Proc. SymCon'03

NuSBDS: Symmetry Breaking made Easy

Iain McDonald

School of Computer Science, University of St Andrews,
Fife KY16 9SS,
Scotland,
iain@dcs.st-and.ac.uk

Abstract. Over the past few years we have seen a marked increase in the interest of symmetry breaking in constraint programming. Many new techniques have been introduced to deal with the problems that symmetrical CSPs create. However, if we are to see symmetry breaking techniques being used by the constraint community in general we need to make sure they are implemented to be easy to use. We present such an implementation: NuSBDS, which has been designed to allow the average constraint programmer to break symmetry in their CSPs quickly and easily.

1 Introduction

A constraint solver is made of many parts. There are many different ways to traverse the search space of a CSP: backtrack search, conflict directed backjumping, limited discrepancy search, depth bounded search etc. There are also many levels of consistency to enforce such as bounds consistency, forward checking and arc or path consistency. Finally there are many popular dynamic variable ordering heuristics we can choose: most constrained first, smallest domain first etc.

At present there are also a large number of different methods of breaking symmetry in CSPs. A symmetry breaking system is quite simply an implementation of one or more symmetry breaking methods that can be used to avoid redundant work performed by the constraint solver.

In the future we hope to see symmetry breaking systems playing an equal role to search and inference techniques and heuristics for highly symmetric problems in constraint solvers. This work will look at the importance of implementing symmetry breaking methods in terms of ease of use and generality. By doing so we hope to suggest ways to take symmetry breaking research into the mainstream of constraint solvers.

2 Current methods of Breaking Symmetry

We first saw interest in symmetry breaking in CSPs in [1] in 1991. Since then, there has been more research into symmetry breaking such as [2–4] however, over the past few years there has been a marked increase in the number of

papers published that deal with the study of symmetries in CSPs [5–11]. All these papers have introduced new techniques for dealing with symmetries in CSPs. However, in the same way that constraint programmers wish to use an arc-consistency algorithm or an all-different constraint without needing to know the theory or implementation behind them, symmetry breaking systems need to be just as easy to use. We will now briefly review the most recent methods of symmetry breaking and highlight how applicable they are to being implemented as a symmetry breaking system.

2.1 SBDS

Symmetry Breaking During Search or SBDS was developed by Backofen and Will [5] and independently by Gent and Smith [6]. This technique involves maintaining a list of symmetries of a given problem and uses them to produce symmetry breaking constraints which are posted upon backtracking from a nogood. The latest ECL^iPS^e release [12] contains an SBDS library and currently this is the only symmetry breaking system distributed with any constraint solver.

Although effective at reducing the run-times of solving symmetric CSPs, the main problem with using SBDS is that the constraint programmer needs to produce an explicit representation of the list of symmetries. For problems that have even a moderate amount of symmetry, a program needs to be written to automate the production of such a list. Also, since the number of symmetries generally varies with respect to the size of the CSP, a list of symmetries is generally valid only for a specific sized instance of a CSP.

2.2 Lex constraints for Models with Matrices

By modelling problems in matrices, a lot of symmetry can be discarded by lexicographically ordering the vertices [11]. The main advantage in using lexicographic ordering constraints is their ease of use. Also, there exists an algorithm that enforces GAC on a single lex constraint in linear time [13], and an algorithm for AC on a chain of lex constraints [14].

The problem with breaking symmetry in this method is that the constraint programmer is restricted at the modelling stage and they can only break symmetries of a certain type i.e. symmetries resulting from the interchangeability of rows and columns of matrices. Also, for an n-dimensional matrix where $n > 1$, not all symmetry is broken.

2.3 SBDD and Global Cut Framework

Symmetry Breaking via Dominance Detection or SBDD was developed by Fahle, Schamberger and Sellman [9] and the Global Cut Framework was developed by Focacci and Milano [8]. The main idea of these works is that if we can map a previously visited set of domains to a subset of the current set of domains in search via some symmetry, we can backtrack from the current state in search.

Puget simplified this problem in [15] to finding a mapping of a failed set of decisions into a subset of the current set of decisions in search.

In order to use these techniques the constraint programmer must write a function called a dominance checker in [9] or a function to produce a Symmetry Removal Cut for a given "family" of symmetries [8]. Therefore, for any symmetric problem that a constraint programmer wishes to solve, they must write additional code in order to break their symmetries.

2.4 GHK-SBDS

A modification to the original SBDS algorithm was introduced by Gent, Harvey and Kelsey in [10]. In this paper, the Computational Group Theory programming language GAP [16] was used to deal with the symmetry in an algebraic way. The symmetries were broken using a similar method to SBDS (via posting constraints during search) however, GHK-SBDS represents symmetries as a group rather than an explicit list. A *generator set* of a group typically has 2 to 6 symmetries in it which makes the representation concise.

The problem with this method is that in order to use the GHK-SBDS algorithm, constraint programmers must have some knowledge of group theory, a pure mathematics discipline, in order to label their CSP and represent its symmetries as a permutation group.

3 NuSBDS

We now introduce a symmetry breaking system for the ILOG Solver constraint solving tool [17]. As such the following sections of example code will be clearer to readers already familiar with the Solver constraint solving tool and more specifically, C++. NuSBDS is based on the GHK-SBDS implementation[1] and as such, can handle over 10^7 symmetries. However, even now this number is not as impressive as results matched by other symmetry breaking techniques. Since NuSBDS is an implementation, it is specific to both ILOG Solver and the GHK-SBDS algorithm. The main theme of this research can be used for different constraint solvers *and* different symmetry breaking algorithms as is mentioned in Section 3.1. The research contribution that NuSBDS makes is two-fold:

1. The group theory used by NuSBDS is hidden from the constraint programmer thus no previous pure mathematics knowledge is needed to use it.
2. A series of *macros* have been written to allow the constraint programmer to describe their groups quickly and easily.

These macros are essentially functions representing types of symmetries. When called they look at the model of the CSP, perform some basic error checking to see if the symmetry can be applied to the model, and internally store the generators for those symmetries. Different macros can be called repeatedly so

[1] NuSBDS uses a simple GHK-SBDS implementation e.g. there are no delayed goals.

that different types of symmetry can be combined to create direct products of groups. This allows the constraint programmer to describe complicated groups by using a few simple commands.

For example, say we have an encoding of the n-queens problem which has n variables, where the value represents which column the queen should be placed in. In this model, the symmetries of a square act on the assignments. Using Solver we require an `IloGoal` object to search on, which we create like so, where `x` is the array of variables and `env` is an `IloEnv` object. The important facts to take from this are that `x` contains a list of constrained integer variables and the `IloGoal` object will allow us to call functions that will search to find values for the variables in `x`.

```
IloGoal goal = IloGenerate(env, x);
```

We can use the following code to break the symmetries of the problem, where `solver` is an `IloSolver` object.

```
Symmetry* sym = new (env) Symmetry(env);
IloIntArray type(env, 1, SQUARE);
IloGoal goal;
if(symBreaking){
    goal = NuSBDSGenerate
        (env, x, sym->setup(x, solver, ASSIGN, type));
} else{
    goal = IloGenerate(env, x);
}
```

The above code contains some new terms that need explanation. Firstly, the `Symmetry` class is a part of NuSBDS. We use this to describe and break the symmetries of the CSP. The `type` object is simply an array of integers. The `#define` statement in C++ was used to associate each macro with an integer, and the `type` array merely contains the list of integers (representing macros) to use. The `NuSBDSGenerate` function takes an additional parameter to the `IloGenerate` function i.e. the `Symmetry` object we created to deal with the symmetry. Note that we must also call the function `setup`, which records which macros to use and whether or not the symmetry acts on assignments (`ASSIGN`) or variables (`VAR`).

As another example, say we have the BIBD problem (CSPLib problem: prob028 [18]) which we can model as a matrix of 0/1 variables, given any solution we can permute the rows and columns to yield another. This matrix may not necessarily be a square so in order to describe the group, we need to tell NuSBDS the number of columns. We can then simply use two macros to describe the symmetries of this problem. Notice that these macros have "rectangle" in their name to show that they potentially act on a non-square matrix[2].

[2] Though all square matrices are also rectangles, a macro for squares is provided for simplicity.

```
Symmetry* sym = new (env) Symmetry(env);
sym->setNumOfColumns(numOfCol);
IloIntArray type
    (env, 2, SYMMETRIC_RECTANGLE_ROW, SYMMETRIC_RECTANGLE_COL);
IloGoal goal;
if(symBreaking){
    goal = NuSBDSGenerate
        (env, x, sym->setup(x, solver, VAR, type));
} else{
    goal = IloGenerate(env, x);
}
```

Given a problem that needs to use more macros, we just create a larger `type` array where each element represents the macro to be used. Currently there are 11 different macros which can be used for either symmetries acting on assignments (`ASSIGN`) or variables (`VAR`). Here is the list of macros for variable symmetry:

- SQUARE - n^2 variables with the symmetry of an $n \times n$ square acting on them
- CYCLE_ROW - n^2 variables make a square where the rows can be cycled
- CYCLE_COL - n^2 variables make a square where the columns can be cycled
- SYMMETRIC_ROW - n^2 variables make a square where the rows are interchangeable
- SYMMETRIC_COL - n^2 variables make a square where the columns are interchangeable
- SYMMETRIC_RECTANGLE_ROW - as SYMMETRIC_ROW but for non-square matrices
- SYMMETRIC_RECTANGLE_COL - as SYMMETRIC_COL but for non-square matrices

Here is the list of macros for variable and value symmetry:

- SQUARE - e.g. n-queens
- SYMMETRIC_VAR - interchangeable variables
- SYMMETRIC_VAL - interchangeable values
- SQUARE_VAR - n^2 variables with the symmetry of an n by n square acting on them

The reader should note that some classes of symmetry can be broken more easily than using the underlying GHK-SBDS algorithm e.g. freely interchangeable values, as found in graph colouring, can be broken using the constraint described in [19], using the technique in [20] or as is noted in [5], only a subset of the constraints posted by SBDS is needed to break all these symmetries. NuSBDS is most useful for combinations of symmetries that result in groups for which there are no efficient methods of dealing with.

3.1 Using macros with other symmetry breaking methods

Though the NuSBDS implementation is specific to GHK-SBDS, it is possible to use the same macro approach to make symmetry descriptions simpler for other symmetry breaking methods.

Using the macros to describe the symmetries of a CSP creates a permutation group representing these symmetries. This is convenient for GHK-SBDS since we use group theory algorithms during search to break symmetry. However, methods such as SBDD try to map a previous failed set of nogoods (or sets of domains) into a subset of the current set of decisions (or set of domains) by solving another CSP i.e. a dominance check. This new CSP repeatedly combines a subset of the symmetries of the problem to find such a symmetry that satisfies the above condition.

In this case, the subset of symmetries used in the dominance check is simply a generator set of the group. Such a group can be created with macros as seen above and it should also be possible to write a function that takes a group as its parameters and outputs a CSP representing a dominance check. Therefore, the approach taken with NuSBDS can be modified for other symmetry breaking systems.

4 Empirical Results

We now present some brief empirical evidence of NuSBDS solving symmetric CSPs. A *Most Perfect Magic Square* is an $n \times n$ matrix of variables with different values from 1 to n^2 that satisfy the following constraints:

1. Each row and column sum to $(n^3 + n)/2$
2. Every 2×2 block of cells (including wrap-around) sum to $2T$ (where $T = n^2 + 1$).
3. Any pair of integers distant $\frac{1}{2}n$ along a diagonal (including wrap-around) sum to T.

This problem has $8n^2$ symmetries which are derived from the symmetries of a square combined with being able to cycle the rows and columns. Describing these symmetries was done very simply in NuSBDS by making the following array act on variables (`VAR`):

```
IloIntArray type(env, 3, SQUARE, CYCLE_ROW, CYCLE_COL);
```

The results of using NuSBDS to break the symmetry in this problem can be found in Table 1.

5 Conclusions and Future Work

NuSBDS is very successful at making the process of describing symmetries quick and easy. Although there is initially a small learning curve when presented with

Parameters		Solver 5.2			NuSBDS		
n	Syms.	Sols.	Time	Backtracks	Sols.	Time	Backtracks
4	128	384	0.48	1594	3	0.07	73
6	288	0	2905.12	4176447	0	14.85	25157

Table 1. Most Perfect Magic Squares. There are trivially no solutions for odd values of n and $n = 8$ is too complex to find all solutions within a reasonable amount of time.

the syntax of using NuSBDS, the constraint programmer needs no prior knowledge of any symmetry breaking technique or group theory. This is a large step forward in being able to include symmetry breaking systems in constraint solvers in general.

There are two main areas where NuSBDS could be improved. Firstly, the symmetry breaking technique should be improved so that larger symmetry groups can be dealt with. Secondly, a more expressive way of describing symmetries should be created so that users can create *wreath* groups such as those that occur in the Social Golfer's problem, and finally users should be able to supply their own macros. This would make NuSBDS a more complete and competent symmetry breaking system.

Acknowledgements

The author would like to thank Warwick Harvey and all the members of the APES research group, most notably Tom Kelsey for continuing to play football even after his arm was broken. Finally I would like to thank the reviewers for their enthusiasm and insightful comments. This work is partly supported by EPSRC grant GR/R29666 and by an EPSRC PhD studentship.

References

1. Eugene Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *American Association for Artificial Intelligence, Vol.1*, pages 227–233, 1991.
2. James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Knowledge Representation'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
3. Belaid Benhamou. Study of symmetry in constraint satisfaction problems. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, Orcas Island, Seattle, USA, 1994.
4. Jean-François Puget. On the satisfiability of symmetrical constrained satisfaction problems. In J. Komorowski and Z. W. Ras, editors, *Methodologies for Intelligent Systems: Proc. of the 7th International Symposium ISMIS-93*, pages 350–361. Springer-Verlag, 1993.

5. Rolf Backofen and Sebastian Will. Excluding symmetries in constraint-based search. In Alex Brodsky, editor, *Principles and Practice of Constraint Programming*, pages 73–87. Springer-Verlag, 1999.
6. Ian P. Gent and Barbara Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.
7. Iain McDonald. Unique symmetry breaking in CSPs using group theory. In Piere Flener and Justin Pearson, editors, *SymCon'01: Symmetry in Constraints*, pages 75–78, 2001. Available from http://www.csd.uu.se/~pierref/astra/symmetry/index.html.
8. Filippo Focacci and Michaela Milano. Global cut framework for removing symmetries. In Toby Walsh, editor, *Principles and Practice of Constraint Programming - CP2001*, pages 77–92. Springer-Verlag, 2001.
9. Torsten Fahle, Stefan Schamberger, and Meinolf Sellman. Symmetry breaking. In Toby Walsh, editor, *Principles and Practice of Constraint Programming - CP2001*, pages 93–107. Springer-Verlag, 2001.
10. Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints: Symmetry breaking during search. In P Van Hentenryck, editor, *Principles and Practice of Constraint Programming*, pages 415–430. Springer-Verlag, 2002.
11. Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming*, pages 462–476. Springer-Verlag, 2002.
12. Mark Wallace, Stefano Novello, and Joachim Schimpf. ECL^{i}PS^e: A platform for constraint logic programming. Technical report, IC-Parc, Imperial College, 1997.
13. Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Global constraints for lexicographic orderings. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming*, pages 93–108. Springer-Verlag, 2002.
14. Mats Carlsson and Nicolas Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report Research Report T2001-18, Swedish Institute of Computer Science, 2002.
15. Jean-François Puget. Symmetry breaking revisited. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming*, pages 446–461. Springer-Verlag, 2002.
16. The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000.
17. ILOG S.A., Gentilly, France. *ILOG Solver, Version 5.0*, 2000.
18. Ian P. Gent and Toby Walsh. Csplib: a benchmark library for constraints. Technical report, Technical report APES-09-1999, 1999. Available from http://www.csplib.org/.
19. Ian P. Gent. A symmetry breaking constraint for indistinguishable values. In Piere Flener and Justin Pearson, editors, *SymCon'01: Symmetry in Constraints*, pages 25–32, 2001. Available from http://www.csd.uu.se/~pierref/astra/symmetry/index.html.
20. Pascal Van Hentenryck, Pierre Flener, Justin Pearson, and Magnus Agren. Tractable symmetry breaking for CSPs with interchangeable values. In *International Joint Conference of Artificial Intelligence*, pages 575–580, Acapulco, Mexico, 2003.