2012

Exploiting Structure in Constraint Propagation

Robert J. Woodward
University of Nebraska-Lincoln, rwoodwar@cse.unl.edu

Shant K. Karakashian
University of Nebraska-Lincoln, shantk@cse.unl.edu

Berthe Y. Choueiry
University of Nebraska-Lincoln, choueiry@cse.unl.edu

Christian Bessiere
University of Montpellier, France, bessiere@lirmm.fr

David B. Marx
University of Nebraska-Lincoln, david.marx@unl.edu

Exploiting Structure in Constraint Propagation

Robert J. Woodward[1] Shant Karakashian[1]
Berthe Y. Choueiry[1] Christian Bessiere[2]
David B. Marx[3]

[1]Constraint Systems Laboratory
University of Nebraska-Lincoln, USA
{rwoodwar|shantk|choueiry}@cse.unl.edu

[2]LIRMM-CNRS
University of Montpellier, France
bessiere@lirmm.fr

[3]Department of Statistics
University of Nebraska-Lincoln, USA
dmarx1@unl.edu

UNL-CSE-2012-0010

November 19, 2012

Abstract

Local consistency properties and algorithms for enforcing them are central to the success of Constraint Processing. In this paper, we explore how to exploit the structure of the problem on the performance of the algorithm for enforcing consistency. We propose various strategies for managing the propagation queue of an algorithm for enforcing consistency, and empirically compare their effectiveness for solving CSPs with backtrack search and full lookahead. We focus our investigations on consistency algorithms that operate on the dual graph of a CSP and demonstrate the importance of exploiting a tree decomposition of the dual graph. Further, we note that exploiting structure is particularly striking on unsatisfiable instances. We conjecture that the approach for queue-management strategies benefits virtually all other propagation algorithms.

Contents

1 Introduction **3**

2 Background **3**
2.1 Graphical representations 4
2.2 Triangulation & tree decomposition 4
2.3 Local consistency properties 5

3 Queue-Management Strategies (QMS) **6**
3.1 Exact strategies 7
3.2 Lazy strategies 8

4 Experimental Results **9**
4.1 Number of completed instances 10
4.2 CPU time results 12
4.3 Summary of results 13
4.4 Non-binary CSPs 13
4.5 Binary CSPs 15

5 Related Work **16**

6 Future Work & Conclusions **17**

1 Introduction

Constraint Satisfaction Problems (CSPs) are in general $\mathcal{NP}$-complete, and backtrack search is the only known sound and complete algorithm for solving them. A common, and useful, technique to prune the search space of a CSP is by enforcing some consistency property. Algorithms for enforcing consistency typically maintain a queue of the variables or constraints that need to be revised. However, the ordering of those elements in the queue is usually random. Information specific to a problem, such as its structure, is typically neglected. One avenue to improve the performance of a propagation algorithm without affecting the consistency level enforced is to 'direct' propagation along the structure of the constraint network. In this paper, we show the benefits of using structure-based approaches for managing the propagation queue of the algorithm for enforcing Relational Neighborhood Inverse Consistency (RNIC) [Woodward *et al.*, 2011]. We propose four structure-based strategies, and, show that the one based on a tree-decomposition [Dechter, 2003] of the dual graph is advantageous. We show that our approach is particularly beneficial on unsatisfiable instances because it allows us to hasten the detection of an inconsistency, and argue that this research direction is useful and deserves more attention.

This paper is structured as follows. Section 2 reviews background information about CSPs. Section 3 introduces the four new queue-management strategies (two exact and two approximative). Section 4 discusses our experimental results, where we conclude that exploiting the structure in the queue has significant benefit. Section 5 reviews the state of the art in queue-management strategies. Finally, Section 6 concludes this paper.

2 Background

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a set of domains, and $\mathcal{C}$ is a set of constraints. Each variable $V_i \in \mathcal{V}$ 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 specified by a relation R_i defined on a subset of the variables, called the scope of the relation. Given a relation R_i, a tuple $\tau_i \in R_i$ is a vector of allowed values for the variables in the scope of R_i. Solving a CSP corresponds to finding an assignment of a value to each variable such that all the constraints are satisfied.

2.1 Graphical representations

A binary CSP is represented by its *constraint graph* where the vertices are the variables of the CSP and the edges represent the constraints. A non-binary CSP is similarly represented by its *hypergraph* where the hyperedges represent the non-binary constraints. Another graphical representation of a non-binary CSP is the *primal graph*, where the vertices are the CSP variables and edges connect every two vertices corresponding to variables in the scope of a relation [Dechter, 2003]. The dual encoding of a CSP, $\mathcal{P}$, is a binary CSP whose variables are the relations of $\mathcal{P}$, their domains are the tuples of those relations, and the constraints enforce *equalities* over the shared variables. The representation of this encoding as a graph is the *dual graph* of the CSP. Figure 1, illustrates the hyper, primal, and dual graphs of a small non-binary CSP where $\mathcal{V} = \{A, \ldots, F\}$ and the relations are $R_1, \ldots, R_6$.

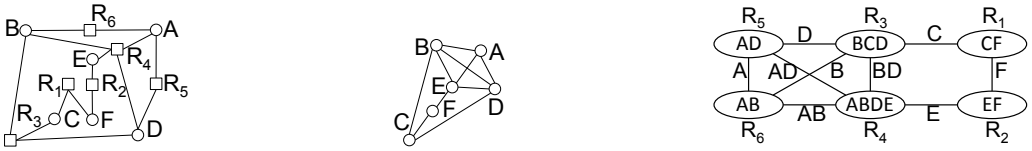

Figure 1: Hyper, primal, and dual graphs of a small CSP.

2.2 Triangulation & tree decomposition

Graph triangulation adds an edge (a chord) between two non-adjacent vertices in every cycle of length four or more [Golumbic, 2004]. While minimizing the number of edges added by the triangulation process is NP-hard, MIN-FILL is an efficient heuristic commonly used for this purpose [Kjærulff, 1990; Dechter, 2003].

Notice, the dual graph of Figure 1 has a cycle of length four (R_1, R_2, R_3, and R_4). One possible triangulation of the dual graph would be to add an edge from R_1 to R_4, as illustrated in Figure 2. This triangulation is not unique, and the edges added depends on the heuristic used.

A perfect elimination ordering on a graph is an ordering of the vertices such that, for each vertex v, v and the neighbors of v that occur after v in the ordering form a clique. If a graph is triangulated, then it is guaranteed to have a perfect elimination ordering [Fulkerson and Gross, 1965].

A tree decomposition of a CSP is an encoding of the constraint network [Dechter, 2003]. The tree decomposition is defined by a triple $\langle T, \chi, \psi \rangle$ of a

Figure 2: Triangulating a dual graph.

CSP $P = (X, D, C)$, where $T = (V, E)$ is a tree, and for each node $v \in V$ in the tree, χ is the variable labeling function, $\chi(v) \subseteq X$, and ψ is the relation labeling function, $\psi(v) \subseteq C$. These labeling functions determine which CSP variables and constraints appear in which nodes of the tree. The tree nodes are thus *clusters* of variables and constraints. A tree decomposition must satisfy two conditions:

1. Each constraints $c \in C$ appears in at least one node $v \in V$ in the tree where all of its variables are in that vertex, $scope(c) \subseteq \chi(v)$.

2. All the vertices where a variable $x \in X$ appears, $\{v \in V | x \in \chi(v)\}$, induces a subtree of T.

2.3 Local consistency properties

CSPs are in general $\mathcal{NP}$-complete and solved by search. To reduce the severity of the combinatorial explosion, they are usually 'filtered' by enforcing a given local consistency property [Bessiere, 2006]. One such property is Neighborhood Inverse Consistency (NIC) introduced in [Freuder and Elfe, 1996] for binary CSPs. NIC ensures that every value in the domain of a variable can be extended to a solution of the subproblem induced by the variable and all the constraints in its neighborhood. Algorithms for enforcing NIC operate by filtering the domains of the variables. These consistency properties are called domain consistency properties, as they are filtering the domains of the variables. In [Woodward *et al.*, 2011], we extended the definition to NIC to Relational Neighborhood Inverse Consistency (RNIC), which ensures that every tuple in every relation can be extended in a consistent assignment to all the relations in its neighborhood. Algorithms for enforcing RNIC operate by filtering the tuples of the relations. Because RNIC is filtering the relations instead of the domains, this type of consistency property is called a relational consistency property. We also introduced three variations of RNIC:

enforcing on the minimal dual graph[1] (wRNIC), a triangulation of the dual graph (triRNIC), or doing both (wtriRNIC). We also proposed a selection strategy (selRNIC) for selecting which variation of RNIC to use based on the density of the dual graph.[2] We showed that in a statistically significant manner selRNIC is able to select the correct variation of RNIC to enforce on a given problem. In [Woodward *et al.*, 2011], we also showed that enforcing higher-level consistency, such as selRNIC, is beneficial to solving difficult problems.

3 Queue-Management Strategies (QMS)

We explore the following three directions for ordering the relations:

1. Random ordering of the relations in the propagation queue, as in [Woodward *et al.*, 2011],

2. Using a *perfect elimination ordering* (PEO) of the vertices of some triangulation of the dual graph, and

3. Using an ordering of the maximal cliques of some triangulation of the dual graph, which corresponds to a *tree-decomposition ordering* (TD).

For the example of Figure 3, a perfect elimination ordering obtained by applying the max-cardinality ordering is: $\langle R_6, R_5, R_3, R_4, R_2, R_1 \rangle$ and the maximal cliques ordering is: $\langle C_1, C_2, C_3 \rangle$, where $C_1 = \{R_3, R_4, R_5, R_6\}$, $C_2 = \{R_1, R_3, R_4\}$, and $C_3 = \{R_1, R_2, R_4\}$.

If the graph is not already triangulated, we first triangulate the dual graph. In this step, the effect of the triangulation does not effect the neighborhoods of the dual variables as the triangulation is used only in the computation of the queue ordering. In order to triangulate the dual graph, we use the MinFill heuristic [Kjærulff, 1990; Dechter, 2003]. We use a perfect elimination ordering (PEO) obtained by applying the max-cardinality ordering (MCO) of [Tarjan and Yannakakis, 1984] to the triangulated dual graph. Using this PEO, we find the sequence of maximal cliques [Gavril, 1972].

[1]The minimal dual graph is a dual graph where no alternate path exists between any two vertices such that the shared variables appear in every vertex in the path [Janssen *et al.*, 1989].

[2]The density of a graph of n vertices is equal to the ratio of the number edges in the graph over the maximum number of edges possible ($\frac{n(n-1)}{2}$).

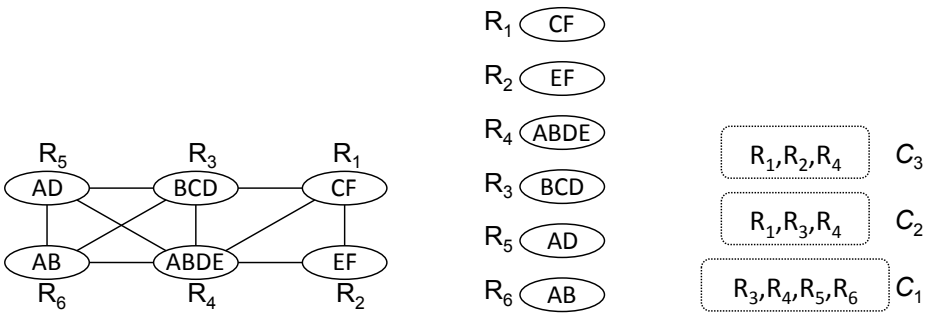

Figure 3: A triangulated dual graph (left) along with a perfect elimination ordering (center) and a maximal cliques ordering (right) where the orderings proceed from bottom to top.

We study the impact of the above orderings in three exact strategies and two approximate strategies (lazy) for managing the propagation queue of the RNIC algorithm in a backtrack search for finding the first solution of a CSP. Below, we describe three exact strategies and two lazy strategies.

3.1 Exact strategies

ProcessQ [Woodward *et al.*, 2011] uses two types of queues: A queue of the relations to be revised ($\mathcal{Q}_R$, which was denoted as $\mathcal{Q}$ in [Woodward *et al.*, 2011]), and for each relation, a queue of tuples for which a support must be found ($\mathcal{Q}_t(R)$). Note that $\mathcal{Q}_R$ is static.[3] However, a relation R is processed only when its $\mathcal{Q}_t(R)$ is not empty. We consider the three exact following strategies:

1. *The random ordering* (QMS_r): The order of the relations in $\mathcal{Q}_R$ is random.

2. *The perfect elimination ordering* (QMS_{PEO}): This ordering aligns the relations in $\mathcal{Q}_R$ following the perfect elimination ordering explained above, and processes them back and forth in that order until quiescence (i.e., until all the tuples in the relations have appropriate supports).

[3]As in AC-1 [Mackworth, 1977] and unlike the queues of most modern consistency algorithms.

3. *The tree decomposition ordering* (QMS_{TD}): This strategy maintains an additional queue, $\mathcal{Q}_C$, that is formed as follows:

   - For each maximal clique, C, a queue of the relations in this clique, $\mathcal{Q}_R(C)$, where the relations are stored in an arbitrary order.
   - Relations are listed in the queues of all the maximal cliques where they appear.
   - The queue $\mathcal{Q}_C$ is a queue of the queues of those maximal-cliques, $\mathcal{Q}_R(C)$, aligned in the tree-decomposition ordering introduced above.
   - A relation R is revised only when its queue of tuples, $\mathcal{Q}_t(R)$, is not empty.

   The cliques are processed back forth in the order they are listed in $\mathcal{Q}_C$ until quiescence. Each time that a clique is considered, its queue is processed in an arbitrary ordering until quiescence before we can move to the next clique in the sequence. During search, $\mathcal{Q}_C$ starts at the shallowest clique that the relations on the instantiated variable participates in.

Note that all three strategies above enforce the same consistency property, RNIC where the neighborhoods are from the original CSP. When the problem is unsolvable, the strategies may differ in the amount of tuples removed before discovering the problem is inconsistent.

3.2 Lazy strategies

The QMS_{TD} strategy, described in Section 3.1 enforces RNIC, it:

1. processes each clique in the order considered,
2. iterates over the relations in the clique in an arbitrary order until quiescence,
3. then moves to the next cliques,
4. while traversing the ordering back and forth until quiescence.

In this section, we investigate if and when weakening the filtering of QMS_{TD} can reduce the solving time of the CSP. We examine weakening this strategy in two different ways, resulting in two 'lazy' strategies:

1. QMS_{LTD}: QMS_{LTD} relaxes Step 4 above, traversing the cliques only once, from bottom to top.

2. QMS_{L^2TD}: QMS_{L^2TD} relaxes Step 2 and 4 above, that is, it traverses the relations in the cliques only once, in a random order, and traverses the cliques also only once, from bottom to top.

The two lazy strategies are strictly weaker than the exact strategies.

4 Experimental Results

We study the impact, on the CPU time, of the queue-management strategies (QMS) of Table 1 when enforcing RNIC-based consistency in a pre-processing step and as full lookahead during backtrack search. We compute the orderings during the pre-processing step, and use the same ordering throughout search. Our search procedure uses dynamic variable ordering (domain/degree) and stops after finding the first solution. We consider the following consistency properties: RNIC, wRNIC, triRNIC, wtriRNIC. We

Table 1: Proposed queue-management strategies.

Exact	QMS_r	The relations are ordered arbitrarily in the propagation queue.
	QMS_{PEO}	The relations are ordered using a perfect elimination ordering. The order is traversed back and forth until quiescence.
	QMS_{TD}	The propagation queue is the sequence of maximal cliques [Gavril, 1972]. Each clique, which is a list of relations, is revised until quiescence; the cliques are revised in sequence, back and forth until quiescence.
Lazy	QMS_{LTD}	Same as QMS_{TD}, however, the sequence of cliques is traversed only once.
	QMS_{L^2TD}	Same as QMS_{TD}, but traversing each clique only once and the relations in a clique only once and in a random order.

also include selRNIC, which is nothing but one of the preceding four properties determined by an automatic selection policy [Woodward *et al.*, 2011]. Thus, *in practice, the performance of only selRNIC matters*. The results of the four properties are reported to facilitate a finer analysis. For each strategy, the reported CPU time is the sum of the time for pre-processing and finding the first solution.

We ran our experiments on benchmark problems from the CSP Solver Competition,[4] testing 1128 CSP instances (613 non-binary and 515 binary)[5]. The tested problems varied in size, ranging from 5 dual variables (constraints in the hypergraph) with 10 dual edges, to 5,714 dual variables with 291,734 dual edges. The time limit per instance was 90 minutes and the memory limit 7 GB. In this paper, we present our results separately for non-binary and binary CSPs, and satisfiable and unsatisfiable ones.

When used at the pre-processing stage, we noticed that the lazy approaches, QMS_{LTD} and QMS_{L^2TD}, ran consistently quicker than the exact strategies because they executed fewer revisions. However, they provided less filtering and yielded larger search spaces. Extensive testing showed that the saving in CPU time during pre-processing that the lazy strategies provided was insignificant compared with the loss of filtering power. Therefore, in the results reported in this paper, the problems for evaluating QMS_{LTD} and QMS_{L^2TD} were pre-processed with QMS_{TD}, and the lazy strategies were applied as lookahead.

4.1 Number of completed instances

Tables 2 and 3 show the number of instances completed within the allotted time for non-binary and binary CSPs, respectively. On non-binary CSPs, clearly selRNIC using QMS_{TD} solves the most number of instances. Thus, it appears that selecting the correct algorithm (which is done by selRNIC) is the most significant decision, and the choice of the queue-management strategy becomes critical when the best algorithm is chosen. On binary CSPs,

[4] All constraints are normalized, `http://www.cril.univ-artois.fr/CPAI09/`

[5] The non-binary benchmarks tested are: aim-50/100/200, dag-rand, dubois, jnhSat/Unsat, lexVg, modifiedRenault, ogdVg, pret, rand-10-20-10, rand-3-20-20(-fcd), ssa, ukVg, varDimacs, and wordsVg. The binary benchmarks tested are: bqwh-15-106, coloring, composed-25-1-2/25/40/80, composed-25-10-2/20/25/40/80, domino, geom, graphColoring-mug/myciel, graphColoring-sgb-book/miles, hanoi, langford/2, QCP-10/15, and QWH-10/15.

Table 2: Non-binary CSPs: Number of completed instances, out of 613 instances tested.

	Satisfiable					Unsatisfiable					$\sum$
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**selRNIC**
QMS_r	93	98	54	79	101	164	167	178	199	209	310
QMS_{PEO}	94	81	81	50	113	176	202	202	194	213	326
QMS_{TD}	90	79	79	51	**114**	181	211	211	205	**221**	**335**
QMS_{LTD}	90	80	80	51	107	181	210	210	205	219	326
QMS_{L^2TD}	88	78	78	51	109	179	211	211	203	220	329

Table 3: Binary CSPs: Number of completed instances, out of 515 instances tested.

	Satisfiable					Unsatisfiable					$\sum$
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**selRNIC**
QMS_r	155	**166**	40	74	154	**105**	63	15	96	**105**	**259**
QMS_{PEO}	156	163	40	82	156	102	63	23	95	102	258
QMS_{TD}	152	155	38	68	153	102	63	51	95	102	255
QMS_{LTD}	112	140	38	69	113	102	59	50	96	102	215
QMS_{L^2TD}	150	140	38	75	151	102	58	50	96	102	253

the result is not as clear. On satisfiable instances, wRNIC using QMS_r solves the largest number of instances, whereas, RNIC and selRNIC using QMS_r show a tie. The table shows that wRNIC is not competitive because of its poor performance on unsatisfiable instances (wRNIC is too weak to detect inconsistency), and the selection policy of selRNIC is immune to this weakness. Again, we conclude that selecting the right correct algorithm is the most significant decision. Focusing on selRNIC, there is not a significant difference between the queue-management strategies (except QMS_{LTD}). Thus, the choice of the queue-management strategy is not as crucial for binary as it is for non-binary CSPs (the one additional instance QMS_r solved was near the time-threshold, and as we will see later, the average CPU time for using

QMS_r was indeed higher).

4.2 CPU time results

We report the CPU time results in Tables 4 to 7. Our goal is to analyze the effect of the new queue-management strategies on a *given* consistency algorithm. The comparison across algorithms is irrelevant for the objectives of this study and was already investigated in [Woodward *et al.*, 2011], where selRNIC was shown to be the winner. Indeed, selRNIC (highlighted in gray in Table 4 to 7) is itself a portfolio of the four other algorithms. Thus, the reader should examine the results in Tables 4, 5, 6, and 7 for *each consistency algorithm separately from all others*, comparing the numbers that are in the same column and *not* the numbers across columns.

- *CPU time:* The results are shown in Table 4 (non-binary) and Table 6 (binary). When search fails to complete within the time limit, the corresponding data point is considered to be *right-censored*. It is then appropriate to run a survival data analysis on the results [Lee, 1992]. It would be statistically unsound to 'penalize' failed runs by some arbitrary factor of the timeout limit. We chose to analyze the data using right-censored data with the Kaplan-Meier estimator to estimate the survival function (the probability that a given algorithm and strategy have finished running after a given time) [Kaplan and Meier, 1958]. The Kaplan-Meier estimator does not make any assumption about the distribution of the data. The mean CPU times are computed by estimating the area under each of survival functions [Allison, 1995].

- *Significance rankings:* The strategies are ranked by statistical significance into equivalence classes of CPU-time performance. The likelihood ratio-test as well as the non-parametric Wilcoxon test were used to test differences between the strategies for a given algorithm [Lawless, 1982; Klein and Moeschberger, 1997]. We used a significance level of 0.05 after using Sidak's method for adjusting the probabilities to account for error in multiple comparisons [Sidak, 1967]. These pair-wise differences are testing if the survival functions for each strategy for a given algorithm are statistically different. The results are reported in Tables 5 and 7, where entries must be compared along the same column. Rank A indicates the best performance, and rank D the worse.

4.3 Summary of results

Below we summarize our findings before discussing them:

1. Exploiting the structure of the problem (here the dual graph) in the management of the propagation queue of a consistency algorithm is beneficial in general and should not be ignored.

2. Lazy strategies are less effective or equivalent than their exact counterparts,[6] thus, they do not constitute an interesting alternative.

3. While improvements are visible on both binary and non-binary CSPs, they are more compelling on the latter than on the former.

4. The benefit of exploiting the structure during constraint propagation is particularly striking on unsatisfiable instances. We believe that the identification of this benefit is an important contribution.

4.4 Non-binary CSPs

Table 4 gives the results of the CPU time for non-binary CSPs. The analysis

Table 4: Average CPU time (ms) for non-binary CSPs.

Satisfiable instances					
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**
QMS_r	3,196,905	**3,005,325**	4,209,710	**2,987,451**	2,993,836
QMS_{PEO}	**2,829,078**	3,534,640	3,534,640	4,236,640	2,944,574
QMS_{TD}	3,216,125	3,443,329	**3,443,329**	4,127,530	**2,887,435**
QMS_{LTD}	3,315,147	3,535,693	3,535,693	4,218,954	3,029,751
QMS_{L^2TD}	3,369,269	3,551,685	3,551,685	4,128,261	3,024,953
Unsatisfiable instances					
QMS_r	2,890,903	2,316,135	2,585,909	**2,102,904**	1,968,668
QMS_{PEO}	**2,409,765**	2,098,659	2,098,659	2,315,073	1,968,376
QMS_{TD}	2,566,678	1,983,243	1,983,243	2,282,629	1,849,360
QMS_{LTD}	2,573,975	**1,970,401**	**1,970,401**	2,290,885	**1,838,448**
QMS_{L^2TD}	2,509,176	1,974,085	1,974,085	2,228,188	1,848,069

was executed for the various queue strategies of each algorithm separately

[6] With one exception for triRNIC for unsatisfiable binary CSPs, see Table 7.

from all other algorithms. Thus, the numbers must be compared only along the same column, and values across the columns are not comparable. The numerical results shown in Table 4 is reported for reference. The discussion focuses on the statistical analysis of the data.

Table 5 shows the impact of the various queue-management strategies on the performance of an individual consistency algorithm (ranking along a column).

Table 5: Significance rankings for non-binary CSPs.

	Satisfiable					Unsatisfiable				
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**
QMS_r	**A**	**A**	C	**A**	**A**	C	B	B	**A**	B
QMS_{PEO}	**A**	B	**A**	B	**A**	**A**	**A**	**A**	B	**A**
QMS_{TD}	B	B	**A**	B	**A**	B	**A**	**A**	**A**	**A**
QMS_{LTD}	B	B	**A**	B	B	B	**A**	**A**	B	**A**
QMS_{L^2TD}	B	B	B	B	B	B	**A**	**A**	**A**	**A**

- For selRNIC on unsatisfiable instances any of the strategies that exploit the structure are significantly better than the arbitrary strategy. For selRNIC on satisfiable instances, the exact strategies, QMS_{PEO} and QMS_{TD}, are equivalent to the random strategy, QMS_r. Therefore, the benefit of exploiting the structure is more advantageous on unsatisfiable instances, but does not degrade its quality on satisfiable instances.

- On satisfiable and unsatisfiable instances, either QMS_{PEO} or QMS_{TD} are statistically better than, or equivalent to, QMS_r, with the exception of wRNIC and wtriRNIC on satisfiable instances.

- The lazy strategies, QMS_{LTD} and QMS_{L^2TD}, do not show significant improvement over QMS_{TD}. (The lazy strategies are either statistically equivalent or worse.) Therefore, there is no significant benefit for using the lazy strategies.

Looking at the number of instances completed in Table 2, selRNIC and QMS_{TD} can solve the largest number of instances. The CPU time for selR-

NIC and QMS_{TD} confirm our claim that selRNIC using QMS_{TD} essentially outperforms all strategies.

4.5 Binary CSPs

The CPU times for binary CSPs are given in Table 6 and the significance classes in Table 7.

Table 6: Average CPU time (ms) for binary CSPs.

Satisfiable instances					
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**
QMS_r	8,726,586	**655,795**	1,940,681	3,607,532	8,916,430
QMS_{PEO}	6,118,663	695,165	**1,902,550**	3,622,475	6,186,426
QMS_{TD}	**2,319,532**	849,787	2,134,283	**3,538,485**	2,299,738
QMS_{LTD}	2,943,891	1,504,313	2,053,600	3,707,704	**2,295,135**
QMS_{L^2TD}	2,321,664	1,463,449	1,936,013	3,558,812	2,302,040
Unsatisfiable instances					
QMS_r	1,438,841	**1,813,683**	2,184,951	1,890,259	1,432,093
QMS_{PEO}	**896,639**	2,412,611	3,282,667	1,728,061	**850,289**
QMS_{TD}	1,585,428	2,144,222	2,886,106	1,823,050	1,579,925
QMS_{LTD}	1,789,383	2,868,932	1,850,859	**1,694,733**	1,679,963
QMS_{L^2TD}	1,491,684	3,300,720	**1,809,207**	2,005,330	1,502,701

Table 7: Significance rankings for binary CSPs.

	Satisfiable					Unsatisfiable				
Strategy	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**	**RNIC**	**wRNIC**	**triRNIC**	**wtriRNIC**	**selRNIC**
QMS_r	C	**A**	B	C	B	B	**A**	B	**A**	B
QMS_{PEO}	C	B	**A**	C	B	**A**	C	B	**A**	**A**
QMS_{TD}	**A**	C	C	**A**	**A**	B	B	B	**A**	B
QMS_{LTD}	**A**	D	C	B	**A**	B	D	**A**	**A**	B
QMS_{L^2TD}	**A**	D	B	B	**A**	B	D	**A**	B	B

- Interestingly, on binary instances, selRNIC performs best using QMS_{PEO} for unsatisfiable instances. However, selRNIC performs best on QMS_{TD} for satisfiable instances.

- The lazy strategies were significantly better than the other strategies only on unsatisfiable instances running triRNIC. The reason for the boost in performance here is that the neighborhood sizes RNIC considers is very large after triangulation on these problems. Therefore, it is becomes very costly to check the triangulated neighborhoods, and the lazy strategies show their benefit.

- Generally speaking, the improvements brought about by queue-management strategies on binary CSPs are not as dramatic as those on non-binary CSPs, however the former do not contradict the latter.

5 Related Work

Wallace and Freuder investigated various ordering heuristics for the propagation queue of arc consistency showing a reduction of the number of constraint checks [1992]. Unlike the strategies studied in this paper, which exploit the structure of the (dual) network, their heuristics considered the properties of individual domains and constraints. Laburhe [2000] and Schulte and Stuckey [2004] ordered propagation queues by prioritizing the constraints based on the time complexity of their processing. Thus, the queue ordering is based on cost and predefined rules, and not the structure of the problem, as we propose in this paper. Schulte and Stuckey in [2008] used the semantics of the constraints/propagators in re-ordering the queue, but did not exploit the structure of the problem. Lagerkvist and Schulte [2009] also studied the propagation order of constraints, but required the user to specify the ordering. Francis and Stuckey [2007] investigated propagation ordering on problems with articulation points, which is less general than tree decomposition.

Freuder linked the width of the constraint network to the consistency level necessary in a relation that guarantees a backtrack-free search [1982]. This approach was extended by Dechter and Pearl to directional path consistency (DPC) [1987] where propagation proceeds along a fixed ordering. Planken et al. [2008] used DPC on a perfect elimination ordering of some triangulation of the constraint graph of a binary CSP in order to propagate Partial Path

Consistency (PPC) [Bliek and Sam-Haroud, 1999], which requires adding constraints, thus modifying the constraint graph. Their work is restricted to the Simple Temporal problem [Dechter *et al.*, 1991]. The work presented in this paper exploits the information gained from triangulation/tree decomposition but does not alter the topology of the constraint network or add any constraints to the problem.

6 Future Work & Conclusions

Exploiting the structure of the problem in the management of the propagation queue of a consistency algorithm is beneficial in general and should not be ignored. Importantly, exploiting the structure during constraint propagation is particularly striking on unsatisfiable instances. This result is important as it indicates that inconsistency can be detected locally in the problem, instead of arbitrarily looking for the inconsistency.

Our approach of exploiting the structure for the propagation queues opens the door to the investigation of other methods for detecting and exploiting structure. Of the queue-management strategies we studied in this paper, we experimentally determined the best strategy is the exact strategy using the tree decomposition of the dual graph for selRNIC on both binary and non-binary instances. The lazy strategies showed no significant improvement during search. Future work is to evaluate if the exact tree-decomposition strategy remains a significant improvement when applied to other consistency algorithms, such as R($*$,m)C of [Karakashian *et al.*, 2010]. The queue-management strategies studied in this paper were all static queues, future work is to study the effect of dynamic queue-management strategies. We believe that the use of queue-management strategies that exploit the structure of the problem will benefit virtually all propagation algorithms.

Consistency properties and their algorithms are central to CP, and perhaps best distinguish this discipline from other fields that study the same problems. Research has focused on defining new properties, proposing new algorithms, improving the performance of known ones, and theoretically characterizing the relationship between the consistency level and the tractability of the CSP. This work adds to the large body of literature on consistency properties and improving their propagation algorithms, but investigating a new dimension of the propagation algorithm: the queue. The use of more sophisticated queue-management strategies shows that we are one step closer

to dynamically recognizing the problems we have at hand, and exploit them to our advantage.

Acknowledgments

We are grateful to Elizabeth Claassen and David B. Marx of the Department of Statistics at the University of Nebraska-Lincoln (UNL) for their help with designing the statistical analysis. Experiments were conducted on the equipment of the Holland Computing Center at UNL. Robert Woodward was partially supported by a National Science Foundation (NSF) Graduate Research Fellowship grant number 1041000. This research is supported by NSF Grant No. RI-111795.

References

[Allison, 1995] Paul Allison. *Survival Analysis Using the SAS System: A Practical Guide*. SAS Publishing, 1995.

[Bessiere, 2006] Christian Bessiere. *Handbook of Constraint Programming*, chapter Constraint Propagation. Elsevier, 2006.

[Bliek and Sam-Haroud, 1999] Christian Bliek and Djamilla Sam-Haroud. Path Consistency for Triangulated Constraint Graphs. In *Proceedings of the 16 th International Joint Conference on Artificial Intelligence*, pages 456–461, Stockholm, Sweden, 1999.

[Dechter and Pearl, 1987] Rina Dechter and Judea Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34:1–38, 1987.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Francis and Stuckey, 2007] Kathryn Francis and Peter J. Stuckey. Constraint propagation for loose constraint graphs. In *22nd ACM Symposium on Applied Computing (ACM SAC 07)*, pages 334–335, 2007.

[Freuder and Elfe, 1996] Eugene C. Freuder and Charles D. Elfe. Neighborhood Inverse Consistency Preprocessing. In *Proceedings of AAAI-96*, pages 202–208, Portland, Oregon, 1996.

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

[Fulkerson and Gross, 1965] Delbert R. Fulkerson and O. A. Gross. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics*, 15 (3):835–855, 1965.

[Gavril, 1972] Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1 (2):180–187, 1972.

[Golumbic, 2004] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 2004. Annals of Discrete Mathematics, Vol 75.

[Janssen *et al.*, 1989] Philippe Janssen, Philippe Jégou, B. Nougier, and Marie-Catherine Vilarem. A Filtering Process for General Constraint-Satisfaction Problems: Achieving Pairwise-Consistency Using an Associated Binary Representation. In *IEEE Workshop on Tools for AI*, pages 420–427, 1989.

[Kaplan and Meier, 1958] Edward L. Kaplan and Paul Meier. Nonparametric Estimation from Incomplete Observations. *Journal of the American Statistical Association*, 53:457–481, 1958.

[Karakashian *et al.*, 2010] Shant Karakashian, Robert Woodward, Christopher Reeson, Berthe Y. Choueiry, and Christian Bessiere. A First Practical Algorithm for High Levels of Relational Consistency. In *24th AAAI Conference on Artificial Intelligence (AAAI 10)*, pages 101–107, 2010.

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

[Klein and Moeschberger, 1997] John P. Klein and Melvin L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer, 1997.

[Laburhe, 2000] François Laburhe. CHOCO: implementing a CP kernel. In *Proceedings of TRICS 2000*, pages 71–85, 2000.

[Lagerkvist and Schulte, 2009] Mikael Z. Lagerkvist and Christian Schulte. Propagator Groups. In 15th *International Conference on Principle and Practice of Constraint Programming (CP'09)*, volume 5732 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2009.

[Lawless, 1982] Jerald F. Lawless. *Statistical Models and Methods for Lifetime Data*. John Wiley & Sons, New York, NY, 1982.

[Lee, 1992] Elisa T. Lee. *Statistical Methods for Survival Data Analysis*. John Wiley & Sons, New York, NY, second edition, 1992.

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[Planken *et al.*, 2008] Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. P3C: A New Algorithm for the Simple Temporal Problem. In *Processings of the 18th International Conference on Automated Planning & Scheduling (ICAPS 2008)*, pages 256–263, 2008.

[Schulte and Stuckey, 2004] Christian Schulte and Peter J. Stuckey. Speeding up constraint propagation. In *Proceedings of 10th International Conference on Principle and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*, pages 619–633, Toronto, Canada, 2004.

[Schulte and Stuckey, 2008] Christian Schulte and Peter J. Stuckey. Efficient constraint propagation engines. *Transactions on Programming Languages and Systems*, 31(1), 2008.

[Sidak, 1967] Zbynek Sidak. Rectangular Confidence Regions for the Means of Multivariate Normal Distributions. *Journal of the American Statistical Association*, 62:626–633, 1967.

[Tarjan and Yannakakis, 1984] Robert Endre Tarjan and Mihalis Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[Wallace and Freuder, 1992] Richard J. Wallace and Eugene C. Freuder. Ordering Heuristics for Arc Consistency Algorithms. In *9th Canadian Conference on Artificial Intelligence*, pages 163–169, 1992.

[Woodward *et al.*, 2011] Robert Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Solving Difficult CSPs with Relational Neighborhood Inverse Consistency. In *25th AAAI Conference on Artificial Intelligence (AAAI 11)*, pages 112–119, 2011.