177.

Jégou, P., and Terrioux, C. 2003. Hybrid Backtracking Bounded by Tree-Decomposition of Constraint Networks. *Artificial Intelligence* 146:43–75.

Jégou, P., and Terrioux, C. 2010. A New Filtering Based on Decomposition of Constraint Sub-Networks. In *Proc. of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 10)*, 263–270.

Karakashian, S.; Woodward, R.; Reeson, C.; Choueiry, B. Y.; and Bessiere, C. 2010. A First Practical Algorithm for High Levels of Relational Consistency. In *Proc. of the Conference on Artificial Intelligence (AAAI 10)*, 101–107.

Kask, K.; Dechter, R.; Larrosa, J.; and Dechter, A. 2005. Unifying Tree Decompositions for Reasoning in Graphical Models. *Artificial Intelligence* 166(1-2):165–193.

Kjærulff, U. 1990. Triagulation of Graphs - Algorithms Giving Small Total State Space. Research Report R-90-09, Aalborg University, Denmark.

Paparrizou, A., and Stergiou, K. 2012. An Efficient Higher-Order Consistency Algorithm for Table Constraints. In *Proc. of the Conference on Artificial Intelligence (AAAI 12)*.

Rollon, E., and Dechter, R. 2010. New Mini-Bucket Partitioning Heuristics for Bounding the Probability of Evidence. In *Proc. of the Conference on Artificial Intelligence (AAAI 10)*, 1199–1204.

Samaras, N., and Stergiou, K. 2005. Binary Encodings of Non-binary Constraint Satisfaction Problems: Algorithms and Experimental Results. *Journal of Artificial Intelligence Research (JAIR)* 24:641–684.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of AI Research* 32:565–606.