Symmetry Breaking in Constraint Satisfaction with Graph-Isomorphism: Comma-Free Codes

Justin Pearson
Department of Information Technology
Division of Computer Systems
Uppsala University, Box 337, 751 05 Uppsala, Sweden
justin@docs.uu.se

December 5, 2003

Abstract

In this paper the use of graph isomorphism is investigated within the framework of symmetry breaking in constraint satisfaction problems. A running example of Comma-free codes is used to test the methods. But the technique can be extended to other problems. In particular Symmetry Breaking via Dominance Detection (SBDD) is applied to find Comma-Free Codes. To check if a current partial solution is symmetrically equivalent to a previously found no-good, graph isomorphism is used. In particular the powerful and fast graph isomorphism package `nauty` is used. Experimental results show that for difficult instances SBDD+Nauty out performs lexicographic ordering[1].

1 Introduction

Constraint Satisfaction [1, 5, 14] is a framework for stating and solving combinatorial problems. A constraint satisfaction problem (CSP) is a collection

[1]An earlier version of this paper appeared in the SymCon03 workshop on Symmetry in Constraints held at the CP03.

of variables that are to be assigned values from domains subject to a collection of constraints that restrict possible combinations of values of sets of variables. In general solving CSPs is NP-complete.

Most solution methods for CSPs involve a mixture of backtracking on variable assignment and propagation of local information. Often propagation takes the form of local consistency methods (such are arc-consistency) or specialised propagation algorithms implemented as global-constraints.

Recently [9, 6, 2, 3] there has been much interest using symmetries inherent in problems to speed up search. Essentially symmetrically equivalent branches in the search space are pruned. Often this is achieved by either inserting new constraints on failure [9] to exclude symmetrically equivalent nodes later in the search or checking the current partial against previous nogoods [6].

This paper uses graph-isomorphism to check if a partial assignment is symmetrically equivalent to a previously found nogood this is applied to a running example, comma-free codes. The method of graph isomorphism could be applied to other problems. The rest of paper is organised as follows: Comma-Free codes are introduced, then Symmetry Breaking via Dominance Detection (SBDD) is outlined the required graph is then constructed and finally experimental results are presented.

2 Comma-Free Codes

A *comma-free code* [10, 11, 12] over an alphabet A is a set, $C \subseteq A^*$, of words over A such that given any two words, $w, v \in C$, any sub-word, u, of the concatenation, wv, is not in the code. Here we will be only interested in codes where all the words have the same length. See figure 1 for an example of a comma-free code.

$$00001 \quad 00101 \quad 00110 \quad 11001 \quad 11010 \quad 11110$$

Figure 1: A Comma-Free Code of 6 words of length 5 over the alphabet $\{0, 1\}$

Comma-Free Codes were originally inspired by biology. Genetic information in the cell is stored in DNA, which for a computer scientist is a string of letters from the alphabet A, C, T and G. Via a transcription mechanism proteins are constructed which are chains of amino acids from an alphabet of 20 different acids. One question that exercised biologists was: How are

each of the 20 different acids coded as strings of DNA? One proposal is that the 20 acids are coded as comma-free codes, so as to aid transcription. As it turns out there is a comma-free code of size 20 of words of length 3 over a four letter alphabet. But in reality nature is not so simple: the genetic code is not a comma-free code and in fact is not even a code in the mathematical sense. Figure 2 shows a comma free code over alphabet size 4 with words of length 3.

112	113	114	212	213	214	223	224	312	313
314	323	324	334	412	413	414	423	424	434

Figure 2: Comma-Free code with word length 3 over an alphabet of size 4.

Comma-Free Codes are still interesting from a mathematical point of view. One property of interest in the theory of comma-free codes is: given an alphabet size and a word length what is the maximal number of words a comma-free code can contain?

3 Symmetry Breaking via Dominance Detection (SBDD)

Essentially SBDD [6] prevents symmetrically equivalent no-goods being explored. In more detail, given a failure during search (where a failure is some partial assignment that cannot be extended) the search procedure should guarantee that no symmetrically equivalent partial assignment is ever explored. In SBDD an extra procedure is added to the search procedure which checks if the current partial assignment has already been seen before (dominated) as a no-good.

One way of implementing SBDD is by maintaining a database of previously seen partial assignments and checking the current assignment against all the previous partial assignments for equality modulo symmetry.

Various modifications can be applied to SBDD in reducing the number of no-goods that need to be stored. In particular during depth-first search no-goods below a completed node in the search tree can be removed.

In implementing a search procedure for comma-free codes each no-good and partial assignment will be converted into a graph. In implementing SBDD with the optimisation of removing redundant no-goods the dominance

checking procedure has to check if the graph of the previously found no-good is isomorphic to a subgraph of the current no-good. This procedure is NP-complete. In the implementation of SBDD used in this paper time is traded for space. All the no-goods are kept, none are thrown away as would be done with an optimised implementation of SBDD. This allows graph isomorphism to be used as a dominance check since only no-goods at the same level in the search tree are checked. The complexity of graph isomorphism is not known, but in practice it is often polynomial.

4 Comma-Free Codes: Graph Isomorphism and Symmetry Breaking

To model and find comma-free codes using a CSP, a code is represented as a list of lists where each list represents a word. There are two obvious symmetries with this representation. First, the order of the words does not matter: that is any permutation of the words is still a comma-free code. Secondly given any comma-free code, C, over an alphabet A and any bijection, $f : A \to A$, on the alphabet, applying the bijection to every word in the code (giving the code $\{f(w)|w \in C\}$ where $f(w) = f(w_1) \cdots f(w_k)$ when $w = w_1 \cdots w_k$) is still a comma-free code and finally given any comma free code C the code C^R which contains all the words in C reversed is still a comma free code. In this paper only the first two symmetries are broken.

One way of combating the first symmetry is to order the code words lexicographically using the `lex-chain` [4] global constraint. The second symmetry (the value symmetry) is harder to break efficiently and even harder to break in the presence of the first symmetry. One such approach would be to reformulate the problem as a matrix model and use two lexicographic chain constraints (see [7]) but this would still not break all the symmetry in the problem.

The search procedure implemented to solve the CSP searches for a complete word at a time in the code. To check if the current partial assignment is symmetrically equivalent to a previously found no-good at the same level graphs are constructed for both the partial assignment and the no-good such that the graphs are isomorphic if and only if the assignments are symmetrically equivalent. Isomorphism of the graphs is checked using the `nauty` [13] system, which is able to return a canonical graph such that two graphs are

isomorphic if and only if they have the same canonical graph. Hence if the no-goods are stored as `nauty` canonical graphs then a new partial assignment can be converted into a canonical graph and checked against stored no-goods.

The graph used for isomorphism testing is constructed as follows. Given a partial assignment of n code words of the form:

$$\begin{aligned} W_1 &= w_1^1 \dots w_1^k \\ &\vdots \\ W_n &= w_n^1 \dots w_n^k \end{aligned}$$

where each w_i^j is a letter from the alphabet of the code (because of the way the search procedure works there will be no code words to level n which have only some entries grounded), then a coloured graph with $1 + k$ colours is constructed as follows:

- The set of nodes of the graph is the disjoint union of the sets A (the domain or the alphabet of the code) and the set $\{w_i^j | 1 \leq i \leq n \wedge 1 \leq j \leq k\}$;
- The nodes in A are all the same colour and the members of the disjoint sets $K_i = \{w_j^i | 1 \leq j \leq n\}$ give the k other colours for $1 \leq i \leq k$;
- For every graph, regardless of the code, there are edges from w_i^j to w_i^{j+1} for all $1 \leq i \leq n$ and all $1 \leq j < k$;
- For all $1 \leq i \leq n$ and $1 \leq j \leq k$ if $w_i^j = d$ then there is an edge from d to w_i^j.

In figure 3 an example graph is given for the code $\{001, 101\}$. It is then possible to prove that two codes of the same length are symmetrically equivalent if and only if the two graphs are coloured graph isomorphic. Such an isomorphism gives a bijection on the code words. The separation of the colours gives a bijection on the domain elements. The edges between w_i^j and w_i^{j+1} forces any isomorphism f such that $f(w_i^j) = w_{i'}^{j'}$ forces $f(w_i^{j+1})$ to be $w_{i'}^{j'+1}$. Finally edges between the domain elements and the nodes w_i^j force the isomorphism on domain elements to be an isomorphism of the code words.

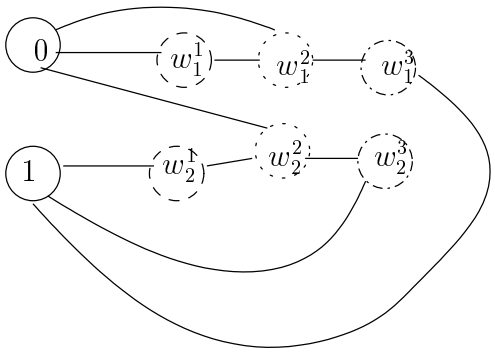

Figure 3: Coloured graph for the code $\{001, 101\}$

5 Results

An implementation of SBDD with graph isomorphism was compared against lexicographically ordering the words. Although lexicographic ordering will not break all the symmetry (that is both the value symmetry and the interchangeability of the words in the code) in many cases it performs well. It is not until relatively difficult instances that SBDD with graph isomorphism competes with lexicographic ordering. In figure 4 results are presented for

Code Length	Time: SBDD	Backtracks	Time: lex	Backtracks
13	1.17	34	0.84	18
14	4.4	96	2.41	98
15	259.09	3145	159.29	4198
16	328.37	3145	208.6	4198
17	1891.27	13608	3729.96	57680
18	2272.76	13608	4591.89	57680
19	2741	13608	5768.78	57680
20	3234.31	13608	7247.42	57680

Figure 4: Results on domain size 4, word length 3, all times reported in seconds.

codes of words of length 3 over a domain of size 4 using Sicstus Prolog on a Pentium 4 2.53Ghz machine (with 512Mb of memory) running Linux; note that in this case 20 is the maximal code length. The search looks for one code satisfying the constraints. The number of backtracks does not refer to the total number of backtracks, but the backtracks at the word level. At each

level in the search tree a complete word is found, thus on a backtrack a new code word is found. To find each code word at each level in the tree Sicstus' normal labelling procedure is used. After code length 16, SBDD with graph isomorphism wins. Also in figure 5 results are shown for code words of length 3 over an alphabet of size 5.

Code Length	Time: SBDD	Backtracks	Time: lex	Backtracks
22	36.02	704	33.58	1092
23	40.16	704	40.43	1092
24	44.74	704	47.74	1092
25	47.93	704	54.07	1092
26	841.82	2926	878.23	10827

Figure 5: Results: Domain size 5, word length 3.

6 Conclusion

Although it might seem that losing the optimisations possible with SBDD on a depth-first search requires many no-goods to be stored and checked, by using graph isomorphism these no-goods can be checked relatively quickly. Also since the symmetry groups in general would be large and constructing the graphs is quite simple this method avoids generating many no-goods as would be done with SBDS or its optimisations [8].

The technique of using graph isomorphism could also be applied to Balanced Incomplete Block Designs and the Social Golfer and this is work in progress. This work was partially supported by a STINT institutional grant IG2001-67 of STINT the grant 221-99-369 of the Swedish Research Council an earlier version of this work was presented at SymCon'03 at CP'03 conference Cork.

References

[1] K.R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.

[2] R. Backofen. Constraint techniques for solving the protein structure prediction problem. In M. Maher and J.-F. Puget, editors, *Proceedings of CP'98*, volume 1520 of *LNCS*, pages 72–86. Springer-Verlag, 1998.

[3] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In J. Jaffar, editor, *Proceedings of CP'99*, volume 1713 of *LNCS*, pages 73–87. Springer-Verlag, 1999.

[4] Mats Carlsson and Nicolas Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002.

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

[6] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Proceedings of CP'01*, volume 2293 of *LNCS*, pages 93–107. Springer-Verlag, 2001.

[7] P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. Van Hentenryck, editor, *Proceedings of CP'02*, volume 2470 of *LNCS*, pages 462–476. Springer-Verlag, 2002.

[8] Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints: Symmetry breaking during search. In P. Van Hentenryck, editor, *Proceedings of CP'02*, volume 1520 of *LNCS*, pages 415–430. Springer-Verlag, 2002.

[9] I.P. Gent and B.M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of ECAI'00*, pages 599–603, 2000.

[10] S.W. Golomb and L.R. Welch. Comma-free codes. *Candian Journal of Mathematics*, 10:202–209, 1958.

[11] B.H. Jiggs. Recent results in comma-free codes. *Canadian Journal of Mathematics*, 15:178–187, 1963.

[12] Nguyen Huong Lam. Completing comma-free codes. *Theoretical Computer Science*, 301:399–415, 2003.

[13] Brendan McKay. `nauty` user's guide (version 2.2). Available via `http://cs.anu.edu.au/people/bdm/nauty/`.

[14] P. Van Hentenryck. Constraint and integer programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002.