The Difference All-Difference Makes *

Kostas Stergiou and **Toby Walsh**
Department of Computer Science
University of Strathclyde
Glasgow, Scotland
{ks,tw}@cs.strath.ac.uk

Abstract

We perform a comprehensive theoretical and experimental analysis of the use of all-different constraints. We prove that generalized arc-consistency on such constraints lies between neighborhood inverse consistency and, under a simple restriction, path inverse consistency on the binary representation of the problem. By generalizing the arguments of Kondrak and van Beek, we prove that a search algorithm that maintains generalized arc-consistency on all-different constraints dominates a search algorithm that maintains arc-consistency on the binary representation. Our experiments show the practical value of achieving these high levels of consistency. For example, we can solve almost all benchmark quasigroup completion problems up to order 25 with just a few branches of search. These results demonstrate the benefits of using non-binary constraints like all-different to identify structure in problems.

1 Introduction

Many real-world problems involve all-different constraints. For example, every fixture for a sports team must be on a different date. Many of the constraint satisfaction toolkits therefore provide specialized algorithms for efficiently representing and, in some case, reasoning about all-different constraints. Alternatively, we can expand all-different constraints into a quadratic number of binary not-equals constraints. However, it is less efficient to do this, and the translation loses some semantic information. The aim of this paper is to show the benefits of keeping with a non-binary representation. We prove that we can achieve much higher levels of consistency in the non-binary representation compared to the binary. We show experimentally that these high levels of consistency can reduce search dramatically.

*The authors are members of the APES research group, http://www.cs.strath.ac.uk/ apes. We thank our colleagues in the group at the Universities of Strathclyde and Leeds, most especially Paul Shaw. The second author is supported by EPSRC award GR/K/65706.

2 Formal background

A constraint satisfaction problem (CSP) is a triple (X, D, C). X is a set of variables. For each $x_i \in X$, D_i is the domain of the variable. Each k-ary constraint $c \in C$ is defined over a set of variables $(x_1, \ldots x_k)$ by the subset of the cartesian product $D_1 \times \ldots D_k$ which are consistent values. An all-different constraint over $(x_1, \ldots x_k)$ disallows the values $D_1 \times \ldots D_k - \{(a_1, \ldots a_k) \mid a_i \in D_i \ \& \forall u \neq v . a_u \neq a_v\}$. A solution is an assignment of values to variables that is consistent with all constraints.

Many lesser levels of consistency have been defined for binary constraint satisfaction problems (see [Debruyne and Bessiere, 1997] for references). A problem is (i, j)-consistent iff it has non-empty domains and any consistent instantiation of i variables can be extended to a consistent instantiation involving j additional variables. A problem is arc-consistent (AC) iff it is $(1, 1)$-consistent. A problem is path-consistent (PC) iff it is $(2, 1)$-consistent. A problem is strong path-consistent iff it is $(j, 1)$-consistent for $j \leq 2$. A problem is path inverse consistent (PIC) iff it is $(1, 2)$-consistent. A problem is neighborhood inverse consistent (NIC) iff any value for a variable can be extended to a consistent instantiation for its immediate neighborhood. A problem is restricted path-consistent (RPC) iff it is arc-consistent and if a variable assigned to a value is consistent with just a single value for an adjoining variable then for any other variable there exists a value compatible with these instantiations. A problem is singleton arc-consistent (SAC) iff it has non-empty domains and for any instantiation of a variable, the problem can be made arc-consistent.

Many of these definitions can be extended to non-binary constraints. For example, a (non-binary) CSP is generalized arc-consistent (GAC) iff for any variable in a constraint and value that it is assigned, there exist compatible values for all the other variables in the constraint [Mohr and Masini, 1988]. Regin gives an efficient algorithm for enforcing generalized arc-consistency on a set of all-different constraints [Régin, 1994]. We can also maintain a level of consistency at every node in a search tree. For example, the MAC algorithm for binary CSPs maintains arc-consistency at each node in the search tree [Gaschnig, 1979]. As a second example, on a non-binary problem, we can maintain generalized arc-consistency (MGAC) at every node in the search tree.

Following [Debruyne and Bessiere, 1997], we call a con-

sistency property A stronger than B ($A \geq B$) iff in any problem in which A holds then B holds, and strictly stronger ($A > B$) iff it is stronger and there is at least one problem in which B holds but A does not. We call a local consistency property A incomparable with B ($A \sim B$) iff A is not stronger than B nor vice versa. Finally, we call a local consistency property A equivalent to B iff A implies B and vice versa. The following identities summarize results from [Debruyne and Bessiere, 1997] and elsewhere: strong PC > SAC > PIC > RPC > AC, NIC > PIC, NIC ~ SAC, and NIC ~ strong PC.

3 Generalized arc-consistency

All-different constraints are network decomposable [Dechter, 1990] (abbreviated to decomposable in this paper) as they can be represented by binary constraints on the same set of variables. In this section, we give some theoretical results which identify the level of consistency achieved by GAC on decomposable constraints like the all-different constraint.

In general, GAC on decomposable constraints may only achieve the same level of consistency as AC on the binary representation. The problem is that decomposable constraints can often be decomposed into smaller constraints. For example, we can decompose an n-ary all-different constraint into $n(n-1)/2$ binary all-different constraints, and enforcing GAC on these only achieves the same level of consistency as AC on the binary representation. We can achieve higher levels of consistency if we prohibit too much decomposition of the non-binary constraints. For example, we can insist that the constraints are *triangle preserving*. That is, we insist that, if there is a triangle of variables in the constraint graph of the binary representation, then these variables must occur together in a non-binary constraint. Binary constraints can still occur in a triangle preserving set of constraints, but only if they do not form part of a larger triangle. Under such a restriction, GAC is strictly stronger than PIC, which itself is strictly stronger than AC.

Theorem 1 *On a triangle preserving set of decomposable constraints GAC is strictly stronger than PIC on the binary representation.*

Proof: Consider a triple of variables, x_i, x_j, x_k and any value for x_i from its generalized arc-consistent domain. The proof divides into four case. In the first, x_i and x_j appear in one constraint, and x_i and x_k in another. As each of these constraints is arc-consistent, we can find a value for x_j consistent with x_i, and for x_k consistent with x_i. As the (non-binary) constraints are triangle preserving, there is no direct constraint between x_j and x_k so the values for x_j and x_k are consistent with each other. Hence, the binary representation of the problem is PIC. The other three cases follow a similar argument. To show that GAC is strictly stronger, consider an all-different constraint on 4 variables each with domains of size 3. This problem is PIC but not GAC. □

A corollary of this result is that GAC on a triangle preserving set of decomposable constraints is strictly stronger than RPC or AC on the binary representation. We can also put an upper bound on the level of consistency that GAC achieves.

Theorem 2 *NIC on the binary representation is strictly stronger than GAC on a set of decomposable constraints.*

Proof: Consider any variable and value assignment. NIC ensures that we can assign consistent values to the variable's neighbors. However, any (non-binary) constraint including this variable has all its variables in the neighborhood. Hence, the (non-binary) constraint is GAC. To prove strictness, consider a problem with five all-different constraints on $\{x_1, x_2, x_3\}$, on $\{x_1, x_3, x_4\}$, on $\{x_1, x_4, x_5\}$, on $\{x_1, x_5, x_6\}$, and on $\{x_1, x_6, x_2\}$. in which x_1 has the unitary domain $\{1\}$ and every other variable has the domain $\{2, 3\}$. This problem is GAC, but enforcing NIC. shows that it is insoluble. □

Finally, GAC on decomposable constraints, is incomparable to strong PC and SAC, even when restricted to triangle preserving sets of constraints.

Theorem 3 *On a triangle preserving set of decomposable constraints, GAC is incomparable to strong PC and to SAC.*

Proof: Consider an all-different constraint on 4 variables, each with the same domain of size 3. The binary representation of the problem is strong PC and SAC, but enforcing GAC shows that it is insoluble.

Consider the problem in the proof of Theorem 2 with five all-different constraints. This problem is GAC, but enforcing strong PC or SAC shows that it is insoluble.

These results are summarized in Figure 1.

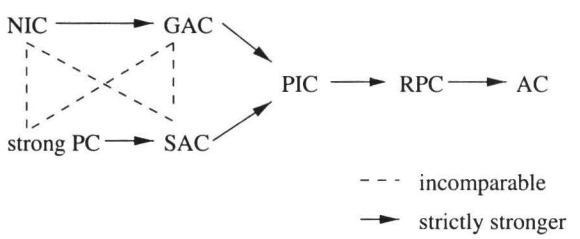

Figure 1: The consistency of GAC on a triangle preserving set of decomposable constraints.

4 Quasigroup problems

Quasigroup problems lend themselves to a non-binary representation using all-different constraints. A quasigroup is a Latin square, a n by n multiplication table in which each entry appears once in every row and column. Quasigroups model a variety of practical problems like tournament scheduling and designing drug tests. Quasigroup completion, the problem of completing a partial filled quasigroup, has been proposed as a constraint satisfaction benchmark [Gomes and Selman, 1997].

An order n quasigroup completion problem can be represented as a non-binary constraint satisfaction problem with n^2 variables, each with a domain of size n. The constraints are $2n$ all-different constraints of size n, one for each row and column, and any number of unitary constraints or preassignments. The special structure of these constraints allows us to prove some tighter results.

Theorem 4 *In quasigroup completion problems, GAC is equivalent to NIC.*

Proof: We need to show that GAC implies NIC. The neighborhood of any variable in an order n quasigroup completion problem are the $2n - 1$ variables that appear in the 2 all-different constraints that contain the variable. As these constraints are GAC, we can find consistent instantiations for each of the variables. In the binary representation, none of these variables have a direct constraint with each other. Hence, this is a consistent instantiation for the neighborhood. □

GAC on quasigroup problems remains strictly stronger than PIC and incomparable to strong PC and to SAC.

Theorem 5 *In quasigroup completion problems, GAC is strictly stronger than PIC.*

Proof: By theorem 1, GAC is stronger than PIC. To show that it is strictly stronger, consider an order 4 quasigroup, in which 3 diagonal elements have domains $\{1\}$, and all the other elements (including the other diagonal element) have domains $\{2, 3, 4\}$. This problem is PIC but is not GAC. □

Theorem 6 *In quasigroup completion problems, GAC is incomparable to strong PC and to SAC.*

Proof: Consider the problem from the last proof. This problem is strong PC and SAC but is not GAC.

Consider an order 3 quasigroup. Let every element have a domain $\{1, 2, 3\}$ except the top right which has the domain $\{1, 2\}$, the bottom left which has the domain $\{1, 3\}$ and the bottom right which has the domain $\{2, 3\}$. This problem is GAC but enforcing strong PC or SAC shows that it is insoluble. □

What can we learn from these results? First, on quasigroup completion problems, we achieve the maximum level of consistency (viz. NIC) possible for a GAC procedure on decomposable constraints. And second, we achieve this at very moderate cost. Regin's algorithm for achieving GAC on a set of all-different constraints has a cost that is polynomial in n. By comparison, enforcing NIC on binary constraints is exponential in the size of the neighborhood (which is $O(n)$ in this case).

5 Maintaining GAC and AC

We now compare an algorithm that maintains GAC on decomposable constraints over one that maintains AC on the binary representation. We say that algorithm A dominates algorithm B if when A visits a node then B also visits the equivalent node in its search tree, and strictly dominates if it dominates and there is one problem on which it visits strictly fewer nodes. Using the previous results, we can reduce our analysis to comparing algorithms that maintain NIC, PIC and AC. In fact, we do better than this and prove some general results about algorithms that maintain any level of consistency stronger than FC in which we just filter domains. This covers algorithms that maintain NIC, PIC and AC, as well those that maintain RPC and SAC. We shall use A-consistent and B-consistent to denote any two such levels of consistency.

We assume throughout a static variable and value ordering. We can then associate each node in the search tree with the sequence of value assignments made. We say that a node $(a_1, \ldots, a_i)$ is A-compatible with another node $(a_1, \ldots, a_j)$ where $j < i$, if enforcing A-consistency at $(a_1, \ldots, a_j)$ does not remove a_i from the domain of the respective variable. First, we give a necessary and sufficient condition for a node to be visited.

Theorem 7 *A node is visited by an algorithm that maintains A-consistency iff it is consistent, it is A-compatible with all its ancestors, and its parent can be made A-consistent.*

Proof: ($\Rightarrow$) The proofs of the first and third conjuncts are similar to those in [Kondrak and van Beek, 1997]. For the second, suppose that node $(a_1, \ldots, a_i)$ is not A-compatible with one of its ancestors and it is visited. Let $(a_1, \ldots, a_j)$, with $j < i$, be the shallowest of those ancestors. Since $(a_1, \ldots, a_j)$ is an ancestor of $(a_1, \ldots, a_i)$, it is also visited. When we visit node $(a_1, \ldots, a_j)$ and A-consistency is enforced, a_i is pruned out from the domain of x_i. Node $(a_1, \ldots, a_{i-1})$ cannot therefore be extended to $(a_1, \ldots, a_i)$. This is a contradiction.

($\Leftarrow$) WLOG assume that node $(a_1, \ldots, a_{i-1})$ is the shallowest node that can be made A-consistent, its child $(a_1, \ldots, a_i)$ is consistent and A-compatible with all its ancestors, but the child is not visited. Since $(a_1, \ldots, a_i)$ is consistent and A-compatible with all its ancestors, a_i is in the domain of x_i. At node $(a_1, \ldots, a_{i-1})$, we do not annihilate any of the domains of future variables because the node can be made A-consistent. The branch will therefore be extended to the remaining values of the next variable x_i. One of these values is a_i and therefore node $(a_1, \ldots, a_i)$ is visited. □

This results lets us rank algorithms in the hierarchy presented in [Kondrak and van Beek, 1997].

Theorem 8 *If A-consistency is (strictly) stronger than B-consistency then maintaining A-consistency (strictly) dominates maintaining B-consistency.*

Proof: All nodes visited by an algorithm that maintains A-consistency, are A-consistent with all their ancestors and have parents that can be made A-consistent. But as A-consistency is stronger than B-consistency, all these nodes are B-consistent with all their ancestors and have parents that can be made B-consistent. Hence maintaining A-consistency dominates maintaining B-consistency. To show strictness, consider any problem that is B-consistent but is not A-consistent. □

From this result, it follows that MGAC on decomposable constraints strictly dominates MAC on the binary representation, and that MAC itself strictly dominates FC. We can also prove the correctness of MGAC and MAC using the following general result.

Theorem 9 *Maintaining A-consistency is correct.*

Proof: Soundness is trivial as only consistent nodes are visited. For completeness, suppose that some n-level node is consistent. Since this node is consistent, all its ancestors are also consistent. WLOG consider the deepest node $k = (a_1, \ldots, a_i)$, $i \geq 1$ that is not visited, and its parent is visited. When node $(a_1, \ldots, a_{i-1})$ is visited A-consistency is enforced, and since all its descendants are consistent, there is no domain wipe-out. Therefore, k is visited. □

6 Experimental results

To demonstrate the practical relevance of these theoretical results, we ran experiments in three domains.

6.1 Quasigroup completion

Gomes and Selman have proposed random quasigroup completion problems as a benchmark that combines some of the best features of random and structured problems [Gomes and Selman, 1997]. For these problems, there is a phase transition from a region where almost all problems are soluble to a region where almost all problems are insoluble as we vary the percentage of variables preassigned. The solution cost peaks around the transition, with approximately 42% of variables preassigned [Gomes and Selman, 1997].

We encoded the problem in ILOG Solver, a C++ constraint toolkit which includes Regin's algorithm for maintaining GAC on all-different constraints. We used the Brelaz heuristic for variable selection (as in [Gomes and Selman, 1997]) and Geelen's promise heuristic for value ordering (as in [Meseguer and Walsh, 1998]). Gomes et al. observed that search costs to solve random quasigroup completion problems can be modeled by a "heavy-tailed" distribution [Gomes *et al.*, 1997]. We therefore focus on the higher percentiles. Table 1 gives branches explored to complete an order 10 quasigroup with $p\%$ of entries preassigned, maintaining either AC on the binary representation or GAC on the all-different constraints. We see a very significant advantage for MGAC over MAC. With a random value ordering, the worst case for MGAC was also 2 branches. CPU times reflect the difference in explored branches. For example, some instances at the phase transition for quasigroups of order 20 were solved by MGAC in seconds, while MAC took hours.

p	MAC		MGAC	
	100th	90th	100th	90th
10	163	1	1	1
20	*	1	1	1
30	*	15	2	1
35	*	124	2	1
40	*	1726	2	1
42	*	*	2	1
45	*	*	2	1
48	*	2771	2	1
50	5692	1263	2	1
55	324	71	2	1
60	47	7	1	1
70	2	2	1	1
80	2	2	1	1
90	2	2	1	1

Table 1: Percentiles in branches searched to complete a quasigroup of order 10 using either MAC or MGAC. $*$ means that the instance was abandoned after 10000 branches. 100 problems were solved at each data point.

Table 2 shows that, as we increase problem size, almost all the problems remain trivial. The only exception was a single order 25 problem with 42% of its variables preassigned. Search was abandoned at the cuttoff limit of 10,000 branches. Apart from this, all instances were solved in less than 5 branches. This is a significant improvement over the results of [Gomes *et al.*, 1997] where, despite the use of random restarts, problems of order 25 were too expensive to solve, especially at the phase transition.

p	order 10		order 15		order 20		order 25	
	100th	90th	100th	90th	100th	90th	100th	90th
10	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1
30	2	1	1	1	2	1	2	1
40	2	1	2	1	2	1	2	1
42	2	1	2	1	2	1	*	1
45	2	1	3	1	3	1	3	1
48	2	1	2	1	2	1	2	1
50	2	1	2	1	2	1	3	1
60	1	1	1	1	4	1	1	1
70	1	1	1	1	1	1	1	1
80	1	1	1	1	1	1	1	1
90	1	1	1	1	1	1	1	1

Table 2: Percentiles in branches explored to complete quasigroups of order 10, 15, 20 and 25 using MGAC.

6.2 Quasigroup existence

A variety of automated reasoning programs have been used to answer open questions in finite mathematics about the existence of quasigroups with particular properties [Fujita *et al.*, 1993]. Is GAC useful on these problems? We follow [Fujita *et al.*, 1993] and look at the so-called QG3, QG4, QG5, QG6 and QG7 class of problems. For example, the QG5 problems concern the existence of idempotent quasigroups (those in which $a \cdot a = a$ for each element a) in which $(ba \cdot b)b = a$. For the definition of the other problems, see [Fujita *et al.*, 1993]. In these problems, the structure of the constraint graph is disturbed by additional non-binary constraints. These reduce the level of consistency achieved compared to quasigroup completion problems. Nevertheless, GAC significantly prunes the search space and reduces runtimes.

To solve these problems, we again use the Solver toolkit, maintaining either GAC on the all-different constraints, or AC on the binary representation, and the fail-first heuristic for variable ordering. To eliminate some of the symmetric models, as in [Fujita *et al.*, 1993], we added the constraint that $a \cdot n \geq a - 1$ for every element a. Table 3 demonstrates the benefits of MGAC over MAC. In QG3 and QG4, MAC explores twice as many branches as MGAC, in QG5 the difference is orders of magnitude, whilst there is only a slight difference in QG6 and QG7. MGAC dominates MAC in terms of CPU time as well as in terms of explored branches. It would be interesting to identify the features of QG5 that gives MGAC such an advantage over MAC, and those of QG6 and QG7 that lessen this advantage.

We now compare our results with those of FINDER [Slaney, 1992], MACE [McCune, 1994], MGTP [Fujita *et*

Order	QG3		QG4		QG5		QG6		QG7	
	MAC	MGAC	MAC	MGAC	MAC	MGAC	MAC	MGAC	MAC	MGAC
6	7	4	6	4	0	0	0	0	6	4
7	64	48	59	42	5	3	5	2	67	39
8	1,511	821	1,227	707	15	10	9	3	415	314
9	65,001	31,274	88,460	40,582	30	19	36	26	4,837	4,211
10	-	-	-	-	268	74	199	167	94,433	80,677
11	-	-	-	-	1,107	292	2,221	1,876	-	-
12	-	-	-	-	6,832	910	42,248	34,741	-	-
13	-	-	-	-	>1,000,000	27,265	-	4,730,320	-	-

Table 3: Branches explored using MAC on the binary representation and MGAC on the all-different constraints.

al., 1993], SATO [Zhang and M., 1994], and SEM [Zhang and Zhang, 1995]. Table 4 shows that Solver outperforms MGTP and FINDER by orders of magnitude, and explores less branches than SEM. SEM and SATO have sophisticated branching heuristics and complex rules for the symmetry breaking that are far more powerful than the symmetry breaking constraint we use [Zhang and Zhang, 1995]. It is therefore impressive that our simple Solver program is competitive with well-developed systems like SEM and SATO.

To conclude, despite the addition of non-binary constraints that disturb the structure of the constraint graph, MGAC significantly reduces search and runtimes on quasigroup existence problems. We conjecture that the performance of SEM and SATO could be improved by the addition of a specialized procedure to maintain GAC on the all-different constraints.

6.3 Small-worlds problems

Recently, [Watts and Strogatz, 1998] has shown that graphs that occur in many biological, social and man-made systems are often neither completely regular nor completely random, but have instead a "small world" topology in which nodes are highly clustered, whilst the path length between them is small. Walsh has argued that such a topology can make search problems hard since local decisions quickly propagate globally [Walsh, 1999]. To construct graphs with such a topology, we start from the constraint graph of a structured problem like a quasigroup and introduce randomness by deleting edges at random from the binary representation. Deleting an edge at random breaks up an all-different constraint on n variables into two all-different constraints on $n-1$ variables. For example, if $x_1, x_2, x_3 \ldots, x_k$ are all-different and remove the edge between x_1 and x_2 then we are left with all-different constraints on $x_1, x_3 \ldots, x_k$ and $x_2, x_3 \ldots, x_k$.

Figures 2 and 3 show percentiles in the number of branches explored and in CPU time to find the optimal coloring of order 10 quasigroups in which we delete $p\%$ of edges from the binary representation. The hardest problems had 5% of their edges removed. MGAC dominates MAC by orders of magnitude in the hard region both in terms of branches explored and CPU time. All instances were solved by MGAC within 120 seconds while approximately 10% of the instances could not be solved by MAC within 1 hour. As p increases, problems become very easy and both MGAC and MAC quickly find a solution. MAC starts to outperform MGAC in terms of CPU time as the overhead of GAC on the large number of all-different constraints is greater.

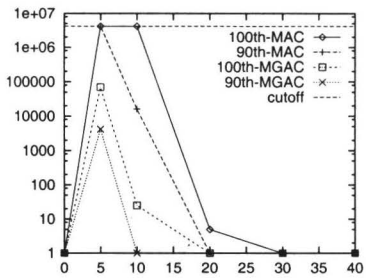

Figure 2: Percentiles in branches explored by MAC and MGAC to color small world problems.

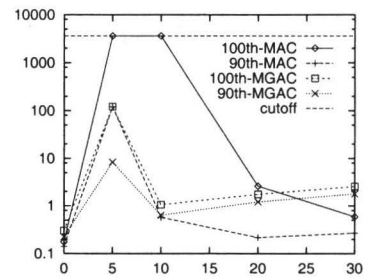

Figure 3: Percentiles of CPU seconds used by MAC and MGAC to color small world problems.

7 Related work

[Gomes and Selman, 1997] solved quasigroup completion problems using the MAC algorithm and a binary representation. They found that a randomization and restart strategy could eliminate the heavy-tailed behavior of the backtracking algorithm. However, they were still not able to consistently solve quasigroup completion problems of order 25 or larger.

[Meseguer and Walsh, 1998] solved quasigroup completion problems using forward checking (FC) on the binary representation. They found that discrepancy and interleaved based methods can reduce the heavy tail. However, their experiments were limited to quasigroups of order 20 and less.

[Bacchus and van Beek, 1998] have compared generalized FC on non-binary constraints with FC on the hidden variable and dual encodings into binary constraints. They show that

Order	Models	Branches					
		MGTP	FINDER	MACE	SATO	SEM	Solver (MGAC)
7	3	9	3	4	5	6	3
8	1	34	13	8	8	11	10
9	0	239	46	14	11	29	19
10	0	7,026	341	37	21	250	74
11	5	51,904	1,728	112	43	1,231	292
12	0	2,749,676	11,047	369	277	8,636	910

Table 4: Branches explored and models found on QG5 problems by a variety of different programs.

a simple extension of FC on the hidden variable encoding dominates generalized FC on the non-binary representation.

8 Conclusions

We have shown experimentally and theoretically the benefits of achieving generalized arc-consistency on decomposable constraints like all-different constraints. Generalized arc-consistency on such constraints lies between neighborhood inverse consistency and, under a simple restriction, path inverse consistency on the binary representation of the problem. On quasigroup completion problems, generalized arc-consistency achieves neighborhood inverse consistency. By generalizing the arguments of [Kondrak and van Beek, 1997], we proved that a search algorithm that maintains generalized arc-consistency on decomposable constraints dominates a search algorithm that maintains arc-consistency on the binary representation. Our generalization also proves the correctness of the algorithms that maintain arc-consistency or generalized arc-consistency. Our experiments demonstrated the practical value of achieving these high levels of consistency. For example, we solved almost all benchmark quasigroup completion problems up to order 25 with just a few branches of search. On quasigroup existence problems, we are competitive with the best programs, despite lacking their specialized branching heuristics and symmetry breaking rules.

What general lessons can be learnt from this study? First, it can be very beneficial to identify structure in a problem by means of a non-binary representation. We can use this structure to enforce higher levels of consistency than can be practical in a binary representation. Second, theory can be motivated by experiment. We were led to attempt our theoretical analysis by the exceptionally good experimental results on quasigroup completion problems. And finally, the all-different constraint really can make a big difference.

References

[Bacchus and van Beek, 1998] F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proc. of AAAI-98*, pages 311–318. 1998.

[Debruyne and Bessiere, 1997] R. Debruyne and C. Bessiere. Some practicable filtering techniques for the constraint satisfaction problem. In *Proc. of IJCAI-97*, pages 412–417. 1997.

[Dechter, 1990] R. Dechter. On the expressiveness of networks with hidden variables. In *Proc. of AAAI-90*, pages 555–562. 1990.

[Fujita *et al.*, 1993] Masayuki Fujita, John Slaney, and Frank Bennett. Automatic generation of some results in finite algebra. In *Proc. of IJCAI-93*, pages 52–57. 1993.

[Gaschnig, 1979] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Tech. rep. CMU-CS-79-124, Carnegie-Mellon University, 1979. PhD thesis.

[Gomes and Selman, 1997] C. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proc. of AAAI-97*, pages 221–226. 1997.

[Gomes *et al.*, 1997] C. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In G. Smolka, editor, *Proc. of CP97*, pages 121–135. 1997.

[Kondrak and van Beek, 1997] G. Kondrak and P. van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[McCune, 1994] W. McCune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Tech. Rep. ANL/MCS-TM-194, Argonne National Laboratory, 1994.

[Meseguer and Walsh, 1998] P. Meseguer and T. Walsh. Interleaved and discrepancy based search. In *Proc. of ECAI-98*. Wiley, 1998.

[Mohr and Masini, 1988] R. Mohr and G. Masini. Good old discrete relaxation. In *Proc. of ECAI-88*, pages 651–656, 1988.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of AAAI-94*, pages 362–367. 1994.

[Slaney, 1992] J. Slaney. FINDER, Finite Domain Enumerator: Notes and Guide. Tech. Rep. TR-ARP-1/92, Australian National University, 1992.

[Walsh, 1999] T. Walsh. Search in a small world. In *Proc. IJCAI-99*, 1999.

[Watts and Strogatz, 1998] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[Zhang and M., 1994] H. Zhang and Stickel M. Implementing the Davis-Putnam Algorithm by Tries. Tech. rep., University of Iowa, 1994.

[Zhang and Zhang, 1995] J. Zhang and H. Zhang. SEM: a System for Enumerating Models. In *Proc. of IJCAI-95*, pages 298–303, 1995.