Reformulating R$(*, m)$C with Tree Decomposition

Shant Karakashian, Robert J. Woodward and **Berthe Y. Choueiry**
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln, USA
Email: {shantk|rwoodwar|choueiry}@cse.unl.edu

Abstract

Local consistency properties and algorithms for enforcing them are central to the success of Constraint Processing. Recently, we have demonstrated the importance of higher levels of consistency and the effectiveness of their algorithms for solving difficult problems (Karakashian et al. 2010; Woodward et al. 2011). In this paper, we introduce two reformulation techniques for improving the effectiveness of our algorithm for the relational consistency property R$(*, m)$C (Karakashian et al. 2010). Both techniques exploit a tree decomposition of the constraint network of a Constraint Satisfaction Problem (CSP), which is a tree embedding of the network. Our first reformulation technique exploits the structure of the decomposition tree and the state of the backtrack search to omit unnecessary steps from our algorithm and improve its performance. Our second contribution is new relational consistency property called T-R$(*, m, z)C$ that is strictly stronger than R$(*, m)$C. This property is achieved by modifying the structure of the constraint network and adding new redundant constraints to the CSP at the intersection of two vertices of the tree decomposition (Rollon and Dechter 2010). We demonstrate the advantages of the proposed two reformulations for finding all the solutions of a CSP using the technique known as Backtracking with Tree Decomposition (BTD) (Jégou and Terrioux 2003).

1 Introduction

Consistency propagation algorithms are at the heart of solving Constraint Satisfaction Problems (CSPs). Arc consistency has been widely used and improved over the years, and recent studies demonstrated the effectiveness of high levels of consistency properties (Karakashian et al. 2010; Woodward et al. 2011). In this paper, we present the algorithm ProcessMQ as an improvement of the queue management strategy, ProcessQ, for enforcing the relational consistency property R$(*, m)$C (Karakashian et al. 2010). We also propose a new consistency property called T-R$(*, m, z)$C.

The algorithm that we introduced in (Karakashian et al. 2010) enforces R$(*, m)$C by checking the extension of every tuple in every $m-1$ related relations, and deleting the tuple when the check fails. ProcessQ, which manages the relation updates, loops through all the relations until quiescence. ProcessMQ is a reformulation of the algorithm ProcessQ that does not loop when certain condition is met, and omits unnecessary checks during Backtracking with Tree Decomposition (BTD) (Jégou and Terrioux 2003).

A tree decomposition of a CSP is a tree embedding of the constraint network (Dechter 2003). The nodes of the tree are *clusters* of CSP variables and relations. The intersection of two adjacent clusters in the tree is, a *separator*, is the set of variables common to the two clusters. A tree decomposition can have the property where the variables in the intersection of every two clusters are a subset of the scope of a relation in one of the clusters. When this situation holds, we say that the *separator is covered by the relation*. When every separator is covered by a single relation, R$(*, m)$C can be enforced by ordering the relations and checking them in only two passes: starting from the relations in the leaf clusters, via the parents to the root, and back to the leaf clusters. Quiescence is reached without having to loop. For example, the hinge (Cohen, Jeavons, and Gyssens 2008) and hinge+ (Zheng and Choueiry 2005) tree decompositions fulfill this condition. The reformulated algorithm ProcessMQ exploits this situation to reduce the propagation effort without affecting the filtering effectiveness.

The reformulated algorithm also exploits the state of BTD (Jégou and Terrioux 2003) to omit unnecessary propagation steps from R$(*, m)$C. BTD restricts the variable ordering so that the variables in a parent cluster are instantiated before the variables in its children. When this ordering is followed and all the variables in a cluster are instantiated, the subtrees rooted at the children of this cluster can be treated as independent problems. Unlike ProcessQ, ProcessMQ omits checking relations in subtrees other than the subtree rooted at the cluster in which the variable is instantiated, thus localizing the propagation effort.

It is not in general the case that every separator is covered by a relation in a tree decomposition. For example, the tree-clustering technique of (Dechter and

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Pearl 1989), which we use in this paper, does not guarantee the condition. After generating a a tree decomposition, one could add new relations to cover the separators by joining existing relations in the cluster. However, when the number of variables in the separator is large (i.e., a large size separator), adding a single relation to cover the separator may not be practical because of the space overhead. We propose to use the mini-bucket partitioning heuristic of Rollon and Dechter (2010) to generate possibly several redundant relations that *partially* cover the variables in a separator. This operation is reformulation of the original CSP by the addition of redundant constraints that boost propagation without changing the set of solutions of the problem. R$(*,m)$C on the reformulated CSP is strictly stronger than it is on the original problem, resulting in a new consistency property. We denote this property by T-R$(*,m,z)$C, where the parameter z is an input integer parameter to the mini-bucket partitioning algorithm of (Rollon and Dechter 2010), the largest scope of any relation generated by the algorithm.

Our contributions can be summarized as follows:

1. Reformulation of ProcessQ to ProcessMQ to improve the performance of enforcing R$(*,m)$C
2. Introduction of a new relational consistency property T-R$(*,m,z)$C
3. Experimental evaluation on benchmark problems

This paper is organized as follows. Section 2 gives some background information. Section 3 reviews related work. Section 4 describes the property R$(*,m)$C and the algorithm ProcessQ for enforcing it. Section 5 describes the reformulated algorithm ProcessMQ and Section 6 presents the new property T-R$(*,m,z)$C. Section 7 discusses our experimental results and Section 8 concludes this 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 relations. 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 in $\mathcal{C}$ is defined by a relation R specified over the *scope* of the constraint, which are the variables subject to the constraint, as a subset of the Cartesian product of the domains of those variables. A tuple $\tau{\in}R$ 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). A solution to the CSP is an assignment of a value to each variable 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, the vertices represent the variables of the CSP and the hyperedges represent the scopes of the constraints. Figure 1 shows the hypergraph of a CSP where $\mathcal{X}=\{A,B,C,D,E,F,G\}$. In

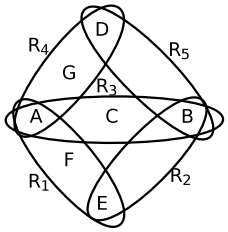

Figure 1: Hypergraph.

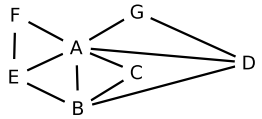

Figure 2: Primal graph.

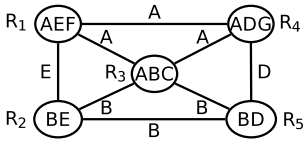

Figure 3: Dual graph.

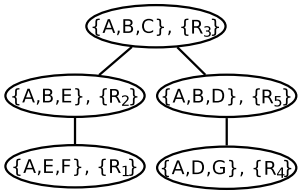

Figure 4: A tree decomposition.

the *primal graph*, the vertices represent the variables and (binary) edges link every two variables that appear in the scope of some constraint, see Figure 2. In the *dual graph*, the vertices represent the relations of the CSP and edges connect two vertices corresponding to relations whose scopes overlap, see Figure 3. The dual CSP is thus a binary CSP where: (1) variables are the relations of the original CSP; (2) the variable domains are the tuples of the corresponding relations; and (3) the constraints enforce *equalities* over the shared variables. We denote by φ a combination of m constraints that induce a connected component in the dual graph, and by Φ the set of all such combinations of size m in the CSP. Finally, π and $\bowtie$ denote the relational operators project and join, respectively.

A tree decomposition of a CSP is defined by a triple $\langle\mathcal{T},\chi,\psi\rangle$, where $\mathcal{T}=(\mathcal{V},\mathcal{E})$ is a tree with a set $\mathcal{V}$ of vertices, also called clusters, and a set $\mathcal{E}$ of edges. χ is a function that associates with each cluster $v\in\mathcal{V}$ a set of variables $\chi(v)\subseteq\mathcal{X}$, and ψ is a function that associates with each cluster v a set of relations $\psi(v)\subseteq\mathcal{C}$. A tree decomposition is a tree embedding of the CSP that satisfies the following two conditions: (1) For each relation R there is at least one cluster $v\in\mathcal{V}$ where $R\in\psi(v)$ and $scope(R)\subseteq\chi(v)$; and (2) For every variable $x\in\mathcal{X}$, the clusters where x appears induce a connected subtree of $\mathcal{T}$. Figure 4 shows a tree decomposition of the CSP of Figure 1. Assuming that $\mathcal{T}$ is a rooted tree, the separator of a cluster $v_i\in\mathcal{V}$ is defined as $sep(v_i)=\chi(v_i)\cap\chi(v_p)$, where v_p is the parent of v_i. A tree is characterized by its treewidth $tw=max_{v\in\mathcal{V}}|\chi(v)|-1$ and its hyperwidth $hw=max_{v\in\mathcal{V}}|\psi(v)|$. Many tree-decomposition techniques exist. In this paper, we use the tree-clustering technique of (Dechter and Pearl 1989). In summary, (1) we triangulate the primal graph of the CSP using the min-fill heuristic (Kjærulff 1990); (2) we identify the maximal cliques in the resulting chordal graph using the MaxCliques algorithm (Golumbic 1980); (3) we

use the identified maximal cliques to form the clusters of the tree decomposition; (4) we connect the clusters to form a tree using the JoinTree algorithm (Dechter 2003). In our work, we choose the root of the tree as the cluster that minimizes the longest chain to a leaf. Further, we place each relation R in the cluster v_i where $scope(R) \subseteq \psi(v_i)$ but $scope(R) \not\subseteq \psi(v_j)$ where v_j is the parent of v_i.

To reduce the severity of the combinatorial explosion, CSPs are usually filtered by enforcing a given local consistency property. One such common property is Generalized Arc Consistency (GAC). A CSP is GAC iff for every relation, any value in the domain of any variable in the scope of the relation can be extended to a tuple satisfying the relation. Using the terminology introduced in (Debruyne and Bessiere 1997), we say that a consistency property p is *stronger* than another p' if in any CSP where p holds, p' also holds. Further, we say that p is *strictly stronger* than p' if p is stronger than the p' and there exists at least one CSP in which p' holds but not p. Similarly to (Bessiere, Stergiou, and Walsh 2008), we say that p and p' are *equivalent* when p is stronger than p' and vice versa. In practice, when a consistency property is stronger (respectively, weaker) than another, enforcing the former never yields less (respectively, more) pruning than enforcing the latter on the same problem.

3 Related Work

Algorithms for enforcing local consistency allow us to effectively reduce the size of the search space and the cost of the search effort while having typically a polynomial cost. Increasing the level of consistency enforced is in general problematic and unpopular because it may require the generation of an exponential number of constraints, prohibitively affecting the space requirements and increasing the computational cost. Those issues worsen in the presence of non-binary constraints.

On the conceptual level, relational(i, m)-consistency is a parameterized relational-consistency property proposed in (Dechter and van Beek 1997), and m-wise consistency was proposed in the area of Relational Databases (Gyssens 1986). Recently, we reintroduced m-wise consistency with an algorithm for enforcing it that demonstrated practical benefits (2010).

Tree clustering proposed by Dechter and Pearl (1989) is a tree decomposition method for binary CSPs similar to the decomposition that we use in this work. Hypertree decomposition is more general than tree clustering, it has algorithms for finding the optimal decomposition proposed by Gottlob and Samer in (2009) and heuristic methods with better runtime and close to optimal results proposed in (Dermaku et al. 2008).

Backtrack search with tree decomposition (BTD) was introduced by (Jégou and Terrioux 2003) and exploits the tree decomposition of CSP during the search process. First, BTD follows the ordering of the variables in the clusters for instantiation while allowing dynamic ordering within a cluster. By doing so, BTD exploits the fact that the search effort is bounded by the treewidth of the tree decomposition. Further, BTD generates structural goods and nogoods at the separators. Those nogoods (respectively, goods) are inconsistent (respectively, consistent) partial assignments that, when recorded, can be used to avoid visiting the subtrees rooted at the corresponding separator when the same partial assignments are encountered again at the separator. BTD has been successfully used for solving CSPs (Jégou and Terrioux 2003) and for counting number of solutions to a CSP (Favier, de Givry, and Jégou 2009).

Mini-Bucket Elimination (MBE) algorithm is a powerful tool used to solve optimization problems (Dechter and Rish 2003; Marinescu and Dechter 2007). MBE partitions a bucket (i.e., set of relations) into mini-buckets, such that the number of variables in each mini-bucket is bounded by an input parameter z before eliminating variables. We use the partitioning algorithm GreedyPartition of (Rollon and Dechter 2010) to generate new relations at the separators of the tree decomposition. The complexity of GreedyPartition is $\mathcal{O}(e^3 d^z)$, where e is the number of relations in the bucket, d is the domain size of the variables, and z is the parameter limiting the mini-bucket size. We treat each cluster of the tree decomposition as a bucket, and use GreedyPartition to generate a relation corresponding to the join of the relations of each mini-bucket. We enhance the filtering power of R$(*, m)$C by generating the new relations, hence yielding a new relational consistency property: T-R$(*, m, z)$C.

4 Background Information on R$(*, m)$C

The consistency property R$(*, m)$C is a strong and parameterized relational consistency property defined over the relations of a problem (Karakashian et al. 2010). This section recalls its definition from (Karakashian et al. 2010) and summarizes the algorithm for enforcing it.

Definition 1 *A set of m relations $\mathcal{R} = \{R_1, \cdots, R_m\}$ with $m \geq 2$ is said to be $R(*, m)C$ iff every tuple in each relation $R_i \in \mathcal{R}$ can be extended to the variables in $\bigcup_{R_j \in \mathcal{R}} scope(R_j) \setminus scope(R_i)$ in an assignment that satisfies all the relations in $\mathcal{R}$ simultaneously. A network is $R(*, m)C$ iff every set of m relations, $m \geq 2$, is $R(*, m)C$.*

Informally, in every given set φ of m relations, every tuple τ in every relation $R \in \varphi$ can be extended to a tuple τ' in each $R' \in \varphi \setminus \{R\}$, such that all those tuples form a consistent solution to the relations in φ. R$(*, m)$C can be enforced by filtering the existing relations, and without introducing any new relations to the CSP. This operation will be repeatedly applied to all combinations of m relations $\{R_1, \cdots, R_m\}$ until quiescence:

$$\forall R_i \in \{R_1, \cdots, R_m\}, R_i = \pi_{scope(R_i)}(\bowtie_{j=1}^{m} R_j) \quad (1)$$

After enforcing R$(*, m)$C on a constraint network, variable domains are filtered by projecting the filtered re-

lations on the domains of the variables. This process (domain filtering) is always applied by default unless explicitly stated. These domain reductions do not break the R$(*, m)$C property. Therefore, if a network is R$(*, m)$C, domain filtering by projecting relations on domains or by GAC cannot enable further relations filtering by R$(*, m)$C (Karakashian et al. 2010).

4.1 An Algorithm for Enforcing R$(*, m)$C

Our algorithm for enforcing R$(*, m)$C is discussed in detail in (Karakashian et al. 2010). We summarize it here. We define $S_{\tau,\varphi}$, the *support* of tuple $\tau \in R$, to be the set of tuples that verify the condition: $\forall R' \in \varphi \setminus \{R\}$, $\exists \tau' \in S_{\tau,\varphi}$, $\tau' \in R'$, and the tuples in $S_{\tau,\varphi} \cup \{\tau\}$ agree on all shared variables. For each tuple $\tau \in R$ without a valid support, the function SearchSupport determines $S_{\tau,\varphi}$ as the first solution of a backtrack search procedure on the subproblem induced by the relations in φ and where R is assigned τ. The support is recorded and stays active as long as all its constituent tuples are active. SearchSupport uses forward checking and dynamic variable ordering (domain/degree). The function ProcessQ loops through all the relations and checks for the valid support of each tuple in the relation. When no valid support is found for a tuple, the tuple is deleted. ProcessQ stops when all 'active' tuples have a valid support.

5 Enforcing R$(*, m)$C with Tree Decomposition

Given a tree decomposition of the CSP and the state of the search, we describe how we improve the execution of R$(*, m)$C. Section 5.1 informally describes how unnecessary checks in ProcessQ can be avoided. Section 5.2 describes ProcessMQ, which implements this improvement by maintaining local propagation queues. Section 5.3 justifies how relations are assigned to clusters in the tree decomposition. Section 5.4 explains how the local queues are updated during search. Finally, Section 5.5 discusses the correctness of ProcessMQ.

5.1 Intuition

Consider the hypergraph shown in Figure 1, its dual graph shown in Figure 3 and a tree decomposition shown in Figure 4. Let Φ be the set of all m combinations of connected relations in the dual graph shown.

In a pre-processing step, we can enforce R$(*, m)$C by starting from the leaves, up to the root, and then back to the leaves and be guaranteed to reach quiescence rather than looping through the relations. This simplification is possible because the condition that each separator be covered is satisfied. Note that our tree decomposition does not guarantee the condition.

After instantiating the variables A, B, and C, which form the root cluster, and propagating the effects of these instantiations on the children clusters, the tuples in R_1 and R_2 that surviving the filtering process cannot be affected by deletions of tuples in R_4 and R_5. The converse also holds. Therefore, even though R_2 and R_5 may appear in some combination in Φ, this combination need not be revised.

5.2 ProcessMQ

We modify ProcessQ into ProcessMQ to exploit the tree decomposition and the state of the search in the tree. ProcessMQ differs from ProcessQ in that it uses a propagation queue for each cluster instead of the global propagation queue for the entire CSP.

Given a CSP $P=(\mathcal{X}, \mathcal{D}, \mathcal{C})$, we first generate Φ the set of all m constraints that are connected in the dual graph. Next, we compute a tree decomposition of the CSP $\langle \mathcal{T}, \chi, \psi \rangle$ with $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ using tree clustering (Dechter and Pearl 1989). We generate all relation-combination pairs (R, φ) for every relation in the CSP and every combination $\varphi \in \Phi$ where $R \in \varphi$. We generate a local propagation queue $\mathcal{Q}_v$ for each cluster v in the tree, which we initialize with the relation-combination pairs (R, φ) for all the relations in the cluster, $R \in \psi(v)$. ProcessMQ orders the local propagation queues in the post-order traversal of the corresponding tree clusters (i.e., from the children to the root). ProcessMQ processes each local queue $\mathcal{Q}_v$ once in that sequence, the last queue processed being the one assigned to the root cluster. Then, ProcessMQ proceeds again in the reverse order, from the root to the leaves. Unless all separators are covered, the above process may need to be repeated until either all queues are empty or inconsistency is detected because all tuples in some relation have been removed. At each queue $\mathcal{Q}_v$, ProcessMQ examines one relation-combination pair from $\mathcal{Q}_v$ at a time. It iterates over the active tuples of R and uses SearchSupport to seek a support for each such tuple $\tau \in R$. When no support is found, ProcessMQ removes τ from R and, for each combination φ such that $R \in \varphi$, for each $R' \in \varphi \setminus \{R\}$, it adds the pair (R', φ') where $R, R' \in \varphi'$ to the queue $\mathcal{Q}_{v'}$ of the cluster v' where R' appears (i.e., $R' \in \psi(v')$).

5.3 Assigning Relations to Clusters

As described in Section 2, a constraint R is added to a cluster v if its scope is a subset of the variable set of the cluster (i.e., $scope(R) \subseteq \chi(v_i)$) but not a subset of the parent cluster (i.e., $scope(R) \not\subseteq \psi(v_j)$, v_j parent of v_i). The reason for this choice is best demonstrated on a CSP that is already R$(*, m)$C but R$(*, m)$C still needs to be verified. Consider the example where a relation R is such that $scope(R) \subseteq \chi(v_1)$, $scope(R) \subseteq \chi(v_2)$, $scope(R) \subseteq \chi(v_3)$. Let v_1 be the parent of v_2 and v_3. If R is added to all the three clusters, then R is checked three times. By putting R only in v_1, we are traversing the tuples in R only once.

5.4 ProcessMQ during Search

ProcessMQ is used during backtracking with tree decomposition (BTD) (Jégou and Terrioux 2003) to enforce full lookahead. BTD restricts the variable ordering so that the variables in a parent cluster are instanti-

ated before the variables in its children. Inside a cluster the variables are ordered according to some heuristic such as minimizing domain over degree heuristic. After a variable A in some cluster v is instantiated, all the relations R where A appears are filtered to remove all the tuples disagreeing with the instantiation of A. Further, for each combination φ such that $R \in \varphi$ and each relation $R' \in \varphi \setminus \{R\}$, the pair (R', φ') is added to the queue $\mathcal{Q}_{v'}$ where $R, R' \in \varphi'$, $R \in \psi(v')$, and v' is in the subtree rooted at v. During lookahead, when ProcessMQ deletes a tuple that has no support, relations R' that do not belong to a cluster in the subtree rooted at v are not added to the queue.

Consider again the tree decomposition in Figure 4. Assume that variables A, B, and C have been instantiated. After instantiating variable E, some tuples in R_1 and R_2 may be deleted, however, only R_1 and R_2, paired with the combination $\{R_1, R_2\}$, are added to the queue. In particular, R_4 will not be added to the queue despite the existence of the combination $\{R_1, R_4\}$. Similarly, when a tuple is deleted from R_1 or R_2 by ProcessMQ, no relations other than R_1 and R_2 are added to the queue.

5.5 Correctness

With respect to ProcessQ, ProcessMQ changes only the ordering in which the relation-combination pairs are checked by storing them in different queues. The correctness of ProcessMQ follows directly from the correctness of ProcessQ proved in (Karakashian et al. 2010). When ProcessMQ is used during search, the relation-combination pairs that are not added to a queue can be safely ignored and need not be revised. These relations and those that lost tuples share only the instantiated variables. Hence, the tuples in the ignored relations will be supported as long as the same variable instantiation is maintained.

6 T-R$(*, m, z)$C

T-R$(*, m, z)$C is a new relational consistency property that is strictly stronger than R$(*, m)$C. It differs from R$(*, m)$C because it creates new relations and changes the topology of the graph. It does not add new relations exhaustively like RmC (Dechter and van Beek 1997), but only at specific locations, namely at the separators of a given tree decomposition. Thus, the property depends on the tree decomposition used and is defined by it. For the sake of simplicity, however, the particular tree decomposition is not included in the notation.

The new relations are added by joining existing relations in a cluster and projecting the result on the variables that are common to two adjacent clusters. For the purpose of generating the new redundant relations, we first compute a tree decomposition of the CSP as described in Section 2, except for assigning the relations to the clusters. In this process, we place a relation R in a cluster v_i if the scope of R is a subset of the set of variables associated with the cluster $scope(R) \subseteq \psi(v_i)$, but not a subset of any cluster v_c $scope(R) \not\subseteq \psi(v_j)$ where v_c is a child of v_i.

After placing the relations, we process the clusters from the leaves up to the root. At each cluster, we apply the procedure GreedyPartition of (Rollon and Dechter 2010) to generate the new relations at the separator of the cluster then place the generated relations in the parent cluster. GreedyPartition is a greedy algorithm that takes a set of relations, a parameter z, a set of variables X_p and returns a set of relations over X_p that are heuristically as tight as possible while not generating any intermediate relation with an arity exceeding z. The algorithm proceeds by joining pairs of heuristically chosen relations as long as the scope of the join does not exceed the fixed size z. Then projects the joined relations on X_p. If z is large enough, all the relations will be joined into a single relation and then projected on X_p. At each cluster v, we pass to the algorithm $\psi(v)$, $\chi(v) \cap \chi(v')$, and z, where v' is the parent of v if v is not the root, and is the root cluster if v is the root. If v has children, then $\psi(v)$ will also include the newly generate relations passed from the children. The parameter z is the parameter passed to the consistency property T-R$(*, m, z)$C.

Let $\mathcal{C}_\delta$ be the set of all generated relations. The reformulated CSP is P_r=$(\mathcal{X}, \mathcal{D}, \mathcal{C}')$, where $\mathcal{C}_r = (\mathcal{C} \cup \mathcal{C}_\delta) \setminus \{R | R \in \mathcal{C}, \exists R' \in \mathcal{C}_\delta, scope(R) \subset scope(R')\}$. We use the dual graph of P_r to generate the set of combinations Φ necessary for enforcing R$(*,m)$C. The tree decomposition of the CSP is substituted by $\langle \mathcal{T}, \chi, \psi_r \rangle$, where ψ_r is defined as before, associates each relation in $\mathcal{C}_r$ to the cluster if the scope of the relation is subset to the set of variables associated with that cluster but not subset to the set of variables associated with the parent cluster.

Theorem 1 *T-R$(*, m)C$ is strictly stronger than R$(*, m)C$.*

Proof: Let $\mathcal{G}$ be the dual graph of the original problem and $\mathcal{G}'$ be the dual graph of the reformulated problem by adding the separator relations. Every node and every edge in $\mathcal{G}$ will also appear in $\mathcal{G}'$. $\mathcal{G}$ will have additional nodes and edges. Now consider the relations that are omitted. Let R_{ig} be such a relation. $\exists R_{ni} \in \mathcal{G}'$, $scope(R_{ig}) \subseteq scope(R_{ni})$. Hence, the ignored relation will in fact be replaced with a relation whose scope is a superset of the scope of the ignored relation, and consequently will have at least all the edges incident on it in $\mathcal{G}'$ that were in $\mathcal{G}$. Thus T-R$(*, m)$C will check all partial assignments that R$(*, m)$C does, and therefore is stronger than R$(*, m)$C.

Moreover, T-R$(*, m, z)$C is strictly stronger than R$(*, m)$C. Below, we provide an example that is R$(*,m)$C but not T-R$(*,m,z)$C. Let $\mathcal{P}$ be the Boolean CSP shown in Figure 5 with five variables A, B, C, D, and E and the four all equals relations: R_1, R_2, R_3, and R_4. The primal graph of $\mathcal{P}$ is shown in Figure 6, and the dual graph is given in Figure 7. The edge added to the primal graph by the triangulation algorithm is shown in dotted line. For a tree decomposition shown

in Figure 9, the reformulated problem will have the additional separator relation R_5 with scope $\{A, E\}$ provided the parameter z used for partitioning is at least 4. The dual graph of the reformulated problem is given in Figure 8. Let $\mathcal{P}_{TR}$ and $\mathcal{P}_R$ be the problems after T-R($*$,m,z)C and R($*$,m)C are enforced on $\mathcal{P}_{TR}$, and $\mathcal{P}_R$, respectively. The partial assignment $\langle(A, 0), (E, 1)\rangle$ is consistent in $\mathcal{P}_R$ because $\mathcal{P}_R$ has no relation between A and E. However, this partial assignment violates the relation $R_5 = \{\langle 0, 0\rangle, \langle 1, 1\rangle\}$ which is added in $\mathcal{P}_{TR}$ by T-R($*$,m,z)C. Thus, T-R($*$,m,z)C is strictly stronger than R($*$,m)C. □

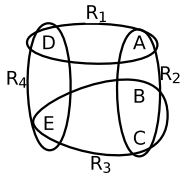

Figure 5: Hypergraph.

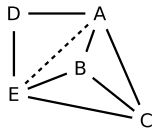

Figure 6: Primal graph.

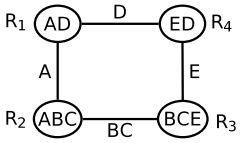

Figure 7: Dual graph.

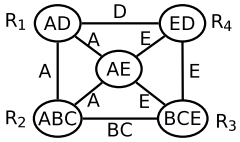

Figure 8: Dual graph.

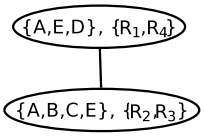

Figure 9: A tree decomposition.

6.1 Weakened Form

In the dual graph, edges enforce the equality of the shared variables of two adjacent vertices. Janssen et al. (1989) and Dechter (2003) observed that an edge between two vertices is redundant if there exists an alternate path between the two vertices such that the shared variables appear in every vertex in the path. Such redundant edges can be removed without modifying the set of solutions. Janssen et al. (1989) introduced an efficient algorithm for computing the minimal dual graph by removing redundant edges. Many minimal graphs may exist, but they are all guaranteed to have the same number of remaining edges. In the example of Figure 3, the edges (R_3, R_4) and (R_2, R_5) are redundant.

Removing redundant edges from the dual graph reduces the size of Φ but also weakens R($*, m$)C, which we denote in(Karakashian et al. 2010) wR($*, m$)C. Similarly, we define the weakened from of T-R($*, m, z$)C and call it wT-R($*, m, z$)C by considering the minimal dual graph for generating the set of combinations Φ. wT-R($*, 2, z$)C is strictly stronger than wR($*, m$)C. For $m > 2$, we expect wT-R($*, m, z$)C to filter more tuples than wR($*, m$)C as is demonstrated in the experiments, but theoretically they are incomparable because the minimal dual graphs may be different and consider different partial assignments.

7 Experimental Results

In this section, we demonstrate the performance of ProcessMQ by comparing it to ProcessQ. We also show the effect of the new property wT-R($*, m, z$)C. We conducted the experiments on benchmarks from the CSP Solver Competition[1]. We limited each run for each instance to one hour, and used BTD maintaining wR($*, m$)C or wT-R($*, m, z$)C to find all the solutions to the problems. We follow the variable ordering imposed by the tree decomposition and use domain/degree variable ordering heuristic within a cluster.

We conducted the experiments using three configurations. In the first configuration, BTD maintains wR($*, m$)C during search enforced using ProcessQ. In the second configuration it also maintains wR($*, m$)C, but using ProcessMQ. In those two configurations, we tried the values 2, 3, and 4 for m, and chose the best result for each instance. In the third configuration, BTD maintains wT-R($*, m, z$)C using ProcessMQ. We tried the values 2, 3 and 4 for m and the values 5, 7 and 9 for z, and choose the best result.

We report the results for the three configurations in Table 1. For each benchmark, we give the number of instances, the average number of variables, and the average treewidth tw of our tree decomposition.

The next columns correspond to the three configurations. For each benchmark the table has three rows. On the first row the number of completed instances is reported in each configuration. On the second and third rows the average and the maximum times in seconds are reported. The average and the maximum are over the best results of each configuration per instance. When no value is reported, it means the configuration was excluded due to few or no instances solved for that benchmark.

Observing the results we notice that the new algorithm ProcessMQ improved the runtime for several of the considered benchmarks. The improvements were in the average and maximum time. Although the improvement of the average time per benchmark is not very significant, the improvement in the maximum time indicates that the benefits of the new algorithm are visible for difficult problems that take more time to solve. With the one hour time limit per run on an instance, more instances were solved using ProcessMQ in benchmarks rand-3-20-20, ukVg and wordsVg, while ProcessQ helped to solve one more instance in benchmark lexVg.

The impact of wT-R($*, m, z$)C was most significant for the aim-100 and aim-200 benchmarks. It helped solve more instances with significant reductions in both average and maximum times. This result indicates that

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

Table 1: Performance on benchmark problems in terms of number of instances completed (**#C**), average time (t_{avg}) in seconds and the maximum time (t_{max}) in seconds.

	Parameters				ProcessQ	ProcessMQ	
Benchmark	**#inst**	**#vars**	tw		**wR(*,b)C**	**wR(*,b)C**	**wT-R(*,b,b)C**
aim-100	24	100	50.83	**#C**	21	21	**24**
				t_{avg}	128.72	127.60	**6.73**
				t_{max}	1,595.39	1,728.64	**95.05**
aim-200	24	200	104.92	**#C**	17	17	**22**
				t_{avg}	246.35	252.48	**238.99**
				t_{max}	3,352.54	3,452.98	**1,540.94**
dubois	13	98.08	3.00	**#C**	13	13	13
				t_{avg}	0.03	0.03	**0**
				t_{max}	0.13	0.14	**0.02**
lexVg	57	129.00	82.00	**#C**	**57**	56	20
				t_{avg}	454.40	**415.25**	
				t_{max}	3,365.20	**3,040.05**	
modifiedRenault	50	110.00	10.00	**#C**	50	50	50
				t_{avg}	**20.39**	21.23	39.82
				t_{max}	**365.62**	378.58	991.83
ogdVg	59	134.00	85.00	**#C**	15	15	15
				t_{avg}	283.27	**242.06**	266.74
				t_{max}	1,834.11	**1,508.27**	1,720.97
pret	8	105.00	4.00	**#C**	**8**	**8**	**8**
				t_{avg}	0.05	0.05	**0.01**
				t_{max}	0.09	0.09	**0.02**
rand-10-20-10	20	20	13.00	**#C**	20	20	20
				t_{avg}	**0.21**	0.32	0.99
				t_{max}	**0.26**	0.38	1.32
rand-3-20-20	50	20	13.00	**#C**	13	**14**	0
				t_{avg}	2,191.56	**1,949.87**	
				t_{max}	3,481.04	**3,145.77**	
renault	2	101.00	9.00	**#C**	**2**	**2**	**2**
				t_{avg}	31.33	32.47	**30.31**
				t_{max}	49.25	51.32	**35.51**
ssa	7	1,024.00	15.00	**#C**	**9**	**9**	**9**
				t_{avg}	**42.75**	47.75	56.74
				t_{max}	**190.75**	222.37	274.2
travellingSalesman-20	15	61	20.00	**#C**	**14**	**14**	0
				t_{avg}	**689.88**	773.55	
				t_{max}	**2,788.37**	3,259.90	
travellingSalesman-25	15	76	25.00	**#C**	**4**	**4**	0
				t_{avg}	**271.24**	272.27	
				t_{max}	393.46	**382.88**	
ukVg	56	134.00	85.00	**#C**	15	**17**	**17**
				t_{avg}	484.98	452.22	**443.55**
				t_{max}	2,800.58	2,533.19	**2,450.08**
wordsVg	50	134.00	85.00	**#C**	44	**46**	43
				t_{avg}	174.92	**163.39**	164.24
				t_{max}	2,673.81	2,558.64	**2,526.50**

the cost of processing the additional relations is outweighed by the gain from the time spent on backtracking. For most of the other benchmarks, higher levels of consistency were not useful, and in most cases the cost of generating new relations was detrimental for the successful completion of the process within the one hour time limit or in many cases the memory was not sufficient. Despite the limited success of wT-R$(*, m, z)$C, we believe that revising the parameters more intelligently, we would be able to extract better results specially on

problems where low levels of consistency are not effective.

8 Conclusions and Future Work

In this work we presented a reformulation of the algorithm presented in (Karakashian et al. 2010) for enforcing the relational consistency property R$(*, m)$C. We also proposed a new relational consistency property T-R$(*, m, z)$C that is strictly stronger than R$(*, m)$C and is achieved by reformulating the CSP. The experimental results demonstrated the benefits of the new algorithm and the consistency property on various benchmark problems. The results showed the importance of choosing the right values for the parameters in T-R$(*, m, z)$C. This problem is beyond the scope of this work.

In the future, we plan to study the choice of the parameters that control the consistency level. This choice can be static by analyzing the problem and can also be decided and updated dynamically during search.

Acknowledgments

Experiments were conducted at the Holland Computing Center facility of the University of Nebraska. This material is based in part upon works supported by the National Science Foundation under Grant No. RI-1117956.

References

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

Cohen, D. A.; Jeavons, P.; and Gyssens, M. 2008. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences* 74(5):721–743.

Debruyne, R., and Bessiere, C. 1997. Some Practical Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. of the 15 th IJCAI*, 418–423.

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

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for bounded inference. *J. ACM* 50(2):107–153.

Dechter, R., and van Beek, P. 1997. Local and Global Relational Consistency. *Theor. Comput. Sci.* 173(1):283–308.

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

Dermaku, A.; Ganzow, T.; Gottlob, G.; Mcmahan, B.; Musliu, N.; and Samer, M. 2008. Heuristic Methods for Hypertree Decomposition. In *Proceedings of the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, MICAI '08, 1–11. Springer-Verlag.

Favier, A.; de Givry, S.; and Jégou, P. 2009. Exploiting Problem Structure for Solution Counting. In *Proceedings of* 15^{th} *International Conference on Principle and Practice of Constraint Programming (CP 09)*, 335–343.

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

Gottlob, G., and Samer, M. 2009. A Backtracking-Based Algorithm for Hypertree Decomposition. *J. Exp. Algorithmics* 13:1:1.1–1:1.19.

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

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

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

Karakashian, S.; Woodward, R.; Reeson, C.; Choueiry, B. Y.; and Bessiere, C. 2010. A First Practical Algorithm for High Levels of Relational Consistency. In 24^{th} *AAAI Conference on Artificial Intelligence (AAAI 10)*, 101–107.

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

Marinescu, R., and Dechter, R. 2007. Best-first and/or search for graphical models. In 22^{nd} *AAAI Conference on Artificial Intelligence (AAAI 07)*, 1171–1176.

Rollon, E., and Dechter, R. 2010. New Mini-Bucket Partitioning Heuristics for Bounding the Probability of Evidence. In 24^{th} *AAAI Conference on Artificial Intelligence (AAAI 10)*.

Woodward, R.; Karakashian, S.; Choueiry, B. Y.; and Bessiere, C. 2011. Solving Difficult CSPs with Relational Neighborhood Inverse Consistency. In 25^{th} *AAAI Conference on Artificial Intelligence (AAAI 11)*, 1–8.

Zheng, Y., and Choueiry, B. Y. 2005. New Structural Decomposition Techniques for Constraint Satisfaction Problems. In et al., B. F., ed., *Recent Advances in Constraints*, volume 3419 of *Lecture Notes in Artificial Intelligence*. Springer. 113–127.