Arc Consistency and Quasigroup Completion

Paul Shaw   Kostas Stergiou   Toby Walsh

APES Research Group
Department of Computer Science
University of Strathclyde
Glasgow G1 1XH, Scotland

{ps,ks,tw}@cs.strath.ac.uk

April 7, 1998

Abstract

Quasigroup completion is a recently proposed benchmark constraint satisfaction problem that combines the features of randomly generated instances and highly structured problems. A quasigroup completion problem can be represented as a CSP with n^2 variables, each with a domain of size n. The constraints can be represented either by $2n$ all different n-ary constraints or by binary pairwise constraints, giving a constraint graph with $2n$ cliques of size n. We present a comparison between the two representations and show that the $n-$ary representation reduces the cost of solving quasigroup completion problems drastically.

1 Introduction

Quasigroup completion [GS97b, GS97a, GSC97] is a recently proposed benchmark constraint satisfaction problem that combines the features of randomly generated instances and highly structured problems. A quasigroup is an ordered pair $(Q, \cdot)$, where Q is a set and $(\cdot)$ is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q. The constraints on a quasigroup are such that its multiplication table forms a Latin square. That is, each element occurs exactly once in every row or column of its n by n multiplication table. The order n of the quasigroup is the cardinality of the set Q. Quasigroup completion problem is the NP-complete problem of determining whether the remaining entries of a partially filled n by n table can be filled in such a way that a full quasigroup multiplication table is obtained.

A quasigroup completion problem can be represented as a CSP with n^2 variables, each with a domain of size n. The constraints can be represented by $2n$ all different n-ary constraints (one for each row and column). Alternatively, we can use binary "not equal to" constraints, giving a constraint graph with $2n$ cliques of size n. We present a comparison of the two representations and show that the $n-$ary representation can make quasigroup completion a largely trivial problem.

2 Constraint propagation

In [GS97b], quasigroup completion problems were solved by maintaining arc consistency on the $2n$ cliques of binary constraints. By comparison, [MW98] uses a forward checking algorithm on the binary constraints. A third method to solve quasigroup completion problems is by maintaining general arc consistency on the $n-$ary all different constraints using the algorithm of [Reg94]. We first show that forward checking on the binary constraints is less powerful than enforcing arc consistency on the binary constraints, which in turn is less powerful than enforcing general arc consistency on the all different constraints.

Consider, for instance, the following completion problem,

{r}	{r,g}
{r,g}	{r,g}

Forward checking gives,

{r}	{g}
{g}	{r,g}

By comparison, enforcing arc consistency on the binary constraints completes the quasigroup,

{r}	{g}
{g}	{r}

The example can be generalized to any order n.

Enforcing general arc consistency on the all different constraints is strictly stronger than enforcing arc consistency on the binary constraints. Consider the following quasigroup completion problem,

{r}	{r,g,b}	{r,g,b}
{r,g,b}	{r}	{r,g,b}
{r,g,b}	{r,g,b}	{r,g,b}

Enforcing arc consistency on the binary representation gives,

{r}	{g,b}	{g,b}
{g,b}	{r}	{g,b}
{g,b}	{g,b}	{r,g,b}

However, enforcing general arc consistency on the all different constraints filters out two more values in the bottom right square,

{r}	{g,b}	{g,b}
{g,b}	{r}	{g,b}
{g,b}	{g,b}	{r}

This example can be generalized to any order n,

{1}	{1,...n}	...	{1,...n}	{1,...n}
{1,...n}	{1}	...	{1,...n}	{1,...n}
⋮	⋮	⋮	⋮	⋮
{1,...n}	{1,...n}	...	{1}	{1,...n}
{1,...n}	{1,...n}	...	{1,...n}	{1,...n}

3 Random Preassignment

The random preassignment of some of the variables in the multiplication table introduces perturbations in the structured set of original constraints. [GS97b] uses a backwards checking method to generate problems with random preassignments. First, they randomly select the desired number of variables to be preassigned. For each variable they iterate over its possible values until a value consistent with all the previous assignments is found. If no such value exists, the current attempt to generate a problem fails and a new one is initiated. Using this method, a randomly generated quasigroup of order n with $p\%$ preassignments will have $p * n^2/100$ variables assigned and all the other variables will have their domains intact.

Alternatively, in [MW98] the possible random assignments are forward checked against the unassigned variables until a consistent assignment is found. This suggests two other ways to generate random problems. The first is to propagate using arc consistency on the binary constraints for every assignment, and the second is to propagate using general arc consistency on the $n-$ary all different constraints. A stronger propagation method will perform more domain pruning. A side effect of the last three methods is that some variables that are not randomly preassigned may become bound because of constraint propagation. Using these methods, a randomly generated quasigroup of order n with $p\%$ preassignments will have $p * n^2/100$ variables assigned and some of the other variables may have their domains pruned. In our implementation of the fourth method, preassignment stops when the number of variables that are bound, either through random assignment or propagation, reaches the desired number. If constraint propagation causes more variables to become bound than the desired number then the instance is “thrown away”.

[GS97b] observed that the cost of solving a quasigroup completion problem with random preassignment peaks around the point where 42% of the variables have been preassigned. They also observed a phase transition from a region where almost all problems are soluble to a region where almost all problems are insoluble around the same point.

4 Search Algorithm

Quasigroup completion was encoded using ILOG Solver, a powerful C++ constraint programming library [Pug94]. [GS97b, GSC97] have used the standard Solver backtracking algorithm with either the Brelaz [Bre79] or the r-Brelaz

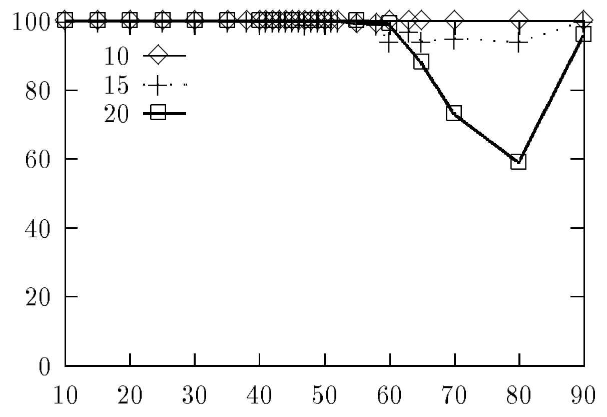

Figure 1: Percent satisfiable problems for quasigroups of order 10, 15 and 20 as a function of the preassignment percentage with initial assignments made using general arc consistency.

heuristic for variable selection, and random value selection. We have used the same algorithm but used *Geelen's promise* heuristic for value ordering [Gee92]. This value ordering heuristic was also used in [MW98]. This heuristic selects the value v at each variable x so that v least reduces the possibility of finding consistent values for the remaining unassigned variables. To achieve that, we select the value v that maximizes the product of the number of values that support v in all variables that are involved in a constraint with x. We say that these variables are *adjacent* to x. In terms of quasigroup completion, the variables that are involved in a constraint with a variable x are the variables in the same row or column with x. For a value v of variable x, the values that support v in an adjacent variable x_i are all the values in the domain of x_i except v. This means that if v is assigned to x then all the values in the domain of x_i, apart from v, can be assigned to x_i without violating a constraint. In addition, we used Solver's powerful all different propagator to achieve general arc consistency on the $n-$ary all different constraints. This propagation algorithm is based on the filtering algorithm presented in [Reg94].

5 Experimental Results

Our experiments show that when general arc consistency is used during random preassignment, almost all generated instances for every preassignment percentage are easy. Figures 1 and 2 demonstrate the effects of two different methods of preassigning variables on the percentage of satisfiable instances. Figure 1 shows what happens when random preassignments are propagated using general arc consistency, while Figure 2 corresponds to random preassignment using Gomes and Selman's backward checking method. Note that when the backward checking preassignment method is used, the way the constraints are represented (i.e., $n-$ary or binary) does not affect the percentage of consistent instances. At each data point, 100 instances were solved.

Experiments with Gomes and Selman's random preassignment method show

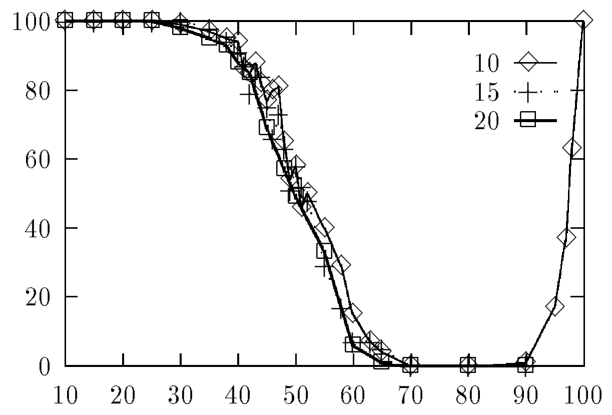

Figure 2: Percent satisfiable problems for quasigroups of order 10, 15 and 20 as a function of the preassignment percentage with preassignments made using the backward checking model of Gomes and Selman.

that enforcing general arc consistency on $n-$ary all different constraints instead of arc consistency on binary constraints reduces the cost of solving the problem drastically. This is demonstrated in Table 1, giving the percentiles in backtracks required to complete a quasigroup of order 10 with $p\%$ of its entries preassigned.

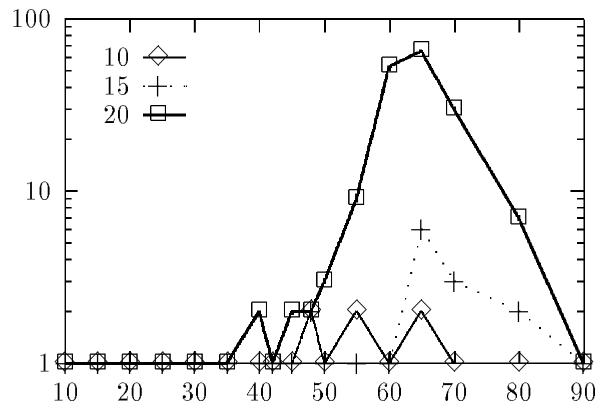

Figure 3: 90th percentile in branches explored to complete a quasigroup as a function of the preassignment percentage, with general arc consistency used to perform both preassignments and completion.

Figures 3 and 4 show how the cost of quasigroup completion scales when using general arc consistency to perform both preassignments and completion.

Figure 3 gives the 90th percentile in the number of backtracks required to find a completion or to show that none exists for quasigroups of order 10, 15 and 20, while Figure 4 gives the 100th percentile. Along the horizontal axis is the percentage of preassigned variables.

Table 2 shows how the cost of quasigroup completion scales when the random preassignments are done using the backward checking method, and general arc consistency on the all different constraints is used during the search. Note

p	$AC_g - AC_g$		$BC - AC_g$		$BC - AC_b$	
	100th	90th	100th	90th	100th	90th
10	0	0	0	0	163	0
20	0	0	0	0	*	0
30	0	0	1	0	*	14
35	0	0	1	0	*	123
40	1	0	1	0	*	1725
42	2	0	1	0	*	*
45	1	0	1	0	*	*
48	5	1	1	0	*	2770
50	1	0	1	0	5691	1262
55	4	1	1	0	323	70
60	2	0	0	0	46	6
65	15	3	0	0	7	2
70	5	2	0	0	1	1
80	1	0	0	0	1	1
90	0	0	0	0	1	1

Table 1: 100th and 90th percentiles in backtracks required to complete a quasigroup of order 10. $AC_g - AC_g$ corresponds to generating and solving using general arc consistency, $BC - AC_g$ corresponds to generating using Gomes and Selman's backward checking method and solving using general arc consistency, and $BC - AC_b$ corresponds to generating preassignments with the backward checking method and solving using arc consistency on binary constraints. A $*$ means that the instance was abandoned after 10000 leaf nodes had been visited. 100 problems were solved at each data point.

that for order 25 when 42% of the variables were preassigned, there was one extremely hard instance that was abandoned after 10000 backtracks. Apart from that, all instances were solved with less than 4 backtracks. The propagation algorithm we have used allowed us to solve easily almost all the generated instances for quasigroups of order 25 using a complete search method. This is a very significant improvement over the results of [GSC97] where despite the use of random restarts to enhance performance problems of order 25 were too expensive to solve, especially at the phase transition.

6 Conclusion

We have presented a comparison between the binary and $n-$ary representation of quasigroup completion problems. We have shown that enforcing general arc consistency on the $n-$ary constraints is strictly stronger than enforcing arc consistency on the binary constraints, and this is in turn strictly stronger than forward checking. We have studied the impact of such stronger constraint propagation on the cost to solve problems. Enforcing general arc consistency

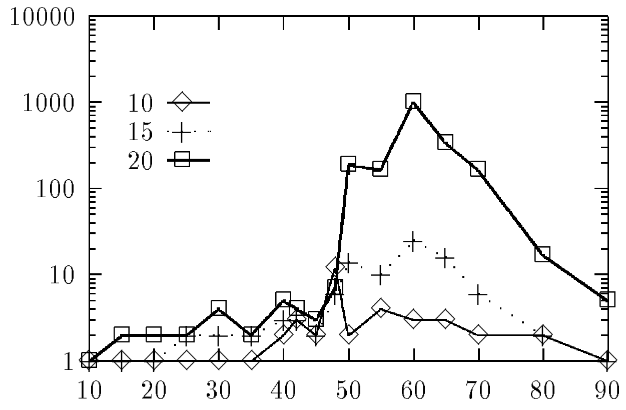

Figure 4: 100th percentile in branches explored to complete a quasigroup as a function of the preassignment percentage, with general arc consistency used to perform both preassignments and completion.

makes almost all problems easy. By comparison, problems can be very hard to solve if we just enforce arc consistency on the binary constraints.

There are two important lessons to be learnt from this study. First, while [GS97b] argue that it is desirable to add structure to random problems, you have to be careful that the structure you add is not exactly that which your propagators can use. And second, we should not restrict research to algorithms for binary constraints since some problems can be hard in a binary representation but relatively easy when expressed using n-ary constraints. In our future work, we intend to see if similar lessons apply to other related problems (e.g. sports scheduling and exam time-tabling).

References

[Bre79] D. Brelaz. New Methods to color the vertices of a graph. *JACM*, 22(4):251–256, 1979.

[Gee92] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI92)*, pages 31–35, 1992.

[GS97a] C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of UAI-97*, Providence, RI., USA, 1997.

[GS97b] C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proceedings of the AAAI-97 National Conference*, pages 221–226, Providence, RI., USA, 1997.

[GSC97] C. P. Gomes, B. Selman, and N. Crato. Heavy-tailed probability distributions in combinatorial search. In *Proceedings of CP-97*, pages 121–135, Vienna, Austria, 1997.

p	order 10		order 15		order 20		order 25	
	100th	90th	100th	90th	100th	90th	100th	90th
10	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
30	1	0	0	0	1	0	1	0
35	1	0	1	0	1	0	1	0
40	1	0	1	0	1	0	1	0
42	1	0	1	0	1	0	*	0
45	1	0	2	0	2	0	2	0
48	1	0	1	0	1	0	1	0
50	1	0	1	0	1	0	2	0
55	1	0	1	0	2	0	3	0
60	0	0	0	0	3	0	0	0
65	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0

Table 2: 100th and 90th percentiles in backtracks required to complete quasigroups of order 10, 15, 20 and 25 using the backward checking method for preassignment and general arc consistency when searching.

[MW98] P. Meseguer and T. Walsh. Interleaved and discrepancy based search. 1998. Submitted to ECAI-98.

[Pug94] J. F. Puget. A C++ Implementation of CLP. Technical Report 94-01, ILOG S.A., Gentilly, France, 1994.

[Reg94] J. C. Regin. A filtering algorithm for constraints of difference in csps. In *Proceedings of AAAI-94*, pages 362–367, 1994.