

B. Benhamou, B.Y. Choueiry, and B. Hnich
(Eds.)

Proceedings of the Seventh International Workshop on Symmetry in Constraint Satisfaction Problems

The Seventh International Workshop on Symmetry in Constraint Satisfaction Problems of the Thirteenth International Conference on Principles and Practice of Constraint Programming, CP 2007, RI, Providence, USA, September 23–27, 2007.

Preface

Symmetries frequently occur in Constraint Satisfaction Problems (CSPs). When undetected, search redundantly explores symmetric parts of the search space, which causes thrashing. The idea of exploiting symmetry to prevent this misbehavior in search was discussed by Glaisher as far back as 1874. In recent years, new techniques to detect and/or break symmetry have been proposed. However, the characterization, automatic detection, and exploitation of symmetry and its various forms (e.g., local, dynamic, and weak) still present many challenges.

The International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon) has been organized since 2001 during the International Conference on Principles and Practice of Constraint Programming (CP). The workshop provides an opportunity for researchers to present advances in the theory and discuss applications and case studies that exhibit some form of symmetry. Following the success of the previous years, the Seventh SymCon is held during CP 2007 on September 23, 2007 in Rhode Island, Providence (USA).

This year, the program includes twelve accepted papers, presented as technical talks. Six papers are invited for long presentations, the six others for short presentations. SymCon 07 also features an invited talk by Dr. Stephan Szeider. The invited talk focuses on symmetry reasoning in resolution systems. The accepted papers investigate a range of problems related to symmetry detection in CSP models (and instances), and symmetry breaking in local and backtrack search. Symmetry-breaking techniques are investigated in various contexts including optimization, unsatisfiability, quantified Boolean formulas, and for the sub-graph isomorphism problem. Value symmetry, minimal ordering constraints for some families of variable symmetries, local symmetry and weak symmetry are also studied. Finally, the usefulness of these techniques are demonstrated for the Multiple Knapsack problem and in the Balanced Academic Curriculum Problem.

As co-chairs of SymCon 07 and editors of this volume, we would like to take the opportunity to thank the invited speaker, Stephan Szeider, the authors who submitted a paper to this volume, and the members of the program committee who carefully reviewed the papers and provided feedback to the co-chairs. Their contributions and hard work are crucial for ensuring a high-quality technical program. We are also grateful to the CP 2007 Workshop/Tutorial Program Chair, Pedro Meseguer, as well as the CP 2007 Conference Chairs, Laurent Michel and Meinlof Sellmann, for their help in the organization of SymCon 07. Finally, we would like to express our gratitude to the two doctoral students, Lionel Paris and Mohamed Réda Saidi, who built and maintained the website of SymCon 07.

We hope that the present volume is useful for anyone interested in the advances and new trends in Symmetry and Constraint Satisfaction Problems.

August 2007

B. Benhamou, B.Y. Choueiry, and B. Hnich

Organization

Program Chairs

Belaïd Benhamou Université de Provence (Aix-Marseille I), France
Berthe Y. Choueiry University of Nebraska-Lincoln, USA
Brahim Hnich Faculty of Computer Science, Izmir University of Economics, Turkey

Program Committee

Fadi A. Aloul American University of Sharjah, U.A.E.
Gilles Audemard Université d'Artois, France
Rolf Backofen Albert-Ludwigs-Universität, Germany
Pierre Flener Sabanci University, Turkey and Uppsala University, Sweden
Zeynep Kiziltan University of Bologna, Italy
Derek Long University of Strathclyde, Glasgow, UK
Igor Markov University of Michigan, U.S.A.
Pedro Meseguer Universitat Autònoma de Barcelona, Spain
Michela Milano University of Bologna, Italy
Justin Pearson Uppsala University, Sweden
Karen Petrie Oxford University, UK
Steve Prestwich University College Cork, Ireland
Jean-François Puget ILOG, France
Lakhdar Sais Université d'Artois, France
Pierre Seigel University of Provence, Aix-Marseille I, France
Meinolf Sellmann Brown University, U.S.A.
Barbara Smith Cork Constraint Computation Centre, Ireland
Pascal van Hentenryck Brown University, U.S.A.
Toby Walsh University of New South Wales, Australia

Table of Contents

Invited Talk: Without Loss of Generality – Symmetric Reasoning for Resolution Systems	5
<i>S. Szeider</i>	
Symmetry Breaking in Local Search for Unsatisfiability	9
<i>F. Aloul, I. Lynce, and S. Prestwich</i>	
Efficient Symmetry Breaking Predicates for Quantified Boolean Formulas	14
<i>G. Audemard, S. Jabbour, and L. Saïs</i>	
Dynamic Detection and Elimination of Local Symmetry in CSPs	22
<i>B. Benhamou and M.R. Saidi</i>	
A Novel Approach For Detecting Symmetries in CSP Models	30
<i>B. Domoen, M. Garcia de la Banda, C. Mears, and M. Wallace</i>	
Exploiting Symmetry in Multiple Knapsack Problems	39
<i>A.S. Fukunaga</i>	
Minimal Ordering Constraints for Some Families of Variable Symmetries .	47
<i>A. Grayland, I. Miguel, and C. Rouney-Dougal</i>	
A CP Approach to the Balanced Academic Curriculum Problem	56
<i>J.N. Monette, P. Schaus, S. Zampelli, Y. Deville, and P. Dupont</i>	
Substitutability Based Domain Decomposition for Constraint Satisfaction	64
<i>W. Naanaa</i>	
Symmetry-Breaking Formulas for Groups with Bounded Orbit Projections	72
<i>A. Roy</i>	
Breaking Value symmetry	76
<i>T. Walsh</i>	
Symmetry in Constraint Optimization	84
<i>T. Walsh</i>	
Symmetry Breaking in Subgraph Isomorphism	90
<i>S. Zampelli, Y. Deville, M.R. Saidi, and B. Benhamou</i>	
Author Index	97

Without Loss of Generality – Symmetric Reasoning for Resolution Systems

(Extended Abstract for SymCon'07 Invited Talk)

Stefan Szeider

Department of Computer Science
Durham University
Durham, UK
stefan.szeider@durham.ac.uk

Abstract

Krishnamurthy [1985] introduced symmetry rules that make the informal “without loss of generality” reasoning available for resolution-based systems. The homomorphism rules of Szeider [2005] are more powerful variants of Krishnamurthy’s rules and can save in certain cases an exponential number of inference steps over symmetry rules. In this talk we will review the concepts of symmetry and homomorphism rules for resolution-based systems and discuss various questions and results that arise in that context.

1 Resolution

Resolution is a reasoning method that is particularly suited for automated reasoning as it rests on one single rule of inference:

If from a set of clauses one can derive the clauses $C \cup \{x\}$ and $D \cup \{-x\}$ then the *resolution rule* (Res) allows to derive also the clause $C \cup D$.

A *clause* is a set of literals and represents a disjunction; a literal is a variable or a negated variable. The clauses in the given set are considered as *axioms* and can be derived immediately. It is well known that a set of clauses is unsatisfiable if and only if one can derive the empty clause from it using the resolution rule. A derivation of the empty clause is called a *refutation*.

Resolution is fundamental for many automated reasoning systems. Some unsatisfiable sets of clauses are “hard” for resolution: an exponential number of resolution steps is required to derive the empty clause. A famous hard example for resolution is the “pigeon hole clause set” PH_n which encodes (the negation of) the fact that $n + 1$ pigeons do not fit into n holes if each hole can hold at most one pigeon. Using variable $x_{i,j}$ to represent the proposition “pigeon i sits in hole j ”

we can define PH_n as the set of the following clauses: the clauses $\{x_{i,1}, \dots, x_{i,n}\}$ for $1 \leq i \leq n + 1$ (“pigeon i sits in some hole”), and the clauses $\{\neg x_{i,j}, \neg x_{i',j}\}$ for $1 \leq i < i' \leq n + 1, 1 \leq j \leq n$ (“pigeons i and i' cannot sit in the same hole”). Haken [1985] has shown that it requires $2^{\Omega(n)}$ resolution steps to obtain the empty clause from PH_n .

Such exponential lower bounds for resolution imply that satisfiability solvers that are based on the Davis-Putnam-Logmann-Loveland procedure (DPLL) [Davis *et al.*, 1962] need exponential time for these instances, independent of the branching heuristics used (cf. [Coock and Mitchel, 1996]). As demonstrated by Mitchel [1998] exponential resolution lower bounds are also significant for the running time of constraint solvers.

2 Symmetries and Homomorphisms

If we take into account that PH_n is highly symmetric, we can make the following inductive argument: Assume that PH_n is satisfiable. *Without loss of generality*, assume that pigeon $n + 1$ sits in hole n ; thus we set variable $x_{n+1,n}$ to true. This leaves us with PH_{n-1} . Repeating this step several times we obtain PH_1 which is evidently unsatisfiable.

Krishnamurthy [1985] suggested certain *symmetry rules* that formalize this type of reasoning for resolution-based systems. He distinguished between two variants of symmetry rules: a *global symmetry rule* that takes into account symmetries of the given set of axioms as a whole, and a more powerful *local symmetry rule* that takes into account what axioms were actually used for deriving a certain clause. We will describe the rules more detailed below.

Krishnamurthy’s symmetry rules are special cases of *homomorphism rules*; the latter were introduced by Szeider [2005] using the concept of *CNF homomorphism* [Szeider, 2003].

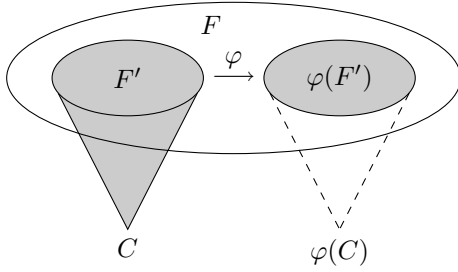


Figure 1: Illustration for the homomorphism rule. The homomorphism rule allows to obtain $\varphi(C)$ in one single step; $\varphi(C)$ could be derived from $\varphi(F')$ but this derivation (indicated by dashed lines) can be omitted.

Let F and G be sets of clauses. A *homomorphism from F to G* is a mapping φ from the literals of F to the literals of G such that the following conditions hold ($\varphi(C)$ denotes the set $\{\varphi(x) : x \in C\}$).

1. $\varphi(\neg x) = \neg\varphi(x)$ for all variables x of F (“ φ preserves complements”);
2. $\varphi(C) \in G$ for all clauses $C \in F$ (“ φ preserves clauses”).

For example, let $F = \{\{x\}, \{\neg y\}, \{\neg x, y\}\}$ and $G = \{\{z\}, \{\neg z\}\}$. The mapping φ with $\varphi(\neg x) = \varphi(y) = z$, and $\varphi(x) = \varphi(\neg y) = \neg z$, is a homomorphism from F to G . Note that $\varphi(\{\neg x, y\}) = \{z\}$, thus a homomorphism can “shrink” clauses.

If φ is a homomorphism from F to G and $F' \subseteq F$, then we write $\varphi(F') = \{\varphi(C) : C \in F'\}$. A *symmetry* φ is an injective homomorphisms; *i.e.*, a homomorphism where $\varphi(x) \neq \varphi(y)$ whenever $x \neq y$. A homomorphism is *nontrivial* if it differs from the identity mapping.

It is easy to see that if F is unsatisfiable and there exists a homomorphism from F to G , then G is unsatisfiable as well [Szeider, 2003].

Now we can formulate the following rule (see Figure 1 for an illustration).

Give a set F of clauses. Assume that we can derive a clause C from a subset $F' \subseteq F$. If φ is a homomorphism from F' to F , then the *local homomorphism rule* (LHR) allows to derive the clause $\varphi(C)$.

The soundness of this rule can be seen as follows: we consider a sequence C_1, \dots, C_n of clauses, with $C_n = C$, that corresponds to a derivation of C from F' . Applying φ we obtain the sequence $\varphi(C_1), \dots, \varphi(C_n)$, $\varphi(C_n) = \varphi(C)$. It is not difficult to see that the latter sequence contains a valid derivation of $\varphi(C)$ from the clauses in $\varphi(F')$. However, $\varphi(F') \subseteq F$, hence there exists a derivation of $\varphi(C)$ from F .

The argument outlined above suggests to consider refutations that use the homomorphism rule as *succinct representations* of resolution refutations.

Restricting the local homomorphism rule in various ways leads to the following less general rules:

- The *global homomorphism rule* (GHR) arises from the local one by requesting that φ is an *endomorphism* of F , that is, φ is a homomorphism from F to F .
- Krishnamurthy’s *local symmetry rule* (LSR) arises from the local homomorphism rule by requesting that φ is a *monomorphism* from F' to F , *i.e.*, $\varphi(x) \neq \varphi(y)$ whenever $x \neq y$.
- Krishnamurthy’s *global symmetry rule* (GSR) arises from the local homomorphism rule by requesting that φ is an *automorphism* of F , *i.e.*, φ is both a monomorphism as well as an endomorphism of F .

Thus one can consider the following five resolution-based systems: Res, Res+GSR, Res+LSR, Res+GHR, and Res+LHR (*e.g.*, the system Res+GSR uses the resolution rule together with the global symmetry rule).

3 Comparison of the systems

We say that an unsatisfiable set F of clauses is *easy* for system A if one can derive the empty clause from F using a polynomial number of inference steps of system A . If F requires an exponential number of steps, we say that F is *hard* for A . How are the above resolution-based systems related to each other? Is system A *strictly stronger* than system B in the sense that every instance that is easy for B is also easy for A , but some instances are easy for A and hard for B ? Or are two systems A and B *incomparable* in the sense that there are instances that are easy for A and hard for B and instances where the converse prevails?

For the five systems under consideration the following is known (the first two statements have been established in [Urquhart, 1999] and [Arai and Urquhart, 2000], respectively; statements 3-6 have been established in [Szeider, 2005]).

1. Res+GSR is strictly stronger than Res.
2. Res+LSR is strictly stronger than Res+GSR.
3. Res+GHR is strictly stronger than Res+GSR.
4. Res+LHR is strictly stronger than Res+GHR.
5. Res+LHR is strictly stronger than Res+LSR.
6. Res+LSR and Res+GHR are incomparable.

The diagram in Figure 2 illustrates the relationships between the systems.

In the following let us briefly review the proof ideas used for establishing the above comparison results. Recall from above that the pigeon hole clause sets are hard instances for Res. For Res+GSR however, PH_n has a short refutation as the following inductive argument, given by Urquhart [1999], shows.

Evidently PH_1 has a short proof. Now consider PH_n for $n > 1$. Using the resolution rule we can derive from $\{\neg x_{n+1,n}, \neg x_{1,n}\} \in \text{PH}_n$ and $\{x_{1,1}, \dots, x_{1,n-1}, x_{1,n}\} \in \text{PH}_n$ the clause $\{x_{1,1}, \dots, x_{1,n-1}, \neg x_{n+1,n}\}$. Similarly we can derive all the clauses $\{x_{i,1}, \dots, x_{i,n-1}, \neg x_{n+1,n}\}$ for

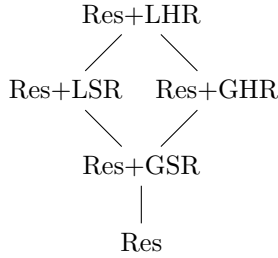


Figure 2: The relative strength of the considered reasoning systems.

$i = 2, \dots, n$. Thus we have the clauses of PH_{n-1} except that some contain additionally the literal $\neg x_{n+1,n}$. By induction hypothesis there is a short Res+GSR-derivation of the empty clause from PH_{n-1} . This gives rise to a short Res+GSR-derivation of the unit clause $\{\neg x_{n+1,n}\}$ from PH_n . Evidently there is an automorphism φ of PH_n such that $\varphi(x_{n+1,n}) = x_{n+1,n-1}$. Hence the global symmetry rule allows to derive the clause $\varphi(\{\neg x_{n+1,n}\}) = \{\neg x_{n+1,n-1}\}$. Similarly, we can use the global symmetry rule to derive all the clauses $\{\neg x_{n+1,n-2}\}, \dots, \{\neg x_{n+1,1}\}$. Resolving all these unit clauses with the clause $\{x_{n+1,1}, \dots, x_{n+1,n}\} \in \text{PH}_n$ yields the empty clause.

Thus PH_n is hard for Res and easy for Res+GSR.

For the further comparison results one uses the following strategy. One “disguises” the clauses of PH_n in a certain way, making the symmetry/homomorphism rule under consideration inapplicable. The disguise is formed in such a way that the original clauses of PH_n can be obtained from the disguised clauses by means of a few resolution steps. This property is used to show that the disguised PH_n remains hard for Res.

The simplest disguising technique is to replace a clauses C by the clause $C \cup \{x\}$ and to add the unit clause $\{\neg x\}$; the original clauses C can be res-established by resolving $C \cup \{x\}$ with $\{\neg x\}$. Applying this technique one can disguise PH_n in such a way that every clause has a unique size (except for unit clauses), making the global symmetry rule inapplicable.

The disguise can be designed in such a way that various symmetry/homomorphism rules cannot be applied any more. If a certain symmetry/homomorphism rule X is not applicable for the disguised PH_n , then we have an instance that is hard for the system Res+X. If we can design the disguise in such a way that rule X can still be applied but another more restricted rule Y cannot, we have shown that system Res+X is strictly stronger than system Res+Y.

There exists a disguise that makes the local homomorphism rule, the strongest of the rules considered, inapplicable [Szeider, 2005]. Hence there are hard instances for the system Res+LHR.

4 Algorithmic Aspects

In our above considerations we have asked for the *existence* of short refutations; we have not worried about the question of how a short refutation can actually be found. This “proof complexity” perspective has the advantage that hardness results apply to all possible algorithms that search for refutations: Even if we had an ideal heuristics that always tells us the best next move in our derivation, the running time of such an algorithm would still be exponential if the instance under consideration has no short refutation.

An algorithm that searches for proofs of one of the four systems considered in Section 2, one needs to find symmetries/homomorphisms in order to apply the respective rule.

The following decision problems arise in the context of applying the various symmetry/homomorphism rules.

- P1 Has a given set F of clauses a nontrivial automorphism?
- P2 Given a set F of clauses and a subset F' of F ; is there a nontrivial monomorphism from F' to F ?
- P3 Has a given set F of clauses an endomorphism that is not an automorphism?
- P4 Given a set F of clauses and a subset F' of F ; is there a nontrivial homomorphism from F' to F ?

Problems P1, P2, P3, and P4 are associated with applications of the rules GSR, LSR, GHR, and LHR, respectively. All problems belong to NP as one can easily verify whether a given homomorphism has the required property. Except for problem P1, all other problems are known to be NP-hard (NP-hardness of P2 is shown in [Boy de la Tour and Demri, 1995]; NP-hardness of problems P3 and P4 is shown in [Szeider, 2005]). As observed by Boy de la Tour and Demri [1995], problem P1 is polynomial-time equivalent to the graph automorphism problem GA, which asks whether a given graph has a nontrivial automorphism. GA is a natural problem in NP that appears to be not solvable in polynomial time. However, it is believed that GA is not NP-complete since otherwise the Polynomial Hierarchy would collapse to its second level [Schöning, 1988]. Thus the weakest of the rules, GSR, is apparently computationally less costly than the other rules. Moreover, one can compute first the automorphism group at a preprocessing stage (cf. the discussion in [Boy de la Tour and Demri, 1995]). Some experimental results on the algorithmic use of a restricted version of GSR have been reported by Benhamou and Saïs [1994].

In summary, the worst-case complexities of the problems associated with applications of the symmetry/homomorphism rules, except for the weakest rule GSR, are not very encouraging, and a direct use of the rules in automated deduction seems difficult. However, it is conceivable that in certain situations one has additional information on the given instance that allows an efficient computation of homomorphisms and symmetries. This aspect seems to be of particular relevance if

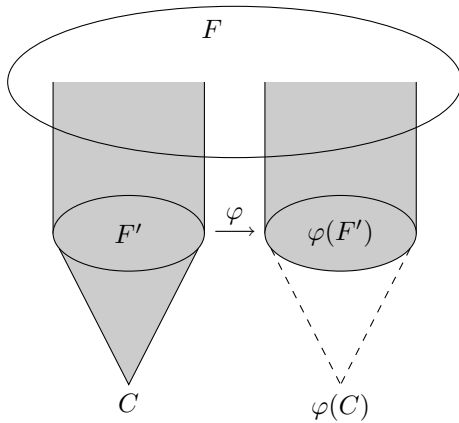


Figure 3: Illustration for the dynamic homomorphism rule.

the instance encodes a problem from a different domain, such as constraint satisfaction.

5 Dynamic Rules

The homomorphisms/symmetries of the above rules act on the clauses of the input set F , the axioms. However, one could apply first some inference steps in order to derive some clauses, and then apply the homomorphism/symmetry rules using homomorphisms/symmetries that act on the derived clauses. These considerations lead to the following definition (see Figure 3 for an illustration).

Given a set F of clauses. Assume that we can derive from F the sets F' and F'' , and assume that, in turn, we can derive from F' the clause C . If φ is a homomorphism from F' to F'' , then the *dynamic homomorphism rule* (DHR) allows to derive the clause $\varphi(C)$.

The dynamic homomorphism rule is discussed in [Pitassi, 2003] and [Szeider, 2005].

The soundness of this rule can be established similarly as the soundness of the local homomorphism rule. In this more general setting, the above approach for finding hard instances does not work anymore: one can always derive the original PH_n from its disguise; once PH_n is derived, one can obtain the empty clause via a polynomial number of inference steps using resolution and symmetry rules that act on the derived clauses.

Currently we do not know any hard instances for the system $\text{Res}+\text{DHR}$.

References

[Arai and Urquhart, 2000] Noriko H. Arai and Alasdair Urquhart. Local symmetries in propositional

logic. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods (Proc. TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Computer Science*, pages 40–51. Springer Verlag, 2000.

[Benhamou and Sais, 1994] Belaid Benhamou and Lakhdar Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12:89–102, 1994.

[Boy de la Tour and Demri, 1995] Thierry Boy de la Tour and Stéphane Demri. On the complexity of extending ground resolution with symmetry rules. In *IJCAI-95, Vol. 1, 2 (Montreal, PQ, 1995)*, pages 289–295. Morgan Kaufmann, San Francisco, CA, 1995.

[Cook and Mitchel, 1996] Stephen A. Cook and David G. Mitchell, Finding hard instances of the satisfiability problem: a survey, In *Satisfiability problem: theory and applications* (Piscataway, NJ, 1996), pages 1–17, AMS, 1997.

[Davis et al., 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Comm. ACM*, 5:394–397, 1962.

[Haken, 1985] Armin Haken. The intractability of resolution. *Theoret. Comput. Sci.*, 39:297–308, 1985.

[Krishnamurthy, 1985] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.

[Mitchell, 1998] David G. Mitchell. Hard problems for CSP algorithms. In *15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI'98/IAAI'98)*, pages 398–405. The MIT Press, 1998.

[Pitassi, 2003] Toniann Pitassi. Theory open problems session. Technical report, Department of Computer Science, University of Toronto, June 2003. <http://www.cs.toronto.edu/~toni/TheoryProblems/open.ps>.

[Schöning, 1988] Uwe Schöning. Graph isomorphism is in the low hierarchy. *J. of Computer and System Sciences*, 37:312–323, 1988.

[Szeider, 2003] Stefan Szeider. Homomorphisms of conjunctive normal forms. *Discr. Appl. Math.*, 130(2):351–365, 2003.

[Szeider, 2005] Stefan Szeider. The complexity of resolution with generalized symmetry rules. *Theory Comput. Syst.*, 38(2):171–188, 2005.

[Urquhart, 1999] Alasdair Urquhart. The symmetry rule in propositional logic. *Discr. Appl. Math.*, 96/97:177–193, 1999.

Symmetry Breaking in Local Search for Unsatisfiability

Fadi Aloul

Dept of Computer Engineering,
American University of Sharjah,
UAE
faloul@aus.edu

Inês Lynce

IST/INESC-ID,
Technical University
of Lisbon, Portugal

ines@sat.inesc-id.pt

Steven Prestwich

Cork Constraint Computation
Centre, Dept of Computer Science,
University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract

Symmetry breaking can greatly speed up the search for solutions and proofs of unsatisfiability, but has been shown to have a bad effect on local search for solutions. Recently a new form of local search has been used to prove unsatisfiability of SAT problems, based on randomised resolution with greedy heuristics. An interesting question is: does symmetry breaking speed up this form of local search? We present experimental evidence that it does, making randomised refutation a promising new application of symmetry breaking.

1 Introduction

Complete SAT algorithms may be based on resolution or backtracking. Resolution provides a complete proof system by refutation [Robinson, 1965]. The first resolution algorithm was the Davis-Putnam (DP) procedure [Davis and Putnam, 1960] which was then modified to the Davis-Putnam-Logemann-Loveland (DPLL) backtracking algorithm [Davis *et al.*, 1962]. Because of its high space complexity, resolution is often seen as impractical for real-world problems, but there are problems on which general resolution proofs are exponentially smaller than DPLL proofs [Ben-Sasson *et al.*, 2004]. Incomplete SAT algorithms are usually based on local search following early work by [Gu, 1992; Selman *et al.*, 1992]. On some large satisfiable problems, local search finds a solution much more quickly than complete algorithms. Genetic and other evolutionary algorithms have also been applied to SAT but do not yet rival local search. A new form of local search was recently described that is able to prove *unsatisfiability*: the RANGER [Prestwich and Lynce, 2006] and GUNSAT [Audemard and Simon, 2007] algorithms apply resolution to SAT instances in a randomised way, using greedy heuristics and other techniques to speed up the search, in the hope of deriving the empty clause.

Symmetry-breaking has proved to be very effective when combined with complete solvers, by reducing the size of the search space. Probably the simplest and most popular approach is to add constraints to the problem formulation, so that each equivalence class of solutions to the original problem corresponds to a single solution in the new problem. A formal framework for this approach is given

in [Puget, 1993]. In constraint programming, symmetry breaking is usually applied manually by the modeller in an instance-independent way, whereas in SAT it is usually applied in an automated, instance-dependent way via generic tools that detect graph automorphisms [Aloul *et al.*, 2003; Crawford *et al.*, 1996]. On some problems the instance-dependent approach is best [Ramani *et al.*, 2006] while on others the opposite holds [Lynce and Marques-Silva, 2007]. Symmetry breaking can also be used to reduce the size of a resolution proof, for example short inductionless proofs of the pigeon-hole principle can be constructed using symmetry [Krishnamurthy, 1985].

Symmetry breaking has also been used in genetic algorithms (though not for SAT to the best of our knowledge). A symmetric optimisation problem has multiple optimum solutions that are symmetrically equivalent, and applying recombination to them may yield offspring with very poor fitness. Symmetry breaking in this context involves designing more complex genetic operators and problem models [Galinier and Hao, 1999], or using clustering techniques [Pelikan and Goldberg, 2000]. However, the use of symmetry-breaking constraints seems to have a bad effect on local search (randomised, non-population-based) algorithms [Prestwich, 2003]. This is true even if we ignore any runtime overheads due to symmetry breaking constraints, and measure only search steps. The reasons for this phenomenon are not completely understood, but detailed experiments show that symmetry breaking constraints transform symmetric solutions into deep local minima, thus decreasing the solution density and increasing the number of local minima, and also reduce the relative sizes of basins of attraction of global minima [Prestwich and Roli, 2005].

Given the above background, what effect should we expect symmetry breaking clauses to have on local search for unsatisfiability as in RANGER and GUNSAT? On one hand, symmetry breaking clauses usually have a bad effect on local search; on the other hand, we might expect them to speed up proof of unsatisfiability even if that proof is generated non-systematically. This paper investigates this question: Section 2 provides background on this new class of local search algorithms, Section 3 reports the results of our experiments, and Section 4 concludes the paper.

2 Local search for unsatisfiability

Local and backtrack search have complementary strengths and weaknesses. Local search has superior scalability on many large problems, but it cannot (in its usual form) prove unsatisfiability. Backtrack search and resolution-based algorithms are (usually) complete, and backtrack search’s use of unit propagation, clause learning, dedicated data structures and other methods enables it to outperform local search on some highly-structured problems. This complementarity has inspired research on hybrid approaches such as the use of unit propagation in local search, and more flexible backtracking strategies.

An interesting question is: can local search be applied to *unsatisfiable* problems? Such a method might be able to refute (prove unsatisfiable) SAT problems that defy complete algorithms. The first such algorithm that we know of is RANGER [Prestwich and Lynce, 2006], which explores a space of multisets of resolvents using general resolution, and aims to derive the empty clause non-systematically but greedily. It will eventually refute any unsatisfiable instance while using only bounded memory (by exploiting a recent theoretical result of [Esteban and Torán, 2001]). It can refute some problems more quickly than current DPLL and systematic resolution algorithms, though on most benchmarks it is currently uncompetitive.

The RANGER architecture is shown in Figure 1. It has six parameters: the formula ϕ , three probabilities p_i, p_t, p_g , the width w and the size k of the formula ϕ_i . RANGER begins with any sub-multiset $\phi_1 \subseteq \phi$ (we interpret ϕ, ϕ_i as multisets of clauses). It then performs iterations i , each either replacing a ϕ_i clause by a ϕ clause (with probability p_i) or resolving two ϕ_i clauses and placing the result r into ϕ_i . In the latter case, if r is tautologous or contains more than w literals then it is discarded and $\phi_{i+1} = \phi_i$. Otherwise a ϕ_i clause must be removed to make room for r : either (with probability p_g) the removed clause is the longer of the two parents of r (breaking ties randomly), or it is randomly chosen. In the former case, if r is longer than the parent then r is discarded and $\phi_{i+1} = \phi_i$. At the end of the iteration, any satisfiability-preserving transformation may (with probability p_t) be applied to ϕ, ϕ_{i+1} or both. If the empty clause has been derived then the algorithm terminates with the message “unsatisfiable”. Otherwise the algorithm might not terminate, but a time-out condition (omitted here for brevity) may be added.

Local search algorithms usually use *greedy* local moves that reduce the value of an objective function, and *plateau traversal* moves that leave it unchanged. However, they must also allow non-greedy moves in order to escape from local minima. This is often controlled by a parameter known as *noise* (or *temperature* in simulated annealing). RANGER’s goal is to derive the empty clause, and a necessary condition for this to occur is that ϕ_i contains at least some small clauses. We call a local move *greedy* if it does not increase the number of literals in ϕ_i . This is guaranteed on line 10, so increasing p_g increases the greediness of the search, reducing the proliferation of large resolvents.

RANGER has a useful convergence property: for any unsatisfiable SAT problem with n variables and m clauses,

```

1 RANGER( $\phi, p_i, p_t, p_g, w, k$ ):
2    $i \leftarrow 1$  and  $\phi_1 \leftarrow \{\text{any } k \text{ clauses from } \phi\}$ 
3   while  $\phi_i$  does not contain the empty clause
4     with probability  $p_i$ 
5       replace a random  $\phi_i$  clause by a
        random  $\phi$  clause
6     otherwise
7       resolve random  $\phi_i$  clauses  $c, c'$  giving  $r$ 
8       if  $r$  is non-tautologous and  $|r| \leq w$ 
9         with probability  $p_g$ 
10        if  $|r| \leq \max(|c|, |c'|)$  replace the
          longer of  $c, c'$  by  $r$ 
11        otherwise
12        replace a random  $\phi_i$  clause by  $r$ 
13        with probability  $p_t$ 
14        apply any satisfiability-preserving
          transformation to  $\phi, \phi_i$ 
15    $i \leftarrow i + 1$  and  $\phi_{i+1} \leftarrow \{\text{the new formula}\}$ 
16   return UNSATISFIABLE

```

Figure 1: The RANGER architecture

RANGER finds a refutation if $p_i > 0, p_i, p_t, p_g < 1, w = n$ and $k \geq n + 1$ (for a proof see [Prestwich and Lynce, 2006]). The space complexity of RANGER is $O(n + m + kw)$. To guarantee convergence we require $w = n$ and $k \geq n + 1$ so the complexity becomes at least $O(m + n^2)$. In practice we may require k to be several times larger, but a smaller value of w is often sufficient.

Lines 13–14 provide an opportunity to apply helpful satisfiability-preserving transformations to ϕ or ϕ_i or both (if we do not aim for a pure resolution refutation). We apply the subsumption and pure literal rules in several ways. Using ϕ_i clauses to transform ϕ , a feature we shall call *feedback*, preserves useful improvements for the rest of the search. (We believe that for these particular transformations we can set $p_t = 1$ without losing completeness, but we defer the proof until a later paper.) Note that if ϕ is reduced then this will soon be reflected in the ϕ_i via line 5 of the algorithm.

A related algorithm is GUNSAT [Audemard and Simon, 2007] which has a similar architecture but interesting differences. For example, whereas RANGER aims for a high rate of rather unintelligent local moves, GUNSAT takes longer to make more intelligent moves based on a more complex objective function. GUNSAT also uses extended resolution while RANGER uses general resolution. The two algorithms have not yet been compared empirically.

3 Experiments

We now evaluate the effects of symmetry breaking on RANGER. The results are shown in Figure 2. All results are medians over 10 runs with a cutoff time of 1000 seconds. #Steps denotes the number of RANGER iterations, #Time the CPU time taken, #V and #C the number of variables and clauses (respectively) in the SAT instances. Experiments were performed on an Intel Xeon 3 GHz with 4GB RAM running Linux. The RANGER implementation is as described in [Prestwich and Lynce, 2006] with parameter settings $k=10V, w=V, p_i=0.1, p_t=0.9$ and $p_g=0.95$. The prob-

lem sets are as follows:

- `Chnl` problems represent large unsatisfiable instances that model the routing of X wires in the N channels of field-programmable integrated circuits. Assuming that each channel accepts up to one wire, since $X > N$ the instances are unsatisfiable [Aloul *et al.*, 2003].
- `Hole` represent the famous pigeon hole instances, where the goal is to place X pigeons in N holes.¹ Again each hole can hold up to one pigeon, and since $X > N$ the instances are unsatisfiable.
- `Pipe` represent difficult unsatisfiable instances that model the functional correctness requirements of modern out-of-order microprocessor CPUs. The instances were generated by Miroslav Velev [Velev and Bryant, 2001].
- `X` encodes verification problems of two exclusive-or chains. The instances were generated by Lintao Zhang and Sharad Malik.²
- `Urq` are unsatisfiable randomized instances based on expander graphs [Urquhart, 1987].
- The biological instances `b2ar`, `ace`, `non-uniform` and `hapmap` come from the haplotype inference problem. Given a set of genotypes, described using a string alphabet $\{0, 1, 2\}$ the main goal is to identify the minimum number of haplotypes, which are described over with a string over the alphabet $\{0, 1\}$ such that each genotype is explained by a pair of haplotypes. The CNF encoding for this problem is described in [Lynce and Marques-Silva, 2006] as well as the problem instances we have used.

Symmetry was broken by the Shatter system [Aloul *et al.*, 2006; 2003]. Symmetry breaking took only a small fraction of a second, which is not included in our results.

The FPGA and hole instances clearly show a huge improvement due to symmetry breaking. The others (of which five are denoted (1) . . . (5) for space reasons) were unrefuted within the time limit, with or without symmetry breaking. Thus we have no evidence that symmetry breaking harms RANGER performance, but some evidence that it improves performance. A possible explanation is that the symmetry breaking clauses allow smaller refutations, which are easier to discover than large refutations.

However, it is interesting to note that the instances that RANGER could not refute contain few new variables when adding symmetry breaking (they are not required to model the *phase shift symmetries* of most of these instances), while those it did refute contain many new variables. This might indicate that the improvement was not completely due to symmetry breaking, but also to the use of additional variables, which might have a similar effect to the auxiliary variables introduced in *extended resolution*. (Extended resolution allows the definition of new SAT variables via the *extension rule*

¹DIMACS Challenge benchmarks, 1996

<ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf>

²SAT 2002 Competition

<http://www.satlive.org/SATCompetition/submittedbenchs.html>

without symmetry breaking					
instance	#V	#C	#Lit	#Steps	#Time
chnl10_11	220	1122	2420	n/a	>1000
chnl10_12	240	1344	2880	n/a	>1000
chnl10_13	260	1586	3380	n/a	>1000
chnl11_12	264	1476	3168	n/a	>1000
chnl11_13	286	1742	3718	n/a	>1000
chnl11_20	440	4220	8800	n/a	>1000
hole7	56	204	448	n/a	>1000
hole8	72	297	648	n/a	>1000
hole9	90	415	900	n/a	>1000
hole10	110	561	1210	n/a	>1000
hole11	132	738	1584	n/a	>1000
hole12	156	949	2028	n/a	>1000
Urq3_5	46	470	2912	n/a	>1000
x1.1_16	46	122	364	n/a	>1000
2pipe	892	6695	18637	n/a	>1000
(1)	776	3725	10045	n/a	>1000
(2)	110	428	992	n/a	>1000
(3)	187	643	1584	n/a	>1000
(4)	156	522	1141	n/a	>1000
(5)	223	880	2363	n/a	>1000

with symmetry breaking					
instance	#V	#C	#Lit	#Steps	#Time
chnl10_11	728	3077	9103	2700218	11.605
chnl10_12	796	3487	10213	2999725	18.605
chnl10_13	864	3917	11363	3326896	27.67
chnl11_12	878	3847	11291	4417899	33.85
chnl11_13	953	4321	12561	5155214	50.905
chnl11_20	1478	8255	22683	948902	1.55
hole7	153	567	1663	241347	0.35
hole8	199	776	2261	352256	0.535
hole9	251	1026	2967	626528	1.015
hole10	309	1320	3787	948902	1.55
hole11	373	1661	4727	1153560	2.1
hole12	443	2052	5793	1784522	3.825
Urq3_5	46	500	2941	n/a	>1000
x1.1_16	48	142	385	n/a	>1000
2pipe	1246	8137	23571	n/a	>1000
(1)	780	3746	10073	n/a	>1000
(2)	120	449	1022	n/a	>1000
(3)	205	694	1670	n/a	>1000
(4)	160	531	1153	n/a	>1000
(5)	223	913	2395	n/a	>1000

Key:

- (1) `1dlx_c_mc_ex_bp_f`
- (2) `bio-b2ar-simp-b2ar_5.01`
- (3) `bio-ace-simp-ace_5.07`
- (4) `non-uniform-simp-nonunif-10_50.06`
- (5) `hapmap-simp-test_chr21_HCB_30`

Figure 2: Experiments on RANGER with and without symmetry breaking

[Tseitin, 1983]. It can lead to exponentially smaller proofs, but is used even less than general resolution because there are no known heuristics for generating the new variables.)

We hope to resolve this issue by further experimentation in future work, either by finding instances without auxiliary variables for which symmetry breaking helps randomised refutation, or by testing the effects of new auxiliary variables on the unrefuted instances. If the explanation turns out to be a form of extended resolution then we may enhance RANGER from general resolution to extended resolution, along the lines of GUNSAT.

4 Conclusion

This work is only at a preliminary stage, but already shows the promise of symmetry breaking in randomised refutation. Local search for unsatisfiability appears to be an exception to the rule that “symmetry breaking is bad for local search”.

In retrospect this is perhaps unsurprising: if symmetry breaking allows smaller refutations then these may be easier to find by *any* resolution algorithm, whether systematic or randomised. Moreover, we have not actually applied symmetry breaking to the space explored by the local search algorithm: to do this we would have to restrict the search so that it excludes refutations that are symmetric in some sense to other refutations. This might indeed harm local search performance.

In fact the usual arguments based on solution density and global basins of attraction do not hold when refuting an UNSAT problem. Adding symmetry breaking clauses to a SAT problem increases the number of possible resolution refutations, so there are more search states from which greedily applying resolution leads directly to the empty clause. In other words, in this context symmetry breaking *increases* the size of the basin of attraction of each solution (defined here as a search state containing the empty clause).

Acknowledgements

This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075, and partially supported by Fundação para a Ciência e Tecnologia under research projects POSC/EIA/61852/2004 and POSI/SRI/41926/01.

References

- [Aloul *et al.*, 2003] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Transactions on Computer Aided Design*, 22(9):1117–1137, 2003.
- [Aloul *et al.*, 2006] F. Aloul, K. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006.
- [Audemard and Simon, 2007] G. Audemard and L. Simon. Gunsat: A greedy local search algorithm for unsatisfiability. In *20th International Joint Conference on Artificial Intelligence*, 2007. Poster.
- [Ben-Sasson *et al.*, 2004] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of treelike and general resolution. *Combinatorica*, 24(4):585–603, 2004.
- [Crawford *et al.*, 1996] M. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *5th International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159, 1996.
- [Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.
- [Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
- [Esteban and Torán, 2001] J. L. Esteban and J. Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001.
- [Galinier and Hao, 1999] P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
- [Gu, 1992] Jun Gu. Efficient local search for very large-scale satisfiability problems. *Sigart Bulletin*, 3(1):8–12, 1992.
- [Krishnamurthy, 1985] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22:327–337, 1985.
- [Lynce and Marques-Silva, 2006] I. Lynce and J. P. Marques-Silva. Sat in bioinformatics: Making the case with haplotype inference. In *9th International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 136–141. Springer, 2006.
- [Lynce and Marques-Silva, 2007] I. Lynce and J. P. Marques-Silva. Breaking symmetries in sat matrix models. In *10th International Conference on Theory and Applications of Satisfiability Testing*, volume 4501 of *Lecture Notes in Computer Science*, pages 22–27. Springer, 2007.
- [Pelikan and Goldberg, 2000] M. Pelikan and D. E. Goldberg. Genetic algorithms, clustering, and the breaking of symmetry. In *6th International Conference on Parallel Problem Solving from Nature*, 2000.
- [Prestwich and Lynce, 2006] S. D. Prestwich and I. Lynce. Local search for unsatisfiability. In *9th International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2006.
- [Prestwich and Roli, 2005] S. D. Prestwich and A. Roli. Symmetry breaking and local search spaces. In *2nd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3524 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2005.
- [Prestwich, 2003] S. D. Prestwich. Negative effects of modeling techniques on search performance. *Annals of Operations Research*, 118:137–150, 2003.
- [Puget, 1993] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *International Symposium on Methodologies for Intelligent Systems*, volume 689 of *Lecture Notes in Computer Science*, pages

- 350–361. Springer-Verlag, 1993. J. Komorowski, Z. W. Ras (eds.), *Methodologies for Intelligent Systems*.
- [Ramani *et al.*, 2006] A. Ramani, I. L. Markov, K. A. Sakallah, and F. A. Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26:289–322, 2006.
- [Robinson, 1965] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.
- [Selman *et al.*, 1992] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *10th National Conference on Artificial Intelligence*, pages 440–446. MIT Press, 1992.
- [Tseitin, 1983] G. Tseitin. On the complexity of derivation in propositional calculus. *Automation of Reasoning: Classical Papers in Computational Logic*, 2:466–483, 1983.
- [Urquhart, 1987] A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.
- [Velev and Bryant, 2001] M. N. Velev and R. E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *Design Automation Conference*, pages 226–231, 2001.

Efficient Symmetry Breaking Predicates for Quantified Boolean Formulae

Gilles Audemard and Said Jabbour and Lakhdar Sais

CRIL - Université d'Artois - Lens, France

{audemard,jabbour,sais}@cril.univ-artois.fr

Abstract

Many reasoning task and combinatorial problems exhibit symmetries. Exploiting such symmetries has been proved useful in reducing the search space. In this paper, a formal approach for symmetry breaking in quantified boolean formula is proposed. It make use of a new efficient technique for encoding the additional symmetry predicates in prenex clausal form. The new asymmetric formula is equivalent to the original one with respect to the validity. Experimental evaluation shows significant improvements over a wide range of QBF instances.

1 Introduction

Solving Quantified Boolean Formulae (QBF) has become an attractive and important research area over the last years. Such increasing interest might be related to different factors, including the fact that many important artificial intelligence (AI) problems (planning, non monotonic reasoning, formal verification, etc.) can be reduced to QBF which is considered as the canonical problem of the PSPACE complexity class. Another important reason comes from the recent impressive progress achieved in the practical resolution of the satisfiability problem. Many solvers for QBFs have been proposed recently (e.g. [Giunchiglia *et al.*, 2001b; Letz, 2002; Zhang and Malik, 2002; Biere, 2004; Benedetti, 2005a]), most of them are obtained by extending satisfiability results. This is not surprising, since QBFs is a natural extension of the satisfiability problem (deciding whether a boolean formula in conjunctive normal form is satisfiable or not), where the variables are universally or existentially quantified.

It is well known that exploiting and removing symmetries is an important task to deal with the intractability of many combinatorial problems such as constraint satisfaction problems (CSP) [Cohen *et al.*, 2005] and satisfiability of boolean formula (SAT) [Benhamou and Sais, 1994; Crawford, 1992; Aloul *et al.*, 2002]. One of the most popular approach for breaking symmetry consists in adding new constraint, called Symmetry Breaking Predicates (SBP), to the original formula or to the

constraint network. This approach independently proposed by Crawford [Crawford, 1992] and Puget [Puget, 1993] is solver independent and is performed in a pre-processing step.

Extending this approach to QBFs is a very challenging task. The problem rise from the presence of universal quantifier, while in SAT or CSP all the variables can be seen as existentially quantified. Adding the classical symmetry breaking predicates to the QBF formula do not preserve the validity of the original formula. Recently, two promising approaches for breaking symmetry in QBF have been proposed [Audemard *et al.*, 2004; 2007]. In [Audemard *et al.*, 2004], the symmetry breaking predicates are not added to the original formula leading to a hybrid QBF-SAT formula. A QBF solver handling such hybrid formula is then proposed. Consequently, this first approach is clearly solver dependent. More recently, another technique was proposed in [Audemard *et al.*, 2007]. The authors proposed an original preprocessing technique, called `sbp4qbf`, which extends the symmetry breaking approach introduced by [Crawford, 1992] and improved by [Aloul *et al.*, 2002] for the satisfiability problem. It consists in rewriting the quantifier prefix and adding new constraints in order to deal with the universal quantifiers.

In this paper, we a new formal approach for QBF symmetry breaking is proposed. Furthermore, we introduce a new and efficient way for encoding SBPs. Our approach allows us to propagate redundant existential literals and to removes universal symmetrical literals by considering some interpretations as models of the original formula.

The paper is organized as follows. After some preliminary definitions on quantified boolean formulae, symmetry framework in QBFs is presented. In section 3, we formalize symmetry breaking predicates for QBF and propose an efficient encoding. In section 4, an experimental evaluation of our approach is described. An experimental evaluation (section 4) and comparison (section 5) with the approach proposed in [Audemard *et al.*, 2007] is presented before concluding.

2 Technical Background

2.1 Quantified boolean formulae

Let \mathcal{P} be a finite set of propositional variables. Then, $\mathcal{L}_{\mathcal{P}}$ is the language of quantified Boolean formulae built over \mathcal{P} using ordinary boolean formulae (including propositional constants \top and \perp) plus the additional quantification (\exists and \forall) over propositional variables.

In this paper, we consider quantified boolean formula Φ in the prenex clausal form $\Phi = Q_1 X_1, \dots, Q_m X_m \psi$ (in short $QX\psi$, QX is called the prefix and ψ the matrix) where $Q_i \in \{\exists, \forall\}$, X_1, \dots, X_m are disjoint sets of variables and ψ a boolean formula in conjunctive normal form. Consecutive variables with the same quantifier are grouped. Without loss of generality, we can consider Q_m as the existential quantifier. We define $Var(\Phi) = \bigcup_{i \in \{1, \dots, k\}} X_i$ the set of variables of Φ . A literal is the occurrence of propositional variable in either positive (l) or negative form ($\neg l$). $Lit(\Phi) = \bigcup_{i \in \{1, \dots, k\}} Lit(X_i)$ the set of complete literals of Φ , where $Lit(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$. Finally, we note ψ_X , the propositional formula ψ simplified by the assignation of all literals of X .

The semantic of QBF is associated with the notion of policy. A policy is a set of propositional models which respect some conditions. For a QBF with n universal variables, a policy contains 2^n propositional models. For more details, please refer to [Benedetti, 2005b] or [Coste-Marquis *et al.*, 2006].

2.2 Symmetries in Quantified Boolean Formulae

Let $\Phi = Q_1 X_1, \dots, Q_m X_m \psi$ be a QBF and σ a permutation over the literals of Φ i.e. $\sigma : Lit(\Phi) \mapsto Lit(\Phi)$. The permutation σ on Φ is then defined as follows: $\sigma(\Phi) = Q_1 \sigma(X_1), \dots, Q_m \sigma(X_m) \sigma(\psi)$. For example, if ψ is in clausal form then $\sigma(\psi) = \{\sigma(c) | c \in \psi\}$ and $\sigma(c) = \{\sigma(l) | l \in c\}$.

Definition 1 *Let $\Phi = Q_1 X_1, \dots, Q_m X_m \psi$ be a quantified boolean formula and σ a permutation over the literals of Φ . σ is a symmetry of Φ iff*

1. $\forall x \in Lit(\Phi), \sigma(\neg x) = \neg \sigma(x)$
2. $\sigma(\Phi) = \Phi$ i.e. $\sigma(\psi) = \psi$ and $\forall i \in \{1, \dots, m\} \sigma(X_i) = X_i$.

Let us note that each symmetry σ of a QBF Φ is also a symmetry of the boolean formula ψ . The converse is not true. So the set of symmetries of Φ is a subset of the set of symmetries of ψ .

A symmetry σ can be seen as a list of cycles $(c_1 \dots c_n)$ where each cycle c_i is a list of literals $(l_{i_1} \dots l_{i_{n_i}})$ st. $\forall 1 \leq k < n_i, \sigma_i(l_{i_k}) = l_{i_{k+1}}$ and $\sigma_i(l_{i_{n_i}}) = l_{i_1}$.

It is well known that breaking all symmetries might lead in the general case to an exponential number of clauses [Crawford *et al.*, 1996]. In this paper, for efficiency and clarity reasons, we only consider symmetries with binary cycles. Our approach can be extended to symmetries with cycles of arbitrary size.

Detecting symmetries of a boolean formula is equivalent to the graph isomorphism problem [Crawford, 1992; Crawford *et al.*, 1996] (i.e. problem of finding a one to one mapping between two graphs G and H). This problem is not yet proved to be NP-Complete, and no polynomial algorithm is known. In our context, we deal with graph automorphism problem (i.e. finding a one to one mapping between G and G) which is a particular case of graph isomorphism. Many programs have been proposed to compute graph automorphism. Let us mention NAUTY [McKay, 1990], one of the most efficient in practice.

Recently, Aloul *et al.* [Aloul *et al.*, 2002] proposed an interesting technique that transforms CNF formula ψ into a graph G_ψ where vertices are labeled with colors. Such colored vertices are considered when searching for automorphism on the graph (i.e. vertices with different colors can not be mapped with each others).

In [Audemard *et al.*, 2004], a simple extension to QBFs formulae is given. Such extension is simply obtained by introducing a different color for each set of vertices whose literals belong to the same quantifier group. In this way, literals from different quantifier groups can not be mapped with each others (see the second condition of the definition 1). Then, to detect such symmetries, NAUTY is applied on the graph representation of the QBF.

3 Breaking symmetries in QBFs

Symmetry breaking has been extensively investigated in the context of constraint satisfaction and satisfiability problems. The different approaches proposed to break symmetries can be conveniently classified as dynamic and static schemes. Dynamic breaking generally search and break symmetries using breaking predicates or not [Benhamou and Sais, 1994; Gent and Smith, 2000]. Static breaking schemes refer to techniques that detect and break symmetries in a preprocessing step. For SAT, symmetries are generally broken by generating additional constraints, called symmetry breaking predicates (SBP) [Crawford, 1992; Aloul *et al.*, 2002]. Such SBP eliminates all models from each equivalence class of symmetric models, except one. However, in the general case, the set of symmetry predicates might be of exponential size. In [Aloul *et al.*, 2002], Aloul *et al* extend the approach of Crawford [Crawford, 1992] by using group theory and the concept of non-redundant generators, leading to a considerable reduction in the SBP size.

We briefly recall the symmetry breaking technique introduced by Crawford in [Crawford *et al.*, 1996]. Let ψ be a CNF formula and $\sigma = (x_1, y_1) \dots (x_n, y_n)$ a symmetry of ψ . The sbp_σ associated to σ is defined as follows:

$$\begin{aligned} x_1 &\leq y_1 \\ (x_1 = y_1) &\rightarrow x_2 \leq y_2 \\ \dots & \\ (x_1 = y_1) \dots (x_{n-1} = y_{n-1}) &\rightarrow x_n \leq y_n \end{aligned}$$

The sbp_σ defined above expresses that, when for all $i \in \{1, \dots, k-1\}$ x_i and y_i are equivalent (take the same truth value) and x_k is *true*, then y_k must be assigned to *true* in order to avoid the exploration of isomorphic interpretations. In the next section, an extension of this approach to QBF is described.

3.1 Motivation

In the following example, we show the main difficulty behind the extension of SAT symmetry breaking predicates (SBP) to QBFs.

Example 1 Let $\Phi = \forall x_1 y_1 \exists x_2 y_2 \psi$ be a QBF where $\psi = (x_1 \vee \neg x_2) \wedge (y_1 \vee \neg y_2) \wedge (\neg x_1 \vee \neg y_1 \vee x_2 \vee y_2)$. The permutation $\sigma = \{(x_1, y_1)(x_2, y_2)\}$ is a symmetry of Φ . Breaking the symmetry σ using the traditional approach, induces the following $sbp_\sigma : (\neg x_1 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_2) \wedge (y_1 \vee \neg x_2 \vee y_2)$. As all the literals of the clause $(\neg x_1 \vee y_1)$ are universally quantified, the new obtained QBF by adding conjunctively the sbp_σ to the original one leads to an invalid QBF formula.

To overcome this main drawback, we will demonstrate that for symmetries containing only universal cycles, we can add disjunctively the symmetry breaking predicates. The resulting formula is combined with the one obtained for existential symmetries leading to a general QBF symmetry breaking approach.

In this paper, we propose two possible ways to handle symmetry breaking predicates in quantified boolean formulas. The first one, produces a non clausal matrix. Whereas, the second one produce a formula in prenex clausal form usually used by most of the available QBF solvers.

3.2 Breaking Symmetries : Theoretical approach

Let us now give a formal description of our approach for breaking symmetries in QBF.

Definition 2 Let $\Phi = Q_1 X_1 \dots Q_i X_i \dots Q_m X_m \psi$ be a QBF and σ a symmetry of Φ . We define $\sigma \uparrow X_i$ as the sub-sequence of the symmetry σ restricted to the cycles involving variables from X_i . Then, the symmetry σ can be rewritten following the prefix ordering as $\{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ such that $\sigma_i = \sigma \uparrow X_i$. When σ follow the prefix ordering, it is called *p-ordered*.

In the sequel, symmetries are considered to be p-ordered.

Example 2 Let $\Phi = \exists x_2 y_2 \forall x_1 y_1 \exists x_3 y_3 (\neg x_1 \vee y_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee y_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_3 \vee \neg y_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (\neg x_1 \vee \neg y_1 \vee \neg x_2 \vee \neg y_2)$. Φ has a symmetry $\sigma = \{(x_1, y_1)(x_2, y_2)(x_3, y_3)\}$. Reordering σ with respect to the prefix leads to $\sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ st. $\sigma_1 = \sigma \uparrow X_1 = (x_2, y_2)$, $\sigma_2 = \sigma \uparrow X_2 = (x_1, y_1)$ and $\sigma_3 = \sigma \uparrow X_3 = (x_3, y_3)$.

Definition 3 Let Φ be a QBF, σ a symmetry of Φ and $c = (x, y)$ is a cycle of σ . A cycle c is called *universal* (resp. *existential*) if x and y are universally quantified (resp. existentially quantified). A symmetry σ is called

universal (resp. *totally universal*) if it contains at least one universal cycle (resp. all cycles are universals), otherwise it is called *existential*.

For existential symmetries of a QBF, classical SBP [Crawford *et al.*, 1996] can be translated linearly to a CNF formula thanks to new additional variables. The obtained set of clauses can be added to the QBF matrix while preserving its validity. This is formalized in property 1. The main problem arises when breaking universal symmetries (see example 1).

Property 1 Let $\Phi = QX\psi$ be a quantified boolean formula and σ an existential symmetry of ψ . Then Φ and $QX(\psi \wedge sbp_\sigma)$ are equivalent with respect to the validity.

In a first step, our reasoning concerns totally universal symmetries. Let Φ be a QBF formula and $\sigma = \{(x_1, y_1)\}$ such that (x_1, y_1) is a universal cycle. The two interpretations $\{x_1, \neg y_1\}$ and $\{\neg x_1, y_1\}$ are symmetric. As x_1, y_1 are universally quantified, we can check one of them and consider the other one as a model of ψ . When x_1 is assigned to *true*, thanks to the sbp_σ we can propagate $y_1 = \text{true}$. Instead of finding a logical formulation implying $y_1 = \text{true}$ when $x_1 = \text{true}$, we transform ψ in such a way that when x_1 is assigned to *true* and y_1 to *false* then ψ becomes *true*. As the sbp_σ is *false* under the interpretation $\{x_1, \neg y_1\}$ and *true* otherwise, the formula $QX(\psi \vee \neg sbp_\sigma)$ is asymmetric respect to σ and is equivalent to Φ with respect to validity.

In the following property, we generalize this idea to a totally universal symmetry of arbitrary size.

Property 2 Let $\Phi = QX\psi$ be a QBF formula and σ a totally universal symmetry of Φ . Then, $\Phi = QX\psi$ and $\Phi' = QX(\psi \vee \neg sbp_\sigma)$ are equivalent with respect to the validity.

Proof :

proof of \rightarrow : It is obvious, since each model of ψ is also a model of $\psi \vee \neg sbp_\sigma$. Then a policy of Φ is also a policy of $QX(\psi \vee \neg sbp_\sigma)$.

proof of \leftarrow : Given \mathcal{I} a policy of $QX(\psi \vee \neg sbp_\sigma)$, we built a policy \mathcal{I}' of Φ . Let $I \in \mathcal{I}$, two cases can occur :

- $I \models \psi$, then $\mathcal{I}' = \mathcal{I} \cup I$
- $I \not\models \psi$, then, $I \models \neg sbp_\sigma$, $\exists c = (x_i, y_i) \in \sigma$ where c is universal and I is of the form $(x_1, \dots, x_i, \neg y_i, \dots)$. As \mathcal{I} is a total policy of $\psi \vee \neg sbp_\sigma$ then $\exists I' \in \mathcal{I}$ of the form $(x_1, \dots, \neg x_i, y_i, \dots)$ such that $I' \models \psi \vee \neg sbp_\sigma$ and $I' \not\models \neg sbp_\sigma$. Consequently, $I' \models \psi$ and $\sigma(I') \models \psi$ where $\sigma(I')$ is of the form $(x_1, \dots, x_i, \neg y_i, \dots)$. Then $\mathcal{I}' = \mathcal{I} \cup \sigma(I')$.

By construction, it is clear that \mathcal{I}' is a policy of Φ .

Example 3 Let $\Phi = \forall x_1 \exists x_2 y_2 (x_1 \vee x_2 \vee \neg y_2) \wedge (x_1 \vee \neg x_2 \vee y_2) \wedge (\neg x_1 \vee \neg x_2 \vee y_2) \wedge (\neg x_1 \vee x_2 \vee \neg y_2)$ Φ has a symmetry $\sigma = \{(x_1, \neg x_1)\}$. As $\{(x_1, \neg x_1)\}$ is a universal cycle, and $sbp_\sigma = (\neg x_1)$, we have: $\Phi' = \Phi \vee \neg sbp_\sigma = \Phi \vee x_1$. Translating Φ' to clausal form we obtain : $\Phi' = \forall x_1 \exists x_2 y_2 (x_1 \vee x_2 \vee \neg y_2) \wedge (x_1 \vee \neg x_2 \vee y_2)$

The problem arises from symmetry with existential and universal cycles. the new formula $\Phi' = QX(\psi \vee \neg sbp_\sigma)$ described in property 2 is not equivalent to Φ wrt. validity. Let us consider a counter example. Let $\sigma = \{(x_1, y_1) \dots (x_n, y_n)\}$ be a symmetry such that $\sigma \uparrow X_1 = (x_1, y_1)$ and $Q_1 = \exists$. If we start by assigning x_1 to *true* and y_1 to *false*, the sbp_σ becomes *false*, and $(\psi \vee \neg sbp_\sigma)$ is valid. This conclusion is wrong because it means that all formulae Φ which have a symmetry $\sigma = \{(x_1, y_1) \dots (x_n, y_n)\}$ such that $\sigma \uparrow X_1 = (x_1, y_1)$ and $Q_1 = \exists$ are valid.

In conclusion, let σ a symmetry of Φ . If all cycles of σ are existentially quantified then Φ and $(\Phi \wedge sbp_\sigma)$ are equivalent and if all cycles of σ are universally quantified then Φ and $(\Phi \vee \neg sbp_\sigma)$ are equivalent. To deal with symmetries containing both universally and existentially quantified cycles, one need to alternate sub parts of $(\Phi \wedge sbp_\sigma)$ and $(\Phi \vee \neg sbp_\sigma)$.

Definition 4 Let $\Phi = Q_1 X_1 \dots Q_m X_m \psi$ be a QBF and $\sigma = \{\sigma_1 \dots \sigma_m\}$ a symmetry of Φ . We define $sbp_\sigma \uparrow X_i$ the projection of sbp_σ on the set X_i

Example 4 Let $\Phi = \forall x_1 y_1 x_2 y_2 \exists x_3 y_3 \psi$ be a QBF formula such that $\sigma = \{(x_1, y_1)(x_2, y_2)(x_3, y_3)\}$ a symmetry of Φ .

$$\begin{aligned} sbp_\sigma \uparrow X_1 &= x_1 \leq y_1 \\ &\quad (x_1 = y_1) \rightarrow x_2 \leq y_2 \\ sbp_\sigma \uparrow X_2 &= (x_1 = y_1) \wedge (x_2 = y_2) \rightarrow x_3 \leq y_3 \end{aligned}$$

Using the previous definitions, we can now describe the general formulation of symmetry breaking for quantified boolean formulae as follows:

Property 3 Let $\Phi = Q_1 X_1 \dots \exists X_m \psi$ be a QBF and $\sigma = \{\sigma_1 \dots \sigma_m\}$ a symmetry of Φ . Φ and Φ' are equivalent with respect to validity where $\Phi' = Q_1 X_1 \dots \exists X_m (\dots ((\psi \wedge [sbp_\sigma \uparrow X_m]) \vee \neg [sbp_\sigma \uparrow X_{m-1}]) \wedge [sbp_\sigma \uparrow X_{m-2}]) \vee \dots)$

Proof : Suppose that Q_1 is universal.

We can restrict the symmetry breaking predicates to X_1 . All cycles of $\sigma \uparrow X_1$ are universal. Then by property 2, Φ and $Q_1 X_1 \dots Q_m X_m (\psi \vee \neg [sbp_\sigma \uparrow X_1])$ are equivalent. Let $\sigma_1 = (x_1, y_1) \dots (x_p, y_p)$. and $\eta_1 = (x_1 = y_1) \wedge \dots \wedge (x_p = y_p)$. If $\eta_1 = \text{true}$ then $\{\sigma_2 \dots \sigma_m\}$ is a symmetry of ψ and also a symmetry of $QX(\psi \vee \neg [sbp_\sigma \uparrow X_1])$. By considering only σ_2 , $QX\psi$ is equivalent to $QX(\psi \vee \neg [sbp_\sigma \uparrow X_1]) \wedge (\neg \eta_1 \vee sbp_{\sigma_2})$. From the definition 4, we can easily conclude that $sbp_\sigma \uparrow X_2 = (\neg \eta_1 \vee sbp_{\sigma_2})$. Finally we have $QX\psi$ is equivalent to $(\psi \wedge [sbp_\sigma \uparrow X_2]) \vee (\neg [sbp_\sigma \uparrow X_1] \wedge [sbp_\sigma \uparrow X_2])$ As $(\neg [sbp_\sigma \uparrow X_1] \rightarrow [sbp_\sigma \uparrow X_2])$ $QX\psi$ and $QX(\psi \wedge [sbp_\sigma \uparrow X_2] \vee \neg [sbp_\sigma \uparrow X_1])$ are equivalent wrt validity. The proof follow by iterating the above process. Similarly the proof can be obtained when the first quantifier Q_1 is existential.

Let us note that in the case of existential symmetries σ_e , $\Phi' = QX(\psi \wedge sbp_{\sigma_e})$ which is the formula of property 1. And in the case of totally universal symmetries σ_u

$\Phi' = QX(\psi \vee \neg sbp_{\sigma_u})$ which is the formula associated to property 2. The following example illustrate how the formula Φ' is built.

Example 5 Let us consider the QBF Φ of the example 2. $\sigma = \{(x_1, y_1)(x_2, y_2)(x_3, y_3)\}$ is a symmetry of Φ . Φ is rewritten as :

$$\begin{aligned} \Phi' &= \exists x_1 y_1 \forall x_2 y_2 \exists x_3 y_3 \\ &((\psi \wedge [sbp_\sigma \uparrow X_3]) \vee \neg [sbp_\sigma \uparrow X_2]) \wedge [sbp_\sigma \uparrow X_1] \\ &\text{where:} \\ &sbp_\sigma \uparrow X_1 = x_1 \leq y_1 \\ &sbp_\sigma \uparrow X_2 = (x_1 = y_1) \rightarrow x_2 \leq y_2 \\ &sbp_\sigma \uparrow X_3 = (x_1 = y_1) \wedge (x_2 = y_2) \rightarrow x_3 \leq y_3 \end{aligned}$$

This reasoning can easily be generalized to a set of symmetries $S = \{\sigma^1, \dots, \sigma^k\}$ by considering projections of all symmetries of S on X_i .

Let us note that the formula Φ' is not in prenex clausal form. As most of the available QBF solvers take as input a QBF formula in prenex clausal form, one need to translate Φ' to this most used standard format. For the other few solvers using general representation ([Zhang, 2006; Sabharwal et al., 2006]) this translation is not necessary. Unfortunately, this transformation could be time and space consuming. To overcome this problem, we propose in the next section an linear time transformation approach.

3.3 Breaking Symmetries : clausal form

Let $\Phi = QX\psi$ be a quantified boolean formula and $\sigma = \{(x_1, y_1) \dots (x_n, y_n)\}$ a symmetry of Φ . Suppose $(x_1 = y_1)$ and $(x_2 = y_2) \dots (x_{i-1} = y_{i-1})$, our proposed transformation distinguish the two following cases :

1. $Qx_i, Q = \exists$: if $x_i = \text{true}$ then y_i has to be assigned to *true*
2. $Qx_i, Q = \forall$: $x_i = \text{true}$ and $y_i = \text{false}$, then we must satisfy ψ under this interpretation.

By considering all the symmetry σ , there exists a set of interpretations which we want to consider as models of ψ . These expected models are collected in a set called I_σ defined in the following definition.

Definition 5 Let $\Phi = QX\psi$ be a quantified boolean formula and $\sigma = (x_1, y_1), \dots, (x_n, y_n)$ a symmetry of Φ . We define I_σ as the set of following interpretations: $I_\sigma = \bigcup_{i=1..n} \{(x_1 = y_1), \dots, (x_{i-1} = y_{i-1}), x_i, \neg y_i\} \mid \text{such that } x_i \text{ is universal}\}$

In order to break symmetries of Φ , we want to satisfy the above two cases, i.e. propagate existential literals thanks to sbp_σ and consider all interpretations of I_σ as models of ψ . To this end, we introduce a new existential variable r_σ added to the innermost quantifier group and we rewrite ψ as follows: $(\psi \vee \neg r_\sigma) \wedge f_\sigma(r_\sigma, sbp_\sigma)$. The function f_σ , allows us to break the symmetry σ . It forces r_σ to be *false* for all interpretations belonging to I_σ and *true* otherwise. Furthermore, it induces propagations for existential cycles.

Definition 6 Let $\sigma = \{(x_1, y_1) \dots (x_{lu}, y_{lu}) \dots (x_n, y_n)\}$ be a symmetry of

Φ such that $\forall i > lu$, (x_i, y_i) is an existential cycle (lu the rank of the last universal cycle in σ). The function $f_\sigma(r, sbp_\sigma)$ is defined as follow:

1. if $0 < i < lu$ and $(x_1 = y_1) \wedge \dots \wedge (x_{i-1} = y_{i-1})$ then:
 - if (x_i, y_i) is a universal cycle:
 $x_i \wedge \neg y_i \rightarrow \neg r$ and $\neg x_i \wedge y_i \rightarrow r$
 - else
 $x_i \rightarrow y_i$ and $\neg x_i \wedge y_i \rightarrow r$
2. if $i = lu$ and $(x_1 = y_1) \wedge \dots \wedge (x_{i-1} = y_{i-1})$ then :
 $x_{lu} \wedge \neg y_{lu} \rightarrow \neg r$ and $\neg x_{lu} \rightarrow r$ and $(x_{lu} = y_{lu}) \rightarrow r$
3. if $i > lu$ and $(x_1 = y_1) \wedge \dots \wedge (x_{i-1} = y_{i-1})$ then :
 $x_i \rightarrow y_i$

Example 6 Let $\Phi = \exists x_1, y_1 \forall x_2, y_2 \exists x_3, y_3 \psi$ be a QBF formula and $\sigma = \{(x_1, y_1), (x_2, y_2)\}$ a symmetry of Φ . Using the formula $f_\sigma(r_\sigma, sbp_\sigma)$, the different values taken by r_σ according to the different interpretations of x_1, x_2, y_1, y_2 are depicted in Figure 1.

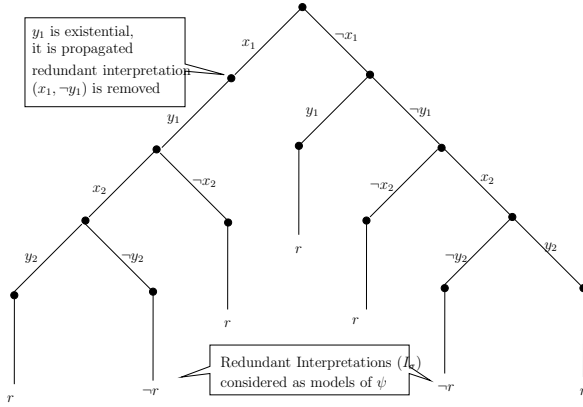


Figure 1: $f_\sigma(r_\sigma, sbp_\sigma)$: a search tree

Finally, we have the following property.

Property 4 Let $\Phi = QX\psi$ be a QBF and σ a symmetry of Φ , then Φ and $\Phi^r = QX\exists r (\psi \vee \neg r) \wedge f_\sigma(r, sbp_\sigma)$ are equivalent wrt. validity.

Example 7 Let us consider the QBF

$\Phi = \forall x_1, y_1, x_2, y_2 \exists x_3, y_3 \psi$ and $\sigma = \{(x_1, y_1), (x_2, y_2)\}$ a symmetry of Φ .

$\Phi^r = \forall x_1, y_1, x_2, y_2 \exists x_3, y_3 \alpha_1, \alpha_2, r$

$$\begin{array}{ll}
(\psi \vee \neg r) & \wedge \\
(\neg \alpha_1 \vee x_2 \vee y_2 \vee \alpha_2) & \wedge (x_1 \vee y_1 \vee \alpha_1) \wedge \\
(\neg \alpha_1 \vee \neg x_2 \vee \neg y_2 \vee \alpha_2) & \wedge (\neg x_1 \vee \neg y_1 \vee \alpha_1) \wedge \\
(\neg \alpha_1 \vee \neg x_1 \vee \neg x_2 \vee y_2 \vee \neg r) & \wedge (\neg x_1 \vee y_1 \vee \neg r) \wedge \\
(\neg \alpha_1 \vee y_1 \vee \neg x_2 \vee y_2 \vee \neg r) & \wedge (x_1 \vee \neg y_1 \vee r) \wedge \\
(\neg \alpha_1 \vee x_2 \vee \neg r) & \wedge (\neg \alpha_1 \vee \neg \alpha_2 \vee r)
\end{array}$$

The variables α_1 (resp. α_2) is introduced to express that $(x_1 = y_1)$ (resp. $(x_1 = y_1) \wedge (x_2 = y_2)$). The last 10 clauses are those of $f_\sigma(r, sbp_\sigma)$.

In the following, we show how to generalise the previous transformation to an arbitrary set of symmetries. Let Φ a QBF and $S = \{\sigma^1 \dots \sigma^n\}$ the set of symmetries of Φ . For each universal symmetry σ^i , we introduce an additional variable r_σ^i to break it. For the set S , the equivalent formula is obtained by adding all the symmetries of Φ as described in the following definition.

Definition 7 Let Φ be a QBF and S a set of symmetries of Φ and $U = \{\sigma^1 \dots \sigma^p\}$ the subset of S containing only universal symmetries. we define

$$\Phi^r = QX\exists r_\sigma^1 r_\sigma^2 \dots r_\sigma^p r (\psi \vee \neg r) \wedge f_U(r, SBP_U) \wedge SBP_\exists$$

where

- $r = \wedge (r_\sigma^1, r_\sigma^2 \dots r_\sigma^p)$
- $f_U(r, SBP_U) = \bigwedge_{\sigma^i \in U} f_{\sigma^i}(r_\sigma^i, sbp_{\sigma^i})$
- SBP_\exists (resp. SBP_U) is the SBP corresponding to all existential (resp. universal) symmetries.

Property 5 Let Φ a QBF formula and S the set of all its symmetries, then Φ and Φ^r are equivalent with respect to validity.

In the sequel, we propose some useful simplification of our proposed transformation that we use in our experimental evaluation.

Let $\Phi = Q_1 X_1 \dots Q_m X_m$ be a QBF and $U = \{\sigma^1 \dots \sigma^p\}$ be the set of universal symmetries of Φ such that $Var(U) \in \{X_i \cup \dots \cup X_n\}$. Let c be a clause of ψ such that $Var(c) \in \{X_1 \cup \dots \cup X_{i-1}\}$. then it is not necessary to add $\neg r$ in c . Indeed, if $\Phi_{X_1 \dots X_{i-1}}$ is the QBF obtained after the assignment of all the variables in $\{X_1 \cup \dots \cup X_{i-1}\}$, then σ remains a symmetry of $\Phi_{X_1 \dots X_{i-1}}$.

Obviously, the new built formula allows us to avoid some isomorphic interpretations. However, adding $\neg r$ to all clauses of ψ eliminates some useful propagation. For example, if $(x \vee y)$ is a clause such that neither x and y belong to the symmetries of Φ then $(x \vee y \vee \neg r)$ is the new clause of Φ^r . When x is assigned to *false*, then y is propagated in ψ , which is not the case in Φ^r . Furthermore, in the worst case, if y is assigned to *false*, $\neg r$ is propagated and the solver will lose time in checking $f_U(r, sbp_U)$ i.e. to prove the inconsistency induced by the *sbp*.

Not surprisingly, in practice adding $\neg r$ in all clauses of ψ make generally Φ^r more difficult to be solved than Φ . To avoid such a case, the goal is to add $\neg r$ just in some chosen clauses and leave the others unchanged.

For our experiments, as binary clauses might be helpful during search, we add $\neg r$ only in non binary clauses, containing at least one literal appearing in a universal symmetry.

Let $\Phi = Q_1X_1 \dots Q_mX_m$ be a QBF and $\sigma = \{(x_1, y_1) \dots (x_n, y_n)\}$ a symmetry of Φ . Suppose that exists a cycle (x_p, y_p) of σ such that x_p and y_p are pure literals. In this case, if we generate the *sbp* of σ , then the literals x_p and y_p are not pure. As the pure literals play an important role in QBF solving [Giunchiglia *et al.*, 2004], this might have a great impact on the efficiency of the QBF solvers. To avoid this problem, we propose the two following possibilities. The first one, consists in eliminating the cycle (x_p, y_p) from σ and propagating x_p and y_p following their quantifiers. The second one consists in reducing σ to the set of cycles $\{(x_1, y_1) \dots (x_{p-1}, y_{p-1})\}$. For our experiments, we choose this second option.

Finally, if σ is of the form $(x, \neg x)$ where x is a universal literal, then x is requantified existentially and the clauses $c = (\neg x)$ is added to the matrix.

Complexity

Adding new existentially quantified variables, the size of the classical *SBP* is known to be linear in the size of symmetry. According to the formula of $f_\sigma(r_\sigma, sbp_\sigma)$, its size remain linear in the size of the symmetry σ . In the worst case, for a totally universal symmetry $\sigma = \{(x_1, y_1) \dots (x_n, y_n)\}$, we introduce one variable and four clauses for each cycle. Consequently, the size of $f_\sigma(r_\sigma, sbp_\sigma)$ is in $O(5n)$.

4 Experiments

The experimental results reported in this section are obtained on a Xeon 3.2 GHz (2 GB RAM) and performed on a large panel of symmetric instances available from [Giunchiglia *et al.*, 2001a]. This set of QBF instances contains different families like `toilet`, `k_*`, `FPGA`, `qshifter`. As a comparison, we run the state-of-the-art DPLL-like solver SEMPROP [Letz, 2002] on QBF instances with and without breaking symmetries. The time limit is fixed to 900 seconds. Results are reported in seconds. The symmetry computation time is not reported (less than one second in most cases). *U* indicates if the instances contain universal symmetries, *NB_S* represents the number of symmetries, *SAT* indicates if the instances is valid or not. Φ represents the time needed to solve the instances without breaking symmetries and Φ^r the time to solve the instances after breaking symmetries with the following restrictions. First, in a case of existential symmetry, we keep only the first cycle. Second, in a case of universal symmetry, we cut the symmetries from the last universal cycle.

Table 1 gives results on a large variety of QBF instances. Breaking symmetries with our method significantly improves SEMPROP performances on many QBF instances leading to more solved instances. As an example, for the `biu_*` family, without breaking symmetries, SEMPROP solves one instance whereas adding symmetry breaking predicates, we are able to solve 8 instances. For `C5135_*` family we can solve one instance. On `toilet_c*` family the cpu time decreases in more than one order of magnitude when using symmetry breaking predicates.

Furthermore, the existence of universal symmetries seems to be an important factor for reducing the search time, especially when these symmetries occurs in the outermost quantifier. It is the case for example for the `biu` family.

Not surprisingly, we have also noticed that symmetries between literals occurring in the innermost quantifier (recall that it is existential) group are useless. Indeed, such symmetries does not lead to a great reduction in the search tree, since their corresponding variables are assigned last i.e when the formula is considerably reduced by the previous assignments. Finally, for QBF families containing only existential symmetries, breaking them does not lead to a great improvement in the search time except in some cases (see for example families (`k_grz`), `toilet_a`).

5 Comparison between our encoding and sbp4qbf algorithm

Let us now compare our method with `sbp4qbf` [Audemard *et al.*, 2007].

To break symmetries, the authors of `sbp4qbf` propose a new order of the quantifier. For example, let Φ be a QBF and $\{(x_1, y_1)\}$ a symmetry of Φ such that, $Var(X_1) = \{(x_1, y_1)\}$, and $Q_1 = \forall$. The equivalent *sbp* is $x_1 \leq y_1$. So, if x_1 is assigned to *true*, then we can directly assign y_1 to *true* and if y_1 is assigned with *false*, x_1 can be assigned to *false*. From this algorithm, the new prefix is rewritten as $\forall x_1 \exists \alpha_1 \forall y_1' \exists y_1 Q_2 X_2 \dots Q_m X_m \psi \wedge sbp \wedge qsbp$ where *qsbp*, are the added predicates to preserve the equivalence. This new prefix constrains x_1 to be assigned before y_1 . With our method no restriction is needed. The only modification of the prefix concern the set of existential variables added to the innermost quantifier. In `sbp4qbf`, when there is two universal symmetries like $\sigma^1 = \{r_1, (x_1, y_1) \dots\}$ and $\sigma^2 = \{r_2, (z_1, -y_1) \dots\}$ such that y_1 is a universal literal. To keep the equivalence, σ^2 is reduced to $\sigma'^2 = \{r_2\}$ and a part of the symmetry σ^2 is removed. It is not the case here with our proposed method.

Obviously, the main difference between `sbp4qbf` and our method is in the way used to manage the universal literals. Indeed, considers a simple universal symmetry $\{(x_1, y_1)\}$, if x_1 is assigned to *true*, y_1 is propagated in `sbp4qbf`, where with our method, $\{x_1, \neg y_1\}$ is considered as a model which is an other alternative to the algorithm proposed in [Audemard *et al.*, 2007].

At the end, we compared experimentally our results with the ones obtained in [Audemard *et al.*, 2007]. Table 2 provides more detailed results on the different QBF families. The second column (*NB*) represents the number of instances in each family. The third column (*U*) indicates if the instances contain universal symmetries (*Y*) or not (*N*). For each family, *S* and *TT* represents the total number of solved instances and the total run-time needed for solving all the instances (900 seconds are added for each unsolved one) respectively and finally Φ^S represent the broken formula using the algorithm described in [Au-

Instances	U	NB_S	SAT	Φ^r	Φ
biu.mv.xl_ao-p005-IPF02-c02	Y	15	F	—	1.45
biu.mv.xl_ao-p005-OPF03-c09	Y	15	T	17.83	—
biu.mv.xl_ao-p010-IPF02-c07	Y	15	T	1.25	—
biu.mv.xl_ao-p010-IPF02-c08	Y	15	T	0.83	—
biu.mv.xl_ao-p010-IPF05-c05	Y	94	T	0.15	—
biu.mv.xl_ao-p010-IPF05-c07	Y	28	T	16.72	—
biu.mv.xl_ao-p010-IPF05-c10	Y	91	T	0.16	—
biu.mv.xl_ao-p010-OPF02-c08	Y	91	T	1.36	—
biu.mv.xl_ao-p010-OPF02-c10	Y	91	T	1.35	—
C5315.blif_0.10_1.00_0_1_out_exact	Y	37	T	788.41	—
k_path_p-10	Y	4	F	18.73	39.48
k_path_p-11	Y	4	F	20.35	71.39
k_path_p-12	Y	7	F	130.25	357.70
k_path_p-13	Y	8	F	203.14	643.14
k_path_p-14	Y	8	F	—	39.64
k_path_p-14	Y	8	F	—	39.64
k_path_n-19.qdimacs	Y	11	T	—	579.72
ncf_8_16_4_u.4	Y	1	F	168.39	176.67
ncf_8_16_4_u.9	Y	1	F	313.67	336.21
ncf_8_16_8_d.2	Y	1	F	121.71	129.42
ncf_8_16_8_d.4	Y	1	F	168.28	180.17
ncf_8_16_8_d.6	Y	1	F	157.22	169.51
ncf_4_16_4_d.6	Y	1	F	19.28	3.06
ncf_16_32_4_u.9	Y	1	T	70.45	61.36
ncf_4_16_2_u.8	Y	2	T	90.19	1.69
ncf_8_16_8_edau.3	Y	1	F	305.25	318.73
ncf_8_16_8_edau.6	Y	1	T	455.41	479.48
ncf_8_16_8_euad.7	Y	1	T	669.75	717.30
lut4_3_fAND	Y	11	T	0.28	26.74
lut4_AND_fXOR	Y	11	F	423.40	—
lut4_2_fXOR	Y	9	T	4.51	0.07
ken.flash08.C-f4	Y	1	F	438.24	8.26
term1.blif_0.10_0.20_0_0_inp_exact	Y	3	F	151.88	160.19
TOILET7.1.iv.13	Y	2	F	12.82	70.37
TOILET6.1.iv.11	Y	2	F	1.28	4.39
TOILET7.1.iv.14	Y	2	T	11.83	7.07
toilet_c_10_01.15	Y	2	F	2.48	12.05
toilet_c_10_01.16	Y	2	F	7.47	48.31
toilet_c_10_01.17	Y	2	F	23.75	158.30
toilet_c_10_01.18	Y	2	F	77.86	570.81
toilet_c_10_01.19	Y	2	F	259.48	—
toilet_c_10_01.20	Y	2	T	1.87	—
S-adeu-26	Y	46	F	8.49	14.69
S-adeu-37	Y	46	F	1.21	3.32
S-adeu-40	Y	46	F	7.51	12.53
S-edau-26	Y	46	F	4.17	6.23
S-edau-43	Y	46	F	1.66	2.71
S-adeu-23	Y	48	F	37.77	4.85
S-adeu-40	Y	46	F	7.51	12.53
T-adeu-14	Y	46	F	7.47	25.70
T-adeu-26	Y	46	F	2.04	4.08
T-adeu-40	Y	46	F	1.65	2.97
T-edau-14	Y	46	F	2.97	6.65
x40.11	N	2	T	687.46	728.11
k_branch_n-9	N	1	T	291.31	292.66
k_branch_p-10	N	1	F	170.64	182.57
k_branch_p-11	N	1	F	331.90	334.92
k_branch_p-12	N	1	F	192.48	206.89
k_branch_p-9	N	1	F	38.88	41.47
k_poly_p-20	N	2	F	314.37	334.85
k_poly_p-21	N	2	F	348.21	349.71
k_poly_p-9	N	2	F	21.59	22.92
k_ph_n-10	N	1	T	651.07	—
k_grz_n-10	N	6	T	148.00	161.22
k_grz_n-12	N	6	T	729.89	874.60
k_grz_n-13	N	6	T	360.63	425.38
k_grz_n-15	N	8	T	325.21	338.32
k_grz_n-16	N	8	T	298.36	326.51
k_grz_n-17	N	8	T	255.17	263.69
k_grz_p-10	N	4	F	153.96	164.11
k_grz_p-11	N	4	F	79.35	84.79
k_grz_p-12	N	4	F	191.12	205.59

Table 1: SEMPROP with and without symmetries breaking

demard *et al.*, 2007]. In general case, `sbp4qbf` seems to be more efficient than our method but it exists instances, where our method outperforms `sbp4qbf`.

family	NB	U	Φ		Φ^S		Φ^r	
			S	TT	S	TT	S	TT
fpga	8	Y	6	1834	7	921	6	2231.71
biu	23	Y	1	19801	8	13668	8	13539
toilet_c	53	Y	51	2656	53	49	53	374.61
C5315	4	Y	0	3600	0	3600	1	3488
k_path	40	Y	28	12915	34	7999	24	15001.79
qshifter	6	Y	6	67	6	61	6	40
TOILET	7	Y	6	988	6	926	6	926
term1	6	Y	6	174	6	177	6	164.13
strategic	100	N	86	13482	86	13477	86	13477
k_branch	42	N	21	20096	21	20069	21	20069
k_lin	21	N	5	14553	5	14551	5	14551
k_grz	37	N	23	15242	24	14997	24	14997
k_poly	42	N	42	2031	42	2068	42	2068
toilet_a	22	N	22	12	22	20	22	20
TOTAL	411		303	107420	320	92583	310	100945

Table 2: Results on different QBF families

6 Conclusion

In this paper, a new approach to break symmetries in quantified boolean formulae is proposed. To preserve the equivalence, this new formula allows to propagate the existential literals from the `sbp` and remove universal symmetrical literals by introducing an auxiliary variable and considering some interpretations as models of the original formula. Experiments show the interest behind breaking these symmetries. Important improvement are obtained on different QBF families. As future work, we want to use our theoretical approach on non clausal solvers.

References

- [Aloul *et al.*, 2002] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. Technical report, University of Michigan, 2002.
- [Audemard *et al.*, 2004] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified boolean formulas. In *proceedings of SAT*, pages 257–262, 2004.
- [Audemard *et al.*, 2007] G. Audemard, S. Jabbour, and L. Sais. Symmetry breaking for quantified boolean formulas. In *proceedings of IJCAI*, pages 2262–2267, 2007.
- [Benedetti, 2005a] M. Benedetti. sKizzo: a Suite to Evaluate and Certify QBFs. In *Proc. of CADE*, pages 369–376, 2005.
- [Benedetti, 2005b] Marco Benedetti. Extracting certificates from quantified boolean formulas. In *IJCAI*, pages 47–53, 2005.
- [Benhamou and Sais, 1994] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102, 1994.

- [Biere, 2004] Armin Biere. Resolve and expand. In *proceedings of SAT (Selected Papers)*, pages 59–70, 2004.
- [Cohen *et al.*, 2005] David A. Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. In *proceedings of CP*, pages 17–31, 2005.
- [Coste-Marquis *et al.*, 2006] Sylvie Coste-Marquis, Hélène Fargier, Jérôme Lang, Daniel Le Berre, and Pierre Marquis. Representing policies for quantified boolean formulae. In *proceedings of KR*, pages 286–297, 2006.
- [Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luck, and A. Roy. Symmetry-breaking predicates for search problems. In *proceedings of KR*, pages 148–159, 1996.
- [Crawford, 1992] J. Crawford. A theoretical analysis of reasoning by symmetry in first order logic. In *Proceedings of Workshop on Tractable Reasoning, AAAI*, pages 17–22, 1992.
- [Gent and Smith, 2000] I. Gent and B. Smith. Symmetry breaking in constraint programming. In *Proceedings of ECAI*, pages 599–603, 2000.
- [Giunchiglia *et al.*, 2001a] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. <http://www.qbflib.org>.
- [Giunchiglia *et al.*, 2001b] E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of IJCAR*, pages 364–369, 2001.
- [Giunchiglia *et al.*, 2004] Enrico Giunchiglia, Masimo Narizzano, and Armando Tachella. Monotone literals and learning in qbf reasoning. In *Proceedings of CP*, pages 260–273, 2004.
- [Letz, 2002] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux*, pages 160–175, 2002.
- [McKay, 1990] B. McKay. nauty user’s guide (version 1.5). Technical report, 1990.
- [Puget, 1993] J.F. Puget. On the satisfiability of symmetrical constraint satisfaction problems. In *proceedings of ISMIS*, pages 350–361, 1993.
- [Sabharwal *et al.*, 2006] Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. Qbf modeling: Exploiting player symmetry for simplicity and efficiency. In *in proceedings of SAT*, pages 382–395, 2006.
- [Zhang and Malik, 2002] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the CP*, pages 200–215, 2002.
- [Zhang, 2006] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *in proceedings of AAAI*, 2006.

Dynamic Detection and Elimination of Local Symmetry in CSPs

Belaïd Benhamou and Mohamed Réda Saïdi

Laboratoire des Sciences de l'Information et des Systèmes (LSIS)

Centre de Mathématiques et d'Informatique

39, rue Joliot Curie - 13453 Marseille cedex 13, France

email: Belaïd.Benhamou@cmi.univ-mrs.fr, saïdi@cmi.univ-mrs.fr

Abstract

Many research works on symmetry in CSPs appeared recently. But, most of them deal only with the global symmetry¹ of the studied problem and give no strategy that can be used to detect and eliminate local symmetry². Eliminating global symmetry is shown to be useful, but exploiting only these symmetries could not be sufficient to solve some hard locally symmetrical problems. That is, a problem can have few or no initial symmetries and become very symmetrical at some nodes during the search. In this paper we study a general principle of semantic symmetry and define a syntactic symmetry which is a sufficient condition for semantic symmetry. We define a weakened form of this syntactic symmetry, and study three local symmetry detection and elimination strategies and compare them to global symmetry elimination. Experiments confirm that local symmetry breaking is profitable for CSP solving.

1 Introduction

As far as we know, the principle of symmetry is first introduced by Krishnamurty [Krishnamurty, 1985] to improve resolution in propositional logic. Symmetries for Boolean constraints are studied in depth in [Benhamou and Sais, 1992; 1994; Benhamou *et al.*, 1994], the authors showed how to detect them and proved that their exploitation is a real improvement for several automated deduction algorithms. The notion of interchangeability in CSPs is introduced in [Freuder, 1991] and symmetry for CSPs are studied in [Puget, 1993; Benhamou, 1994].

Since that, many research works on symmetry have appeared. For instance, the static approach used by James Crawford *et al.* in [Crawford *et al.*, 1996] for propositional logic theories consists in adding constraints to break the global symmetries of the problem. This technique has been

¹The symmetry of the initial problem appearing at the root of the search tree

²The symmetry of the resulting CSP at a node of the search tree corresponding to a partial instantiation

improved in [Aloul *et al.*, 2003] and extended to 0-1 Integer Logic Programming in [Aloul *et al.*, 2004].

Since a great number of constraints could be added, some researchers proposed to add the constraints during the search. In [Backofen and Will, 1999; Gent and Smith, 2000; Gent *et al.*, 2002], authors post some conditional constraints which remove the symmetric of the partial interpretation in case of backtracking. In [Focacci and Milano, 2001; Fahle *et al.*, 2001; Puget, 2002; Gent *et al.*, 2003], authors proposed to use each subtree as a no-good to avoid exploration of some symmetric interpretations and the group equivalence tree conceptual for value symmetry elimination is introduced in [Roney-Dougal *et al.*, 2004]. These techniques are called respectively SBDS, SBDD and GE-Tree.

Recently, a method which breaks symmetries between the variables of an AllDiff constraint is studied in [Puget, 2005c], a nice method which eliminates all value symmetries in surjection problems is given in [Puget, 2005b], and a work gathering all the known different symmetry definitions to *solution symmetry* or to *constraint symmetry* is done in [Cohen *et al.*, 2005]. More recently, in [Puget, 2006], Puget studied a new lex constraints symmetry breaking method in the term of dynamic lex leader solutions, and in [Walsh, 2006] Walsh studied various new propagators to break various symmetries among them the one acting simultaneously on both variables and values.

One drawback of all these approaches is that only the symmetry of the initial problem is considered (the global symmetry) and no method that allows dynamic detection and exploitation of local symmetry is given. Recently, researchers called this, conditional symmetry [Gent *et al.*, 2005; Zampelli *et al.*, 2006].

In this paper we developed the general concept of semantic symmetry for CSPs that Benhamou first initiated in [Benhamou, 1994]. We also study and extend the principle of syntactic symmetry that we prove to be a sufficient condition for semantic symmetry. We show how local symmetry is detected and eliminated during search, and show how its removal simplifies the search space of tree search algorithms.

This paper is organized as following: CSP background is given in Section 2. *Semantic symmetry* is defined in Section 3. Section 4 discusses the notion of *syntactical symmetry* which is a sufficient condition for *semantic symmetry*. Section 5 shows how symmetry is detected and eliminated

locally during search and how a tree search method (here Forward Checking) takes advantage of symmetrical values to reduce its search space. In section 6 we evaluate the proposed techniques by experimental results and Section 7 concludes the work.

2 Background

A CSP is a quadruple $\mathcal{P} = (V, D, C, R)$ where: $V = \{v_1, \dots, v_n\}$ is a set of n variables; $D = \{D_1, \dots, D_n\}$ is the set of finite discrete domains associated to the CSP variables, D_i includes the set of possible values of the CSP variable v_i , d_i denotes the fact that the value d belongs to the domain D_i , $C = \{C_1, \dots, C_m\}$ is a set of m constraints each involving a subset of the CSP variables. A binary constraint is a constraint which involves at most two variables v_i, v_j , and is denoted by C_{ij} ; $R = \{r_1, \dots, r_m\}$ is a set of relations corresponding to the constraints of C . r_i represents the list of value tuples permitted by the constraint $C_i \in C$. A binary CSP \mathcal{P} (a CSP involving only binary constraints) can be represented by a constraint graph $G(V, E)$ where the set of vertices V is the set of the CSP variables and each edge $(v_i, v_j) \in E$ connects the variables v_i and v_j involved in the constraint $C_{ij} \in C$. The microstructure [Freuder, 1991; Jegou, 1993; Cohen *et al.*, 2005] of the CSP \mathcal{P} is a graph $\mathcal{M}_{\mathcal{P}}(V \times \cup_{i \in [1, n]} D_i, \acute{E})$, where each edge of \acute{E} corresponds either to a tuple allowed by a specific constraint or to an allowed tuple because there is no constraint between the associated variables. An instantiation $\mathcal{I} = (\langle v_1, a_1 \rangle, \langle v_2, a_2 \rangle, \dots, \langle v_n, a_n \rangle)$ is the variable assignment $\{v_1 = a_1, v_2 = a_2, \dots, v_n = a_n\}$ where a value a_i of the domain D_i is assigned to the variable v_i . A constraint $C_i \in C$ is satisfied by \mathcal{I} if the projection of \mathcal{I} on the variables involved in C_i is a tuple of r_i . The instantiation \mathcal{I} is consistent if it satisfies all the constraints of C , thus \mathcal{I} is a solution of the CSP. An instantiation of a subset of the CSP variables V is called a partial instantiation, it defines a nogood when it is inconsistent. Each partial instantiation \mathcal{I} defines a node $n_{\mathcal{I}}$ in the search tree which corresponds to the local CSP $\mathcal{P}_{\mathcal{I}}$ resulting from \mathcal{P} by considering \mathcal{I} and its induced propagations. An instantiation is total if it is defined on all the CSP variables. Given a CSP, the main question is to decide its consistency or to find its set of solutions. We assume that the reader knows a minimal background on permutations and groups. For the sake of simplicity we restricted the study to binary CSPs, however, the notion of symmetry remains valuable for non-binary CSPs as well.

3 Semantic symmetry

Because we are interested in two problems in CSPs: the problem of finding a solution and the problem of finding all the solutions of the CSP, we define two levels of semantic symmetry.

Definition 3.1 (Semantic symmetry for consistency) *Two variable-value pairs $\langle v_i, b_i \rangle \in V \times D_i$ and $\langle v_j, c_j \rangle \in V \times D_j$ are symmetrical for consistency iff the following assertions are equivalent:*

1. *There is a solution of the CSP which assigns the value b_i to the variable v_i ;*

2. *There is a solution of the CSP which assigns the value c_j to the variable v_j .*

Variable-value pairs can be not only symmetrical for consistency, but symmetrical for the set of all solutions as well. Thus, if $sol(\mathcal{P})$ denotes the set of solutions of the CSP \mathcal{P} , then we define a second level of semantic symmetry as follows:

Definition 3.2 (Semantic symmetry for all solutions) *Two variable-value pairs $\langle v_i, b_i \rangle \in V \times D_i$ and $\langle v_j, c_j \rangle \in V \times D_j$ are symmetrical for $sol(\mathcal{P})$ if and only if each solution of the CSP assigning the value b_i to v_i can be mapped into a solution assigning the value c_j to v_j and vice-versa.*

This means that the set of solutions in which the assignment $v_i = b_i$ participates is isomorphic to the one in which $v_j = c_j$ participates. These are symmetrical solutions.

Remark 3.1 1. *If the variables v_i and v_j designate a same variable, then both previous definitions concern symmetry of values in a same domain.*

2. *Symmetry for all solutions implies symmetry for consistency.*

Identifying semantic symmetry is clearly time consuming, since this requires solving the problem. We study in the next section the syntactical symmetry notion which is more tractable computationally and which is a sufficient condition to handle semantic symmetry.

4 Syntactic symmetry

In [Benhamou, 1994], the author studied syntactical symmetry of values of a same CSP domain variable, here syntactic symmetry is extended to the possible variable-value pairs of the CSP. This leads to a similar definition as the one of constraint symmetry given in [Cohen *et al.*, 2005]

Definition 4.1 *A syntactical symmetry of a CSP $\mathcal{P} = (V, D, C, R)$ having the microstructure $\mathcal{M}_{\mathcal{P}}$, is a mapping $\sigma : V \times \cup_{i \in [1, n]} D_i \rightarrow V \times \cup_{i \in [1, n]} D_i$, that preserves the edges and the non-edges of $\mathcal{M}_{\mathcal{P}}$.*

Remark 4.1 *A syntactical symmetry of a CSP \mathcal{P} is an automorphism of its microstructure $\mathcal{M}_{\mathcal{P}}$. The set of syntactic symmetries of a CSP \mathcal{P} is identical to the set of its constraint symmetries [Cohen *et al.*, 2005] which is equivalent to the automorphism group $Aut(\mathcal{M}_{\mathcal{P}})$ of its microstructure. Syntactical symmetries preserve the solutions of the CSP.*

Definition 4.2 *Two variable-value pairs $\langle v_i, b_i \rangle \in V \times D_i$ and $\langle v_j, c_j \rangle \in V \times D_j$ are syntactically symmetrical iff there exists a syntactical symmetry σ of \mathcal{P} , such that $\sigma(\langle v_i, b_i \rangle) = \langle v_j, c_j \rangle$.*

Theorem 4.1 *If two variable-value pairs $\langle v_i, b_i \rangle \in V \times D_i$ and $\langle v_j, c_j \rangle \in V \times D_j$ are syntactically symmetrical, then they are semantically symmetrical for all solutions of the CSP.*

Proof 1 *It is based on the fact that syntactical symmetry preserves solutions.*

4.1 The weakened syntactic symmetry conditions

A weakened symmetry condition has been defined in [Benhamou and Saïdi, 2006; 2005] for the restricted framework of Not-equals CSPs and had been shown to be useful in practice. Here, we show how to extend this weakened condition to General CSPs. Before doing that, we define the notion of assignment trees and failure trees corresponding to the enumerative search method used to prove the consistency of a considered CSP.

Definition 4.3 We call an assignment tree of a CSP \mathcal{P} corresponding to a given search method and a fixed variable ordering, a tree which gathers the history of all the variable assignments made during search, where the nodes represent the variables of the CSP and where the edges outgoing from a node v_i are labeled by the different values used to instantiate the corresponding CSP variable v_i .

The root of the tree is the first variable in the ordering. In this work, the considered search method is Forward Checking [Haralik and Elliot, 1980].

In an assignment tree of a CSP, a path connecting the root of the tree to a node defines a partial instantiation \mathcal{I} of the CSP. The variables of the partial instantiation \mathcal{I} are the nodes of the considered path. The last node $n_{\mathcal{I}}$ of the path corresponds to the last affected variable in the instantiation or to a variable having an empty domain.

We associate to each inconsistent partial instantiation, corresponding to a given path in the assignment tree, a failure tree defined as follows:

Definition 4.4 Let T be an assignment tree of the CSP \mathcal{P} , $\mathcal{I} = (\langle v_1, a_1 \rangle, \langle v_2, a_2 \rangle, \dots, \langle v_i, a_i \rangle)$ an inconsistent partial instantiation of the variables v_1, v_2, \dots, v_i corresponding to the path $\{v_1, v_2, \dots, v_i\}$ in T . We call a failure tree of the instantiation \mathcal{I} , the sub-tree of T denoted by $T_{\mathcal{I}}$ such that:

1. The root of the tree T and the root of the sub-tree $T_{\mathcal{I}}$ are joined by the path corresponding to the instantiation \mathcal{I} ;
2. All the CSP variables corresponding to the leaf nodes of $T_{\mathcal{I}}$ have empty domains.

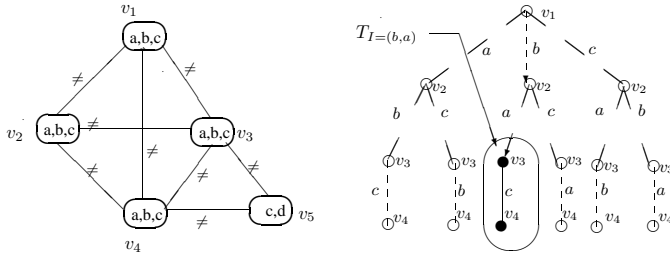


Figure 1: The constraint graph of a graph coloring instance, its assignment tree and the failure tree of $\mathcal{I}=(\langle v_1, b \rangle, \langle v_2, a \rangle)$

Example 4.1 Take the CSP on the left part of Figure 1 and apply a Forward Checking process on it w.r.t the variable ordering $\{v_1, v_2, v_3, v_4, v_5\}$. The right part of Figure 1 illustrates the assignment tree of the considered CSP. If we take

the partial instantiation $\mathcal{I} = (\langle v_1, b \rangle, \langle v_2, a \rangle)$, then the failure tree $T_{\mathcal{I}}$ of the instantiation \mathcal{I} is the part of the assignment tree shown in a box.

We can now give the weakened conditions of syntactic symmetry. The main idea is to weaken the syntactic symmetry conditions when an inconsistent partial instantiation is generated during the search. That is, for a local CSP $\mathcal{P}_{\mathcal{I}}$ where the partial instantiation \mathcal{I} is inconsistent, the conditions of syntactic symmetry (Definition 4.1) are restricted to only the variables involved in the failure tree $T_{\mathcal{I}}$, rather than to all the un-instantiated variables.

Theorem 4.2 Let $\mathcal{P}(V, C, D, R)$ be a CSP, $\mathcal{I}_0 = (\langle v_1, a_1 \rangle, \dots, \langle v_{i-1}, a_{i-1} \rangle)$ a partial instantiation of $i-1$ variables instantiated before the current variable v_i such that the extension $\mathcal{I} = \mathcal{I}_0 \cup \{\langle v_i, a_i \rangle\}$ is inconsistent, $T_{\mathcal{I}}$ is the failure tree of \mathcal{I} and $\text{Var}(T_{\mathcal{I}})$ the set of the variables corresponding to the nodes of $T_{\mathcal{I}}$. If $\langle v_i, a_i \rangle$ is syntactically symmetrical to $\langle v_j, b_j \rangle$ in the CSP $\hat{\mathcal{P}}_{\mathcal{I}_0}$ derived from $\mathcal{P}_{\mathcal{I}_0}$ by restricting its set of variables to $\text{Var}(T_{\mathcal{I}}) \cup \{v_i\}$, then the extension $\mathcal{J} = \mathcal{I}_0 \cup \{\langle v_j, b_j \rangle\}$ is inconsistent.

Proof 2 The CSP $\hat{\mathcal{P}}_{\mathcal{I}_0}$ is $\mathcal{P}_{\mathcal{I}_0}$ where the set of variables is restricted to $\text{Var}(T_{\mathcal{I}}) \cup \{v_i\}$. By the hypothesis, $T_{\mathcal{I}}$ is a failure tree of \mathcal{I} in \mathcal{P} . This implies that the assignment of the value a_i to v_i leads to a failure in $\hat{\mathcal{P}}_{\mathcal{I}_0}$. In other words, $\langle v_i, a_i \rangle$ does not participate in any solution of $\hat{\mathcal{P}}_{\mathcal{I}_0}$. By the hypothesis, the pair $\langle v_i, a_i \rangle$ is symmetrical to $\langle v_j, b_j \rangle$ in $\hat{\mathcal{P}}_{\mathcal{I}_0}$. This implies that $\langle v_j, b_j \rangle$ does not participate in any solution of $\hat{\mathcal{P}}_{\mathcal{I}_0}$. Thus $\langle v_j, b_j \rangle$ does not participate in any solution of $\mathcal{P}_{\mathcal{I}_0}$ as well. This implies that the partial instantiation $\mathcal{J} = \mathcal{I}_0 \cup \{\langle v_j, b_j \rangle\}$ is inconsistent in \mathcal{P} . (QED)

Remark 4.2 In the case of a failure during search, the symmetry conditions are restricted to only the variables participating in the failure.

Theorem 4.2 gives an interesting weakening of the conditions of syntactic symmetry that we can use when the current partial instantiation leads to an inconsistency. By using the new conditions, some symmetries not captured by the normal conditions can result from these weakened conditions. Let us consider for instance the CSP of Figure 1. If we take the partial instantiation $\mathcal{I}_0 = \langle v_1, b \rangle$ and the inconsistent extension $\mathcal{I} = \mathcal{I}_0 \cup \{\langle v_2, a \rangle\}$, then the pairs $\langle v_2, a \rangle$ and $\langle v_2, c \rangle$ are symmetrical. That is, the two values a and c of the domain of the current variable v_2 are symmetrical w.r.t the weakened conditions (Theorem 4.2) applied on the CSP $\hat{\mathcal{P}}_{\mathcal{I}_0}$ involving the set of variables $\text{Var}(T_{\mathcal{I}}) \cup v_2 = \{v_2, v_3, v_4\}$, whereas the normal symmetry conditions are not verified on the CSP $\mathcal{P}_{\mathcal{I}_0}$ involving the set of all un-instantiated variables $\{v_2, v_3, v_4, v_5\}$. The branch corresponding to the assignment of c to v_2 is not explored during search thanks to Theorem 4.2. This defines a more powerful symmetry cut that we use to shorten CSP search trees.

Besides, this weakening property can be used in other known symmetry breaking methods [Benhamou and Sais, 1992; 1994; Benhamou et al., 1994]. That is, symmetry conditions have to be checked only on the variables involved in

the failure when a partial instantiation is shown to be inconsistent.

Below we show how local symmetry is detected and eliminated, and how a tree search method (here, Forward Checking) can take advantage of symmetry.

5 Local symmetry detection and exploitation

5.1 Symmetry detection and breaking

Local symmetries have to be detected dynamically at each node of the search tree. Dynamic symmetry detection has been studied in CSPs, a local syntactic domain symmetry search method had been given in [Benhamou, 1994].

As an alternative to this symmetry search method, we adapted Saucy [Aloul *et al.*, 2003] to detect local syntactic symmetries and show how to break such symmetries during search. Saucy is a tool for computing the automorphism group of a graph. Other tools like Nauty [McKay, 1981] or the most recent methods AUTOM [Puget, 2005a] or the one described in [Mears *et al.*, 2006] can be adapted to search local symmetry. It is shown in [Puget, 2005a] that AUTOM is the best method. Because the source code of AUTOM is not free, we chose Saucy. Since the syntactic symmetry group of a CSP \mathcal{P} is identical to the automorphism group of its microstructure $\mathcal{M}_{\mathcal{P}}$, we can use Saucy on $\mathcal{M}_{\mathcal{P}}$ to detect the syntactic symmetry group of \mathcal{P} . Saucy returns a set of generators Gen of the symmetry group from which we can deduce each symmetry. Saucy offers the possibility to color the microstructure such that, a node is allowed to be permuted with another node if they have the same color. This restricts the permutations to the nodes having the same color. We use this coloration possibility to guide the symmetry search and detect local value symmetries. The source code of Saucy can be found at (<http://vlsicad.eecs.umich.edu/BK/SAUCY/>).

Symmetry detection:

Consider a CSP \mathcal{P} , and a partial instantiation \mathcal{I} of \mathcal{P} , defining a state in the search corresponding to the current node $n_{\mathcal{I}}$. The main idea is to maintain dynamically the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ of the CSP $\mathcal{P}_{\mathcal{I}}$ corresponding to the local sub-problem defined at each current node $n_{\mathcal{I}}$, then color the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ and compute its automorphism group $Aut(\mathcal{M}_{\mathcal{P}_{\mathcal{I}}})$. The CSP $\mathcal{P}_{\mathcal{I}}$ can be viewed as a new problem corresponding to the unsolved part. Because, computing all the automorphisms of the dynamic microstructure at each node of the search tree may be expensive, two limited symmetry detection strategies are considered in [Benhamou and Saïdi, 2007]. Indeed, two coloration strategies of the microstructure are used: the multi-color strategy and the two-color strategy. Below we summarize both previous strategies and introduce the *one-color* strategy that allows full local symmetry detection and compare it to the two other strategies.

1. **The multi-colors-strategy:** We limited the permutations to only values of the same domains. To do that, a color is associated to each variable. Every node of the microstructure belonging to a variable is colored with the same color. Then, by applying Saucy on this colored microstructure we can get the generator set Gen of the symmetry sub-group existing between values of the same domains of the CSP.

2. **The two-colors-strategy:** Here, we associated to the current variable v_i (under instantiation) one color and all the other variables another color. That is, we colored the dynamic microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ with two colors. All the nodes of the microstructure belonging to the current variable v_i have the first color and all the other nodes the second one. Finally, we applied Saucy to compute the generators of the automorphism sub-group corresponding to this coloration. This returns the generators Gen of the symmetry group allowing variable-value permutations on the other variables different from v_i , but the values of v_i are permuted together.
3. **The one-color-strategy:** The total local symmetry group is reached when using only one color on the microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$. Here, all the nodes of the microstructure are assigned the same color, then when applying Saucy a set of generators Gen representing the full local symmetry group is returned. This approach allows to detect all the syntactical variable-value symmetries of the CSP.

Symmetry elimination:

We use Theorem 4.2 to prune search spaces of tree search methods. According to the different symmetry detection strategies we described previously, we distinguish two elimination methods:

- **Full symmetry elimination:** this method is used when applying the one-color symmetry detection strategy. That is, if the assignment $\langle v_i, b_i \rangle$ of the current variable v_i at a given node $n_{\mathcal{I}}$ of the search tree is shown to participate in no solution of the CSP \mathcal{P} , then all the pairs $\langle v_j, c_j \rangle$ which are symmetrical to $\langle v_i, b_i \rangle$ in $\mathcal{P}_{\mathcal{I}}$ do not (i.e. these pairs are the ones corresponding to the orbit of the conflicted pair $\langle v_i, b_i \rangle$ that can be computed by using only the symmetry group generators). Then we remove the value c_j from the domain of the un-instantiated variable v_j , and prune the sub-space which corresponds to its assignment to v_j in the search tree.
- **Limited symmetry elimination:** this method is used when applying one of the two other strategies. Here, for both strategies, the variable v_i and v_j are the same, and the previous reasoning handles symmetries between values of the domain D_i . The domain D_i of the current variable v_i is partitioned into sub-sets of symmetrical values *w.r.t* the detected local symmetries at the corresponding node of the search tree. To avoid generating local symmetrical solutions, we consider one value from each sub-set of symmetrical values in D_i .

If we need to check CSP consistency only, we stop the search when a first solution is found.

5.2 Symmetry advantage in tree search algorithms

Now, we are in the position to show how these symmetrical values can be used to increase the efficiency of CSP tree search algorithms. We choose in our implementation the Forward Checking method to be the baseline method that we want to improve by local symmetry elimination. The resulting procedure called FC-sym is given in Figure 2.

If \mathcal{I} is an inconsistent partial instantiation in which the assignment $\langle v_i, d_i \rangle$ of the current variable v_i is shown to participate in no solutions of the CSP \mathcal{P} , then according to Theorem 4.2, all the pairs $\langle v_j, d_j \rangle$ which are symmetrical to $\langle v_i, d_i \rangle$ in $\mathcal{P}_{\mathcal{I}}$ do not. Thus we remove d_j from the domain of v_j , and prune the sub-space which corresponds to its assignment to v_j .

```

Procedure FC-sym( $D, \mathcal{I}, k$ );  $\{\mathcal{I} = [\langle v_1, d_1 \rangle, \langle v_2, d_2 \rangle, \dots, \langle v_k, d_k \rangle]\}$ 
begin
  if  $k = n$  then  $\mathcal{I}$  is a solution, return( $\mathcal{I}$ )
  else
    begin
      for each  $v_i \in V$ , such that  $C_{i,k} \in C$ ,  $v_i \in \text{future}(v_k)$  do
        for each value  $d_i \in D_i$  do
          if  $(d_i, d_k) \notin r_{i,k}$  then
            delete  $d_i$  from  $D_i$ ;
        if  $\forall v_i \in \text{future}(v_k)$ ,  $D_i \neq \emptyset$  then
          begin
             $v_{k+1} = \text{next-variable}(v_k)$ 
            repeat
              take  $d_{k+1} \in D_{k+1}$ 
               $D_{k+1} = D_{k+1} - \{d_{k+1}\}$ 
               $\mathcal{I} = \mathcal{I} \cup \{\langle v_{k+1}, d_{k+1} \rangle\}$ ;
               $\mathcal{J} = \text{FC-sym}(D, \mathcal{I}, k+1)$ ;
               $\mathcal{I} = \mathcal{I} - \{\langle v_{k+1}, d_{k+1} \rangle\}$ ;
              if  $\mathcal{J} \in \text{Sol}(\mathcal{P})$  then  $\text{Gen} = \text{Saucy}(\mathcal{P}_{\mathcal{I}})$ ;
              else  $\text{Gen} = \text{Saucy}(\mathcal{P}_{\text{Var}(\mathcal{T}_{\mathcal{I}}) \cup v_{k+1}})$ ;
               $\text{SymClass}(\langle v_{k+1}, d_{k+1} \rangle) = \text{orbit}(\langle v_{k+1}, d_{k+1} \rangle, \text{Gen})$ ;
               $D_{k+1} = D_{k+1} - \text{SymClass}(\langle v_{k+1}, d_{k+1} \rangle)$ 
            until  $D_{k+1} = \emptyset$ 
          end
        end
      end
    end
  end;

```

Figure 2: Forward Checking method with symmetry

The function $\text{orbit}(\langle v_{k+1}, d_{k+1} \rangle, \text{Gen})$ is elementary, it computes the orbit of the pair $\langle v_{k+1}, d_{k+1} \rangle$ from the set of generators Gen returned by Saucy.

6 Experiments

Now, we shall investigate the performances of our search techniques by experimental analysis. We choose for our study some classical problems to show the local symmetry behavior in CSP resolution. We expect that symmetry breaking will be more profitable in real-life applications. Here, we tested and compared five methods:

1. **No-sym**: search without symmetry breaking;
2. **Global-sym**: search with global symmetry breaking restricted to values in a same domain. The same symmetries as the ones considered in the GE-tree method, with a slight difference that we break only global symmetries between values in a same domain;
3. **Local-sym0**: search with full local symmetry detection and elimination. This method implements the one-color strategy. one-color (see Section 5.1).
4. **Local-sym1**: search with local value symmetry breaking. This method implements the multi-colors strategy (see Section 5.1).
5. **Local-sym2**: search with restricted local variable-value symmetry breaking. This method implements the two-colors strategy (see Section 5.1).

on different problems: random graph coloring problems, Dimacs graph coloring instances and n -Queens problems. An

implementation of the *Local – sym1* strategy in GECODE system is successfully used in [Zampelli *et al.*, 2007] to break local symmetry in the subgraph pattern matching problem. The common baseline search method for the five previous methods is Forward Checking. The complexity indicators are the number of nodes of the search tree and the CPU time. The time needed for computing symmetries is added to the total CPU time. The source codes are written in C and compiled on a Pentium 4, 2.8 GHZ and 1 Go of RAM.

6.1 Random graph coloring problems

Random graph coloring problems are generated with respect to the following parameters: (1) n : the number of vertices (variables), (2) *Colors*: the number of colors (domain values) and (3) d : the density which is a number between 0 and 1 expressed by the ratio : the number of constraints (the number of edges in the constraint graph) to the number of all possible constraints. For each test corresponding to some fixed values of the parameters n , *Colors* and d , a sample of 100 instances are randomly generated and the measures (CPU time, nodes) are taken on the average.

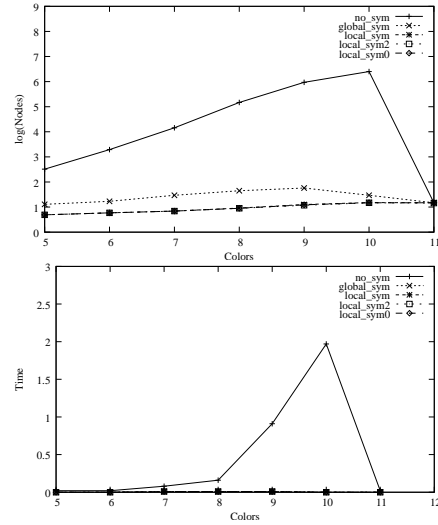


Figure 3: Node and Time curves where $n = 15$ and $d = 0.9$

Figure 3, shows the performances of the five methods in number of nodes of the search tree, respectively, in CPU time (in seconds) on random graph coloring problems, whose number of variables is fixed to $n = 15$ and the density to $d = 0.9$. We reported here experiments on instances having high density, because they are the hardest instances, and symmetry presents a similar behavior for average and weak density instances. The curves on the top are plotted in a logarithmic scale, they represent the performances in number of nodes *w.r.t* the number of colors. The ones on the bottom are plotted in the usual scale and express the performances in CPU time *w.r.t* the number of colors. As expected, we can see that all the methods exploiting symmetry outperform dramatically the search without symmetry (No-sym) in both the number of nodes and the CPU time. We can also see on the node curves that local symmetry elimination (Local-

sym0, Local-sym1 and Local-sym2) reduces more the search tree than global symmetry elimination (Global-sym). That is, local symmetries are more frequent during the search than the global symmetries stabilizing the partial instantiation. The tree methods exploiting local symmetries have the same behavior in number of nodes; their node curves are almost identical. We can distinguish on the CPU time curve of No-sym a critical region where the instances are harder. All the methods using symmetry solved these instances in less than 0.1 seconds, then their CPU time curves are confused with x-axis and do not appear.

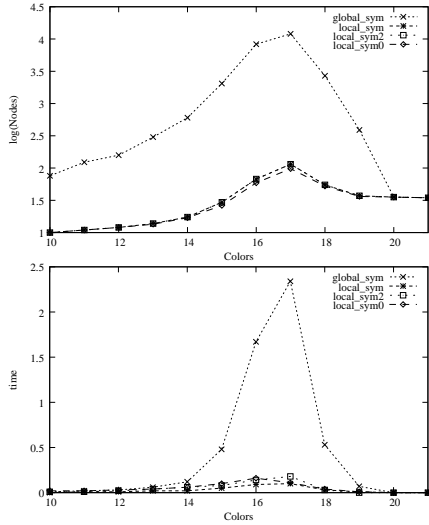


Figure 4: Node and Time curves of the three symmetry methods on random graph coloring where $n = 35$ and $d = 0.9$

Since Figure 3 does not allow a CPU time comparison of the methods exploiting symmetry, we reported in Figure 4 the practical results of the methods: Global-sym, Local-sym0, Local-sym1 and Local-sym2, on the random graph coloring problem where the number of variables is increased to $n = 35$ and where we keep the same density ($d = 0.9$) as in Figure 3.

We can see on the node curves (the curves on the top plotted in a logarithmic scale) that Local-sym0, Local-sym1 and Local-sym2 detect and eliminate more symmetries than the Global-sym method. Once again, the reason is that the local symmetry detected at a node during the search by these methods includes the global symmetry stabilizing the partial instantiation at that node exploited by Global-sym. The node curves of both Local-sym1 and Local-sym2 compare well, and the one of Local-sym0 looks slightly better than both them. From the CPU time curves (the curves on the bottom), we can see that near the peak of difficulty, all of Local-sym0, Local-sym1 and Local-sym2 are faster than Global-sym. We can also see that Local-sym0 is slightly faster than both Local-sym1 and Local-sym2. Therefore, Local symmetry elimination is profitable for solving random graph coloring instances in the hard region and outperforms dramatically global symmetry breaking on these problems.

Instance	k	No-sym		Global-sym		Local-sym0	
		Nodes	Time	Nodes	Time	Nodes	Times
myciel4	5	30,976	0.16	2,764	0.03	992	0.0
myciel5	6	-	-	8,040,259	59.84	1,516,699	16.87
anna	11	-	-	3,403	0.59	151	0.03
david	11	-	-	3,896	0.23	106	0.0
queen7.7	7	2,452	0.01	513	0.02	502	0.0
queen8.8	9	-	-	10,629,131	262.54	859,087	23.76
school1	14	-	-	-	-	45,183	11.82
school1_nsh	14	-	-	-	-	892,372	174.76
2-Insertion_3	4	832,150	1.02	277,408	0.73	19,646	0.07
2-FullIns_3	5	2,294,396	7.63	193,347	1.14	25,049	0.4
mugg88_1	4	-	-	-	-	1,981,671	23.87
mugg88_25	4	-	-	-	-	440,676	5.62
mugg100_1	4	-	-	-	-	1,031,382	14.13
mugg100_25	4	-	-	-	-	1,651,598	17.2
zeroin.i.1	49	-	-	-	-	268	36.8
zeroin.i.2	30	-	-	-	-	262	7.22
zeroin.i.3	30	-	-	-	-	262	7.23
multsol.i.2	31	-	-	-	-	237	11.24
multsol.i.3	31	-	-	-	-	237	11.1
le450_5a	5	178,753	13.88	170,123	13.75	165,169	35.2
le450_5b	5	1,349	0.11	1,110	0.09	967	0.22
le450_5c	5	1,984	0.15	1,984	0.17	1,975	0.34
le450_5d	5	5,795	0.54	4,563	0.34	3,433	0.69
DSJC125.1	5	55,358	0.85	43,773	1.34	40,542	1.46

Instance	k	Local-sym1		Local-sym2		Local-sym0	
		Nodes	Time	Nodes	Time	Nodes	Times
myciel4	5	1,260	0.01	1,260	0.04	992	0.0
myciel5	6	2,413,556	22.21	2,406,945	25.36	1,516,699	16.87
anna	11	168	0.05	168	0.08	151	0.03
david	11	124	0.03	124	0.03	106	0.0
queen7.7	7	502	0.01	502	0.0	502	0.0
queen8.8	9	1,399,436	29.7	1,396,774	30.16	859,087	23.76
school1	14	76,192	17.28	75,985	17.85	45,183	11.82
school1_nsh	14	1,487,287	257.57	1,486,523	270.4	892,372	174.76
2-Insertion_3	4	135,953	0.48	115,737	0.52	19,646	0.07
2-FullIns_3	5	49,202	0.59	48,076	0.65	25,049	0.4
mugg88_1	4	2,882,284	53.91	2,882,284	93.55	1,981,671	23.87
mugg88_25	4	881,784	6.74	881,784	9.4	440,676	5.62
mugg100_1	4	3,325,453	24.85	3,325,453	40.15	1,031,382	14.13
mugg100_25	4	2,727,178	17.3	2,727,178	30.92	1,651,598	17.2
zeroin.i.1	49	268	7.0	268	35.49	268	36.8
zeroin.i.2	30	262	0.75	262	3.675	262	7.22
zeroin.i.3	30	262	0.76	262	3.675	262	7.23
multsol.i.2	31	237	0.85	237	10.14	237	11.24
multsol.i.3	31	237	0.9	237	10.14	237	11.1
le450_5a	5	167,787	32.0	167,703	32.23	165,169	35.2
le450_5b	5	927	0.11	927	0.19	967	0.22
le450_5c	5	1,983	0.31	1,975	0.34	1,975	0.34
le450_5d	5	3,452	0.62	3,452	0.68	3,433	0.69
DSJC125.1	5	40,809	1.44	40,809	1.48	40,542	1.46

Table 1: Results on some Dimacs graph coloring benchmarks

6.2 Dimacs graph coloring benchmarks

Here, we tested and compared the five methods on some graph coloring benchmarks taken from the Dimacs challenge (<http://mat.gsia.cmu.edu/COLOR04/>).

Table 1 shows the results of the methods on some of the benchmarks. It gives the instance, the chromatic number found (k), the number of nodes of the search tree and the CPU time for each method. We seek for each instance the minimal number k of colors needed to color the vertices of the corresponding graph (called the chromatic number). The search of the chromatic number consists in proving the consistency of the problem with k colors (the existence of a k -coloration of the graph); and in proving its inconsistency when using $k - 1$ colors (a $(k-1)$ -coloration does not exist). The symbol "-" means that the corresponding method does not solve the instance in one hour.

Table 1 shows that both No-sym and Global-sym are not able to solve several instances under the time limit, but Global-sym is better than No-sym in both numbers of nodes and CPU time on these problems. We can see that all the methods exploiting local symmetry are in general better than Global-sym in both the number of nodes and the CPU time.

That is, local symmetry is more profitable than global symmetry on these problems. We can also see that Local-sym1 and Local-sym2 eliminates nearly the same symmetries, but Local-sym0 eliminate more symmetries than both them, and is still faster than all of them. This confirms that eliminating local domain value symmetries by using the one-color strategy implemented in Local-sym0 is a good alternative on these problems.

6.3 The n -Queens Problems

Finding *all* solutions of the n -queens problem is still a challenge. We compared the five methods on some instances of this problem.

n	No-sym			Global-sym			Local-sym0		
	Sols	Nodes	Time	Sols	Nodes	Times	Sols	Nodes	Times
8	92	1,360	0.0	46	680	0.01	45	660	0.01
9	352	5,399	0.0	179	2,800	0.0	168	2,614	0.05
10	724	19,744	0.03	362	9,872	0.03	353	9,592	0.12
11	2,680	85,939	0.1	1,382	43,958	0.07	1,305	41,972	0.6
12	14,200	416,828	0.28	7,100	208,414	0.25	6,839	203,120	2.67
13	73,712	2,154,845	2.69	37,361	1,093,606	1.99	35,310	1,050,401	12.88
14	365,596	11,799,746	46.95	51,726	5,899,873	20.65	43,245	5,750,997	78.85

n	Local-sym1			Local-sym2			Local-sym0		
	Sols	Nodes	Time	Sols	Nodes	Times	Sols	Nodes	Times
8	45	664	0.01	45	662	0.01	45	660	0.01
9	172	2,645	0.02	168	2,625	0.05	168	2,614	0.05
10	355	9,656	0.07	353	9,640	0.08	353	9,592	0.12
11	1,309	42,154	0.25	1,305	42,078	0.31	1,305	41,972	0.6
12	6,883	204,901	2.05	6,839	203,611	2.19	6,839	203,120	2.67
13	35,525	1,055,366	11.44	35,312	1,053,053	11.58	35,310	1,050,401	12.88
14	44,334	5,777,244	69.6	43,257	5,765,594	75.6	43,245	5,750,997	78.85

Table 2: Results on the n -queens problem

Table 2 summarizes the results obtained. For each method we give the number of computed solutions (*Sols*), the number of nodes, and the CPU time in seconds. Note that for the methods exploiting symmetry, the number of solutions (*Sols*) is the number of non-symmetrical solutions found *w.r.t* the applied symmetry breaking strategy. The number of solutions of No-sym is the total set of solutions of the problem. We can see that all of local-sym0, Local-sym1 and Local-sym2 represent the set of solutions slightly in a more compact way than Global-sym. This means that Local-sym0, Local-sym1 and Local-sym2 compact some local symmetrical solutions in addition to the global symmetrical ones compacted by Global-sym. Now, if we compare globally the methods in number of solutions, in the number of nodes and in CPU time, the Global-sym method seems to be the best on the average. Indeed, global symmetry is sufficient to solve efficiently the n -queens problems and local value symmetry does not abound like in the graph coloring. We believe that local variable symmetry will be more profitable for n -queens.

7 Discussion and conclusions

Here, we extended symmetry detection and elimination to local symmetry. That is, the symmetries of each sub-CSP defined at a given node of the search tree and which is derived from the initial CSP by considering the partial instantiation corresponding to that node. We adapted Saucy to compute this local symmetry by maintaining dynamically the microstructure of the sub-CSP defined at each node of the search tree. Unlike the methods using GAP tools, here local symmetry detection is fully automated. Saucy is called with the

microstructure of the local sub-CSP as the input graph, and then return the set of generators of the automorphism group of the microstructure which is shown to be equivalent to the local symmetry group of the considered sub-CSP. We proposed three coloration strategies in order to guide local symmetry search: the multi-color strategy restricts local symmetry search to permutations of values of the same domain, the two color strategy allows some variable-value permutations but limited, and the one-color consider all variable-value permutations. All the local symmetry strategies are implemented and exploited in the tree search method *FC* to prove either CSP consistency or to compute the not-local symmetrical solutions of the CSP. Experimental results confirmed that local symmetry breaking is profitable for CSP solving and improves global symmetry breaking in most of the considered problems.

As a future work, we are interested to adapt our symmetry results for other look-ahead CSP methods like MAC, and export local symmetry breaking to other research domains like biology or operational research to tackle real life applications.

Another interesting point, is to extend our approach to variable local symmetry breaking. One can try to detect local variable symmetries and post dynamic constraints to break them, it will be important to consider the possibilities of combining local variable and local value symmetries.

References

- [Aloul *et al.*, 2003] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakalallah. Solving difficult sat instances in the presence of symmetry. *In IEEE Transaction on CAD*, vol. 22(9), pages 1117–1137, 2003.
- [Aloul *et al.*, 2004] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakalallah. Symmetry breaking for pseudo-boolean satisfiability. *In ASPDAC'04*, pages 884–887, 2004.
- [Backofen and Will, 1999] R. Backofen and S. Will. Excluding symmetries in constraint-based search. *In Principle and Practice of Constraint Programming - CP'99*, 1999.
- [Benhamou and Saïdi, 2005] Belaïd Benhamou and Mohamed Réda Saïdi. Some improvements in symmetry elimination in not-equals binary constraint networks. *In Proceedings of the satellite workshop of CP 2005, Symmetry and Constraint Satisfaction Problems (SymCon'05)*, pages 1–7, Sitges, Spain, October 2005.
- [Benhamou and Saïdi, 2006] Belaïd Benhamou and Mohamed Réda Saïdi. Reasoning by dominance in not-equals binary constraint networks. In LNCS Springer, editor, *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP-2006)*, pages 670–674, Cité des Congrès - Nantes, France, septembre 2006.
- [Benhamou and Saïdi, 2007] Belaïd Benhamou and Mohamed Réda Saïdi. Local symmetry breaking during search in cps. *In Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-2007)*, Providence, USA, september 2007. To appear.

- [Benhamou and Sais, 1992] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *Eleventh International Conference on Automated Deduction, Saratoga Springs, NY, USA*, 1992.
- [Benhamou and Sais, 1994] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning (JAR)*, 12:89–102, 1994.
- [Benhamou *et al.*, 1994] B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language. In *Proceedings of STACS'94, Caen France*, pages 71–82, 1994.
- [Benhamou, 1994] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of the 2nd International workshop on Principles and Practice of Constraint Programming - PPCP'94*, 1994.
- [Cohen *et al.*, 2005] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. In *proceedings of CP*, pages 17–31, 2005.
- [Crawford *et al.*, 1996] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.
- [Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.
- [Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.
- [Gent and Smith, 2000] I. P. Gent and B. M. Smith. Symmetry breaking during search in constraint programming. In *In Proceedings of ECAI'2000*, 2000.
- [Gent *et al.*, 2002] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 415–430. Springer Verlag, 2002.
- [Gent *et al.*, 2003] I. P. Gent, W. Hervey, T. Kesley, and S. Linton. Generic sbdd using computational group theory. In *Proceedings CP'2003*, 2003.
- [Gent *et al.*, 2005] I. P. Gent, T. Kelsey, S. A. Linton, I. McDonald, I. Migeul, and B. Smith. Conditional symmetry breaking. In *Principle and Practice of Constraint Programming - CP 2005*, pages 256–270, 2005.
- [Haralik and Elliot, 1980] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, pages 263–313, 1980.
- [Jegou, 1993] P. Jegou. Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In *In Proceedings of AAAI'93*, 1993.
- [Krishnamurty, 1985] B. Krishnamurty. Short proofs for tricky formulas. *Acta informatica*, (22):253–275, 1985.
- [McKay, 1981] B McKay. Practical graph isomorphism. In *Congr. Numer. 30*, pages 45–87, 1981.
- [Mears *et al.*, 2006] C. Mears, M. Garcia de la Banda, and M. Wallace. On implementing symmetry detection. In *The CP 2006 Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'06)*, pages 1–8, Cité des Congrès - Nantes, France, septembre 2006.
- [Puget, 1993] J. F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *In Proceedings of ISMIS'93, LNAI 689*, 1993.
- [Puget, 2002] J.F. Puget. Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2002.
- [Puget, 2005a] J F Puget. Automatic detection of variable and value symmetries. In *LNCS Springer, editor, Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 474–488, Sitges, Spain, october 2005.
- [Puget, 2005b] J.F. Puget. Breaking all value symmetries in surjection problems. In *Proceedings of CP'05*, pages 490–504, 2005.
- [Puget, 2005c] J.F. Puget. Breaking symmetries in all different problems. In *proceedings of IJCAI*, pages 272–277, 2005.
- [Puget, 2006] J.F. Puget. Dynamic lex constraints. In *proceedings of CP'06*, pages 453–467, 2006.
- [Roney-Dougal *et al.*, 2004] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. A. Linton. Tractable symmetry breaking using restricted search trees. In *Proceedings of ECAI'04*, pages 211–215, 2004.
- [Walsh, 2006] T. Walsh. General symmetry breaking constraints. In *proceedings of CP'06*, pages 650–664, 2006.
- [Zampelli *et al.*, 2006] S. Zampelli, Y. Deville, and P. Dupont. Symmetry breaking in subgraph pattern matching. In *SymCon'06*, pages 35–42, 2006.
- [Zampelli *et al.*, 2007] Stéphane Zampelli, Yves Deville, Mohamed Réda Saïdi, Belaid Benhamou, and Pierre Dupont. Breaking local symmetries in subgraph pattern matching. In *The International Symmetry Conference (ISC 2007)*, Edinburgh, Scotland, january 2007.

A Novel Approach For Detecting Symmetries In CSP Models

B. Demoen

Katholieke Universiteit Leuven, Belgium
bmd@cs.kuleuven.ac.be

M. Garcia de la Banda, C. Mears, M. Wallace

Monash University, Australia
mbanda,cmears,wallace@infotech.monash.edu.au

Abstract

While several powerful methods exist for automatically detecting symmetries in *instances* of constraint satisfaction problems (CSPs), methods for detecting symmetries in CSP *models* are constrained to finding relatively simple symmetries, or operating on a narrow range of problems. Herein, a new approach for detecting symmetries in CSP models is presented. The approach is based on first applying powerful methods to a sequence of instances of the model, and then reasoning on the resulting instance symmetries to infer symmetries of the model. Several case studies show that this approach deserves further exploration.

1 Introduction

Constraint satisfaction problems (CSPs) can often be separated into two parts. The *model* specifies the variables, their domains, and the set of constraints that operate on the variables, but is usually parametrised by the particular number of variables, values and, thus, constraints. The *data* provides concrete values to these parameters. As a result, the model in itself represents a *class* of CSPs, while the model plus the data specifies an *instance* of that class (i.e., a particular CSP).

For example, a graph colouring model might be defined in terms of a number of variables (i.e., nodes), each coloured according to a given set of values (i.e., allowed colours), and a set of constraints indicating that no edge can join nodes of the same colour. An instance of the problem is specified by supplying a particular graph (number of nodes and edges among them) and the set of colours. Note that each instance might have a different number of variables and/or constraints.

Solving a CSP can be made more efficient by exploiting the symmetries of the problem. This is because, during search, one can omit parts of the search space that are symmetric to others already explored. If these already explored parts led to a solution, one would have avoided the time spent in searching for symmetric solutions, since they can be generated by applying the symmetries to the already found solutions. If they led to

failure, one would have avoided the time spent in discovering failure yet again.

Considerable progress has been made in the automatic detection of symmetries of CSPs and their exploitation in speeding up the search (e.g., [Mears *et al.*, 2006; Cohen *et al.*, 2005; Puget, 2005; Walsh, 2006; Romani and Markov, 2005; Sellmann and Hentenryck, 2005; Mancini and Cadoli, 2005; Roney-Dougal *et al.*, 2004; Frisch *et al.*, 2003; Gent *et al.*, 2003; Puget, 2002; Gent and Smith, 2000; Haselböck, 1993]). Unfortunately, the most powerful methods [Puget, 2005; Cohen *et al.*, 2005] can only be applied to a given instance, rather than to a model. Therefore, the symmetries detected can only be used to accelerate the solving process for that instance and, as a result, the cost of detecting them cannot be amortised over all instances of the class.

Furthermore, the computation costs of these methods grow with the size of the problem instance in such a way as to render them impractical for real-size instances. For example, the most powerful and generic technique for symmetry detection [Cohen *et al.*, 2005] requires a representation of the “micro-structure” of the instance – a graph with a node for every possible value of every variable, and a hyper-edge between each set of compatible (or incompatible) nodes. Clearly, the size of this structure can grow dramatically with the size of the problem instance.

While some automatic symmetry detection methods are defined for models [Roy and Pache, 1998; van Hentenryck *et al.*, 2005], to our knowledge, they can only detect a relatively small set of “simple” symmetries (i.e., piecewise value and piecewise variable interchangeability), and heavily depend on the precise form of the problem model (i.e., use of global constraints). Instead, we build on powerful symmetry detection techniques designed for problem instances to discover symmetries for models. This is achieved by (1) using symmetry detection methods on a series of small problem instances to elicit candidate symmetries, (2) parametrising these candidate symmetries to be defined over the model rather than over a particular instance, and (3) determining whether these are indeed problem model symmetries.

2 Background and Definitions

A CSP is a tuple (X, D, C, dom) where X represents a set of variables, D a set of domains, C a set of constraints, and where dom is a function from X to D , so that $dom(x) \in D$ denotes the domain of variable $x \in X$. By an abuse of notation, when all variables have the same domain, D will simply denote this domain and the dom -function is usually omitted.

Example 1 Consider the Latin square problem of size 3, which involves a 3×3 square where each of the 9 elements in the square must take a value from $[1..3]$, in such a way that each value occurs exactly once in each row and exactly once in each column. The associated CSP can be defined as

$$\begin{aligned} X &= \{x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}\} \\ D &= \{1, 2, 3\} \\ C &= \{x_{11} \neq x_{12}, x_{11} \neq x_{13}, x_{12} \neq x_{13}, x_{21} \neq x_{22}, \\ &\quad x_{21} \neq x_{23}, x_{22} \neq x_{23}, x_{31} \neq x_{32}, x_{31} \neq x_{33}, \\ &\quad x_{32} \neq x_{33}, x_{11} \neq x_{21}, x_{11} \neq x_{31}, x_{21} \neq x_{31}, \\ &\quad x_{12} \neq x_{22}, x_{12} \neq x_{32}, x_{22} \neq x_{32}, x_{13} \neq x_{23}, \\ &\quad x_{13} \neq x_{33}, x_{23} \neq x_{33}\} \end{aligned}$$

where x_{ij} represents the element in row i , column j . \square

For a given CSP, a *literal* lit is of the form $x = d$ where $x \in X$ and $d \in dom(x)$. We will use $var(lit)$ to denote its variable x . We denote the set of all literals of a CSP P by $lit(P)$. An *assignment* A is a set of literals. An assignment over a set of variables $V \subseteq X$ has exactly one literal $x = d$ for each variable $x \in V$. An assignment over X is called a *complete* assignment.

A constraint c is defined over a set of variables, denoted by $vars(c)$, and specifies a set of *allowed* assignments over $vars(c)$. An assignment over $vars(c)$ that is not allowed by c is *disallowed* by c . An assignment A over $V \subseteq X$ *satisfies* constraint c if $vars(c) \subseteq V$ and the projection of A over $vars(c)$ (i.e., $\{lit \in A | var(lit) \in vars(c)\}$), is allowed by c . A *solution* is a complete assignment that satisfies every constraint in C .

A *solution symmetry* f for a CSP P is a permutation of $lit(P)$ that preserves the set of solutions [Cohen *et al.*, 2005], i.e., a bijection from literals to literals that maps solutions to solutions. A *constraint symmetry* is a solution symmetry that preserves the set of constraints. Two important kinds of solution symmetries are induced by either permuting variables or values.

A permutation f of the set X of variables induces a permutation p_f of literals by defining $p_f(x = d)$ as the literal $f(x) = d$. A *variable symmetry* is a permutation of the variables whose induced literal permutation is a solution symmetry [Puget, 2002]. Since the inverse of any such permutation is also a symmetry, we will use $\langle x_1, x_2, \dots, x_n \rangle \leftrightarrow \langle x_{1'}, x_{2'}, \dots, x_{n'} \rangle$, where $\{x_1, \dots, x_n\}, \{x_{1'}, \dots, x_{n'}\} \subset X$ to denote the symmetry which maps each x_i to $x_{i'}$ leaving the remaining variables in X unchanged.

A set of domain permutations $f_{dom(x)}$, one for each $x \in X$, induces a permutation p_f of literals by defining

$p_f(x = v)$ as the literal $x = f_{dom(x)}(v)$. A *value symmetry* is a set of domain permutations whose induced literal permutation is a solution symmetry [Puget, 2002]. We will use $\langle d_{i1}, d_{i2}, \dots, d_{in} \rangle \leftrightarrow \langle d_{i1'}, d_{i2'}, \dots, d_{in'} \rangle$, where $\{d_{i1}, d_{i2}, \dots, d_{in}\} = dom(x_i) = \{d_{i1'}, d_{i2'}, \dots, d_{in'}\}$, to denote a value symmetry for $x_i \in X$. A *variable-value* symmetry is any solution symmetry that is not a variable or a value symmetry. Note that it is not necessarily a composition of a variable and a value symmetry.

Example 2 The problem of Example 1 has:

- *variable symmetries that swap any two columns:*
 $\langle x_{11}, x_{21}, x_{31} \rangle \leftrightarrow \langle x_{12}, x_{22}, x_{32} \rangle$, $\langle x_{11}, x_{21}, x_{31} \rangle \leftrightarrow \langle x_{13}, x_{23}, x_{33} \rangle$, and $\langle x_{12}, x_{22}, x_{32} \rangle \leftrightarrow \langle x_{13}, x_{23}, x_{33} \rangle$.
- *similar variable symmetries that swap any two rows.*
- *variable-value symmetries that transpose the rows, column and value dimensions, and correspond to flipping the 3×3 square using a diagonal.* \square

Several methods [Romani and Markov, 2005; Puget, 2005; Cohen *et al.*, 2005] have been proposed to automatically detect the symmetries of a CSP instance by constructing a (hyper-)graph representation of the CSP instance, and using graph automorphism techniques to detect symmetries. Our approach uses the technique of Mears *et al.* [Mears *et al.*, 2006] since it is more powerful than that of Puget [Puget, 2005] without being as computationally demanding as that of Cohen *et al.* [Cohen *et al.*, 2005]. However, any such method can be used. The idea is to (a) represent every literal as a node, (b) represent every assignment disallowed by a constraint as a hyper-edge, and (c) add an edge between every two literals $x = d_1$ and $x = d_2$ where $d_1 \neq d_2$.

Example 3 Consider the CSP provided in Example 1. The associated graph (left hand side of Figure 1) has $9 \times 3 = 27$ nodes (labelled $\binom{i,j}{k}$) representing the 27 literals $x_{i,j} = k$ where $i, j, k \in [1..3]$, and $(18 \times 3) + (9 \times 3)$ edges representing the 3 assignments disallowed by each of the 18 constraints, and the 3 extra edges needed to disallow each pair of values of the 9 variables. \square

Given a hyper-graph (V, E) , where V is a set of nodes, and E a set of unweighted and undirected hyper-edges, an *automorphism* f of graph (V, E) is a permutation of the nodes (i.e., a bijection among nodes) such that $\forall \{n_i, \dots, n_j\} \in E : \{f(n_i), \dots, f(n_j)\} \in E$. For a CSP problem P the graph has a node for each literal in $lit(P)$. Since a graph automorphism is a permutation of the nodes of the graph, it has a direct interpretation as a permutation of the literals in $lit(P)$. In particular, each graph automorphism corresponds to a symmetry of P . Thus, in an abuse of terminology, we will sometimes use *symmetry of a graph* as a shorthand for *automorphism of the graph* associated to a CSP.

Standard tools, such as Saucy [Darga *et al.*, 2004], can compute the automorphisms of a graph instance, and return the resulting symmetry group (i.e., all possible symmetries) by means of a set of generators (i.e., a possibly minimal set of symmetries that can be used to generate all other elements).

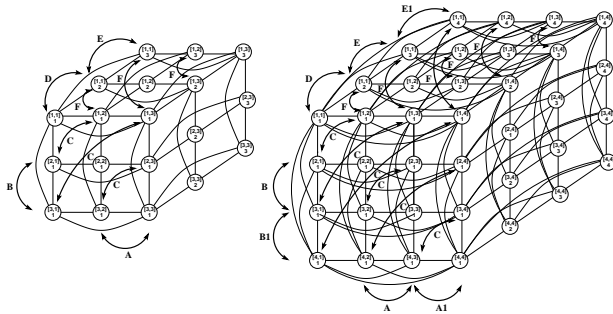


Figure 1: Graphs and generators for LatinSquare[3] and LatinSquare[4]

Example 4 For the graph of the Latin square problem of size 3 given in Example 3 Saucy returns the following (non-minimal) set of generators:

- A** $\langle n_{121}, n_{122}, n_{123}, n_{221}, n_{222}, n_{223}, n_{321}, n_{322}, n_{323} \rangle \leftrightarrow \langle n_{131}, n_{132}, n_{133}, n_{231}, n_{232}, n_{233}, n_{331}, n_{332}, n_{333} \rangle$
- B** $\langle n_{211}, n_{212}, n_{213}, n_{221}, n_{222}, n_{223}, n_{231}, n_{232}, n_{233} \rangle \leftrightarrow \langle n_{311}, n_{312}, n_{313}, n_{321}, n_{322}, n_{323}, n_{331}, n_{332}, n_{333} \rangle$
- C** $\langle n_{121}, n_{122}, n_{123}, n_{131}, n_{132}, n_{133}, n_{231}, n_{232}, n_{233} \rangle \leftrightarrow \langle n_{211}, n_{212}, n_{213}, n_{311}, n_{312}, n_{313}, n_{321}, n_{322}, n_{323} \rangle$
- D** $\langle n_{111}, n_{121}, n_{131}, n_{211}, n_{221}, n_{231}, n_{311}, n_{321}, n_{331} \rangle \leftrightarrow \langle n_{112}, n_{122}, n_{132}, n_{212}, n_{222}, n_{232}, n_{312}, n_{322}, n_{332} \rangle$
- E** $\langle n_{112}, n_{122}, n_{132}, n_{212}, n_{222}, n_{232}, n_{312}, n_{322}, n_{332} \rangle \leftrightarrow \langle n_{113}, n_{123}, n_{133}, n_{213}, n_{223}, n_{233}, n_{313}, n_{323}, n_{333} \rangle$
- F** $\langle n_{112}, n_{113}, n_{123}, n_{212}, n_{213}, n_{223}, n_{312}, n_{313}, n_{323} \rangle \leftrightarrow \langle n_{121}, n_{131}, n_{132}, n_{221}, n_{231}, n_{232}, n_{321}, n_{331}, n_{332} \rangle$

where, for reasons of space, node n_{ijk} represents literal $x_{i,j} = k$. The meaning of these generators, illustrated in the left hand side of Figure 1, is as follows: **A** indicates that column 2 can be swapped with column 3, **B** that row 2 can be swapped with row 3, **C** that the square can be reflected across the top-left/bottom-right diagonal, **D** that value 1 can be swapped with value 2, **E** that value 2 can be swapped with value 3, and **F** that the second dimension of the square can be swapped with the value dimension.

The combination of these generators results in the symmetries given in Example 2. For example, swapping columns 1 and 2 ($\langle x_{11}, x_{21}, x_{31} \rangle \leftrightarrow \langle x_{12}, x_{22}, x_{32} \rangle$) can be achieved by first applying **F**, then **D**, and then **F**. \square

Note that, in order to be able to use these automorphism tools, we need to convert each hyper-edge into a set of binary edges. This can easily be done [Puget, 2005] by (a) creating a new kind of node (a *constraint* node) with an edge to every literal in the disallowed assignment, (b) assigning a different colour to each kind of node (e.g., black to constraint nodes and white to the rest), and (c) extending the concept of automorphism to ensure that the colour of a node is preserved.

3 Parametrising the CSP and its associated graph

There is no standard notation for distinguishing between a CSP model and a CSP instance. Herein, we shall denote a CSP model as $CSP[Data]$, where $Data$ represents the parameters to the model, and the instance as $CSP[d]$, where d represents their particular values.

For simplicity, herein we will use mathematical notation to represent CSP models. However, any high-level modelling language, such as ESO [Mancini and Cadoli, 2005], OPL [Hentenryck, 1999], Essence [Frisch *et al.*, 2007], Esra [Flener *et al.*, 2004], and Zinc [de la Banda *et al.*, 2006], can be used, as long as it explicitly separates the model from the data, has multi-dimensional arrays of finite domain variables, and supports iteration over these arrays.

Example 5 The Latin square problem of Example 1 can be parametrised on the size N of the board as $LatinSquare[N]$, and can be modelled as:

$$\begin{aligned} X[N] &= \{square_{ij} | i, j \in [1..N]\} \\ D[N] &= [1..N] \\ C[N] &= \{square_{ij} \neq square_{ik} | i, j \in [1..N], k \in [j+1..N]\} \cup \\ &\quad \{square_{ji} \neq square_{ki} | i, j \in [1..N], k \in [j+1..N]\} \end{aligned}$$

which defines $N \times N$ integer decision variables ($square_{ij}$) with values in $[1..N]$, and conjoins the inequality constraints for every row (i) and column (j). \square

While being able to obtain the graph associated to an instance is useful, our aim is to determine the symmetries for the model. Thus, we are interested in obtaining a graph that can capture all instances of the model, i.e., a *parametrised graph* that, when instantiated for a given value, yields the graph associated to the problem instance. Note that, the parametrised graph is simply a syntactic construct that represents a class of graphs, much as the parametrised model represents a class of instances. We denote by $G[Data]$ the parametrised graph obtained from model $CSP[Data]$, and by $G[d]$ the graph of instance $CSP[d]$. Furthermore, we would like the graph specification to capture some of the knowledge about the structure of the parametrised problem.

Formally, the parametrised graph $G[Data]$ obtained for model $CSP[Data] = (X[Data], D[Data], C[Data])$ can be obtained as follows:

- $G[Data] = \langle V, E_v \cup E_c \rangle$
- $V = \{x_i = d_i | x_i \in X[Data], d_i \in dom(x_i)[Data]\}$, i.e., V contains a node for every literal in the model.
- $E_v = \{\{x = d_i, x = d_j\} | x \in X[Data], d_i, d_j \in dom(x)[Data], i \neq j\}$, i.e., an edge exists between every two nodes that map a variable to different values.
- $E_c = \bigcup_{c \in C[Data]} \{A | var(A) = var(c), A \text{ is an assignment disallowed by } c\}$, i.e., a hyper-edge exists for every disallowed assignment A of every constraint c , and connects the nodes associated to all literals in A .

Example 6 The parametrised graph $G[N]$ associated to the $LatinSquare[N]$ model of Example 5 is computed as follows. The set of literals that can be extracted from the model is $\{square_{ij} = v \mid i, j, v \in [1..N]\}$. This yields the associated set of nodes $\{n_{ijv} \mid i, j, v \in [1..N]\}$, in the parametrised graph. Note that the nodes in $G[N]$ maintain some of the knowledge about the structure of $LatinSquare[N]$ thanks to the reuse of the i and j identifiers. This is important not only to automate the construction of the edges in $G[N]$, but as we will see later, to parametrise symmetries belonging to graph instances.

E_v is defined as $\{\{n_{ijv_1}, n_{ijv_2}\} \mid i, j, v_1, v_2 \in [1..N], v_1 \neq v_2\}$, while E_c is obtained by transforming the two constraints in the model into the set of assignments they disallow. If $c \in C[N]$ and $vars(c) = \langle square_{i_1j_1}, square_{i_2j_2} \rangle$, then there is an edge $\{n_{i_1j_1v_1}, n_{i_2j_2v_2}\}$ for each v_1, v_2 such that $\langle square_{i_1j_1} = v_1, square_{i_2j_2} = v_2 \rangle$ is disallowed by c . Formally:

$$E_c = \{\{n_{ijv}, n_{ikv}\} \mid i, j, v \in [1..N], k \in [j + 1..N]\} \cup \{\{n_{jiv}, n_{kiv}\} \mid i, j, v \in [1..N], k \in [j + 1..N]\} \square$$

Given a parametrised graph, it is also possible to express a *parametrised permutation* of its nodes. For example, the following is a parametrised permutation f of $G[N]$ for $LatinSquare[N]$: $f(n_{ijv}) = n_{jiv}, \forall i, j, v \in [1..N]$. If a parametrised permutation is an automorphism of the parametrised graph for all possible parameter values, then we call it a symmetry of the parametrised graph. Thus, we can use S_{Data} to denote the group of symmetries of the parametrised graph $G[Data]$ associated to model $CSP[Data]$

4 Computing the candidate symmetries for a model

The idea is to compute the symmetries of several small instances of the model, and use them to elicit parametrised permutations that are likely to be symmetries of the model itself. This poses two main challenges: (a) to find automorphisms $f[d1]$, $f[d2]$ etc. of small graphs $G[d1]$, $G[d2]$ etc. that are (likely to be) instances of a single symmetry $f[Data]$ of the parametrised graph $G[Data]$, and (b) to compute from several such permutations, a parametrised permutation of which they are all instances. Both challenges are explored in this section.

4.1 Models ordered by the subgraph relationship

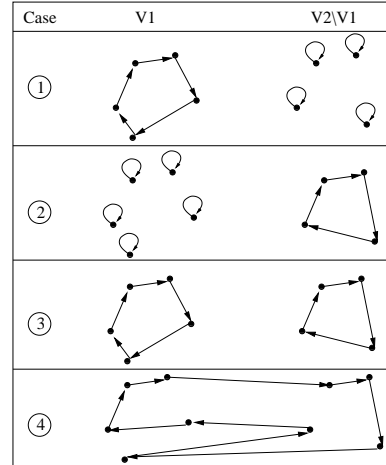
When instantiated, a parametrised permutation of the parametrised graph $G[Data]$ becomes a permutation $f[d1]$ of graph $G[d1]$ and $f[d2]$ of graph $G[d2]$. This and the following subsections explore the relationship between $f[d1]$ and $f[d2]$.

We will only consider a limited, but important class of *ordered* models where graphs of different instances can be ordered by a subgraph relationship. Typically, ordered models are parametrised by a sequence of integers, one for each independent “dimension” of the model (naturally, if more than one dimension exists, the ordering is

partial). For example, $LatinSquare[N]$ is ordered since, if $n < m$, $G[n]$ is a subgraph of $G[m]$.

Let $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ be two graphs and let $G_1 \subset G_2$ indicate that G_1 is a subgraph of G_2 . Automorphism f of G_2 can be *restricted* to $G_1 \subset G_2$, denoted by $f|_{G_1}$, if $\forall n \in V_1, f(n) \in V_1$, i.e., if it maps G_1 onto itself. Let G_1 and G_2 be two instances of parametrised graph $G[Data]$, and let S_1 and S_2 denote the group of automorphisms on G_1 and G_2 , respectively. If $G_1 \subset G_2$, S_2 can be partitioned into two sets: the set Old_{12} of automorphisms that can be restricted to G_1 , and the set New_{12} of automorphisms that cannot. Intuitively, Old_{12} contains the automorphisms in G_1 that still exist in G_2 (since they map nodes of G_1 to G_1 and of G_2 to G_2), while New_{12} contains those that have emerged for G_2 (i.e., map some nodes of G_1 to G_2 and vice versa).

We found it useful to visually illustrate the elements of S_2 according to the adjoining Figure, where cases 1, 2 and 3 belong to Old_{12} , while 4 belongs to New_{12} . Case 1 corresponds to a non-trivial permutation of nodes in G_1 , extended with the identity on the new nodes of G_2 . Case 2 corresponds to the trivial identity permutation of nodes in G_1 , extended with a non-trivial permutation on the new nodes of G_2 . Case 3 corresponds to a non-trivial permutation of nodes in G_1 , extended with a non-trivial permutation on the new nodes of G_2 . And case 4 corresponds to a non-trivial permutation of nodes in G_2 that is not an extension of one in G_1 .



Let $D = V_2 \setminus V_1$ be the set of new nodes in G_2 . Candidate symmetries belonging to cases 1, 2 and 3 above are efficiently computed in our approach by using GAP [GAP, 2006] to find the intersection between the group S_2 and the group $\{s_1 \times p \mid s_1 \in S_1, p \in Perm(D)\}$, where $Perm(D)$ denotes the set of all possible permutations among the nodes in D . Candidate symmetries for case 4 are found using a different approach outlined in Section 4.3 below.

Example 7 Consider the graph $G[4]$ associated to $LatinSquare[4]$, shown in the right hand side of Figure 1. Saucy finds 9 generators for this graph. Six of them are simple extensions of the generators found for $LatinSquare[3]$ in Example 4. For example, the extension of

generator **A** is:

$$\mathbf{A} \langle n_{121}, n_{122}, n_{123}, n_{124}, n_{221}, n_{222}, \dots, n_{321}, \dots, n_{421}, \dots \rangle \leftrightarrow \langle n_{131}, n_{132}, n_{133}, n_{134}, n_{231}, n_{232}, \dots, n_{331}, \dots, n_{431}, \dots \rangle$$

and similarly for **B**, **C**, **D**, **E** and **F**. The other three generators found are:

$$\begin{aligned} \mathbf{A} & \langle n_{131}, n_{132}, n_{133}, n_{134}, n_{231}, n_{232}, \dots, n_{331}, \dots, n_{431}, \dots \rangle \leftrightarrow \\ & \langle n_{141}, n_{142}, n_{143}, n_{144}, n_{241}, n_{242}, \dots, n_{341}, \dots, n_{441}, \dots \rangle \\ \mathbf{B1} & \langle n_{311}, n_{312}, n_{313}, n_{314}, n_{321}, n_{322}, \dots, n_{331}, \dots, n_{341}, \dots \rangle \leftrightarrow \\ & \langle n_{411}, n_{412}, n_{413}, n_{414}, n_{421}, n_{422}, \dots, n_{431}, \dots, n_{441}, \dots \rangle \\ \mathbf{E1} & \langle n_{113}, n_{123}, n_{133}, n_{143}, n_{213}, n_{223}, \dots, n_{313}, \dots, n_{413}, \dots \rangle \leftrightarrow \\ & \langle n_{114}, n_{124}, n_{134}, n_{144}, n_{214}, n_{224}, \dots, n_{314}, \dots, n_{414}, \dots \rangle \end{aligned}$$

The generators found by GAP to be in the Old_{34} partition of the symmetry group in $G[4]$ are **A**, **B**, **C**, **D**, **E** and **F**. Note that all these generators correspond to case 3, while **A1**, **B1** and **E1** correspond to case 4. \square

4.2 Eliciting Parametrised Permutations

In order for our implementation to be able to elicit a parametrised permutation from a candidate permutation, every node in the graph must be identified by a sequence i, j, k, \dots of indices which can take any value between 1 and the dimension of the instance (such as nodes n_{ijk} in the $LatinSquare[n]$ instance graph). If so, our implementation currently attempts to “lift” every index by identifying one of two simple situations: (I) an index which is always mapped to itself, and (II) an index which is always mapped to another. While obviously incomplete, these two heuristics alone suffice to handle all case 3 type symmetries for $LatinSquare$. Note that our method does not claim to elicit all possible symmetries. This is not only due to the limits of our heuristics but also due to the limits of the graph representation which only captures constraint symmetries and, thus, might miss some solution symmetries.

Example 8 The generators found by GAP to be in Old_{34} for $LatinSquare[3]$ and $LatinSquare[4]$ in Example 7 can be automatically parametrised as:

$$\begin{aligned} \mathbf{A} & \{n_{i2v} \leftrightarrow n_{i3v} | i, v \in [1..N]\} \\ \mathbf{B} & \{n_{2jv} \leftrightarrow n_{3jv} | j, v \in [1..N]\} \\ \mathbf{C} & \{n_{ijv} \leftrightarrow n_{jiv} | i, j, v \in [1..N]\} \\ \mathbf{D} & \{n_{ij1} \leftrightarrow n_{ij2} | i, j \in [1..N]\} \\ \mathbf{E} & \{n_{ij2} \leftrightarrow n_{ij3} | i, j \in [1..N]\} \\ \mathbf{F} & \{n_{ijv} \leftrightarrow n_{ivj} | i, j, v \in [1..N]\} \end{aligned}$$

where **A**, **B**, **D** and **E** used (I) above, while **C** and **F** used (I) and (II). For example, **A** is parametrised as $\{n_{i21} \leftrightarrow n_{i31}, n_{i22} \leftrightarrow n_{i32}, n_{i23} \leftrightarrow n_{i33} | i \in [1..3]\}$ first, then to $\{n_{i2v} \leftrightarrow n_{i3v} | i, v \in [1..3]\}$, and finally to the parametrised form above by replacing $[1..3]$ by a range that is independent of the instance. Similarly, **C** is parametrised to $\{n_{ij1} \leftrightarrow n_{jii}, n_{ij2} \leftrightarrow n_{jii}, n_{ij3} \leftrightarrow n_{jii} | i, j \in [1..3]\}$, then to $\{n_{ijk} \leftrightarrow n_{jik} | i, j, k \in [1..3]\}$, and finally to the one shown above.

Note that the parametrised version of these generators is identical to that obtained using the generators in $LatinSquare[4]$. One could then argue that there is no need to compute Old_{34} by calling GAP, since one could simply parametrise the generators for the two instances

and then check which ones are identical, thus obtaining parametrised generators known to be in S_4 . While this works for $LatinSquare[N]$, it is too weak in general because it depends crucially on the set of generators chosen for representing the groups. Instead, GAP intersects the groups specified by the generators and is thus independent of the particular choice of generators. \square

As one can see in Example 8, some parametrised generators contain concrete numbers as node identifiers. We take this as an indication of the possibility of further parametrising the generators in Old_{34} . However, these further parametrisations often require us to consider more than two instances of the graph.

Example 9 The generators found by Saucy for instance $LatinSquare[5]$ are the simple extensions of **A**, **A1**, **B**, **B1**, **C**, **D**, **E**, **E1** and **F**, plus three more, which we will call **A2**, **B2**, and **E2**. Again, all generators in instance $LatinSquare[4]$ are found to be in Old_{45} . They can be automatically parametrised as:

$$\begin{aligned} \mathbf{A1} & \{n_{i3v} \leftrightarrow n_{i4v} | i, v \in [1..N]\} \\ \mathbf{B1} & \{n_{3jv} \leftrightarrow n_{4jv} | j, v \in [1..N]\} \\ \mathbf{E1} & \{n_{ij3} \leftrightarrow n_{ij4} | i, j \in [1..N]\} \end{aligned}$$

This is a pattern that can be observed when intersecting any two consecutive instances N and $N+1$: all generators in N are found to be in $Old_{N(N+1)}$ while the number of generators for $N+1$ increases by exactly 3, one per dimension. For example, those for $LatinSquare[5]$ are:

$$\begin{aligned} \mathbf{A2} & \{n_{i4v} \leftrightarrow n_{i5v} | i, v \in [1..N]\} \\ \mathbf{B2} & \{n_{4jv} \leftrightarrow n_{5jv} | j, v \in [1..N]\} \\ \mathbf{E2} & \{n_{ij4} \leftrightarrow n_{ij5} | i, j \in [1..N]\} \square \end{aligned}$$

When the number of generators in Old differs for two different pairs of instances, one must consider which generators to mark as candidates. One possibility is to choose the smaller set in the hope of minimising false candidates. Another is to choose the bigger set in the hope of maximising true candidates. A third is to look for patterns among the generators of the different $Olds$.

Example 10 Each of the three generators obtained outside the Old of each instance belongs to a sequence: **A:A1:A2**, sequence **B:B1:B2**, and sequence **D:E:E1:E2**, all starting with generators that belong to the current Old (**A**, **B** and **D** for $LatinSquare[5]$), and finishing with generators that do not (**A2**, **B2** and **E2**, respectively). Furthermore, each sequence can itself be parametrised resulting in the following candidates:

$$\begin{aligned} \mathbf{A} & \{\{n_{ijv} \leftrightarrow n_{ikv} | i, v \in [1..N]\} | j \in [1..N-1], k = j+1\} \\ \mathbf{B} & \{\{n_{ijv} \leftrightarrow n_{kqv} | j, v \in [1..N]\} | i \in [1..N-1], k = v+1\} \\ \mathbf{C} & \{n_{ijv} \leftrightarrow n_{jiv} | i, j, v \in [1..N]\} \\ \mathbf{D} & \{\{n_{ijv} \leftrightarrow n_{ijw} | i, j \in [1..N]\} | v \in [1..N-1], w = v+1\} \\ \mathbf{F} & \{n_{ijv} \leftrightarrow n_{ivj} | i, j, v \in [1..N]\} \square \end{aligned}$$

Note that none of these parametrised generators contain concrete numbers. Also note that, as indicated later, such sophisticated parametrisations are beyond the capabilities of our current implementation.

4.3 Using *New* to determine other likely candidates

Up to now we have only used *Old* to determine candidates. However, it is possible for an automorphism in *New* to be a likely candidate. To decide whether this is the case, the last step in our search for likely candidates is to parametrise every generator not in *Old* and mark as likely candidates those parametrisations that represent generators of different instances.

Example 11 *The queens problem aims at positioning N queens in an $N \times N$ chess board without one queen attacking another. The following model of $\text{Queens}[N]$ uses N integer variables (each representing the row in which the queen is positioned) with domains in $[1..N]$.*

$$\begin{aligned} X[N] &= \{q_i | i \in [1..N]\} \\ D[N] &= [1..N] \\ C[N] &= \{q_i \neq q_j | i \in [1..N], j \in [i+1..N]\} \cup \\ &\quad \{q_i + i \neq q_j + j | i \in [1..N], j \in [j+1..N]\} \cup \\ &\quad \{q_i - i \neq q_j - j | i \in [1..N], j \in [j+1..N]\} \end{aligned}$$

Its parametrised graph $G[N] = (V, E_c \cup E_v)$ is:

$$\begin{aligned} V &= \{q_{iv} | i, v \in [1..N]\} \\ E_c &= \{\{q_{iv}, q_{jv}\} | i, v \in [1..N], j \in [i+1..N]\} \cup \\ &\quad \{\{q_{iv}, q_{jv_j}\} | i, v_i, v_j \in [1..N], j \in [i+1..N], v_i + i = v_j + j\} \cup \\ &\quad \{\{q_{iv}, q_{jv_j}\} | i, v_i, v_j \in [1..N], j \in [i+1..N], v_i - i = v_j - j\} \\ E_v &= \{\{q_{iv}, q_{jv_j}\} | i, v_i, v_j \in [1..N], v_i \neq v_j\} \end{aligned}$$

where node q_{iv} represents literal $q_i = v$. Figure 2 shows the graph instances $G[4]$ and $G[5]$ together with the generators found by Saucy for $G[4]$:

$$\begin{aligned} \mathbf{A} &\langle q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, q_{42} \rangle \leftrightarrow \\ &\langle q_{14}, q_{13}, q_{24}, q_{23}, q_{34}, q_{33}, q_{44}, q_{43} \rangle \\ \mathbf{B} &\langle q_{12}, q_{13}, q_{14}, q_{23}, q_{24}, q_{34} \rangle \leftrightarrow \\ &\langle q_{21}, q_{31}, q_{41}, q_{32}, q_{42}, q_{43} \rangle \end{aligned}$$

and for $G[5]$:

$$\begin{aligned} \mathbf{A1} &\langle q_{11}, q_{12}, q_{21}, q_{22}, q_{31}, q_{32}, q_{41}, q_{42}, q_{51}, q_{52} \rangle \leftrightarrow \\ &\langle q_{15}, q_{14}, q_{25}, q_{24}, q_{35}, q_{34}, q_{45}, q_{44}, q_{55}, q_{54} \rangle \\ \mathbf{B} &\langle q_{12}, q_{13}, q_{14}, q_{15}, q_{23}, q_{24}, q_{25}, q_{34}, q_{35}, q_{54} \rangle \leftrightarrow \\ &\langle q_{21}, q_{31}, q_{41}, q_{41}, q_{32}, q_{42}, q_{52}, q_{43}, q_{53}, q_{45} \rangle \end{aligned}$$

where \mathbf{B} is an extension of the generator with the same name found for $G[4]$, and $\mathbf{A1}$ is a new generator. As represented visually in Figure 2, generators \mathbf{A} and $\mathbf{A1}$ indicate a reflection around a horizontal axis through the centre of the board, while \mathbf{B} indicates a reflection around the top-left/bottom-right diagonal. As a group, they provide all the symmetries of a square. The generators found for $G[6]$ are, again, an extension of \mathbf{B} and a new generator $\mathbf{A2}$ which also reflects the board through its horizontal axis.

Generator \mathbf{B} is the only generator found by GAP to be in Old_{45} and also the only one in Old_{56} . Its automatic parametrisation results in $\{q_{ij} \leftrightarrow q_{ji} | i, j \in [1..N]\}$. Since (a) the number of generators in Old for $G[4]$ and $G[5]$, and for $G[5]$ and $G[6]$ is the same, and (b) its parametrised version contains no concrete numbers, \mathbf{B} can be marked as a likely candidate.

To decide whether \mathbf{A} , $\mathbf{A1}$ and $\mathbf{A2}$ are likely candidates or not, we parametrise them and check whether their parametrised version is identical. Our current implementation parametrises \mathbf{A} to $\{q_{i1}, q_{i2}\} \leftrightarrow \{q_{i4}, q_{i3}\} | i \in$

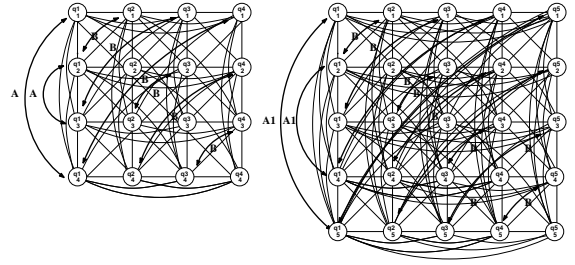


Figure 2: Graph instances for $\text{Queens}[4]$ and $\text{Queens}[5]$

$[1..N]$, $\mathbf{A1}$ to $\{q_{i1}, q_{i2}\} \leftrightarrow \{q_{i5}, q_{i4}\} | i \in [1..N]$, and $\mathbf{A2}$ to $\{q_{i1}, q_{i2}, q_{i3}\} \leftrightarrow \{q_{i6}, q_{i5}, q_{i4}\} | i \in [1..N]$. Though \mathbf{A} , $\mathbf{A1}$ and $\mathbf{A2}$ do not form a sequence such as that in Example 10, a more sophisticated parametrisation would capture the common pattern $\{q_{ij} \leftrightarrow q_{i(N-j+1)} | i, j \in [1..N]\}$ and, thus, that their parametrised version should be marked as likely candidate. \square

5 Summary of the approach

In summary, given a parametrised CSP model $\text{CSP}[Data]$, our approach:

1. obtains the associated parametrised graph $G[Data]$,
2. instantiates $G[Data]$ with at least three consecutive values $G[d_1], G[d_2]$ and $G[d_3]$, for each possible dimension of $Data$,
3. computes Old_{12} for $G[d_1]$ and $G[d_2]$, and Old_{23} for $G[d_2]$ and $G[d_3]$
4. independently parametrises each generator in Old_{12} and Old_{23} ,
 - (a) If the number of generators in Old_{12} and Old_{23} is the same, it checks whether they result in the same set. If so, it marks each parametrised generator in either Old_{12} or Old_{23} as likely candidates. If not (often because of the occurrence of concrete values), it attempts to discover more complex patterns that eliminate these concrete values to yield common parametrisations. If it succeeds, it marks them as likely candidates.
 - (b) If the number of generators is not the same, it attempts to discover sequences that eliminate concrete values to yield common parametrisations. Again, it marks them as likely candidates.
5. computes $\text{New}_i, i \in [1..3]$ as the set of generators in $G[d_i]$ and not in $\text{Old}_{12} \cup \text{Old}_{23}$. It parametrises these three sets of generators and determines likely candidates in the same way as before.
6. determines, for every likely candidate, whether it is a model symmetry.

The last step can be achieved by representing both the CSP model and the candidate in the logic formalism described in [Mancini and Cadoli, 2005], and then making use of theorem proving techniques. Of course, such

technique is in general undecidable. Another approach, which we are currently exploring, is to use graph techniques to prove that a likely candidate is an automorphism of the parametrised graph $G[\text{Data}]$.

This approach has not completely been implemented yet. Currently, our implementation takes several user-specified ECLⁱPS^e instances (rather than a model as in step 2 above), obtains their associated graphs, and computes their symmetry groups. Then, for each two groups of symmetries, we automatically compute *Old* (step 3 above) by calling GAP, as explained in Section 4.1. Each of the generators in the computed *Old* is then automatically parametrised (step 4) by the simple “pattern recognition” program introduced in Section 4.2. The same process is followed for each generator in each *New_i* (step 5), as explained in Section 4.3. We are currently exploring an automatic way of increasing the number of patterns recognised in steps 4 and 5 by using data-mining.

Let us now further illustrate our approach by means of two detailed examples.

Golomb ruler: a set of N integers (marks on the ruler) $0 = a_1 < a_2 < \dots < a_N$ such that the $\frac{N(N-1)}{2}$ differences $a_j - a_i$, $1 \leq i < j \leq N$ are distinct. The problem involves finding a valid set of N marks. The following model for Golomb[N] uses N integer variables (the marks) with domains in $[0..N^2]$, plus $\frac{N(N-1)}{2}$ integer variables (the differences) with domains $[0..N^2]$.

$$\begin{aligned} X[N] &= \{mark_i | i \in [0..N]\} \cup \{diff_{ij} | i \in [1..N], j \in [i+1..N]\} \\ D[N] &= [1..N * N] \\ C[N] &= \{mark_i - mark_j = diff_{ij} | i \in [1..N], j \in [i+1..N]\} \cup \\ &\quad \{diff_{ij} \neq diff_{ik} | i, j \in [1..N], k \in [j+1..N]\} \end{aligned}$$

The parametrised graph associated to Golomb[N] is:

$$\begin{aligned} V &= \{m_{iv} | i \in [1..N], v \in [1..N^2]\} \cup \\ &\quad \{d_{jiv} | i \in [1..N], j \in [(i+1)..N], v \in [1..N^2]\} \\ E_c &= \{\{m_{iv_1}, m_{jv_2}, d_{ijv_3}\} | i \in [1..N], j \in [(i+1)..N], \\ &\quad v_1, v_2, v_3 \in [1..N^2], v_1 - v_2 \neq v_3\} \cup \\ &\quad \{\{d_{ijv}, d_{jiv}\} | i \in [1..N], j \in [(i+1)..N], v \in [1..N^2]\} \\ E_v &= \{\{m_{iv_1}, m_{iv_2}\} | i \in [1..N], v_1, v_2 \in [1..N^2], v_1 \neq v_2\} \cup \\ &\quad \{\{d_{ijv_1}, d_{ijv_2}\} | i \in [1..N], j \in [(i+1)..N], v_1, v_2 \in [1..N^2], \\ &\quad v_1 \neq v_2\} \end{aligned}$$

where node m_{iv} represents literal $mark_i = k$ and node d_{jiv} literal $diff_{ij} = v$. The generator for $G[3]$ is:

$$\begin{aligned} \mathbf{A} &\langle d_{121}, d_{122}, d_{123}, d_{124}, d_{125}, d_{126}, d_{127}, d_{128}, d_{129} \rangle \leftrightarrow \\ &\langle d_{231}, d_{232}, d_{233}, d_{234}, d_{235}, d_{236}, d_{237}, d_{238}, d_{239} \rangle \text{ plus} \\ &\langle m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}, m_{16}, m_{17}, \dots, m_{24} \rangle \leftrightarrow \\ &\langle m_{39}, m_{38}, m_{37}, m_{36}, m_{35}, m_{34}, m_{33}, m_{32}, \dots, m_{25} \rangle \end{aligned}$$

which swaps the lengths of the spaces between the marks, i.e., turns the ruler back-to-front. The generators **A1** for $G[4]$ and **A2** for $G[5]$ follow a similar pattern. While both *Old₃₄* and *Old₄₅* are empty, $New_1 = \{\mathbf{A}\}$, $New_2 = \{\mathbf{A}_1\}$ and $New_3 = \{\mathbf{A}_2\}$. Their independent parametrisation results in three sets with the same number of elements but different parametrisations due to the existence of concrete values. For example, **A** is parametrised to $\{d_{12v} \leftrightarrow d_{23v} | v \in [1..N^2]\} \cup \{\{m_{1v_1}, m_{2v_2}\} \leftrightarrow \{m_{3v'_1}, m_{2v'_2}\} | v_1, v_2, v'_1, v'_2 \in [1..N^2], v_1 = N^2 - v'_1 + 1, v_2 = N^2 - v'_2 + 1\}$. All this

has been achieved by our implementation. A more sophisticated parametrisation would further realise that the number of elements in the three sets is the same, and therefore find the pattern: $\{d_{ijv} \leftrightarrow d_{klv} | i, j, k, l \in [1..N], i = N - k, j = N - l + 2, v \in [1..N^2]\} \cup \{m_{iv_1} \leftrightarrow m_{jv'_1} | i, j \in [1..N], i = N - j + 1, v_1, v'_1 \in [1..N^2], v_1 = N^2 - v'_1 + 1\}$, and mark it as likely candidate.

Social Golfers: aims at constructing a schedule for a band of golfers that, each week, is partitioned into evenly-sized groups. Each pair of golfers may play in the same group at most once. The problem asks for a schedule of W weeks, with G groups per week and P players per group.

$$\begin{aligned} X[N] &= \{players_{wg} | w \in [1..W], g \in [1..G]\} \\ D[N] &= \wp(\{1..P * G\}) \\ C[N] &= \{\{players_{wg} | w \in [1..W], g \in [1..G]\} \cup \\ &\quad \{|players_{wg_1} \cap players_{wg_2}| = 0 | w \in [1..W], \\ &\quad g_1, g_2 \in [1..G], g_1 < g_2\} \cup \\ &\quad \{|players_{w_1g_1} \cap players_{w_2g_2}| \leq 1 | w_1, w_2 \in [1..W], \\ &\quad w_1 < w_2, g_1, g_2 \in [1..G], g_1 < g_2\} \end{aligned}$$

where \wp is the powerset. The parametrised graph associated to Golf[W, G, P] is:

$$\begin{aligned} V &= \{n_{wgp} | w \in 1..W, g \in 1..G, p \in \wp([1..P * G])\} \\ E_c &= \{n_{wgp} | w \in [1..W], g \in [1..G], |p| \neq P\} \cup \\ &\quad \{\langle n_{wg_1p_1}, n_{wg_2p_2} \rangle | w \in 1..W, g_1, g_2 \in G, g_1 < g_2, \\ &\quad p_1, p_2 \in \wp([1..P * G]), |p_1 \cap p_2| \neq 0\} \cup \\ &\quad \{\langle n_{w_1g_1p_1}, n_{w_2g_2p_2} \rangle | w_1, w_2 \in 1..W, w_1 < w_2, g_1, g_2 \in G, \\ &\quad g_1 < g_2, p_1, p_2 \in \wp([1..P * G]), |p_1 \cap p_2| > 1\} \\ E_v &= \{\langle n_{wgp,p}, n_{wgp_2} \rangle | w \in 1..W, g \in 1..G, \\ &\quad p_1, p_2 \in \wp([1..P * G]), p_1 \neq p_2\} \end{aligned}$$

where node n_{wgp} represents literal $players_{wg} = p$. To save space we will not show the concrete form of the generators found for $G[2, 2, 2]$, but their parametrisation:

$$\begin{aligned} \mathbf{A} &\{n_{ija} \leftrightarrow n_{jib} | i \in [1..W], j \in [1..G], a, b \in \wp([1..P * G]), \\ &\quad 1 \in a; b = (a \setminus \{1\}) \cup \{2\}\} \\ \mathbf{B} &\{n_{ija} \leftrightarrow n_{jib} | i \in [1..W], j \in [1..G], a, b \in \wp([1..P * G]), \\ &\quad 2 \in a; b = (a \setminus \{2\}) \cup \{3\}\} \\ \mathbf{C} &\{n_{ija} \leftrightarrow n_{jib} | i \in [1..W], j \in [1..G], a, b \in \wp([1..P * G]), \\ &\quad 3 \in a; b = (a \setminus \{3\}) \cup \{4\}\} \\ \mathbf{D} &\{n_{11v} \leftrightarrow n_{12v} | v \in \wp([1..P * G])\} \\ \mathbf{E} &\{n_{21v} \leftrightarrow n_{22v} | v \in \wp([1..P * G])\} \\ \mathbf{F} &\{n_{1jv} \leftrightarrow n_{2jv} | j \in [1..G], v \in \wp([1..P * G])\} \end{aligned}$$

which corresponds to the following symmetries: golfers 1 and 2 can be swapped (**A**), so can golfers 2 and 3 (**B**), golfers 3 and 4 **C**, groups 1 and 2 in week 1 (**D**), groups 1 and 2 in week 2 (**E**), and weeks 1 and 2 (**F**). The generators for $G[3, 2, 2]$ are simple extensions of those in $G[2, 2, 2]$, plus two more to cover the additional week: the two groups in week 3 can be swapped (**E1**), and weeks 2 and 3 can be swapped **F1**. Similarly, the generators for $G[4, 2, 2]$ are simple extensions of those in $G[3, 2, 2]$ plus, again, two more **E2** and **F2** to cover the additional week. The generators marked as candidates are those in $Old_{\{3,2,2\},\{4,2,2\}}$ are **A,B,C,D,E,F,E1,F1** and **E2**, which leaves **F2** for $New_{\{3,2,2\},\{4,2,2\}}$.

The generators for $G[2, 3, 2]$ include the simple extension of those in $G[2, 2, 2]$, plus those needed to include the extra groups and golfers: golfers 4 and 5 can be swapped

(**C1**), so can golfers 5 and 6 (**C2**), groups 2 and 3 in week 1 (**D1**), and groups 2 and 3 in week 2 (**E'1**). Similarly, the generators for $G[2, 4, 2]$ (once canonicalised by GAP) are simple extensions of those in $G[2, 3, 2]$ plus, again, another four corresponding to the swapping of golfers 6 and 7 (**C3**), 7 and 8 (**C4**), and groups 3 and 4 in week 1 (**D2**), and 3 and 4 in week 2 (**E'2**). The generators in $Old_{\{2,3,2\},\{2,4,2\}}$ are **A,B,C,D,E,F,C1,C2,C4,D1** and **E'1** are marked as candidates, which leaves **C3, D2** and **E'2** for $New_{\{2,3,2\},\{2,4,2\}}$.

The generators for $G[2, 2, 3]$ include the simple extension of those in $G[2, 2, 2]$, plus those needed to include the extra golfers: golfers 4 and 5 can be swapped (**C1**), and so can golfers 5 and 6 (**C2**). The generators for $G[2, 2, 4]$ are again, those needed to include the extra golfers: golfers 6 and 7 can be swapped (**C3**), and so can golfers 7 and 8 (**C4**), and the simple extension of those in $G[2, 2, 3]$ with one surprising exception: **C** is replaced by a different and more complex one. This is the case even after canonicalisation by GAP. **C** is, of course, still a symmetry of the group, it is just not returned by Saucy as a generator.

All this has been achieved by our implementations. A more sophisticated parametrisation would also be able to detect the sequences **D:E:E1:E2, F:F1:F2, D:D:D2, E:E'1:E'2**, and **A:B:C:C1:C2:C3:C4**.

6 Conclusions

Exploiting solution symmetries is often essential in finding solutions to CSPs. Currently, the detection of such symmetries is either restricted to problem instances, or incomplete since the existing methods only detect a small class of symmetries and depend on the syntax of the constraints. Our new approach to symmetry detection for CSPs lifts these restrictions: it can detect symmetries in CSP models and it has the power to capture most - if not all - symmetries. Our approach leverages on existing (and future) symmetry detection methods for CSP instances, by generalising their results to models.

Our approach has been implemented and tested on a small set of models, including those discussed in this paper, although some parts require manual intervention. These case studies show that it can detect model symmetries that could previously only be detected for instances. We are currently investigating how to broaden the tool to discover more complex patterns in symmetries of problem instances, to elicit additional candidate model symmetries. We now plan to integrate techniques to validate or reject candidates, such as theorem proving techniques or graph techniques.

References

[Cohen *et al.*, 2005] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. In Peter van Beek, editor, *LNCS*, volume 3709, pages 17–31, 2005.

- [Darga *et al.*, 2004] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov. Exploiting structure in symmetry generation for cnf. In *Proc. 41st Design Automation Conf.*, pages 530–534, June 2004.
- [de la Banda *et al.*, 2006] Maria Garcia de la Banda, Kim Marriott, Reza Rafieh, and Mark Wallace. The modelling language Zinc. In *Proc. CP'06*, 2006.
- [Flener *et al.*, 2004] P. Flener, J. Pearson, and M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. *LOPSTR'03: Revised Selected Papers*, pages 214–232, 2004.
- [Frisch *et al.*, 2003] Alan M. Frisch, Ian Miguel, and Toby Walsh. CGRASS: A system for transforming constraint satisfaction problems. In Barry O'Sullivan, editor, *Recent Advances in Constraints, Joint ERCIM/CologNet International Workshop on Constraint Solving and Constraint Logic Programming*, volume 2627 of *LNCS*, pages 15–30, 2003.
- [Frisch *et al.*, 2007] A. M. Frisch, M. Grum, C. Jefferson, B. Martinez Hernandez, and I. Miguel. The design of ESSENCE: A constraint language for specifying combinatorial problems. In *IJCAI'07*, 2007.
- [GAP, 2006] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.9*, 2006.
- [Gent and Smith, 2000] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *ECAI 2000 14th European Conference on Artificial Intelligence*, 2000.
- [Gent *et al.*, 2003] I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In *Proc. CP'03*, pages 333–347, 2003.
- [Haselböck, 1993] A. Haselböck. Exploiting interchangeabilities in constraint-satisfaction problems. In *IJCAI'93*, pages 282–289, 1993.
- [Hentenryck, 1999] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.
- [Mancini and Cadoli, 2005] T. Mancini and M. Cadoli. Detecting and breaking symmetries by reasoning on problem specifications. In *Proceedings of the International Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, 2005.
- [Mears *et al.*, 2006] C. Mears, M. Garcia de la Banda, and M. Wallace. On implementing symmetry detection. In *Proc. SymCon 06*, 2006.
- [Puget, 2002] Jean-Francois Puget. Symmetry breaking revisited. In Pascal van Hentenryck, editor, *LNCS*, volume 2470, pages 446–461, 2002.
- [Puget, 2005] Jean-Francois Puget. Automatic detection of variable and value symmetries. In Peter van Beek, editor, *LNCS*, volume 3709, pages 475–489, 2005.

- [Romani and Markov, 2005] Arathi Romani and Igor L. Markov. Automatically exploiting symmetries in constraint programming. *Lecture Notes in Computer Science*, 3419:98–112, 2005.
- [Roney-Dougal *et al.*, 2004] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In *Proc. ECAI'04*, 2004.
- [Roy and Pacheť, 1998] P. Roy and F. Pacheť. Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *ECAI98 Workshop on Non-binary Constraints*, pages 27–33, 1998.
- [Sellmann and Hentenryck, 2005] M. Sellmann and P. Van Hentenryck. Structural symmetry breaking. In *Proc. IJCAI'05*, 2005.
- [van Hentenryck *et al.*, 2005] P. van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Compositional derivation of symmetries for constraint satisfaction. In *SARA'05*, 2005.
- [Walsh, 2006] T. Walsh. General symmetry breaking constraints. In *Proc. CP'06*, 2006.

Exploiting Symmetry in Multiple Knapsack Problems

Alex S. Fukunaga

Global Edge Institute

Tokyo Institute of Technology

fukunaga@is.titech.ac.jp

Abstract

The multiple knapsack problem (MKP) is a classical combinatorial optimization problem. A recent algorithm for some classes of the MKP is bin-completion, a bin-oriented, branch-and-bound algorithm. We investigate mechanisms for detecting and breaking symmetry in the bin-completion search space. We propose path-symmetry and path-dominance, two new symmetry relations which are more powerful generalization of previous MKP symmetry breaking techniques. Experiments show that path-symmetry and path-dominance significantly reduce the number of nodes searched by bin-completion. In particular, a variant of path-symmetry is shown to significantly improve upon the previous state of the art for the MKP with respect to both runtime and nodes searched.

1 Introduction

Consider m containers (bins) with capacities c_1, \dots, c_m , and a set of n items, where each item has a weight w_1, \dots, w_n and profit p_1, \dots, p_n . Packing the items in the containers to maximize the total profit of the items, such that the sum of the item weights in each container does not exceed the container's capacity, and each item is assigned to at most one container is the *0-1 Multiple Knapsack Problem*, or MKP. For example, suppose there are two bins with capacity 10 and 7, and 4 items (9,3),(7,3),(6,7),(1,5), where the first element of each pair is the weight of the item and the second element is the profit of that item. The optimal solution to this MKP instance is to assign (9,3) and (1,5) to the bin with capacity 10, and the item (6,7) to the bin with capacity 7 (total profit = 15). The MKP is a natural generalization of the 0-1 Knapsack Problem where there are m containers of capacities c_1, c_2, \dots, c_m .

Let the binary decision variable x_{ij} be 1 if item j is placed in container i , and 0 otherwise. Then the 0-1 MKP can be formulated as the integer program below, where constraint 2 encodes the capacity constraint for each container, and constraint 3 ensures that each item is assigned to at most one container.

$$\text{maximize } \sum_{i=1}^m \sum_{j=1}^n p_j x_{ij} \quad (1)$$

$$\text{subject to: } \sum_{j=1}^n w_j x_{ij} \leq c_i, \quad i = 1, \dots, m \quad (2)$$

$$\sum_{i=1}^m x_{ij} \leq 1, \quad j = 1, \dots, n \quad (3)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j. \quad (4)$$

The MKP has numerous applications, including task allocation among autonomous agents continuous double-call auctions [Kalagnanam *et al.*, 2001], multiprocessor scheduling [Labbé *et al.*, 2003], vehicle/container loading [Eilon and Christofides, 1971], and the assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices [Labbé *et al.*, 2003]. A special case of the MKP where the profits of the items are equal to their weights, i.e., $p_j = w_j$ for all j is the *Multiple Subset-Sum Problem* (MSSP).

The MKP (including the special case of the MSSP) is strongly NP-complete.¹ Thus, state-of-the-art algorithms for finding optimal solutions are based on branch-and-bound. Previous work has shown that for problems where the ratio of items to bins is relatively small (i.e., $n/m < 4$), the state-of-the-art algorithm is bin-completion, a bin-oriented branch-and-bound algorithm [Fukunaga and Korf, 2007].

The search space explored by bin-completion has many symmetric states. Previous work introduced some techniques for exploiting the symmetry and demonstrated their utility. In this paper, we further investigate methods for exploiting symmetries in the MKP bin-completion algorithm. We propose new techniques that result in significant improvements over the previous state of the art. These techniques are instances of the general symmetry breaking via dominance detection (SBDD) approach [Fahle *et al.*, 2001; Focacci and Milano, 2001].

The paper is organized as follows. We start by reviewing the bin completion algorithm (Section 2. Section 3 defines the

¹In contrast, the single-container 0-1 Knapsack problem is weakly NP-complete, and can be solved in pseudopolynomial time using dynamic programming.

basic framework we use for symmetry detection and breaking, and reviews previous algorithms for exploiting symmetry in the MKP. We then introduce new, generalized symmetry detection techniques (Sections 4-5) which are more powerful than the previous techniques, and we discuss methods for combining various symmetry mechanisms (Section 6). In Section 7, we experimentally evaluate various combinations of symmetry mechanisms. We compare our work with related work on symmetry detection and breaking and in the constraint programming literature in Section 8, and conclude with a discussion of results and directions for future work.

2 Bin-Completion Algorithm for the MKP

Bin-completion is a branch-and-bound algorithm for finding optimal solutions to multi-container assignment problems including the MKP and bin packing problems [Fukunaga and Korf, 2007]. We briefly describe this algorithm. For simplicity, we describe the algorithm in terms of the Multiple Subset-Sum Problem (MSSP), where each item’s profit equals its weight. Thus, whenever possible in the description below, we simply refer to an item by its weight. Generalization of the algorithm to MKP instances where the profits are not the same as the weights is straightforward.

A bin assignment $B_j = (item_1, \dots, item_k)$ is a set of all of the items that are assigned to a given bin j , $1 \leq j \leq m$. Thus, a valid solution to a MKP instance consists of a set of bin assignments, where each item appears in exactly one bin assignment. A bin assignment is *feasible* with respect to a given bin j if the sum of its weights does not exceed the capacity of the bin, c_j . Otherwise, the bin assignment is *infeasible*. We say that a bin assignment S is *maximal* with respect to bin j if S is feasible, and adding any other remaining items would make it infeasible.

The bin-completion algorithm searches a tree where each node at depth d , $1 \leq d \leq m$, represents a maximal, feasible bin assignment. The bin-completion algorithm for the MKP is shown in Figure 2, where each call to `search_MKP` corresponds to a node in the branch-and-bound search tree (e.g., Figure 1).

Nodes are pruned according to an upper bound which is based on a relaxation of the problem by Martello and Toth [1990] (Line 16). Pisinger’s R2 reduction procedure [Pisinger, 1999] is applied at each node (Line 11) in order to try to reduce the problem by eliminating some items for consideration. The `choose_bin` function (Line 18) selects the bin with least remaining capacity, and the undominated bin assignments are sorted (Line 20) in order of non-decreasing cardinality, and ties are broken in order of non-increasing profit. The `symmetric` function (Line 21) applies one of the symmetry detection strategies described in this paper.

Figure 1 shows part of an example bin-completion search tree. In addition to the pruning mechanisms mentioned above, a *dominance criterion* is applied (Fig 2, Line 19) to limit the nodes considered. Given two feasible bin assignments F_1 and F_2 , F_1 *dominates* F_2 if the value of the optimal solution which can be obtained by assigning F_1 to a bin is no worse than the value of the optimal solution that can be obtained by assigning F_2 to the same bin. Bin-completion prunes feasible

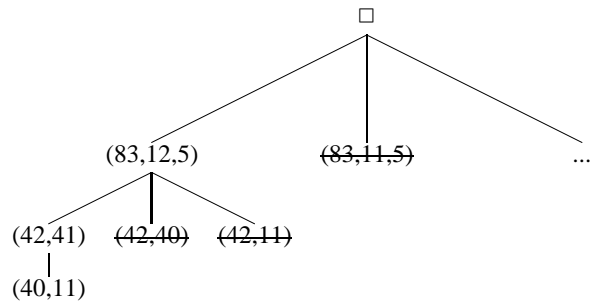


Figure 1: Part of the bin-completion search space for a bin packing instance with capacity 100 and items $\{83,42,41,40,12,11,5\}$. Each node represents a maximal, feasible bin assignment for a given bin. Bin assignments shown with a ~~strickethrough~~, e.g., $(83,11,5)$, are pruned because they are dominated according to the criterion in Proposition 1.

assignments which are dominated according to the following MKP dominance criterion [Fukunaga and Korf, 2007], which is based on the Martello-Toth dominance criterion for bin packing [Martello and Toth, 1990].

Proposition 1 (MKP Dominance Criterion) *Let A and B be two assignments that are feasible with respect to capacity c . A dominates B if B can be partitioned into i subsets B_1, \dots, B_i such that each subset B_k is mapped one-to-one to (but not necessarily onto) a_k , an element of A , and for all $k \leq i$, (1) the weight of a_k is greater than or equal the sum of the item weights of the items in B_k , and (2) the profit of item a_k is greater than or equal to the sum of the profits of the items in B_k .*

For example, given a bin with capacity 10 and items $9,8,7,3,2$, the undominated, feasible bin assignments are $(9),(8,2)$, and $(7,3)$.

3 Exploiting Symmetry

To describe our symmetry breaking mechanisms, which are instances of the general SBDD approach [Fahle *et al.*, 2001; Focacci and Milano, 2001], we first introduce some notation and define the notion of a *nogood*, which is central to all of our symmetry exploitation methods.

Let B^d denote a bin assignment which assigns the elements of set B to a bin at depth d . Thus, $(10,8,2)^1$ and $(10,7,3)^1$ denote two possible bin assignments for a bin at depth 1.

Definition 1 (Nogood) *Let X^d be some node in the bin-completion search tree at depth d . Let E^1, \dots, E^{d-1} be ancestors of X^d at depths $1, \dots, d-1$, respectively. For each such ancestor E_i , we say that every sibling of E^i to the left of E^i in the depth-first bin-completion search tree is a nogood with respect to X^d .*

For example, in Figure 3, $(8,2)^1$ is a nogood with respect to the descendants of $(7,3)^1$ and $(9)^1$; $(7,3)^1$ is a nogood with respect to the descendants of $(9)^1$.

Since bin-completion is a depth-first branch-and-bound algorithm, a nogood denotes a bin assignment (node) whose descendants have been exhaustively searched in the current

```

1 MKP_bin_completion(bins,items)
2 bestProfit =  $-\infty$ 
3 search_MKP(bins,items,0)

4 search_MKP(bins, items,sumProfit)
5 if bins==0 or items == 0
6 /*we have a candidate solution*/
7 if sumProfit > bestProfit
8 bestProfit = sumProfit
9 return
10 /* Attempt to reduce problem by eliminating some items from consideration*/
11 ri = reduce(bins,items)
12 if ri  $\neq$  0
13 search_MKP(bins, items \ ri, sumProfit)
14 return
15 /* Attempt to prune based on Martello-Toth SMKP upper bound */
16 if (sumProfit + compute_upper_bound(items,bins))  $\leq$  bestProfit
17 return

18 bin = choose_bin(bins)
19 undominatedAssignments = generate_undominated(items,capacity(bin))
20 foreach  $A \in$  sort_assignments(undominatedAssignments)
21 if not(symmetric(A))
22 assign A to bin
23 search_MKP(bins \ bin, items \ A,sumProfit+ $\sum_{i \in A}$ profit(i))

```

Figure 2: Outline of bin-completion for the multiple knapsack problem.

search tree. The union of all current nogoods is a concise description of the entire portion of the search tree which has been searched so far. This is similar to the use of the term “nogood” in [Focacci and Shaw, 2002]. The number of possible nogoods at a particular search depth d in the worst case is the number of undominated bin assignments considered at depths $1, \dots, d - 1$.

3.1 2-Swap-Symmetry

The *2-swap-symmetry* method was proposed by Korf in a bin-completion algorithm for the bin packing problem [Korf, 2003].² Suppose we have a MSSP instance with the items $\{9,8,7,3,2,\dots\}$ where all bins have a capacity of 10. A portion of the bin-completion search tree is shown in Figure 3. After exhausting the subproblem below the assignment $(8,2)^1$, and while exploring the subproblem below the assignment $(7,3)^1$, assume we find a solution that assigns $(8,2)^2$. We can swap the pair of items $(8,2)$ from the assignment $(8,2)^2$ with the pair of items $(7,3)$ from the assignment $(7,3)^1$, resulting in a solution with $(8,2)^1$ and $(7,3)^2$ and the same total profit. However, we have already exhausted the subtree below $(8,2)^1$ and we would have found a solution with the same total profit as the best solution in the subtree below $(7,3)^2$. Therefore, we can prune the branch below $(8,2)^2$, because it is redundant (in other words, we have detected that the current partial solution is symmetric to a partial state that has already been exhaustively searched).

Similarly, in Figure 3, the bin assignment $(7,3)^2$ under the

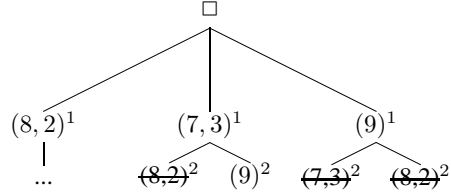


Figure 3: Both of the $(8,2)^2$ bin assignment under the $(7,3)^1$ and the $(9)^1$, as well as the assignment $(7,3)^2$ can all be pruned according to 2-swap-symmetry ($c_1 = 10, c_2 = 10$).

$(9)^1$, as well as the $(8,2)^2$ under the bin $(9)^1$ can both be pruned using 2-swap-symmetry.

More generally, given a bin assignment B^d for the bin at depth d , we can prune B^d if there is a nogood N^g with respect to B^d such that (1) B^d includes all the items in N^g , and (2) if we swap the items in N^g from B^d with the items that are currently assigned to the bin at depth g , both resulting bin assignments are feasible. The correctness follows straightforwardly from the definition of nogoods and the assumption that the nogood represents a node which has been exhaustively searched.

3.2 2-Swap-Dominance

The following *2-swap-dominance symmetry* technique³ [Fukunaga and Korf, 2007] allows even more pruning: Consider the partial search tree in Figure 4. Assume that all bins

²2-swap-symmetry was previously called “nogood pruning”, but we change the terminology in order to avoid confusion.

³2-swap-dominance symmetry was previously called “nogood dominance pruning”.

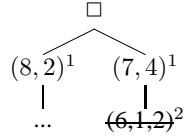


Figure 4: The bin assignment $(6, 1, 2)^2$ can be pruned by 2-swap-dominance ($c_1 = 11, c_2 = 11$).

have capacity 11. Suppose that after exhausting the subproblem below the assignment $(8, 2)^1$, and while exploring the subproblem below the assignment $(7, 4)^1$, we consider the assignment $(6, 2, 1)^2$. We can swap the items $(6, 2, 1)$ from bin 2 with the pair of items $(7, 4)$ from bin 1 and end up with a solution with $(6, 2, 1)^1$ and $(7, 4)^2$. However, according to the MKP dominance criterion, the set of items $(6, 2, 1)$ is dominated by $(8, 2)$, and we have already exhausted the search below the node $(8, 2)^1$, so we can prune the search under $(6, 1, 2)^2$ because it is not possible to improve upon the best solution under $(8, 2)^1$.

More generally, given a bin assignment B^d for depth d , we can prune B^d if there is a nogood N^g with respect to B^d such that (1) N^g dominates B according to the MKP dominance criterion (Proposition 1), and (2) The items in B^d can be swapped with the current items in bin g , such that the resulting bin assignments are both feasible. The correctness follows from the Proposition 1 and the assumption that the nogood represents a node which has been exhaustively searched.

To check whether a candidate assignment B is dominated by some nogood N , our current implementation uses a brute-force algorithm, which in the worst case takes time exponential in the cardinality of the bin assignment for each nogood.

4 Path-Symmetry

The techniques in the previous section only consider symmetries involving three bin assignments: a nogood at depth g , the current node at depth $d > g$, and the ancestor of the current node at depth g . We now generalize the symmetry techniques to detect and break symmetries involving more than these assignments.

Consider the search tree shown in Figure 5. Assume that the capacities for bins 1-4 are 11, 11, 12, and 12, respectively. Assume that we have already exhaustively searched the subtree under $(8, 2)^1$, and we have generated the node $(7, 4)^1, (8, 3)^2, (12)^3, (6, 2, 2)^4$. By rearranging the items in bins 1-3, we can obtain a new set of bin assignments: $(8, 2)^1, (7, 3)^2, (12)^3, (6, 4, 2)^4$. This is a symmetric rearrangement, as the optimal solution under the first set of bin assignments is the same as the optimal solution under the latter set of assignments. Thus, we can prune the node at $(6, 2, 2)^4$. Note that 2-swap-dominance would not allow us to prune this node, since we can not swap the $(7, 4)$ from bin 1 into bin 4 without exceeding bin 4's capacity. We define this symmetry more generally as follows.

Given a bin-completion search tree where we are considering a bin assignment for depth d , we define the *current path* from depth g to depth d as the union of bins $g, g + 1, \dots, d$. The *current path items* are the union of all items in the current

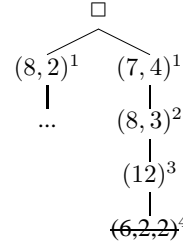


Figure 5: The bin assignment $(6, 2, 2)^4$ can be pruned by Path-Symmetry. ($c_1 = 11, c_2 = 11, c_3 = 12, c_4 = 12$).

path. For example, in Figure 5, if we are at node $(6, 2, 2)^4$, the current path from depth 1 to 4 is the set of bins 1, 2, 3, and 4, and the current path items are 7, 4, 8, 3, 12, 6, 2, 2.

Definition 2 (Path-Symmetry) Let B^d be a candidate bin assignment. Let N^g be a nogood with respect to B^d , and let P be the current path items from depth g to d . We say that there is a path-symmetry with respect to nogood N^g if two conditions hold: (1) every item in N^g is a member of P , and (2) it is possible to (a) assign the items from the current path items corresponding to the items of N^g ($Items(N^g) \subset P$) to bin g , and (b) assign the remaining items ($P \setminus Items(N^g)$) to bins $g + 1, \dots, d$ such that all bins g, \dots, d are feasible.

If there is a path-symmetry between B^d and some nogood N^g as defined above, B^d can be pruned. As with 2-swap symmetry, this follows directly from the definition of nogoods.

Checking the first condition of Definition 2 is straightforward. However, checking the second condition efficiently is not as straightforward, because it is essentially the decision version of a bin packing problem,⁴ where we attempt to pack the items in $P \setminus Items(N^g)$ into bins with capacities c_{g+1}, \dots, c_d . We describe several approaches:

In the first approach, we try to directly solve this bin packing problem using a simple backtracking algorithm (BT). The bin packing problem, like the MKP, is strongly NP-complete, and in the worst case, BT will take time which is $O(n^m)$, where n is the number of items and m is the number of bins. It is possible to avoid backtracking and use a standard bin packing heuristic such as first-fit decreasing (FFD), which has a polynomial complexity. Thus our second approach uses FFD to pack the items $P \setminus Items(N^g)$ into bins $g + 1, \dots, d$. The drawback of heuristics such as FFD is that it is not guaranteed to find a packing of the items into the bins even if one exists. However the symmetry check is still admissible – path-symmetry using a FFD check to test condition (2) may sometimes fail to prune a node that a BT check would have pruned, but will never prune a node that a BT check will not prune.

Another way to approximate the full check for condition (2) for path-symmetry is to limit the set of items that can be swapped among the bins. That is, instead of packing all of the

⁴In the decision version of bin packing, we are given m bins and n items, and the problem is to determine whether all n items can be packed into m bins such that the capacity constraints on all of the bins are not violated.

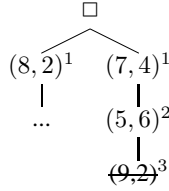


Figure 6: The bin assignment $(9, 2)^2$ can be pruned by Path-Dominance ($c_1 = 11, c_2 = 12, c_3 = 13$)

items $P \setminus Items(N^g)$ into bins $g + 1, \dots, d$, we can “lock” some of the items into their current bins and only consider packing the unlocked items.

We consider a *limited* packing problem where we (a) assign the items from the current path items corresponding to the items of $N^g (Items(N^g) \subset P)$ to bin g , and (b) pack the items $P \setminus Items(N^g)$ into bins $g + 4, \dots, d$, but in contrast to the full packing problem, we lock all of the items in $P \setminus Items(N^g)$ except for the items in bin g . In Figure 5, the unlocked items for this approximation would be the 7 and 4 from bin 1. Thus, this limited packing problem involves packing the 7 and 4 into three bins: bin #2 with remaining capacity 8 (the 8 is moved to bin #1, the original capacity is 11, and there is a 3 which is locked, so the remaining capacity is $11 - 3 = 8$), bin #3 with capacity 0 (no free space because there is a locked 12 occupying the bin), and bin #4 with remaining capacity 4. We can apply either BT or FFD to this limited packing problem. It is easy to see that the original 2-swap-symmetry check is a more restricted case of this approximate approach to path-symmetry.

Thus, we have four implementations path-symmetry, depending on the choice of methods for checking condition 2 in the path symmetry definition: (a) full packing with BT, (b) full packing with FFD, (c) limited packing with BT, and (d) limited packing with FFD.

5 Path-Dominance

Path-Dominance is the generalization of both 2-swap-dominance and path-symmetry. Consider the search tree shown in Figure 6 for an instance where the bin capacities for bins 1-3 are 11, 12, and 13, respectively. Assume that we have already exhaustively searched the subtree under $(8, 2)^1$, and we have generated the current path in the search tree, $(7, 4)^1, (5, 6)^2, (9, 2)^3$. By rearranging the items in bins 1-3, we can obtain a new set of bin assignments: $(7, 2)^1, (5, 6)^2, (9, 4)^3$. This is a symmetric rearrangement, since the optimal solution under the first sequence of bin assignments must be the same as the optimal solution the latter sequence of assignments. Thus, we can prune the node $(9, 2)^3$. More generally:

Definition 3 (Path-Dominance) *Let B^d be a candidate bin assignment. Let N^g be a nogood with respect to B^d , and let P be the current path items from depth g to d . We say that there is a path-dominance symmetry with respect to nogood N^g established at depth g if there exists some $s \subset P$ such two conditions hold: (1) s is dominated by N^g according to the MKP dominance criterion and (2) it is possible to (a) assign*

s to bin g , and (b) assign the remaining items $(P \setminus s)$ to bins $g + 1, \dots, d$ such that all bins g, \dots, d are feasible.

If there is a path-dominance symmetry between B^d and some nogood N^g as defined above, B^d can be pruned. As with 2-swap dominance, this follows from the definition nogoods and Proposition 1.

Our current implementation of path-dominance works as follows. We systematically enumerate every subset s of the current path items such that s is dominated by N^g and is maximal, i.e., there is no other item which can be packed into the N^g . For each such s , we test whether condition (2) of the path-dominance symmetry definition (Definition 3) is satisfied. If so, then a path-dominance has been detected, so the current node can be pruned. The test for condition (2) is the same as the corresponding test for path-symmetry in the previous section. Thus, the same four implementations of the check are considered: (a) full packing with BT, (b) full packing with FFD, (c) limited packing with BT, and (d) limited packing with FFD. In the worst case, this check is executed for each subset s that satisfies condition (1) of Definition 3, so checking for path-dominance can be quite expensive.

6 Combining Symmetry Breaking Strategies

Of the four symmetries defined above, 2-swap-symmetry is the weakest symmetry, and is subsumed by 2-swap-dominance, i.e., every node which would be pruned by 2-swap-symmetry would also be pruned by 2-swap-dominance. Similarly, 2-swap-symmetry is also subsumed by path-symmetry. In turn, 2-swap-dominance and path-symmetry are both subsumed by path-dominance.

However, there is a trade-off between the amount of pruning enabled by a symmetry relation and the amount of overhead incurred at each node in order to detect the symmetry.

To alleviate this trade-off, we combine the strategies by *chaining* some subset of them in a sequence so that the cheapest, least powerful symmetry is applied first. If this prunes the node, then the cost of applying the more powerful symmetries is not incurred. However, if the node is not pruned, then we apply another, more powerful symmetry, and so on.

We compared 11 configurations of the MKP solver, each using a different combination of symmetry mechanisms. These configurations are shown in Table 1. The four symmetry techniques are shown left to right. A “Y” indicates that the symmetry is applied. For the new path symmetries, we also specify whether the limited or full configuration (Section 4) is used, and also whether BT or FFD is used.

The PureBC configuration does not apply any symmetry detection in line 21 of Figure 2. The 2-Sym configuration applies only 2-swap-symmetry. The 2-Dom configuration applies 2-swap-symmetry first; if the node is not pruned, then it applies 2-swap-dominance (this is the configuration labeled “BC+NDP” in [Fukunaga and Korf, 2007]). As a final example, PathDom-Full-BT first tests for Nogood symmetry. If the test fails, then Nogood Dominance symmetry is applied. If that fails, then a full Path Dominance test using backtracking is applied.

Name	2-Swap-Symmetry	2-Swap-Dominance	Path-Symmetry	Limited/Full	BT/FFD	Path-Dominance	Limited/Full	BT/FFD
PureBC	N	N	N	n/a	n/a	N	n/a	n/a
2-Sym	Y	N	N	n/a	n/a	N	n/a	n/a
2-Dom	Y	Y	N	n/a	n/a	N	n/a	n/a
PathSym-Full-BT	Y	Y	Y	Full	BT	N	n/a	n/a
PathSym-Full-FFD	Y	Y	Y	Full	FFD	N	n/a	n/a
PathSym-Lim-BT	Y	Y	Y	Limited	BT	N	n/a	n/a
PathSym-Lim-FFD	Y	Y	Y	Limited	FFD	N	n/a	n/a
PathDom-Full-BT	Y	Y	N	n/a	n/a	Y	Full	BT
PathDom-Full-FFD	Y	Y	N	n/a	n/a	Y	Full	FFD
PathDom-Lim-BT	Y	Y	N	n/a	n/a	Y	Limited	BT
PathDom-Lim-FFD	Y	Y	N	n/a	n/a	Y	Limited	FFD

Table 1: MKP solver configurations - each row corresponds to a combination of symmetry detection mechanisms

7 Experimental Results

We experimentally evaluated the 11 solver configurations described in the previous section, using the following four classes of problems (originally from [Pisinger, 1999]):

- *uncorrelated instances*, where the profits p_j and weights w_j are uniformly distributed in $[min, max]$.
- *weakly correlated instances*, where the w_j are uniformly distributed in $[min, max]$ and the p_j are randomly distributed in $[w_j - (max - min)/10, w_j + (max - min)/10]$ such that $p_j \geq 1$,
- *strongly correlated instances*, where the w_j are uniformly distributed in $[min, max]$ and $p_j = w_j + (max - min)/10$, and
- *multiple subset-sum instances*, where the w_j are uniformly distributed in $[min, max]$ and $p_j = w_j$.

The bin capacities were set as follows: The first $m - 1$ capacities c_i were uniformly distributed in $[0.4 \sum_{j=1}^n w_j/m, 0.6 \sum_{j=1}^n w_j/m]$ for $i = 1, \dots, m - 1$. The last capacity c_m is chosen as $c_m = 0.5 \sum_{j=1}^n w_j - \sum_{i=1}^{m-1} c_i$ to ensure that the sum of the capacities is half of the total weight sum. Degenerate instances were discarded as in Pisinger’s experiments [1999]. That is, we only used instances where: (a) each of the items fits into at least one of the containers, (b) the smallest container is large enough to hold at least the smallest item, and (c) the sum of the item weights is at least as great as the size of the largest container. In our experiments, we used items with weights in the range $[1, 1000]$.

We used instances where the ratio of items to bins is between 2 and 3, because it has been shown that for these instances, bin-completion is clearly the state of the art approach [Fukunaga and Korf, 2007]. We only report comparisons among bin-completion variants due to space; we also ran Pisinger’s Mulknep solver [1999] on the same instances, but Mulknep’s was not competitive with any of the bin-completion variants shown here. See [Fukunaga and Korf, 2007] for a full comparison of bin-completion with Mulknep and MTM, an older solver by Martello and Toth [1990].

The results are shown in Table 2. All experiments were run on a 2.4 GHz Intel Core2 Duo. Each experiment was run on 20 instances (all configurations were run on the same instances). The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (300

seconds/instance). The *time* and *nodes* show average time spent and nodes searched on the successful runs, excluding the failed runs (thus, in the experiments where there were timeouts, the fail column is the most significant result).

First, note that the PathDom-Full-BT and PathSym-full-BT configurations, which implement the full versions of the path dominance and path symmetry relations, respectively, significantly reduce the size of the branch-and-bound tree compared to 2-Dom, which was the previous state of the art [Fukunaga and Korf, 2007]. However, the runtimes are not significantly better – in fact, for many instances, they run much slower than 2-Dom due to the overhead of symmetry detection.

Fortunately, configurations that use a more limited version of path-symmetry and path-dominance fare better: at a cost of some increase in the number of nodes searched, the overhead per node is drastically reduced. In particular, the PathSym-Limited-FFD configuration significantly outperforms the previous state of the art 2-Dom with respect to both runtime and nodes searched. The number of nodes searched by PathSym-Limited-FFD is about an order of magnitude smaller than that of 2-Dom for uncorrelated instances, and about a factor of 2 smaller for subset sum instances, (based on node counts from problems where both configurations solved all of the instances). The runtimes for PathSym-Limited-FFD are a factor of 2-3 faster than 2-Dom, which shows that the increased complexity per search node is more than offset by the search efficiency gained. Thus, bin-completion using PathSym-Limited-FFD is the new state of the art algorithm for optimally solving MKP instances for the class of problems considered here.

Furthermore, the number of nodes searched by PathSym-Limited-FFD is within a factor of 2 of the number searched by PathDom-Full-BT (the most “powerful” configuration tested with respect to the number of nodes searched), despite the fact that PathSym-Limited-FFD uses both a less powerful symmetry relation (path-symmetry as opposed to path-dominance) and uses a limited, approximate mechanism for detecting the symmetry (Limited-FFD) as opposed to Full-BT.

While full path-dominance with backtracking (PathDom-Full-BT) was the most efficient configuration with respect to nodes searched, the overhead of applying full path dominance at each node significantly increases runtime and does not appear worthwhile. We are currently investigating methods which apply path-dominance more selectively (or in combination with path symmetry), but have not yet found a combination of techniques which outperformed the PathSym-Lim-

FFD configuration.

Overall, these results show that exploiting symmetry is a very effective technique for the MKP. Even the simplest configuration which exploits symmetry, 2-Sym, consistently outperforms PureBC (bin-completion without any symmetry detection), by at least a 1-2 orders of magnitude difference in the number of nodes searched, and a consistent runtime speedup of 1-2 orders of magnitude. The new PathSym-Lim-FFD configuration searches 1-4 orders of magnitude fewer nodes, and also run up to 3 orders of magnitude faster than PureBC.

8 Related Work

Our MKP symmetry breaking mechanisms are domain-specific instances of the symmetry breaking via dominance detection (SBDD) approach [Fahle *et al.*, 2001; Focacci and Milano, 2001]. A significant difference is that in addition to detecting equivalences to previously explored subtrees (2-swap-symmetry and path-symmetry), our 2-swap-dominance and path-dominance algorithms also detect partial solutions which are dominated by previously explored subtrees (according to Proposition 1).

Our work is also similar to the pruning technique proposed by Focacci and Shaw [2002] for constraint programming, which was applied to the TSP with time windows. Both methods attempt to prune the search by proving that the current node at depth j , which represents a partial j -variable (bin⁵) solution x , is dominated by some previously explored i -variable (bin) partial solution (nogood bin assignment) q , where $i < j$.

The main difference between our method and Focacci and Shaw's method is the approach used to test for dominance. Focacci and Shaw's method extends q to a j -variable partial solution q' which dominates x . They apply a local search procedure to find the extension q' . In contrast, our methods start with a partial, j -bin solution x and try to transform it to a partial solution x' such that \bar{x}'_i , the subset of x' including the first i bins, is dominated by the i -bin partial solution q . We do this by transforming (via item swaps) the contents of bins $i, i + 1, \dots, j$ in x to derive a feasible partial solution x' such that \bar{x}'_i is dominated by q .

9 Conclusions

This paper investigated techniques for exploiting symmetry in a branch-and-bound algorithm for the multiple knapsack problem. Specifically, we focused on techniques for breaking symmetries in the search space explored by the bin-completion branch-and-bound algorithm. We proposed two new, symmetry breaking mechanisms (path symmetry and path dominance) which are generalizations of previously studied strategies (2-swap symmetry and 2-swap dominance). We showed that the new symmetries are significantly more powerful than their predecessors, reducing the branch-and-bound trees by up to an order of magnitude compared to the previous state of the art, 2-swap dominance. However,

⁵Our analogues of CP variables and values are bins and bin assignments, respectively

we also found that these new symmetry breaking mechanisms added a significant amount of overhead per node, which resulted in slower runtimes compared to the simpler 2-swap dominance symmetry. We showed that approximate (but admissible) implementations of path symmetry and path dominance combined with previous mechanisms (e.g., the PathSym-Lim-FFD configuration) are able to achieve a favorable tradeoff between pruning and per-node complexity, significantly outperforming the previous state of the art on the classes of MKP instances we considered.

There are several directions for future work. Although we have investigated approximate symmetry detection, our focus so far has been on identifying powerful symmetry relations such as path-symmetry and path-dominance. There are opportunities to develop more efficient implementations of path symmetry and path dominance. Although PathSym-Lim-FFD searched an order of magnitude less nodes than 2-Dom on some instances, the runtime speedup is only a factor of 2-3. More efficient implementation strategies may enable path-symmetry to achieve close to an order of magnitude speedup relative to the 2-Dom configuration.

Although path-dominance is our most powerful symmetry relation, the current implementation is not competitive with path symmetry due to the large overhead incurred to check for path-dominance at each node. We are currently investigating improved implementations and approximate detection strategies which make path-dominance more viable as part of a combined (chained) strategy.

References

- [Eilon and Christofides, 1971] S. Eilon and N. Christofides. The loading problem. *Management Science*, 17(5):259–268.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. CP-01*, pages 93–107.
- [Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. CP-01*, 77–92.
- [Focacci and Shaw, 2002] F. Focacci and P. Shaw. Pruning sub-optimal search branch branches using local search. In *Proc. CP-AI-OR*, pages 181–189.
- [Fukunaga and Korf, 2007] A. Fukunaga and R. Korf. Bin-completion algorithms for multicontainer packing, knapsack, and covering problems. *Journal of Artificial Intelligence Research*, 28:393–429.
- [Kalagnanam *et al.*, 2001] J.R. Kalagnanam, A.J. Davenport, and H.S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1:221–238.
- [Korf, 2003] R. Korf. An improved algorithm for optimal bin packing. In *Proceedings of IJCAI*, pages 1252–1258.
- [Labbé *et al.*, 2003] M. Labbé, G. Laporte, and S. Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *Eur. J. of Operational Research*, 149:490–498.
- [Martello and Toth, 1990] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. J. Wiley & Sons.
- [Pisinger, 1999] D. Pisinger. An exact algorithm for large multiple knapsack problems. *Eur. J. of Operational. Res.*, 114:528–541.

Uncorrelated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.951	206419	7	49.436	7545714	1	90.290	8875301	0	9.130	1662504	12	123.233	16112720
2-Sym	0	0.011	1324	0	0.924	84266	2	39.127	2940614	0	0.597	63708	10	56.951	3941467
2-Dom	0	0.010	1059	0	0.600	43875	1	29.319	1735503	0	0.572	47193	10	62.507	3392116
PathDom-Full-BT	0	0.013	188	0	1.513	1924	3	36.779	13483	0	0.786	5031	20	-	-
PathDom-Full-FFD	0	0.006	199	0	0.236	2783	1	15.346	41349	0	0.975	8109	16	89.955	373482
PathDom-Lim-BT	0	0.005	199	0	0.262	2547	0	19.380	40216	0	0.778	7252	14	121.330	317797
PathDom-Lim-FFD	0	0.005	199	0	0.259	2547	0	19.398	40238	0	0.780	7258	14	121.665	318298
PathSym-Full-BT	0	0.005	236	0	0.155	2498	0	29.416	30607	0	0.431	6761	16	170.84	337027
PathSym-Full-FFD	0	0.003	245	0	0.108	3341	0	5.510	70684	0	0.322	10093	9	77.686	978742
PathSym-Lim-BT	0	0.003	239	0	0.099	3141	0	4.230	57896	0	0.284	9422	8	56.068	756632
PathSym-Lim-FFD	0	0.003	239	0	0.097	3142	0	4.208	57962	0	0.284	9432	8	55.568	758425
Weakly Correlated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.717	69033	2	32.962	2328058	15	176.854	12712681	0	4.269	413635	14	125.483	5452363
2-Sym	0	0.026	1418	0	0.608	32680	0	14.699	551669	0	0.762	46564	10	69.61	2197822
2-Dom	0	0.016	787	0	0.302	12273	0	6.532	168474	0	0.703	36682	10	68.902	2024937
PathDom-Full-BT	0	0.015	214	0	0.370	881	1	39.83	4668	0	7.399	5056	18	117.355	109057
PathDom-Full-FFD	0	0.011	236	0	0.295	1251	0	4.969	10967	0	2.134	8894	14	97.217	305538
PathDom-Lim-BT	0	0.010	238	0	0.214	1218	0	3.014	8664	0	1.472	7609	14	78.441	276743
PathDom-Lim-FFD	0	0.009	238	0	0.213	1218	0	3.004	8668	0	1.454	7609	14	77.115	276776
PathSym-Full-BT	0	0.015	254	0	0.096	1248	0	4.904	8417	0	4.643	7061	13	116.37	321874
PathSym-Full-FFD	0	0.007	273	0	0.075	1647	0	1.117	16347	0	0.422	11730	8	94.2817	1250029
PathSym-Lim-BT	0	0.008	273	0	0.070	1623	0	0.825	13094	0	0.348	10562	7	88.891	1172600
PathSym-Lim-FFD	0	0.008	273	0	0.071	1623	0	0.822	13133	0	0.346	10564	7	89.148	1174057
Strongly Correlated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.141	3197	2	28.443	209643	16	202.955	792114	0	0.650	13489	17	192.077	2008851
2-Sym	0	0.021	285	0	1.707	6179	3	39.125	79392	0	0.349	6623	13	145.069	1422360
2-Dom	0	0.013	175	0	1.203	3174	2	33.937	50439	0	0.339	6373	13	136.639	1311413
PathDom-Full-BT	0	0.007	70	0	1.113	653	2	65.039	6341	0	1.177	3674	20	-	-
PathDom-Full-FFD	0	0.008	72	0	0.661	804	0	40.147	13374	0	0.835	4412	19	257.260	126024
PathDom-Lim-BT	0	0.008	72	0	0.675	833	1	24.341	11578	0	0.773	4328	19	180.463	107563
PathDom-Lim-FFD	0	0.008	72	0	0.673	833	1	24.309	11579	0	0.773	4329	19	180.150	107564
PathSym-Full-BT	0	0.006	74	0	0.619	781	1	40.185	8977	0	0.396	4108	20	-	-
PathSym-Full-FFD	0	0.008	75	0	0.646	931	1	25.359	16569	0	0.276	4814	12	130.194	901912
PathSym-Lim-BT	0	0.007	76	0	0.675	979	1	25.779	16267	0	0.273	4810	12	103.099	743737
PathSym-Lim-FFD	0	0.009	76	0	0.670	979	1	26.271	16275	0	0.271	4811	12	103.094	745103
Subset-Sum Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.049	1488	0	4.964	1246450	6	46.046	1386179	0	0.121	3630	4	49.785	1282436
2-Sym	0	0.005	132	0	0.073	1893	0	1.1612	23909	0	0.080	2137	2	30.118	722638
2-Dom	0	0.001	88	0	0.037	941	0	0.545	11534	0	0.078	2096	2	30.067	690547
PathDom-Full-BT	0	0.004	50	0	0.209	325	1	55.56	2178	0	0.709	1456	20	-	-
PathDom-Full-FFD	0	0.002	51	0	0.048	382	0	1.937	3721	0	0.239	1677	10	100.001	99212
PathDom-Lim-BT	0	0.002	51	0	0.042	393	0	1.176	3475	0	0.202	1629	10	73.015	86786
PathDom-Lim-FFD	0	0.003	51	0	0.043	393	0	1.167	3475	0	0.202	1629	10	73.240	86789
PathSym-Full-BT	0	0.002	51	0	0.047	371	0	9.975	3177	0	0.153	1637	18	200.315	45671
PathSym-Full-FFD	0	0.001	52	0	0.023	425	0	0.251	4679	0	0.078	1836	2	38.967	522329
PathSym-Lim-BT	0	0.003	53	0	0.019	451	0	0.254	4903	0	0.077	1838	2	29.725	450926
PathSym-Lim-FFD	0	0.001	53	0	0.020	451	0	0.250	4907	0	0.080	1839	2	29.632	452028

Table 2: Multiple knapsack problem results: Comparison of Bin-Completion with various combinations of 2-Swap-Symmetry, 2-Swap-Dominance Symmetry, Path-Symmetry, and Path-Dominance on random MKP instances. The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (300 seconds/instance). The *time* (seconds on 2.4GHz Intel Core2 Duo) and *nodes* show average time spent and nodes searched on the successful runs, excluding the failed runs.

Minimal Ordering Constraints for some Families of Variable Symmetries

Andrew Grayland School of Comp. Sci., St Andrews, UK. andyg@cs.st-and.ac.uk

Ian Miguel School of Comp. Sci., St Andrews, UK. ianm@cs.st-and.ac.uk

Colva Roney-Dougal School of Maths and Statistics, St Andrews, UK. colva@mcs.st-and.ac.uk

Abstract

Existing methods of breaking all variable symmetries in a constraint satisfaction problem over n variables, by adding lexicographic ordering constraints, can require $n!$ new constraints. This adds an unacceptable overhead to the solving process. In certain cases, for example when the variables are constrained to take distinct values, a reduction to an $O(n)$ set of constraints is possible. This paper studies some commonly-occurring families of groups and shows how the set of ordering constraints can be reduced in each case.

1 Introduction

Constraint programming supports the solution of a combinatorial problem in two stages. First, the problem is characterised or *modelled* as a *constraint satisfaction problem* (CSP): a finite set of decision variables, each with a finite set of potential values, and a set of constraints on the allowed assignments of values to the variables. Second, a constraint solver is used to search for *solutions*: assignments to the decision variables that satisfy all the constraints. Constraint models often contain *symmetries* that partition the set of assignments into equivalence classes. Symmetries can be exploited by restricting the search for a solution to one member of each equivalence class (*symmetry breaking*), dramatically reducing search.

There exist many symmetry breaking techniques. Some methods are *static* in that they break symmetry of the CSP model before search. The most popular static approach is to add symmetry breaking constraints to the model [Crawford *et al.*, 1996]. We can also use *reformulation* [Smith, 2001], where the model is changed into a new model where the symmetries no longer exist. Other methods are *dynamic* and utilise methods to break symmetry during search. Two popular examples of dynamic symmetry breaking are SBDS [Gent *et al.*, 2000] and SBDD [Fahle *et al.*, 2001]. In this paper we examine a successful static symmetry breaking technique and examine ways in which it can be improved upon in specific cases.

Crawford *et al* [Crawford *et al.*, 1996] describes a static symmetry-breaking method that involves adding constraints to the model. This method is called *lex-leader*: one member of each equivalence class is designated as lexicographically least, and a set of lexicographic ordering constraints are added to preclude all other members of that class. The disadvantage of this method is that, for a CSP with n variables, it can produce $n!$ lexicographic ordering constraints. The overhead of adding this number of constraints to the CSP usually outweighs the benefit of breaking the symmetry.

In many cases, this large set of constraints can be reduced to a much smaller set that still breaks all the symmetry [Frisch *et al.*, 2003; Öhrman, 2005]. However, these general reduction methods are themselves prohibitively costly. Puget [Puget, 2005] has identified a special case, where each variable must be assigned a distinct value, in which the set of ordering constraints collapses to just $n - 1$ binary inequalities. This paper follows somewhat in this vein. It considers the mathematical *group* that describes certain symmetries, the associated set of lexicographic ordering constraints necessary to break those symmetries, and how that set can be reduced to a fix-point that we define as minimal.

Further to this the paper discusses combinations of commonly occurring groups. These combinations are known to be much larger than the groups they are constructed from; they therefore pose a similar problem in relation to the number of constraints required to break the symmetries in a CSP with that symmetry group. We examine methods of breaking these symmetries by utilising the minimal sets of constraints defined for the constituent groups.

2 Background

A finite-domain constraint satisfaction problem comprises: a finite set of variables \mathcal{X} ; for each variable $x \in \mathcal{X}$, a finite set of values (its *domain*); and a finite set \mathcal{C} of constraints on the variables. Each constraint $c \in \mathcal{C}$ is defined over a sequence, \mathcal{X}' , of variables drawn from \mathcal{X} . A subset of the Cartesian product of the domains of the members of \mathcal{X}' gives the set of allowed combinations of values. A *complete assignment* maps every variable in a given CSP to a member of its domain.

A *variable symmetry* of a CSP is a bijection $f : \mathcal{X} \rightarrow \mathcal{X}$ such that $\{(x_i, a_i) : 1 \leq i \leq n\}$ is a solution if and only if $\{(f(x_i), a_i) : 1 \leq i \leq n\}$ is a solution.

Any group can be represented by a set G of bijections from a set \mathcal{X} to itself (or *permutations* of \mathcal{X}), such that G is closed under composition of functions and inversion. We are interested in sets of variable symmetries of the CSP. The symmetric group, S_n , is the group whose elements are the set of bijections from $\{1, \dots, n\}$ into itself. A group of permutations is *transitive* if any variable can be mapped to any other.

Having identified a symmetry within a model we identify a set of symmetry-breaking constraints sufficient to break it using the lex-leader method. We first define an ordering on the decision variables, then add constraints that order the assignments to these variables. To illustrate, we consider the symmetric group S_3 acting on variables, x_1 , x_2 , and x_3 . Here the permutation (x_1x_2) means x_3 maps to itself, and x_1 and x_2 map to each other, and $(x_1x_2x_3)$ means x_1 maps to x_2 , x_2 maps to x_3 , and x_3 maps to x_1 :

$$(), (x_1x_2), (x_1x_3), (x_2x_3), (x_1x_2x_3), (x_1x_3x_2)$$

The permutation denoted $()$ is called the *identity*, and represents the mapping of each variable to itself. We choose x_1 to be the most significant variable in the ordering, x_2 the next most significant, and x_3 the least significant. The next step is to add symmetry-breaking constraints to allow only one member of each equivalence class of assignments induced by the symmetry:

$$\begin{aligned} x_1x_2x_3 &\leq_{\text{lex}} x_1x_2x_3, & x_1x_2x_3 &\leq_{\text{lex}} x_2x_1x_3, \\ x_1x_2x_3 &\leq_{\text{lex}} x_3x_2x_1, & x_1x_2x_3 &\leq_{\text{lex}} x_1x_3x_2, \\ x_1x_2x_3 &\leq_{\text{lex}} x_2x_3x_1, & x_1x_2x_3 &\leq_{\text{lex}} x_3x_1x_2 \end{aligned}$$

Notice that there is one constraint per permutation of S_3 . In general, there are $n!$ permutations for the group S_n and so $(n!)$ n -ary constraints are produced by the lex-leader method to break all symmetries.

3 Reducing the Ordering Constraints

Frisch and Harvey [Frisch *et al.*, 2003] describe two rules to reduce the number and arity of constraints whilst maintaining complete symmetry breaking.

1. If c is a constraint of the form $\alpha X\beta \leq_{\text{lex}} \gamma Y\delta$, and $\alpha = \gamma$ logically implies $X = Y$ then we may replace it with $\alpha\beta \leq_{\text{lex}} \gamma\delta$.
2. If C is a set of constraints of the form $C' \cup \{\alpha\beta \leq_{\text{lex}} \gamma\delta\}$, and $C' \cup \{\alpha = \gamma\}$ logically implies $\beta \leq_{\text{lex}} \delta$, then we may replace C with $C' \cup \{\alpha \leq_{\text{lex}} \gamma\}$.

For example, consider the constraint $x_1x_2 \leq_{\text{lex}} x_2x_1$. From the definition of lexicographic ordering, to ensure that the constraint is satisfied we need only compare a pair of variables if each pair of more significant variables are equal. Here, if $x_1 = x_2$ then trivially the second pair *must* be equal. Therefore, by Rule 1 we need only consider the first pair of variables, reducing this constraint

to $x_1 \leq x_2$ without modifying the set of solutions. In the S_3 example given in the previous section, Rule 1 reduces the set of lexicographic ordering constraints to: $x_1 \leq x_2$, $x_2 \leq x_3$, $x_1 \leq x_3$, $x_1x_2 \leq_{\text{lex}} x_2x_3$, $x_1x_2 \leq_{\text{lex}} x_3x_1$. Application of Rule 2 simplifies the constraints further to: $x_1 \leq x_2$, $x_2 \leq x_3$.

Öhrman [Öhrman, 2005] defines a further Rule 3 which states that if we have a set of constraints C of the form $C' \cup \{\alpha X\beta \leq_{\text{lex}} \gamma Y\delta\}$, and $C' \cup \{\alpha = \gamma\}$ logically implies $X = Y$ (or $X < Y$ where $|\beta| = 0$ and $|\delta| = 0$), then we may replace C with $C' \cup \{\alpha\beta \leq_{\text{lex}} \gamma\delta\}$. Rule 3 extends the stated Rules 1 and 2 in that it allows both the consideration of all pairs of variables in any one lex constraint, provided by Rule 1, and the implications derived from considering the entire set of lex constraints, provided by Rule 2. Notice that the support required for removal of the least significant pair, the parenthetical part of the rule which is a restatement of Rule 2, remains essentially different from all other pairs. For this reason we find it useful to separate the action of Öhrman's Rule 3 into Rule 2 and a new Rule 3'.

3' If we have a set of constraints C of the form $C' \cup \{\alpha X\beta \leq_{\text{lex}} \gamma Y\delta\}$, and $C' \cup \{\alpha = \gamma\}$ logically implies $X = Y$, then we may replace C with $C' \cup \{\alpha\beta \leq_{\text{lex}} \gamma\delta\}$.

Definition 1 A set of lexicographic ordering constraints is said to be *minimal* if the set is unchanged under application of Rules 2 and 3'.

Achieving minimality from a factorial number of constraints by mechanical application of Rules 2 and 3' is expensive. In practice, it has proved infeasible as much of the time saved by breaking the symmetries is re-introduced in this pre-processing stage [Öhrman, 2005]. We can derive a lower bound on the number of binary inequality constraints produced by this process.

Theorem 1 Let P be a CSP with decision variables $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$. Assume that P has a transitive group of variable symmetries and that the domains of x_i , $1 \leq i \leq n$, contain more than one value. Then, not considering other constraints in the CSP,

the minimum number of binary \leq constraints required to remove all but one member of each equivalence class of assignments is $n - 1$.

Proof 1 Any constraint graph with $n - 2$ binary constraints is disconnected. Let x_1 be the most significant variable. Let x_i be a decision variable that is not connected to x_1 , and such that x_i is most significant in its component of the constraint graph. Consider the full assignment that assigns $x_i = a$, where a is minimal in the domain of x_i , and $x_j = b$ for $j \neq i$, where $b > a$. Since there is a symmetry mapping x_i to x_1 there is a symmetry mapping this full assignment to a full assignment with $x_1 = a$. Thus there will remain more than one solution from an equivalence class after addition of the $n - 2$ binary constraints, and therefore $n - 2$ binary constraints will not suffice. \square

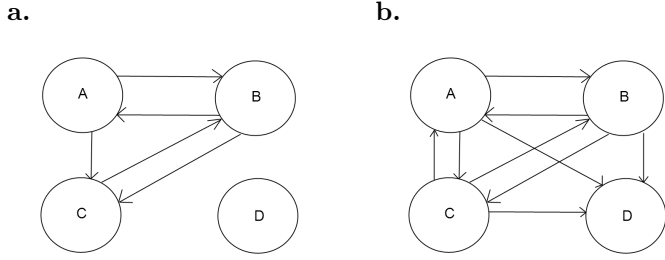


Figure 1: An inequality graph and the transitive closure of that graph.

4 Reduction Rules and Inequality Graphs

We begin by introducing the *inequality graph* as a device to visualise the operation of Rules 1,2 and 3'.

Definition 2 Given a CSP with set of variables \mathcal{X} and set of constraints \mathcal{C} , the corresponding inequality graph G is a directed graph with one node per element of \mathcal{X} . The graph G contains directed edges as follows. If, for some $x_i, x_j \in \mathcal{X}$, \mathcal{C} contains:

1. $x_i \leq x_j$ then G contains an edge from x_i to x_j .
2. $x_i = x_j$ then G contains an edge from x_i to x_j , and an edge from x_j to x_i .
3. $x_i \dots \leq_{\text{lex}} x_j \dots$ then G contains an edge from x_i to x_j .

Given a set of lexicographic constraints \mathcal{C} on variables \mathcal{X} , we consider the process of applying Rule 2 to remove the least significant pair of variables in some $c \in \mathcal{C}$. Following the rule, we begin by adding equality constraints between all more significant pairs of variables $x_i, x_j \in c$. To illustrate, Figure 1 shows the inequality graph for a CSP $C \leq D$ from constraint $ABC \leq_{\text{lex}} BCD$ is under consideration for removal by Rule 2. The assumed equalities, $A = B$ and $B = C$, are represented by dual directed edges between the respective nodes and the inequality $A \leq C$ implied by $ABC \leq_{\text{lex}} CDA$ is represented by a directed edge from A to C .

The antecedent of Rule 2 requires the identification of implied inequality constraints. The identification of such equalities can be characterised partially in terms of taking the transitive closure of the inequality graph.

Definition 3 Consider a directed graph $G = (\mathcal{X}, E)$, where \mathcal{X} is the set of nodes and E is the set of edges. The transitive closure of G is a graph $G' = (\mathcal{X}, E')$ such that for all $x_i, x_j \in \mathcal{X}$ there is an edge (x_i, x_j) in E' if and only if there is a path from x_i to x_j in G [Nuutila, 1995].

Returning to our example, since there is a path from C to A via B , taking the transitive closure of the inequality graph in Figure 1. adds a directed edge from C to A . Since there is now both a directed edge from A to C , and from C to A , it is implied that $A = C$. This allows us to

simplify the constraint $ABC \leq_{\text{lex}} CDA$ to $BC \leq_{\text{lex}} DA$. Consequently, it is now clear that $B \leq D$. Hence, the inequality graph of this new, simpler problem contains a directed edge from B to D . The transitive closure of this graph is shown in Figure 1b. Notice that there is a directed edge from C to D , hence the pair C and D can be removed from $ABC \leq_{\text{lex}} BCD$ via Rule 2.

Generally, as pointed out by Öhrman [Öhrman, 2005], the operation of establishing the antecedent of Rule 2 can be described as follows:

1. Let P be the initial CSP, combined with the equality constraints assumed by Rule 2.
2. Generate the inequality graph G for P and take its transitive closure, G' .
3. If G' contains edges corresponding to equalities not represented explicitly in P , add these equalities to P and go to 2.
4. Otherwise, the antecedent of Rule 2 is satisfied if the corresponding edge is present in G' .

5 Minimal sets of lexicographic ordering constraints for some families of groups

We now consider some specific families of groups. In each case we produce a number of lex constraints that is linear in the number of variables. We have shown that all sets of constraints in this section are minimal but have omitted all minimality proofs for reasons of space.

5.1 Cyclic Groups

If all elements of a group G can be written as powers of some fixed $g \in G$ then G is *cyclic*. In this subsection we produce a set of minimal lex constraints for breaking the cyclic group. We will assume throughout that the elements of the cyclic group are powers of the permutation (x_1, x_2, \dots, x_n) .

Theorem 2 Let P be a CSP with n decision variables, $\{x_1, x_2, \dots, x_n\}$. If the symmetry group of P is the cyclic group of variable symmetries then a complete minimal set A of symmetry breaking constraints is:

$$\begin{aligned}
 x_1 &\leq x_2 \\
 x_1x_2 &\leq_{\text{lex}} x_3x_4 \\
 x_1x_2x_3 &\leq_{\text{lex}} x_4x_5x_6 \\
 &\vdots \\
 x_1x_2 \dots x_{n/2+1} &\leq_{\text{lex}} x_{n/2+2} \dots x_n x_1 x_2 \quad n \text{ even} \\
 x_1x_2 \dots x_{(n+1)/2} &\leq_{\text{lex}} x_{(n+3)/2} \dots x_n x_1 \quad n \text{ odd} \\
 &\vdots \\
 x_1x_2 \dots x_{n-1} &\leq_{\text{lex}} x_n x_1 \dots x_{n-2}
 \end{aligned}$$

Proof 2 We show that A is equivalent to the lex-leader constraints for C_n , namely

$$x_1 \dots x_n \leq_{\text{lex}} x_{k+1} \dots x_n x_1 \dots x_k \quad (1)$$

for $1 \leq k < n$.

We will refer to the i th constraint in A as a_i . To see that the lex-leader constraints imply A , note that each constraint in A is an initial subsequence of one of the lex-leader constraints, and so is certainly implied.

We now prove the converse, we wish to show that A implies (1) for $1 \leq k < n$. The constraint a_k is $x_1 \dots x_k \leq_{\text{lex}} x_{k+1} \dots x_m$, where $m = 2k$ if $k \leq n/2$ and $m = 2k - n$ otherwise. So the initial subsequence of length k of (1) holds, and we may assume without loss of generality that $x_1 = x_{k+1}, x_2 = x_{k+2}, \dots, x_k = x_m$.

We need to prove that under these assumptions

$$x_{k+1} \dots x_n \leq_{\text{lex}} x_{2k+1} \dots x_n x_1 \dots x_k. \quad (2)$$

We have $x_1 \dots x_k = x_{k+1} \dots x_m$, so we can rewrite the left-hand side of (2) as $x_1 \dots x_k x_{m+1} \dots x_n$. The constraint a_{2k} is $x_1 \dots x_{2k} \leq_{\text{lex}} x_{2k+1} \dots x_p$ where $p = (4k - 1 \bmod n) + 1$. Thus the initial subsequence of length k of (2) holds, and we may assume without loss of generality that $k < n/2$, and that $x_1 = x_{k+1} = x_{2k+1}, x_2 = x_{k+2} = x_{2k+2}, \dots, x_k = x_{2k} = x_{3k}$.

If n is divisible by k then by induction (1) holds.

Assume that n is not divisible by k , let $b = \lfloor n/k \rfloor$ and $c = n \bmod k$ so that $n = bk + c$. We must show that

$$x_{bk+1} \dots x_n \leq_{\text{lex}} x_{(b+1)k-n+1} \dots x_k. \quad (3)$$

Note that (3) has length $c < k$, and that $(b+1)k - n + 1 = bk + c - n + k - c + 1 = k - c + 1$.

Consider the constraint a_{k-c} , namely

$$x_1 \dots x_{k-c} \leq_{\text{lex}} x_{k-c+1} \dots x_{2k-2c},$$

and recall that by assumption $x_1 \dots x_c = x_{bk+1} \dots x_{bk+c}$, with $bk + c = n$. If $k - c \geq c$ then the first c pairs of variables in a_{k-c} are precisely what we need to prove.

Therefore, we assume that $k - c < c$, and show that under the additional assumption $x_{bk+1} \dots x_{(b+1)k-c} = x_{k-c+1} \dots x_{2k-2c}$ that

$$x_{(b+1)k-c+1} \dots x_c \leq_{\text{lex}} x_{2k-2c+1} \dots x_k.$$

However our initial assumption that $x_i = x_{k+i}$ for $1 \leq i \leq k$ implies that $x_{(b+1)k-n+i} = x_i$ for $2k - 2c + 1 \leq i \leq k$, hence constraint (1) is implied for $1 \leq k < n$.

Minimality: Omitted. \square

The general personnel scheduling problem is to assign tasks to staff members while satisfying a variety of different constraints. The following simple example exhibits cyclic symmetry.

Given n personnel and δ days, for each day, assign each person to one of the shifts {AM, PM, Night, Holiday}.

Such that:

- **Coverage:** All non-holiday shifts are covered by at least one person.
- **Shift Pattern:** A night shift for a particular person is preceded by either an AM shift or a Holiday.
- **Shift Pattern:** A night shift for a particular person is followed by a Holiday or a PM shift.

The schedule is to be repeated, so day δ can be viewed as preceding day 1.

Given 4 people and three days, a set of cyclic symmetric solutions shown in Figure 2.

Clearly, we can cycle the days but not freely interchange them nor flip the schedule.

We can constrain the per 1 row using lex constraints generated by the general formula for cyclic group symmetry breaking constraints, namely: $d_1 \leq_{\text{lex}} d_2$ and $d_1 d_2 \leq_{\text{lex}} d_3 d_1$.¹

5.2 Dihedral Groups

The *dihedral group* D_n is the symmetries of a regular n -sided shape, for example D_4 is the symmetry group of the square. The reader may note some similarities between the dihedral minimal ordering constraints and the cyclic minimal ordering constraints. This relationship is explained later in the proof of Theorem 3. Let

$$a = (1, 2, \dots, n)$$

and

$$b = (1, n)(2, n-1) \dots (\lfloor n/2 \rfloor, \lceil (n+3)/2 \rceil).$$

All elements of D_n have a unique decomposition as a product of the form $a^i b^j$ for $0 \leq i \leq n-1$ and $0 \leq j \leq 1$. The set of elements for which $j = 0$ is the cyclic group of order n , whereas the elements $a^i b$ all satisfy $(a^i b)^2 = ()$.

We define a set of $2n - 5$ symmetry breaking constraints as follows. First, the constraints a_i from the cyclic group for $2 \leq i \leq n - 3$:

$$\begin{array}{rcl} x_1 x_2 & \leq_{\text{lex}} & x_3 x_4 \\ x_1 x_2 x_3 & \leq_{\text{lex}} & x_4 x_5 x_6 \\ & \vdots & \\ x_1 x_2 \dots x_{n/2+1} & \leq_{\text{lex}} & x_{n/2+2} \dots x_n x_1 x_2 \quad n \text{ even} \\ x_1 x_2 \dots x_{(n+1)/2} & \leq_{\text{lex}} & x_{(n+3)/2} \dots x_n x_1 \quad n \text{ odd} \\ & \vdots & \\ x_1 x_2 \dots x_{n-3} & \leq_{\text{lex}} & x_{n-2} x_1 \dots x_{n-6} \end{array}$$

Then a further $n-1$ constraints γ_i and δ_j . If $n = 2k+1$ is odd we define $s = 2, t = 1$ and $u = 0$, otherwise $n = 2k$ is even, and we let $s = 1, t = 0$ and $u = 1$. Then define γ_i for $1 \leq i \leq k - u$ to be

$$x_1 \dots x_i x_{2i+1} \dots x_{k+i} \leq_{\text{lex}} x_{2i} \dots x_{i+1} x_n \dots x_{k+i+s},$$

and δ_j for $0 \leq j \leq k - 1$ to be

$$x_1 \dots x_j x_{2j+2} \dots x_{k+j+t} \leq_{\text{lex}} x_{2j+1} \dots x_{j+2} x_n \dots x_{k+j+2}.$$

Note that the substrings of variables on the left hand side of γ_i and δ_j occur in increasing order, whilst those on the right hand side occur in decreasing order: if the subscripts at each end of the substring are decreasing on the left hand side, or increasing on the right hand side, then the substring is empty.

¹Although in this example the people in AM are all different and we could use Puget's all-dif constraints, we can imagine a much larger schedule where a person would be required to work more than one AM shift

	Day 1	Day 2	Day 3		Day 2	Day 3	Day 1		Day 3	Day 1	Day 2
per1	AM	N	H		AM	N	H		AM	N	H
per2	PM	H	N	OR	PM	H	N	OR	PM	H	N
per3	N	PM	AM		N	PM	AM		N	PM	AM
per4	H	AM	PM		H	AM	PM		H	AM	PM

Figure 2: Three symmetric schedules for the personnel scheduling problem instance days = 3, people = 4, shifts = {AM, PM, H, N}.

Theorem 3 Let P be a CSP with n decision variables, $\{x_1, x_2, \dots, x_n\}$. If the symmetry group of P is the dihedral group D_n on variables then the set $\{a_i, \gamma_j, \delta_l : 2 \leq i \leq n-3, 1 \leq j \leq k-u, 0 \leq l \leq k-1\}$ of symmetry-breaking constraints is complete and minimal.

Proof 3 The constraints can be divided into two sets. Those derived from the cyclic group symmetry over the same decision variables, namely the a_i s, and those derived from the remainder of the dihedral group permutations, namely the γ_i s and δ_j s.

THE CONSTRAINTS a_i FOR $2 \leq i \leq n-3$.

We have three fewer constraints than for C_n . These three constraints break the symmetry represented by the permutations a , a^{n-2} and a^{n-1} , and would correspond to constraints a_1, a_{n-2} and a_{n-1} . We show that these missing constraints are implied by the remaining constraints.

The constraint corresponding to the permutation a is $x_1 \leq x_2$, which is implied by γ_1 .

The constraint corresponding to a^{n-1} is

$$x_1 \dots x_{n-1} \leq_{\text{lex}} x_n x_1 \dots x_{n-2}.$$

We show that a_{n-1} is implied by the other constraints. Consider the first pair of variables, $x_1 \leq x_n$. Constraint δ_0 has most significant pair $x_2 \leq x_n$, and the implied constraint a_1 states that $x_1 \leq x_2$, which together imply that $x_1 \leq x_n$.

Assuming equality in the first z pairs of variables of a_{n-1} forces

$$x_1 = x_n = x_2 = \dots = x_z \quad \text{where } 2 \leq z \leq n-2.$$

The next pair under consideration is $x_{z+1} \leq x_z$.

First assume that $z = 2i \leq n-2$ is even, then constraint γ_i is

$$x_1 \dots x_i x_{2i+1} \dots x_{k+i} \leq_{\text{lex}} x_{2i} x_{2i-1} \dots x_{i+1} x_n \dots x_{k+i+s},$$

where $s = 2$ or 1 according as n is odd or even. The first i pairs of variables in this constraint have been assumed to be equal, so we deduce that $x_{2i+1} = x_{z+1} \leq x_n = x_z$, as required.

Next assume that $z = 2i+1 \leq n-2$ is odd, then constraint δ_i is

$$x_1 \dots x_i x_{2i+2} \dots x_{k+i+t} \leq_{\text{lex}} x_{2i+1} \dots x_{i+2} x_n \dots x_{k+i+2},$$

where t is 0 or 1 according as n is even or odd. Since the first z pairs of variables in this constraint have been assumed to be equal, we deduce that $x_{z+1} = x_{2i+2} \leq x_n = x_z$, as required. Hence a_{n-1} is implied.

Finally we consider the constraint corresponding to a^{n-2} , namely a_{n-2} which is

$$x_1 x_2 \dots x_{n-2} \leq_{\text{lex}} x_{n-1} x_n x_1 \dots x_{n-4}.$$

The inequality $x_1 \leq x_{n-1}$ is the most significant pair in δ_{k-1} (even values of n) or γ_k (odd values of n). Also, $x_2 \leq x_n$ is the most significant pair in δ_0 . If we assume that $x_1 = x_{n-1}$ and $x_2 = x_n$, then δ_0 gives $x_3 \leq x_{n-1} = x_1$, as required.

Let us now consider the general case $x_{z+1} \leq x_{z-1}$ for $3 \leq z \leq n-2$, so that we assume that $x_1 = x_n = x_3 = x_5 = \dots$ and $x_2 = x_{n-1} = x_4 = \dots$, with the highest subscript (other than n or $n-1$) in an equality class of size greater than 1 being x_z . If $z = 2i+1$ is odd then the constraint δ_i has the first n pairs of variables equal, so we deduce that $x_{2i+1} = x_{z+1} \leq x_n = x_{z-1}$. If $z = 2i$ is even then the constraint δ_{i-1} has first $i-1$ pairs of variables equal. We then find $x_{2i} = x_z = x_n$, so continuing to the next variable we deduce $x_{2i+1} = x_{z+1} \leq x_{n-1} = x_{z-1}$, as required.

Since the missing constraints from the cyclic set, which themselves are known to be complete from Theorem 2, are implied by the additional constraints to break the dihedral symmetry, the reduced set of cyclic constraints are complete with respect to the cyclic symmetry in the dihedral group.

THE REMAINING CONSTRAINTS

Next we show that the reduced set of lex-constraints imply all constraints corresponding to the remaining elements of the dihedral group, namely those of the form $a^i b$. Recall that the elements of the form $a^i b$ have order 2, and so are a product of disjoint 2-cycles. The second variable of each 2-cycle will be deleted from each constraint by Rule 1. Also, when n is even, half of the permutations $a^i b$ fix two variables, which will also be deleted by Rule 1: these correspond to δ_i for $0 \leq i \leq k-1$.

The full set of lex constraints has n symmetry breaking constraints for the remaining dihedral permutations, but that the reduced set only has $n-1$. The constraint which breaks the symmetry

$$b = (1, n)(2, n-1) \dots ([n/2], \lceil (n+2)/2 \rceil)$$

has been completely removed. We now show that this constraint is implied by the reduced set of constraints. The unreduced form of this missing constraint c is

$$x_1 \dots x_n \leq_{\text{lex}} x_n \dots x_1.$$

We consider the implied constraint $a_n : x_1 \dots x_n \leq_{\text{lex}} x_n x_1 \dots x_{n-2}$, which has most significant pair $x_1 \leq x_n$. These are the first pair of variables in c .

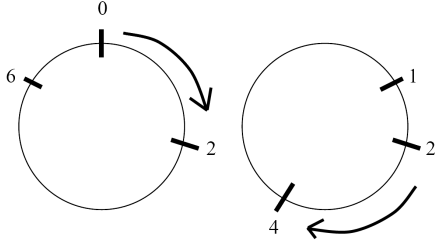


Figure 3: Symmetric solutions to the length 7, 3-tick Circular Golomb Ruler problem.

Consider now an arbitrary pair $x_z \leq x_{n-z+1}$ for $2 \leq z \leq n/2$. When considering this pair for removal we assume that $x_1 = x_n$, $x_2 = x_{n-2}$, \dots , $x_{z-1} = x_{n-z}$. Therefore from a_n and a_1 we deduce $x_2 = x_1 = x_n = x_{n-2}$. We then deduce from a_n and a_i that $x_i = x_1$ for $i \in \{1, \dots, n\}$. Hence the rest of c is implied and can be removed.

We have shown that the reduced set of lex ordering constraints for the dihedral group implies the complete set. Since the converse is clear we conclude that the reduced set of dihedral lex constraints is sound and complete.

Minimality: Omitted. \square

We define the Circular Golomb Ruler Problem as: given a circle with circumference n , place m ticks at integer points around the circle such that all inter-tick distances along the circumference are distinct, $n > 0$ and $m > 0$. Two solutions to the instance of this problem where n is 7 and m is 3 are shown in Figure 3. Clearly, these solutions are symmetric: one can be obtained from the other via rotation. We may also flip the ruler to achieve twice as many symmetries. The symmetry group in the above problem, $n = 7$ $m = 3$, is D_3 . The symmetry breaking constraints required to order the set of ticks, $T = \{t_1, t_2, t_3\}$ are $t_1 \leq t_2$, and $t_1 t_2 \leq_{\text{lex}} t_3 t_1$.

5.3 Symmetric Groups

The symmetric group, S_n , is the group whose elements are the set of bijections from $\{1, \dots, n\}$ into itself; we can freely interchange all variables. Symmetric groups arise frequently as symmetries of CSPs, in particular whenever a set is modelled as a list we introduce the symmetric group on variables. Since the set is unordered but a list is ordered we can freely interchange any variables of the list and get a new list representing the same set.

Theorem 4 Given a CSP with n decision variables $\{x_1, x_2, \dots, x_n\}$ whose symmetry group is S_n on variables, a complete minimal set of symmetry breaking constraints is:

$$x_i \leq x_{i+1} \text{ for } 1 \leq i \leq n-1$$

Proof 4 We first show that the complete set of lex-leader constraints imply our constraints, then show that

the reduced set of constraints implies the lex-leader constraints. Since the lex-leader constraints are complete, the reduced set of constraints are complete.

Lex-leader constraints imply the reduced set of constraints as the reduced set is a subset of the lex-leader constraints.

The reduced set of constraints implies the complete set of lex-leader constraints since the lex-leader constraints break every permutation of S_n , which is every possible permutation of n variables. The reduced set implies that $x_1 \dots x_n$ is sorted, which breaks every possible permutation of n variables.

Minimality: Omitted. \square

5.4 Alternating Groups

Any permutation can be written as a product of cycles of length 2, called *transpositions*. There is usually more than one way of writing any given permutation as a product of transpositions, but the parity of the number of transpositions occurring in all such products is fixed. The *alternating group* A_n on n points is the subgroup of the symmetric group S_n that contains all of the permutations that can be written as a product of an even number of transpositions. It contains exactly half of the permutations of the symmetric group, and hence could be expected to have a similarly small set of minimal lex constraints.

Theorem 5 Let P be a CSP with n decision variables, $\{x_1, x_2, \dots, x_n\}$. If the symmetry group of P is A_n on variables then a complete minimal set C of symmetry breaking constraints is:

$$\begin{aligned} x_{n-2} &\leq x_{n-1} && (c_1) \\ x_{n-2}x_{n-1} &\leq_{\text{lex}} x_nx_{n-2} && (c_2) \\ x_ix_{n-1} &\leq_{\text{lex}} x_{i+1}x_n && (c_{3,i}), 1 \leq i \leq n-3 \end{aligned}$$

Proof 5 We first show that C is implied by lex-leader constraints. The permutation $(n-2, n-1, n) \in A_n$ implies c_1 and the permutation $(n-2, n, n-1)$ implies c_2 . Finally, we have $(x_i, x_{i+1})(x_{n-1}, x_n) \in A_n$ for $1 \leq i \leq n-3$, yielding $c_{3,i}$.

We now consider the converse and show that C implies the full set of lex-leader constraints, of size $(n!/2) - 1$. Consider c_1 and the first pair of variables in $c_{3,i}$, we see that the first $n-1$ variables are sorted. The first pair of variables in c_2 , namely $x_{n-2} \leq x_n$ implies that the variables $x_1, x_2, \dots, x_{n-2}, x_n$ are also sorted. So the only possibly unbroken symmetries move x_{n-1} and x_n . Since the permutation swapping (x_{n-1}, x_n) is odd, for there to be any remaining symmetries either $x_{n-1} = x_{n-2}$, or $x_n = x_{n-2}$, or some of the other variables have equal values.

If $x_{n-1} = x_n = x_{n-2}$ then there are no further symmetries to break, since $x_1 \dots x_n$ is now sorted.

If exactly one of x_{n-1} and x_n is equal to x_{n-2} , then the other one is greater. If $x_{n-2} = x_n < x_{n-1}$ then we are violating c_2 . Therefore $x_{n-2} = x_{n-1}$ so the solution is already minimal.

If $x_{n-1}, x_n \neq x_{n-2}$ then both x_{n-1} and x_n are strictly greater than x_{n-2} so the biggest two values occur at the end of the list of variables. This will be lex minimal under the alternating group unless $x_{n-1} > x_n$ and there are two equal values (say x_i, x_{i+1}) somewhere else in the full assignment. However in this instance we would violate the constraint $c_{3,i}$.

Minimality: Omitted. \square

We are currently aware of no situations where the alternating group arises as symmetry of a CSP model but its similarity to the symmetric group made it interesting from a mathematical point of view.

6 Combining Groups

Often, symmetry groups can be built up out of smaller, easier-to-describe groups. When this occurs we take the product of several groups.

Definition 4 Let $G \leq \text{Sym}(\Omega)$ and $H \leq \text{Sym}(\Delta)$ be groups, with Ω and Δ disjoint sets. The direct product of G and H , written $G \times H$, is the set $\{(g, h) : \forall g \in G, h \in H\}$, with coordinatewise multiplication. Elements of $G \times H$ permute the set $\Omega \cup \Delta$ as follows: $(g, h)(x) = g(x)$ if $x \in \Omega$, and $(g, h)(x) = h(x)$ if $x \in \Delta$.

Consider now the problem of finding two distinct rosters. The symmetry in this problem is the direct product of the two cyclic groups.

Theorem 6 Let P be a CSP whose decision variables are partitioned into two disjoint sets $\chi_1 = \{x_1, \dots, x_n\}$ and $\chi_2 = \{y_1, \dots, y_m\}$. Assume that all symmetries of P are variable symmetries, and that the symmetries of P act independently on χ_1 and χ_2 , with groups G and H of variable symmetries respectively. Let L_G be a minimal set of complete symmetry breaking constraints for G , and let L_H be a minimal set of complete symmetry breaking constraints for H . Then the symmetry group of P is $G \times H$, and a minimal complete set of symmetry breaking constraints for P is $L_G \cup L_H$.

Proof 6 The claim that the symmetry group of P is $G \times H$ is immediate.

We must show three things. Firstly, we show that the lex-leader constraints for $G \times H$ imply $L_G \cup L_H$, secondly that $L_G \cup L_H$ implies all of the lex-leader constraints, and finally that $L_G \cup L_H$ is minimal.

The lex-leader constraints for $G \times H$ will include a constraint c_g for each element of $G \times H$ of the form $(g, 1_H)$ where $g \in G$. Since c_g has all variables y_i in the same positions on each side, by Rule 1 c_g can be reduced to a constraint involving only the x_i s, and hence the set of all such reduced c_g s implies all constraints in L_G . Similarly, the constraints for $G \times H$ will include a constraint c_h for each element $(1_G, h) \in G \times H$, and the constraint c_h can be reduced by Rule 1 to a constraint involving only the y_i s. Hence all constraints in L_H are implied by the lex-leader constraints.

Now we must show the converse. Let $a := (g, h)$ in $G \times H$, let $k = m + n$ and let $z_i \in \chi_1 \cup \chi_2$ for $1 \leq i \leq k$.

We define an action of a on $\{z_i : 1 \leq i \leq k\}$ by $z_{a(i)} = x_{g(j)}$ if $z_i = x_j$ and $z_{a(i)} = y_{h(j)}$ if $z_i = y_j$. Then any lex-leader constraint is of the form:

$$z_1 z_2 \dots z_k \leq_{\text{lex}} z_{a(1)} z_{a(2)} \dots z_{a(k)}.$$

Since by assumption L_G and L_H are complete sets of constraints for G and H , they imply the constraints

$$\begin{aligned} x_1 \dots x_n &\leq_{\text{lex}} x_{g(1)} \dots x_{g(n)} & (A) \\ y_1 \dots y_m &\leq_{\text{lex}} y_{h(1)} \dots y_{h(m)} & (B). \end{aligned}$$

Suppose without loss of generality that $z_1 = x_1$. Then constraint (A) implies that $z_1 \leq_{\text{lex}} z_{a(1)}$, so the first pair of the lex-leader constraint is implied by $L_G \cup L_H$.

Suppose now that the first i pairs of the lex-leader constraint have been assumed to be equal, that is $z_1 = z_{a(1)}$, $z_2 = z_{a(2)}$, \dots , $z_i = z_{a(i)}$. We show that together with $L_G \cup L_H$ this implies that $z_{i+1} \leq z_{a(i+1)}$ and hence by induction that the full lex-leader constraint is implied. We have $z_{i+1} = x_j$ or $z_{i+1} = y_j$, let us assume without loss of generality that $z_{i+1} = x_j$ for some j . Then we must already have assumed that $x_1 = x_{g(1)}$, $x_2 = x_{g(2)}$, \dots , $x_{j-1} = x_{g(j-1)}$, so (A) implies that $z_{i+1} = x_j \leq x_{g(j)} = z_{a(i+1)}$, as required.

We finish the proof by showing that $L_G \cup L_H$ is minimal. This follows from the fact that each constraint in $L_G \cup L_H$ involves only variables from χ_1 or only variables from χ_2 . Hence constraints from L_G do not imply any additional equalities in constraints from L_H , and vice versa. Thus, since L_G and L_H were assumed to be minimal, the same holds for $L_G \cup L_H$. \square

For example take two sets of shifts, $S = \{s_1, s_2, s_3\}$ and $S' = \{s'_1, s'_2, s'_3\}$, defining two distinct shift patterns for per1. The lex constraints required to break the symmetry are $s_1 \leq_{\text{lex}} s_2$, $s'_1 \leq_{\text{lex}} s'_2$, $s_1 s_2 \leq_{\text{lex}} s_3 s_1$ and $s'_1 s'_2 \leq_{\text{lex}} s'_3 s'_1$.

Another common way of combining two groups is the imprimitive wreath product.

Definition 5 Let $G \leq S_n$ and $H \leq S_k$. The imprimitive wreath product of G and H , denoted $G\text{Wr}H$, is a subgroup of S_{nk} . It acts on k copies of the set of size n on which G acts. We have $G\text{Wr}H = \{h(g_1, \dots, g_k) : h \in H, g_i \in G\}$, and these elements permute the set $\{(i, j) : 1 \leq i \leq n, 1 \leq j \leq k\}$ by $h(g_1, \dots, g_k)(i, j) = (g_j(i), h(j))$.

Suppose we have a set A of symmetry breaking constraints for a group G acting on variables x_1, \dots, x_n , and a set B of symmetry breaking constraints for a group H acting on variables Y_1, \dots, Y_k . Then $G\text{Wr}H$ acts on a set of nk variables, x_{ij} , with $1 \leq i \leq n$ and $1 \leq j \leq k$. The group $G\text{Wr}H$ has size $|G|^k \times |H|$, so writing down one constraint for each nontrivial group element is impractical. We now show how to reduce this to $k|A| + |B|$ constraints.

We first post $k|A|$ constraints, namely a copy of A on x_{ij} for each value of j . That is, we lex order each block of variables with respect to G . The arity of these constraints is unchanged.

We then restate the constraints from B so that instead of being statements about the values of sequences of Y s, they are statements about the values of sequences of $x_{1j}x_{2j}\dots x_{nj}$ s. For example, if we previously had a constraint $Y_1 \leq Y_2$ we would replace that with the constraint $x_{11}x_{21}\dots x_{n1} \leq x_{21}x_{22}\dots x_{n2}$. This results in $|B|$ constraints, each of arity n times their original arity.

Theorem 7 *If L_G and L_H are complete sets of symmetry breaking constraints for groups G and H then $L_G \text{Wr} L_H$ is a complete set of symmetry breaking constraints for $G \text{Wr} H$ in the imprimitive action.*

Proof 7 *First we show that the constraints in $L_G \text{Wr} L_H$ are implied by the lex-leader constraints. The group $G \text{Wr} H$ contains elements of the form*

$$1_H(1_G, 1_G, \dots, 1_G, g, 1_G, \dots, 1_G)$$

for each $g \in G$, where g can occur in each coordinate. Using these group elements, applying Rule 1 to the resulting constraints, and then reasoning as in G , we produce each constraint $c_{g,i}$.

The group $G \text{Wr} H$ also contains elements of the form $h(1_G, \dots, 1_G)$ for each $h \in H$. These elements produce all constraints of type c_h .

Next we must show that $L_G \text{Wr} L_H$ implies all of the lex-leader constraints. An arbitrary element of $G \text{Wr} H$ is of the form $a := h(g_1, \dots, g_k)$, and produces the constraint:

$$x_{11} \dots x_{n1} x_{12} \dots x_{nk} \leq_{\text{lex}} x_{g_1(1)h(1)} \dots x_{g_1(n)h(1)} x_{g_2(1)h(2)} \dots x_{g_k(n)h(k)},$$

which we will denote by c_a . We must show that c_a is implied by $L_G \text{Wr} L_H$.

The constraint $y_1 \dots y_k \leq_{\text{lex}} y_{h(1)} \dots y_{h(k)}$ is implied by L_H , since L_H is assumed to be complete. Hence the constraints $L_G \text{Wr} L_H$ imply the constraint

$$x_{11} \dots x_{n1} x_{12} \dots x_{nk} \leq_{\text{lex}} x_{1h(1)} \dots x_{nh(1)} x_{1h(2)} \dots x_{nh(k)},$$

denoted α_h .

For $1 \leq i \leq k$ the constraints L_G imply the constraint

$$x_1 x_2 \dots x_n \leq_{\text{lex}} x_{g_i(1)} x_{g_i(2)} \dots x_{g_i(n)},$$

as they are a complete set of symmetry breaking constraints for G . Hence for $1 \leq i \leq k$ and $1 \leq j \leq k$, the set $L_G \text{Wr} L_h$ implies the constraint

$$x_{1j} x_{2j} \dots x_{nj} \leq_{\text{lex}} x_{g_i(1)j} x_{g_i(2)j} \dots x_{g_i(n)j},$$

denoted $\beta_{g_i,j}$.

We will use these constraints to show that c_a is implied by $L_G \text{Wr} L_H$, considering the variables in blocks of n . Firstly, we have

$$x_{11} x_{21} \dots x_{n1} \leq_{\text{lex}} x_{1h(1)} \dots x_{nh(1)}$$

as the first n variable pairs from α_h . Considering $\beta_{g_1,h(1)}$ we also have

$$x_{1h(1)} \dots x_{nh(1)} \leq_{\text{lex}} x_{g_1(1)h(1)} \dots x_{g_1(n)h(1)}.$$

Combining these two inequalities we deduce that the first n pairs of variables of c_a are implied by $L_G \text{Wr} L_H$.

Suppose that we have shown that the first $n(i-1)$ variable pairs of c_a are implied by $L_G \text{Wr} L_H$, and that we are now considering pairs $n(i-1)+1, \dots, ni$, namely

$$x_{1i} x_{2i} \dots x_{ni} \leq_{\text{lex}} x_{g_i(1)h(i)} x_{g_i(2)h(i)} \dots x_{g_i(n)h(i)}.$$

To consider these variables we assume equality in the preceding $n(i-1)$ variable pairs, so $x_{11} = x_{1h(1)} = x_{g_1(1)h(1)}, x_{21} = x_{2h(1)} = x_{g_1(2)h(1)}, \dots, x_{n(i-1)} = x_{nh(i-1)} = x_{g_{i-1}(n)h(i-1)}$. Considering constraint α_h we now deduce that

$$x_{1i} x_{2i} \dots x_{ni} \leq_{\text{lex}} x_{1h(i)} x_{2h(i)} \dots x_{nh(i)},$$

whereas from constraint $\beta_{g_i,h(i)}$ we deduce that

$$x_{1h(i)} x_{2h(i)} \dots x_{nh(i)} \leq_{\text{lex}} x_{g_i(1)h(i)} x_{g_i(2)h(i)} \dots x_{g_i(n)h(i)}$$

and so the result follows by induction. \square

Examining the model for the rostering problem we see that there is symmetric group symmetry between each person on the roster, i.e. every person can be freely interchanged with any other person to maintain a complete (non)solution. Here the complete symmetry group is $S_n \text{Wr} C_\delta$.

7 Experimental Evaluation

We consider the class scheduling problem: given d days, h hours per day and c classes, where $d \times h$ is divisible by c , find a timetable such that:

1. Each class has $(d \times h)/c$ assigned hours in the schedule.
2. No class has more than 2 hours in any given day.

Here the full symmetry group is the wreath product of two symmetric groups, $S_h \text{Wr} S_d$. The first column shows the progress of a model with no symmetry breaking constraints. The next column shows the same model with the symmetry group broken by lex leader constraints on every symmetric permutation; there are $|S_h|^d \times |S_d|$ such permutations. Finally, the last column shows the same model using the reduced wreath product symmetry breaking constraints; there are $(d \times (h-1)) + (d-1)$ such constraints. Fig 7 shows the results of two test instances of the class scheduling problem. Here the reduced lex constraints are more efficient in terms of memory used, time taken and nodes searched over. We attribute the additional nodes in this case to a feature of the solver used for testing². It is worth noting that larger instances are not listed because the specification for the full lex in those cases was too large to compute. The specifications for both the reduced lex and the no lex versions took a negligible amount of time to create.

²The GACLex algorithm assumes that every variable in any one constraint is unique. This results in simple implications being missed, for example $(A \leq A) = \text{true}$.

$d = 3 \ h = 3 \ c = 3$	No Lex	Full Lex	Reduced Lex
Setup Time (s)	0	0.02	0
Memory (bytes)	232	15784	328
Total Time (s)	0.046	0.2	0.031
Nodes	3304	119	45
Solutions	1512	6	6
$d = 4 \ h = 3 \ c = 6$	No Lex	Full Lex	Reduced Lex
Setup Time (s)	0	0.235	0
Memory (bytes)	376	373624	508
Total Time (s)	74.78	377.25	0.062
Nodes	22462011	57845	6318
Solutions	7484400	715	715

Figure 4: Results from the testing of two class scheduling instances

8 Conclusion

This paper has discussed symmetry breaking in CSPs by adding lexicographic ordering constraints. Given the huge number of such constraints needed in general to break all symmetry, and the intractability of the general methods of reducing this number, we focussed on a number of special cases and showed how the number of ordering constraints necessary in each case can be reduced. To extend this work we intend to evaluate the symmetry breaking constraints proposed in this paper against other effective symmetry breaking techniques. We will integrate this work into the automated modelling system CONJURE [Frisch *et al.*, 2005] so that it is able to break symmetry efficiently as it is introduced.

Acknowledgements Andrew Grayland is supported by an EPSRC CASE studentship, sponsored by Microsoft Research. Ian Miguel is supported by a Royal Academy of Engineering/EPSRC Research Fellowship. Colva Roney-Dougal is partly funded by the Nuffield Foundation.

References

- [Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks and A. Roy. Symmetry-Breaking Predicates for Search Problems. *Proc. KR*, 148–159, 1996.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry Breaking. *Proc. CP*, 225–239, 2001.
- [Frisch *et al.*, 2003] A. M. Frisch and W. Harvey. Constraints for Breaking All Row and Column Symmetries in a Three-by-Two-Matrix. *Proc. Symcon*, 2003.
- [Frisch *et al.*, 2005] A. M. Frisch, C. Jefferson, B. Martinez Hernandez and I. Miguel. The Rules of Constraint Modelling. *Proc. IJCAI*, 109–116, 2005.
- [Gent *et al.*, 2000] I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. ECAI*, 599–603, 2000.

- [Öhrman, 2005] H. Öhrman. *Breaking Symmetries In Matrix Models*. Masters Thesis, Dept. Information Technology, Uppsala University, 2005.
- [Nuutila, 1995] E. Nuutila. *Efficient Transitive Closure Computation in Large Digraphs*. PhD Thesis, Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering Series No. 74, Helsinki 1995.
- [Puget, 2005] J-F. Puget. Breaking Symmetries In All Different Problems. *Proc. IJCAI*, 272–277, 2005.
- [Smith, 2001] B. M. Smith. Reducing Symmetry in a Combinatorial Design Problem. *Technical Research Report, University of Leeds*, 2001.01.

A CP Approach to the Balanced Academic Curriculum Problem

Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville and Pierre Dupont

Department of Computing Sciences and Engineering

Université catholique de Louvain, Belgium

{jmonette,pschaus,sz,yde,pdupont}@info.ucl.ac.be

Abstract

The Balanced Academic Curriculum Problem (BACP) has received little attention in Constraint Programming. Only a few articles deal with this problem with experimental results on the three small instances publicly available in CSPLIB. The present article describes an approach to efficiently solve this challenging problem. Optimal solutions are produced on a variety of randomly generated instances which generalize the CSPLIB test cases. This work describes four contributions to the resolution of this problem: a new branching heuristic, the use of dominance relations, experiments on several balance criteria and several search strategies among which an hybridization of Constraint Programming and Local Search.

1 Introduction

The Balanced Academic Curriculum Problem (BACP) is recurrent in Universities. The goal is to schedule the courses that a student must follow in order to respect the prerequisite constraints between courses and to balance as much as possible the workload of each period. This problem has been introduced first in [Castro and Manzano, 2001] and tackled also in [Hnich *et al.*, 2002]. It has also been studied in [Hnich *et al.*, 2004] with an hybrid CP/ILP approach. More recently, [Lambert *et al.*, 2006] proposed another hybrid approach of Constraint Programming and Genetic Algorithms. Those works deal with the three small instances available on CSPLIB (<http://www.csplib.org>).

This work presents four contributions to address the BACP with Constraint Programming. First, we present a new value ordering heuristic guiding the search toward a balanced solution. Then, we propose to use dominance relations in order to reduce the search tree. Next, other balance criteria such as the minimization of the sum of deviations and square deviations are applied. Fourthly, different search schemes are compared, including an iterative increase of the objective value and the use of Local Search to find quickly a tight upper bound for Branch-and-Bound. Moreover, we present an instance generator

that allows one to run experiments on larger instances with a structure similar to some original ones.

This paper is structured as follows. The next section presents the problem formally and describes the CP model. It describes also a new branching heuristic and the balance criteria that will be compared. Section 3 presents the dominance rules and their application. Section 4 presents the different search strategies used and Section 5 explains the instances generator. Finally, Section 6 presents the experiments and their results before concluding.

2 The BAC Problem

The goal of the BACP is to schedule courses in different periods. Each course has a workload which is expressed in number of credits and a (possibly empty) set of prerequisite courses. A solution is an assignment of courses to periods that satisfies the prerequisite constraints while balancing the workload of periods. The workload of a period is the sum of the credits of the courses taught during this period. Additional constraints limiting the minimum and maximum number of courses and credits for each period were originally introduced in [Castro and Manzano, 2001]. These additional constraints are however not considered here since balanced solutions of the instances in CSPLIB always respect them. Indeed, the limits on the number of credits by period are naturally respected by balanced solutions, unless the problem is unfeasible. The limits on the number of courses by period are respected because the number of credits for each course does not vary much and these limits are quite large in the instances of CSPLIB. Precisely, a BACP instance is characterized by:

- n the number of courses;
- m the number of periods;
- w_i the load of course i for $1 \leq i \leq n$;
- $prerequisites = \{(i, j) \mid i \neq j, 1 \leq i, j \leq n\}$ a set of couples of courses stating that course i is a prerequisite of course j .

Three instances were originally proposed for this problem in CSPLIB with followings characteristics:

Instance	m	n	$prerequisites$	Values for w_i
1	8	46	38	[1, 5]
2	10	42	34	[1, 5]
3	12	66	65	[1, 5]

The BACP is interesting because it is at the boundary of several classes of problems: bin-packing, scheduling and balancing. The bin-packing is a class of problems where a set of objects of different sizes must hold in the smallest set of bags of finite capacity. BACP is a kind of bin-packing problem where the size of the bags is minimized rather than the number of bags. The prerequisite constraints make BACP look also like a scheduling problem. Each course is a unit-time activity whose credit is the consumption of resource and prerequisites are temporal constraints. The periods are time units and the goal is to balance the utilization of the unique resource (the student). Finally, BACP is also an excellent example of balancing problems. The need and the interest for balanced solutions increases in Constraint Programming. Maximization of the satisfaction of a set of customers, minimization of the violations in over-constrained problems, schedule of physicians, share of a scarce resource are some examples where balanced solutions are generally preferred.

BACP is also a hard problem. The satisfaction version of BACP (“Does it exist a solution under a given balance value?”) is a NP-complete problem. This can be shown by reduction from the satisfaction version of the bin-packing problem (“Does it exist a solution with at most a given number of bags?”) that is known to be NP-complete. Objects become courses, credits are sizes and bins are periods. There is no prerequisite. Looking for a solution for BACP solves the corresponding bin-packing problem. As shown in Section 5, the problem becomes easier with the addition of prerequisites. Indeed the number of available periods for each course is reduced.

2.1 CP Model

The CP model has initially been proposed in [Hnich *et al.*, 2002]. Each course is identified by an integer in the interval $[1, n]$. There are three sets of variables:

- $\forall i \in [1, n], P_i$ represents the period of course i .
- $\forall i \in [1, m], L_i$ is the load of the period.
- $\forall (i, j) \in [1, n] \times [1, m], B_{ij}$ makes the link between the two other sets of variables. In particular $B_{ij} = 1$ if and only if course i is given in period j .

The prerequisites constraints are easily stated with the first set of variables using “less than” constraints. The load variables are linked to the binary variables with weighted sums while channeling constraints link the period of the courses and the binary variables.

$$\forall (i, j) \in prerequisites : P_i < P_j.$$

$$\forall 1 \leq j \leq m : L_j = \sum_{i=1}^n B_{ij} \cdot w_i.$$

$$\forall 1 \leq i \leq n, 1 \leq j \leq m : (P_i = j) \Leftrightarrow (B_{ij} = 1).$$

2.2 The Balance Criteria

In previous works [Castro and Manzano, 2001; Hnich *et al.*, 2002], the only balance criterion used for this problem is the minimization of the maximum load that will be denoted by C^{\max} . Mathematically, it is defined as

$$C^{\max} = \max_{1 \leq i \leq m} L_i.$$

The objective of the BACP is to minimize C^{\max} .

As shown in [Schaus *et al.*, 2007], other criteria can be used. The balance can be defined in a generic way by the criterion

$$C(p) = \sum_{i=1}^m |m \cdot L_i - w|^p.$$

where $w = \sum_{i=1}^m L_i = \sum_{i=1}^n w_i$ is the total workload. The objective is now to minimize $C(p)$, i.e. to minimize the sum of the measures of the distance between the load of each period and the average load. In particular, instantiating the parameter p to 1, 2 and ∞ gives the following interpretations:

- $C(1) = \sum_{i=1}^m |m \cdot L_i - w|$ is the sum of deviations from the mean.
- $C(2) = \sum_{i=1}^m (m \cdot L_i - w)^2$ is the sum of square deviations from the mean.
- $C(\infty) = \max_{1 \leq i \leq m} |m \cdot L_i - w|$ is the maximum deviation from the mean.

It as been shown in [Schaus *et al.*, 2007] that neither criterion subsumes the others and there is no a priori reason to prefer one of them. Section 6.1 evaluates how well each criterion approximates the others.

2.3 Variable and Value Ordering Heuristics

The previous work [Hnich *et al.*, 2002] on BACP branches on the variables P_i and uses a classical *first-fail* heuristic that chooses the unassigned variable with the smallest domain. The value heuristic picks the smallest value of its domain.

As the goal is to obtain the most balanced solution, choosing as value the period which is the less heavily loaded ensures that the first solution found will be already quite balanced. Formally, to post the constraint $P_i = v$, v is chosen such that

$$v = \underset{v' \in dom(P_i)}{\operatorname{argmin}} \underline{L}_{v'}$$

where $dom(P_i)$ denotes the domain of the variable P_i and \underline{L}_v is the minimum value in $dom(L_v)$.

3 Dominance Rules

The BACP instances contain many symmetrical solutions. For instances, consider two courses that have the same workload and that are prerequisites of the same courses and have the same courses as prerequisites. In a solution, these two courses can be exchanged giving another solution with the same balance (independently of

the balance criterion used). Situations of this kind may be discovered by the use of dominance rules.

Dominance relations have been shown to be powerful for solving hard problems [Prestwich and Beck, 2004]. The concept of dominance can be seen as a symmetry that is unidirectional. A state s_i of the search tree is said to dominate another state s_j if the best solution that can be found from s_j is no better than the best solution that can be found from s_i . It is useless to explore the subtree rooted in s_j once s_i is explored. In other terms, suppose a property \mathcal{P} such that for every optimal solution with \mathcal{P} there exists an optimal solution with $\neg\mathcal{P}$. The solutions where $\neg\mathcal{P}$ holds dominate the solutions where \mathcal{P} holds. In this case, posting the constraint $\neg\mathcal{P}$ preserves optimality. We refer to [Prestwich and Beck, 2004] for a formal definition of dominance.

3.1 Detecting Dominance Relations

In the context of the BACP, the dominance rules take the form of “less or equal” constraints between the periods of two courses. The conditions to post such a constraint are presented in the following theorem.

Theorem 3.1 *Posting a constraint $P_i \leq P_j$ preserves at least one optimally balanced solution to BACP if the following conditions hold :*

- $w_i = w_j$
- $pred_i \subseteq pred_j$
- $succ_i \supseteq succ_j$

where $pred_i = \{k | (k, i) \in prerequisites\}$ and $succ_i = \{k | (i, k) \in prerequisites\}$ denote respectively the set of prerequisite courses (predecessors) of i and the set of courses for which i is a prerequisite (successors of i).

Proof The first condition ensures that if there exists an assignment for which $P_i = a$ and $P_j = b$, then swapping the periods of i and j gives an equally good assignment. Indeed, whatever the objective, it is based on the workload of the periods that is not changed by swapping two courses with the same credit.

Let us denote P_{p_i} the period of the latest predecessor of i and P_{s_i} the period of the earliest successor of i . If $pred_i \subseteq pred_j$, the latest predecessor of j (whose assigned period is denoted P_{p_j}) cannot be earlier than the latest predecessor of i (whose period is denoted P_{p_i}). Either it is the same course or it is another one that is thus necessarily later. Conversely, $succ_i \supseteq succ_j$ means that the earliest successor of i (P_{s_i}) cannot be later than the earliest successor of j (P_{s_j}). The following inequalities hold in any solution :

- $P_{p_i} < P_i < P_{s_i}$
- $P_{p_j} < P_j < P_{s_j}$
- $P_{p_i} \leq P_{p_j}$
- $P_{s_i} \leq P_{s_j}$

In an optimal solution, either $P_i \leq P_j$ or $P_i > P_j$. To preserve at least one optimal solution when adding the constraint $P_i \leq P_j$, for any solution where $P_i > P_j$

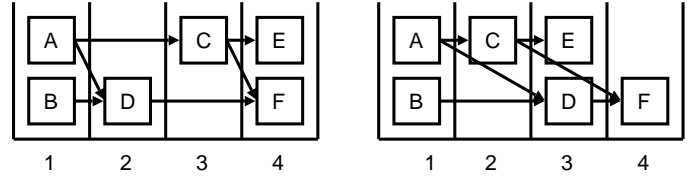


Figure 1: Example of dominance application.

holds, there must exist another optimal solution where $P_i \leq P_j$ holds.

Let us suppose an optimal solution with $P_i = a$, $P_j = b$ and $b < a$. Swapping the values of P_i and P_j (leading to $P_i = b$ and $P_j = a$) keeps the assignment optimal and enforces $P_i \leq P_j$. It remains to show that this assignment is still a solution and verifies the prerequisite constraints. Let us show that a is a valid value for P_j , that is that $P_{p_j} < a < P_{s_j}$. The relations $P_{p_i} < a < P_{s_i}$, $P_{p_j} < b < P_{s_j}$, $b < a$ and $P_{s_i} \leq P_{s_j}$ hold. Using the transitivity of $<$ and \leq , it results in $P_{p_j} < b < a < P_{s_i} \leq P_{s_j} \Rightarrow P_{p_j} < a < P_{s_j}$. The same reasoning can be done to show that b is a valid value for P_i . Thus enforcing $P_i \leq P_j$ preserves at least one optimally balanced solution. ■

Figure 1 presents two situations that induce dominance rules. Squares represent courses with unit credit. Arrows represent the prerequisites between courses. Vertical bins are the periods. The two situations are parts of some solutions to a BACP instance. In the first situation, course C is after course D but they can be swapped without changing the value of the solution. In the second situation, course C is before course D but they cannot be swapped because it would not be a solution anymore (C would not be taught before course E). So the constraint $P_C \leq P_D$ can be posted.

As noted in [Prestwich and Beck, 2004], one has to carefully check that the use of several interacting dominance rules does not suppress all of the optimal solutions. Problems may arise only when two or more courses of equal credit have exactly the same prerequisites and are prerequisite of exactly the same courses. In such a case, there are real symmetric states and the “less or equal” constraint can be posted in either way but only one can be posted. When more than two courses have the same characteristics, the constraints must be posted without creating cycles. For instance, a lexicographic ordering can be used and the constraint $P_i \leq P_j$ is posted only if the additional condition $i < j$ holds.

The detection of dominance relations is performed once per instance at the root of the search tree with a temporal complexity in $\mathcal{O}(n^2p)$, where p is the maximum number of predecessors or successors of a course ($p \ll n$). Indeed, every pair of courses is considered and for each, if they have the same workload, it is necessary to compare the sets of predecessors and successors of the courses, which is linear in the size of these sets.

The number of dominance relations found may be rather important. For the three original instances in CSPLIB, there are respectively 134, 65 and 183 relations for the instance with respectively 8, 10 and 12 periods.

3.2 Applying Dominance Rules

Dominance rules can be used to improve the search. They can simply be posted at the root node of the search tree. However, this method has the drawback (observed also in symmetry breaking) that the first solution found might become inconsistent with the additional constraints. This would possibly force the search process to explore a large part of the search tree not explored otherwise.

Several techniques have been developed to avoid this undesirable phenomenon when dealing with symmetry breaking (see [Gent *et al.*, 2006] for an overview). We choose to reuse the idea of SBDS (Symmetry Breaking During Search) [Gent and Smith, 2003], applying the technique to dominance rules. Basically, SBDS posts constraints after backtracking to avoid to explore search states symmetrical to ones already explored. SBDS has been chosen instead of other dynamic techniques because it seemed the most adapted to our purpose and was easy to implement.

The SBDS technique cannot be applied as such however for dominance rules that are unidirectional. In the context of the dominance rules considered here, a dominance constraint is posted in the right branch of the binary search tree if this constraint is verified in the left branch. Suppose there exists the possible dominance rule $P_i \leq P_j$ and the decision constraint $P_i = a$ is posted during the search. After backtracking, when the opposite constraint ($P_i \neq a$) is posted, the dominance rule $P_i \leq P_j$ is added to the current state if $a < \underline{P}_j$ holds. This dynamic technique removes states dominated by other states already explored but not by states that could be explored later during the search.

4 Search Strategies

4.1 Branch-and-Bound

This section presents the different search strategies that can be used to solve the BACP. The first and most known technique is Branch-and-Bound. It consists in solving the optimization problem to minimize C (where C is the variable representing the criterion to optimize) as a satisfaction problem where the constraint $C < v$ is added. The value v is initially set to some upper bound (*e.g.* $+\infty$) and is changed to the value of C whenever a new solution is found. In this way, successive solutions improve the value of the criterion and when no more solution can be found, the last solution is proved to be an optimal one.

4.2 Including Local Search in Branch-and-Bound

Local Search (LS) can find rapidly a tight upper bound on the value of the objective to start Branch-and-Bound.

We refer to [Hoos and Stützle, 2004] for extensive explanations about LS. The LS algorithm provides a feasible solution that is not necessarily optimal but that should be as close as possible to the optimum. A constraint-based LS approach [Hentenryck and Michel, 2005] is used to model the problem. The LS model is the same as the CP model.

A tabu search heuristic is used. The satisfaction of the prerequisites and the optimization of the balance are combined in an objective function. In order to drive the search toward feasible solutions, the constraint satisfaction term has a much larger weight than the optimization term. A local move consists in choosing the variable P_i and the value allowing the best improvement of the objective function and reassigning the variable to this value. The search is stopped after a fixed number of iterations or when a feasible solution that is known to be optimal is found (perfect balance).

4.3 Iterative Satisfaction Problem

An alternative to Branch-and-Bound is to solve successive satisfaction problems with the constraint $C < v$, incrementing the value of v until a solution is found. The first obtained solution is optimal as the CSPs with smaller values are inconsistency. This technique has been used successfully in Scheduling [Baptiste *et al.*, 2001].

5 Instances Generator

Only three instances of BACP are available in CSPLIB. Moreover they are easy to solve. Therefore a parametrized instances generator has been created. Parameters are fixed to obtain additional instances with a similar structure to the three existing ones as detailed below.

5.1 The Generator

The generator creates instances parametrized by the number of courses, the number of periods, the minimum and maximum possible credits for a course and a probability to have a prerequisite between two courses. Instances are generated according to the following scheme:

1. For each course, a random credit is chosen between the minimum and maximum values.
2. Each course is assigned to a random period.
3. For each pair of courses that are in two successive periods, a prerequisite relation is created according to the given probability.

The instance generation ensures that a solution always exists. The choice of prerequisites only between courses in successive periods excludes some configuration of prerequisite graphs but still offers a wide range of possibilities. The generated solution is usually not optimal and in an optimal solution, prerequisites will not necessary be between courses of successive periods.

5.2 Generated Problem Sets

To obtain instances similar to the original ones in CSPLIB, the ratio between the number of courses and the number of periods is fixed to 5. This is the mean ratio among the three original instances (see Section 2).

Preliminary experiments on instances with 20 periods and 100 courses whose credits range from 3 to 5 enabled us to study how the probability of prerequisites affects the difficulty of the problem. We generated 20 instances for each probability from 0% to 50% by step of 5%. The number of solved instances in less than 30 seconds with C^{\max} and Branch-and-Bound search was considered. These preliminary runs showed that instances are globally easier when they include more prerequisites. Especially, the problems stay hard from 0% to 25% of prerequisites (about 75% of unsolved cases). After this 25% limit, the difficulty of the problem drops rapidly. At 50% of prerequisites all instances are solved.

Two sets of test instances have been generated. The first set is composed of 100 small instances with 5 periods and 25 courses. The probability for prerequisites is set to 30% and the range of credits is [1, 5]. This simple set has been generated to compare the different balance criteria.

The second problem set includes instances with a number of courses varying from 30 to 200 with an increment of 10. The probability to have a prerequisite is fixed to 20%. Instances with different ranges of credit are generated to study the influence of this parameter. There are four classes:

- The range [1, 5] corresponds to the instances in CSPLIB.
- A unit credit for every courses induces easier instances.
- The range [1, 10] is an example of larger interval for which instances are harder.
- The range [3, 5] forbids courses with small credits that can be used to easily fill holes in period not loaded enough.

Generating 10 instances for each configuration (4 classes and 18 sizes), the second set includes a total of 720 instances.

6 Experiments

The aim of this section is to answer the following questions:

- How well does each balance criterion approximate the others?
- How much does the proposed branching heuristic reduce the search space as compared to the one proposed in [Hnich *et al.*, 2002]?
- Which search strategy is more time efficient to solve the BACP?
- How much dominance rules improve search efficiency?

	C^{\max}	$C(1)$	$C(2)$	$C(\infty)$	Average
C^{\max}	0.00	10.62	16.53	0.06	9.07
$C(1)$	2.63	0.00	6.27	0.12	3.00
$C(2)$	0.28	0.00	0.00	0.00	0.09
$C(\infty)$	10.37	18.07	23.66	0.00	17.36
Average	4.43	9.56	15.48	0.06	

Table 1: Comparison of four balance criteria. Each row corresponds to an optimized criterion. Each column corresponds to an evaluated criterion.

All experiments are performed on an Intel® Celeron® 2.8 GHz with 1GB of memory. Our implementation uses the Gecode (<http://www.gecode.org>) constraints library and the Local Search is written in the Comet (<http://www.comet-online.org>) programming language.

6.1 Balance Criteria

The goal of this experiment is to compare the four considered criteria (C^{\max} , $C(1)$, $C(2)$ and $C(\infty)$). The focus is on the quality of the solution in terms of balancing and thus time is not considered.

The 100 instances of the first problem set are successively solved with the four criteria and the best solutions found are evaluated with respect to the four criteria. The search is performed using Branch-and-Bound with the new branching heuristic.

There exists a global constraint for $C(1)$ running in $\mathcal{O}(m)$ introduced in [Schaus *et al.*, 2007] and a less efficient one ($\mathcal{O}(m^2)$) for $C(2)$ presented in [Pesant and Régim, 2005; Schaus *et al.*, 2006]. C^{\max} and $C(\infty)$ use classical constraints whose temporal complexity is $\mathcal{O}(m)$. The total runs took less than 2 seconds for each criterion except for $C(2)$ for which it took 14 seconds.

The result is presented in Table 1 covering four possible optimizations and evaluations. Each row represents an optimized criterion and each column an evaluated criterion. Each table entry reports the average relative difference in percent between the evaluated criterion and the optimum value for that criterion over the 100 instances. Naturally, the values on the diagonal are zero as the instances are solved to the optimum. For example, the value 2.63 at the intersection of row $C(1)$ and column C^{\max} means that the best solution found while optimizing $C(1)$ presents on average a C^{\max} value 2.63% larger than the optimal value for C^{\max} .

On these instances, Table 1 shows that $C(2)$ is the criterion that approximates best the other criteria, as a solution found when $C(2)$ is optimized is also often optimal with respect to the other criteria. The criterion $C(\infty)$ appears to be the worse criterion to approximate the three others. Conversely, $C(2)$ is hardly approximated by the others criteria while C^{∞} is quite well approximated by others. Criteria $C(1)$ and C^{\max} lie between these two extremes with $C(1)$ being a better approximation.

Instance	Nb Leaf Nodes		Nb Int. Sol.	
	first-fail	min-load	first-fail	min-load
8	1172	98	69	3
10	3191	722	58	13
12	30147	2975	87	12

Table 2: First-fail and proposed (“min-load”) heuristics on the original instances with $C(1)$

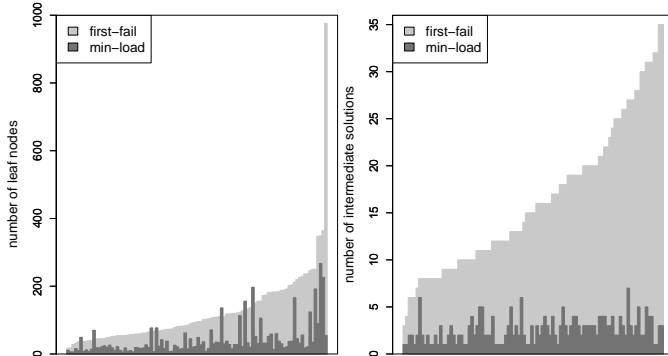


Figure 2: Comparison of heuristics on 100 instances. Left part : Number of leaves. Right part : Number of intermediate solutions.

6.2 Efficiency of the Proposed Branching Heuristic

To analyze the effect of the new value ordering heuristic, it is compared with the *first-fail* heuristic from [Hnich *et al.*, 2002] using the Branch-and-Bound search and with the minimization of $C(1)$ (the minimization with the other criteria gives similar results and are not reported). First the evaluation is achieved on the original instances used in [Hnich *et al.*, 2002] and on the 100 instances of the first generated problem set. The results are presented in Table 2 and Figure 2. The comparison is based on the number of leaf nodes and the number of intermediate solutions found during the Branch-and-Bound search. The left part of Figure 2 presents the number of leaf nodes with both heuristics, sorted by the values obtained with the basic heuristic. The right part presents the results in a similar way for the number of intermediate solutions.

It is clear that the new (“min-load”) branching heuristic improves the search except for 7 out of 100 instances (left part of Figure 2). As expected, the search is rapidly guided toward balanced solutions. Indeed, it can be seen on the right part of Figure 2 that the number of intermediate solutions is always smaller with the new branching heuristic. The same conclusions hold on the original instances in Table 2.

C^{\max}	-	D	DDS
BB	76.4	45.8	83.3
HY	96.5	94.0	95.7
IT	85.8	56.5	85.9

$C(1)$	-	D	DDS
BB	38.9	18.9	45.0
HY	76.5	70.4	76.9
IT	79.7	53.6	80.7

Table 3: Percentage of solved instances (on 720).

6.3 Comparing Search and Dominance Breaking Strategies

This section analyzes both the search strategies and the dominance breaking strategies. This study is performed on the second problem set using the new branching heuristic and with two different criteria, C^{\max} and $C(1)$. We restrict our attention to those two because the first is the most commonly used and the second is a good and fast approximation to the others according to the results of Section 6.1.

There are 3 search strategies (Section 4) referred to as “BB” for Branch-and-Bound, “HY” for the hybrid approach and “IT” for the iterative approach. There are 3 dominance rules strategies (Section 3.2) that are respectively not to apply them (“-”), to post them at the beginning (“D” standing for Dominance), and to use the adapted SBDS (“DDS” for Dominance During Search). This leads to 9 possible combinations.

The percentage of solved instances (on the 720 instances of the second test set) with each setting within the time limit of 30 seconds is presented in Table 3.

The hybrid search outperforms the two other search strategies with any dominance policy when minimizing C^{\max} . More than 96% of the instances can be solved with this technique. Among the two other techniques, the iterative one is overall better than Branch-and-Bound. On the other hand, when optimizing $C(1)$, the largest number of solved instances is obtained with the iterative strategy, followed by the hybrid search.

Concerning the different uses of dominance rules, posting them at the beginning of the search is a bad idea. It is seen with both criteria that column “D” is worse than the two others. On the contrary, the dynamic use of the dominance rules improves the search.

It is interesting to characterize the instances that are harder to solve. On the whole test set, only five instances are not solved within 30 seconds by any combination of the search and dominance strategies for the optimization of C^{\max} . They are all in the $[1, 10]$ class. 47 instances stay unsolved when using the criterion $C(1)$. Most of them (40) have credits ranging in the interval $[3, 5]$. As expected problems with credits in $[3, 5]$ and $[1, 10]$ are harder to solve.

Regarding the search strategies, there are respectively 93, 8 and 58 unsolved instances respectively for “BB”, “HY” and “IT” with any dominance strategy and with

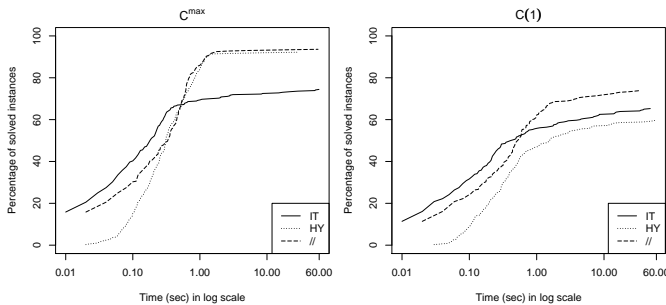


Figure 3: Percentage of solved instances in function of the allocated time.

C^{\max} . These values becomes 385, 104 and 84 when optimizing $C(1)$. These values show that some instances are only solved by one method. The effect of dominance rules is quite unpredictable. Adding dominance constraints increases the number of solved instances within the time constraint but it prevents the resolution of some instances. We have no clear understanding about which instances are easier to solve with and without the use of dominance rules.

Using the hybrid technique, the upper-bound found by LS is optimal in 93% of the solved instances with the criterion C^{\max} and in 77% of the solved instances for the minimization of $C(1)$. LS performs well on this problem with C^{\max} because there are many solutions in the search space and Local Search can easily reach one of them. The results are not as good with the criterion $C(1)$ because optimal solutions are sparser. The time spent in LS never exceeded one second and was less than 500 milliseconds most of the time. For easy instances, most of the work is done by LS to find an upper-bound while for harder instances, only a fraction of the time is spent in LS.

Figure 3 shows the percentage of solved instances among the harder classes ($[3, 5]$ and $[1, 10]$) in function of the allocated time. It compares three strategies for minimizing C^{\max} (left part) and $C(1)$ (right part). The three strategies use dynamic dominance rules and are respectively “IT”, “HY” and “//”. This last strategy supposes that “IT” and “HY” are run in parallel (on a single processor) and that they are both stopped when one of them finds the solution. Thus the time allocated for each search is half the normal time.

When optimizing C^{\max} , the iterative technique solves small problems faster than the hybrid one but the hybrid technique outperforms the iterative search for more complex problems. The method “HY” solves most problems in less than 1.5 seconds and is unable to solve problems after this limit. On the contrary, “IT” still increases the number of solved instances when allocated time increases. The parallel search technique slightly improves the results in comparison to the hybrid technique. This improvement is more important when minimizing $C(1)$. In this case, the parallelism permits to solve 10% more

instances when the allocated time is more than 1 second. Optimizing $C(1)$, “IT” is the technique that solves the most instances in less than 1 second. When the allocated time is more than 1 second, the parallel search becomes better because some hard instances with “IT” are solved with “HY”. The three techniques continue to improve their result when the allocated time increases.

7 Conclusion

This article presents a CP approach to the Balanced Academic Curriculum Problem [Hnich *et al.*, 2002]. It explores several directions to improve the resolution of this problem and it introduces an instance generator and two benchmarks of instances that allows us to validate the proposed improvements.

The first proposition is to consider other balance criteria than the minimization of the maximum load (C^{\max}) used up to now. We propose to minimize the mean deviation ($C(1)$), the mean square deviation ($C(2)$) or the maximum deviation ($C(\infty)$) from the mean workload. The first experiment shows that minimizing the mean square deviation is a good approximation to the other criteria. But existing propagators for $C(2)$ have a higher complexity than for the others criteria. For this reason, we propose to use the criterion $C(1)$ that is the second best approximation to the other criteria. C^{\max} can also be used because its minimization is far more efficient than the one of $C(1)$. A possible direction of future research is to combine these criteria. For instance, one could use C^{\max} to find a first well balanced solution then improve the balance with another criterion like $C(1)$ or $C(2)$.

Our second contribution consists in using an *ad-hoc* branching heuristic that guides the search towards balanced solutions. Experiments show that this new heuristic outperforms the classical *first-fail* heuristic to reach optimal solutions and to prove their optimality.

Next we propose the use of dominance relations to reduce the search. Dominance relations can be thought as one-way symmetries. Dominance relations for BACP are presented. They can be found in $\mathcal{O}(n^2.p)$ (where n is the number of courses and p the maximum number of prerequisites). These relations can be posted at the start of the search or can be dynamically added during the search, in a way similar to SBDS for symmetry breaking. Experiments show that it is counterproductive to post the dominance rules at the start. For the dynamic use of the dominance rules, the effect is less clear. While for some instances it reduces the search size, for other it increases it. The understanding of this phenomenon is a wide subject of research as it is the case in symmetry breaking.

The last contribution presents two alternatives to the simple Branch-and-Bound scheme to find optimal solutions. The first one relies on Local Search to find a good initial upper bound to start the Branch-and-Bound. We illustrate experimentally that this hybrid technique is really efficient for the BACP because the Tabu search often

finds optimal values. The second alternative is to start from below and iteratively increase the upper bound on the optimized criterion until finding a solution. This method performs well because it requires often few iterations before finding a solution. Experiments point out that the hybrid search is by far the best strategy when minimizing C^{\max} , while the iterative technique is a little better for the minimization of $C(1)$.

Although no technique is the panacea, there are only few instances that are not solved by at least one of the studied strategies. Consequently, we think that a portfolio of different search strategies could be the right choice to efficiently solve the BACP. We propose to use the hybrid and iterative search techniques with and without the dynamical use of dominance rules. The analysis of portfolio search techniques [Huberman *et al.*, 1997] for BACP is an interesting direction for future research.

Acknowledgment

The authors wish to thank the anonymous reviewers for their constructive comments. This research is supported by the Walloon Region, project TransMaze (516207).

References

- [Baptiste *et al.*, 2001] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publisher, 2001.
- [Castro and Manzano, 2001] C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM Working Group on Constraints*, 2001.
- [Gent and Smith, 2003] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. *Proceedings of the Third International Workshop on symmetry in Constraint Satisfaction Problems*, pages 55–65, 2003.
- [Gent *et al.*, 2006] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of constraint programming*. Elsevier, 2006.
- [Hentenryck and Michel, 2005] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT Press, 2005.
- [Hnich *et al.*, 2002] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR-2002*, pages 121–131, 2002.
- [Hnich *et al.*, 2004] Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Hybrid modelling for robust solving. *Annals of Operations Research* 130, pages 19–39, 2004.
- [Hoos and Stützle, 2004] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004.
- [Huberman *et al.*, 1997] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, January 3 1997.
- [Lambert *et al.*, 2006] Tony Lambert, Carlos Castro, Eric Monfroy, and Frédéric Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. In *Proceedings of The International Conference on Artificial Intelligence and Soft Computing (ICAISC'06)*, volume 4029 of *Lecture Notes in Artificial Intelligence*, pages 410–419, Zakopane, Poland, 2006. Springer Verlag.
- [Pesant and Régim, 2005] G. Pesant and J.C. Régim. Spread: A balancing constraint based on statistics. *Lecture Notes in Computer Science*, 3709:460–474, 2005.
- [Prestwich and Beck, 2004] Steven Prestwich and J. Christopher Beck. Exploiting dominance in three symmetric problems. *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.
- [Schaus *et al.*, 2006] P. Schaus, Y. Deville, P. Dupont, and J.C. Régim. Simplification and extension of the spread constraint. *Third International Workshop on Constraint Propagation And Implementation*, pages 77–91, 2006.
- [Schaus *et al.*, 2007] P. Schaus, Y. Deville, P. Dupont, and J.C. Régim. The deviation constraint. *4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, pages 260–274, 2007.

Substitutability Based Domain Decomposition for Constraint Satisfaction

Wady Naanaa

Faculté des Sciences de Monastir,
5019 Monastir, Tunisia
naanaa.wady@planet.tn

Abstract

This paper introduces a domain decomposition algorithm which exploits a weak form of neighborhood substitutability called directional substitutability. The main idea is that although two values of a variable may not be neighborhood substitutable if we consider the whole neighborhood of the variable, they could be substitutable if we only focus on a part of the neighborhood induced by a variable ordering. Directional substitutability provides a mean to decompose value domains into chains of directionally substitutable values that can be attempted simultaneously by a tree search.

We report experiments carried out on several binary CSPs which clearly indicate that variations of the Forward-Checking and the Maintaining Arc-Consistency algorithms which exploit directional substitutability or directional interchangeability, a special case of directional substitutability, often outperform the original algorithms.

1 Introduction

Constraint Satisfaction Problems (CSPs) provide a general framework for modeling and solving numerous combinatorial problems. Basically, a CSP consists of a set of variables, each of which can take a value chosen among a set of potential values called its *domain*. The constraints express restrictions on combinations of domain values. The problem is to find an assignment of values to variables, from their respective domains, such that all the constraints are satisfied.

CSPs, which are known to be NP-complete problems, can model problems in various domains. For that reason, many solving algorithms have been developed for them. In this article we are interested in complete methods which have the advantage of finding at least one solution to a problem if such a solution exists.

For large CSPs, finding a solution remains a time consuming task. Moreover, the emergence of new concrete problems reinforces the need for increasingly powerful algorithms.

In [Freuder, 1991], Freuder defined neighborhood interchangeability and substitutability of domain values which are special kinds of symmetry that can be used in reducing the

domains of variables by removing redundant values. Because in a subset of neighborhood interchangeable or substitutable values, one can keep only one value of the subset in the domain of the variable while not affecting the satisfiability of the original problem. This may lead to the exploration of a smaller search tree if neighborhood substitutability occurs frequently in a problem. However, for many problems little substitutability occurs and the overhead due to determining redundant values could offset the benefit. In an attempt to exploit substitutability more intensively, many extensions have been proposed [Bowen and Likitvivanavong, 2004; Lal *et al.*, 2005; Zhang and Freuder, 2004].

This paper introduces a domain decomposition algorithm which exploits a variation of neighborhood substitutability called directional substitutability. The proposed concept differs from neighborhood substitutability in that it assumes a variable ordering and uses it to focus only on a subset of the neighboring variables. That is, when determining directionally substitutable values for the domain of a given variable, the neighborhood of this variable is limited to variables coming earlier in the ordering. This restriction allows directional substitutability to occur more often than neighborhood substitutability, but in compensation it does not allow the removal of redundant values. The reason is that directionally substitutable values are not comparable with regard to the domains of all the neighboring variables and then one cannot keep a single value from each subset of directionally substitutable values while preserving problem satisfiability. Nonetheless, directional substitutability can be used to partition value domains into chains of directionally substitutable values in such a way that values within the same chain could be attempted simultaneously by a tree search algorithm. Hence, directional substitutability can be exploited by a tree search algorithm to assign to each variable a specific subset of its domain at each branch of search instead of using a single value per branch as it is the case in classical search algorithms. This would give a smaller search tree to explore since a variable to which we have affected a subset of directionally substitutable values is considered as instantiated and is discarded from the sub-problem at hand.

The paper is organized as follows: the next section introduces some definitions and notation conventions. Section 3 is

devoted to formally defining directional substitutability, and presenting a search algorithm which exploits directional substitutability. We also provide in the same section, an algorithm for computing minimum width partitions of value domains into chains of directionally substitutable values. Directional interchangeability, a special case of directional substitutability is presented in section 4. An experimental study carried on several binary CSPs is reported in section 5. Section 6 discusses related work and section 7 presents some conclusions and future works.

2 Definitions and Notations

Definition 1 A constraint satisfaction problem (CSP) is given by a triple (X, D, C) where:

1. $X = \{x_1, \dots, x_n\}$ is a finite set of variables.
2. $D = \{D_1, \dots, D_n\}$ is a sequence of value domains so that D_k is the domain of x_k .
3. $C = \{C_1, \dots, C_m\}$ is a set of constraints. Each constraint C_k applies on a sub-set of variables $Var(C_k) = \{x_{k_1}, \dots, x_{k_r}\}$ called the scope of C_k and is defined by a r -ary relation $Rel(C_k) \subseteq D_{k_1} \times D_{k_2} \dots \times D_{k_r}$. $Rel(C_k)$ determines the r -tuples of values accepted by C_k .

The arity of a constraint is the size of its scope. The arity of a problem is the maximum arity over its constraints. In this paper, we focus on binary CSPs, i.e. CSPs with binary and unary constraints only. It must be emphasized however that the proposed concept can be easily extended to cope with non-binary CSPs. Two variables x_i and x_j connected by a binary constraint denoted by $C_{i,j}$ are said to be neighbors. That is, $x_i \in Neighbours(x_j)$ and vice versa. A value $a \in D_i$, (which is also denoted by (x_i, a) when necessary), is compatible with $b \in D_j$ if $(a, b) \in Rel(C_{i,j})$. In this case, a is called a support of b . Given a binary CSP, its constraint graph associates each variable with a node and links any oriented pair of nodes (x_i, x_j) such that $C_{i,j} \in C$. The notion of arc-consistency, a widely studied consistency level, is defined from the constraint graph as follows. An arc (x_i, x_j) in the constraint graph is arc-consistent iff every value in D_i has a support in D_j . A CSP is arc-consistent iff all the arcs of its constraint graph are arc-consistent.

The support set of a value a of a variable x_i is defined by

$$N(x_i, a) = \{(x_j, b) \mid x_j \in Neighbours(x_i) \text{ and } (a, b) \in Rel(C_{i,j})\} .$$

We define the preceding support set of a value a of a variable x_i with respect to an ordering \prec of the variables as follows:

$$\vec{N}(x_i, a) = \{(x_j, b) \mid x_j \in Neighbours(x_i) \text{ and } (a, b) \in Rel(C_{i,j}) \text{ and } x_j \prec x_i\} .$$

Definition 2 A problem $P = (X, D, C)$ is said to be a reduction of $P' = (X, D', C)$ (notation $P \sqsubseteq P'$) iff $D_i \subseteq D'_i$ for each $x_i \in X$.

3 Directional Substitutability (DS)

The notion of directional substitutability is a weak form of neighborhood substitutability. It is defined with regard to an ordering of the problem variables in the following manner

Definition 3 Let P be a binary CSP and let a and b be two values in the domain of a variable x_i . b is said to be directionally substitutable to a with respect to a given ordering of the variables, (notation $a \preceq_{DS}^P b$), iff $\vec{N}(x_i, a) \subseteq \vec{N}(x_i, b)$.

For short, \preceq_{DS} will be used instead of \preceq_{DS}^P if this does not cause any ambiguity. The \preceq_{DS} relation defines a partial order on the domains of each variable. Yet, the domain D_i of a variable x_i can be split into subsets $D_i = \{D_{i,1}, \dots, D_{i,s}\}$ such that the elements of each $D_{i,k}$, $k : 1, \dots, s$ are all pairwise comparable. That is, for any value pair a and b in any $D_{i,k}$, we either have $a \preceq_{DS} b$ or $b \preceq_{DS} a$. Hence, each $D_{i,k}$ is a chain, i.e., a subset of D_i which is totally ordered by \preceq_{DS} . In each chain $D_{i,k}$, we can distinguish a subset of maximum elements

$$\max(D_{i,k}) = \{b \in D_{i,k} \mid \forall a \in D_{i,k}, a \preceq_{DS} b\} .$$

One can distinguish an interesting case in which each value domain of the problem at hand is a single chain of DS values.

Proposition 4 Let P be an arc-consistent binary CSP in which each value domain is a chain of DS values. Then P can be solved in a backtrack-free manner.

Proof: We prove that by selecting the maximum elements from each value domain, we get a solution bundle. That is, each element of $\max(D_1) \times \dots \times \max(D_n)$ is a solution. Suppose there exists an n -tuple of $\max(D_1) \times \dots \times \max(D_n)$ which contains a pair of incompatible values. This means that there exists a pair of variables x_i and x_j such that $a \in \max(D_i)$ is incompatible with $b \in \max(D_j)$. Suppose, without loss of generality that x_i precedes x_j in the considered variable ordering. We have $a \notin \vec{N}(x_j, b)$ since a and b are incompatible and x_i precedes x_j in the ordering. Moreover, $b \in \max(D_j)$ then $\forall c \in D_j, c \preceq_{DS} b$ which is equivalent to $\vec{N}(x_j, c) \subseteq \vec{N}(x_j, b)$. It follows that $\forall c \in D_j, a \notin \vec{N}(x_j, c)$. This means that all the values of D_j are incompatible with a . This latter has therefore no support in D_j . Thus, the arc (x_i, x_j) is not arc-consistent. This contradicts the fact that P is arc-consistent.

Proposition 5 Let P and P' be two CSPs such that $P \sqsubseteq P'$ and let a and b be two values of a variable x_i which are available in both problems. Then, we have

$$a \preceq_{DS}^{P'} b \Rightarrow a \preceq_{DS}^P b .$$

Proof: Let ΔD denotes the subset of values that are available in P' but not in P , that is

$$\Delta D = \bigcup_{x_i \in X} D'_i - D_i$$

and let $\vec{N}^P(x_i, a)$, (resp. $\vec{N}^{P'}(x_i, a)$) be the preceding support set of a in P (resp. in P'). Since $P \sqsubseteq P'$, we have:

$$\vec{N}^P(x_i, a) = \vec{N}^{P'}(x_i, a) - \Delta D . \quad (1)$$

Similarly,

$$\vec{N}^P(x_i, b) = \vec{N}^{P'}(x_i, b) - \Delta D . \quad (2)$$

On the other hand, the relation $a \preceq_{DS}^{P'} b$ is equivalent to $\vec{N}^{P'}(x_i, a) \subseteq \vec{N}^{P'}(x_i, b)$. It follows that,

$$N^{P'}(x_i, a) - \Delta D \subseteq \vec{N}^{P'}(x_i, b) - \Delta D$$

Then, by (1) and (2), we get

$$\vec{N}^P(x_i, a) \subseteq \vec{N}^P(x_i, b)$$

which is equivalent to $a \preceq_{DS}^P b$.

The consequence of proposition 5 is that

Proposition 6 *For any pair of problem $P = (X, D, C)$ and $P' = (X, D', C)$ such that $P \sqsubseteq P'$, if D'_i is a chain of DS values in P' then D_i is a chain of DS values in P .*

Proof: The proof follows immediately from proposition 5 by observing that in a chain, all values are comparable.

3.1 Exploiting Directional Substitutability

In the following, we show how to exploit the concept of DS by conceiving a solving algorithm (MAC-DS) which offers the opportunity of attempting simultaneously several values at each branch of the search tree.

MAC-DS (see Function 7) is typically a 2-way-branching depth-first search algorithm enhanced by a look-ahead schema which maintains arc-consistency. The algorithm takes the problem to be solved and a set of uninstantiated variables as parameters and proceeds as follows. At each node of the search tree, it selects a variable which is not instantiated yet, computes a partition of its domain into chains of DS values and decomposes the current problem into two sub-problems. The first sub-problem is obtained by reducing the domain of the current variable to a selected chain of DS values ($\mathcal{D}_{i,k}$ in the pseudo-code). The arc-consistency of the resulting problem is therefore restored by means of an AC algorithm. Then, a recursive call is performed in order to consider the future variables. If the selected chain does not lead to a solution, a domain restoration process is performed. Then, the attempted chain is discarded from the domain of the current variables. Next, the arc-consistency of the resulting problem is restored again before performing a recursive call to consider the remaining alternatives.

In case the algorithm succeeds to instantiate all variables, it executes an additional polynomial process (line 2). Its role is to extract solutions from the selected chains. We prove in the following that by selecting the maximum of each selected chains, we get a solution bundle of the CSP.

Function 7 MAC-DS($(X, D, C), Y$)

1. **if** $Y = \emptyset$ **then**
2. return(MaximumValues(D))
3. **else**
4. $x_i \leftarrow \text{Select}(Y)$
5. $\mathcal{D}_i \leftarrow \text{DS-Partition}((X, D, C), Y, x_i)$

6. $\mathcal{D}_{i,k} \leftarrow \text{Select}(\mathcal{D}_i)$
7. $D_i \leftarrow \mathcal{D}_{i,k}$
8. AC(X, D, C)
9. **if** no empty domain **then**
10. $I \leftarrow \text{MAC-DS}((X, D, C), Y - \{x_i\})$
11. **if** $I \neq \emptyset$ **then**
12. return(I)
13. Restore(D)
14. $D_i \leftarrow D_i - \mathcal{D}_{i,k}$
15. AC(X, D, C)
16. **if** no empty domain **then**
17. $I \leftarrow \text{MAC-DS}((X, D, C), Y)$
18. **if** $I \neq \emptyset$ **then**
19. return(I)
20. Restore(D)
21. return(\emptyset)

MAC-DS explores a binary search tree since it is a 2-way-branching DFS algorithm. As can be seen from the parameters of the algorithm, a node of the search tree can be characterized by a couple (P, Y) , where P is a problem and $Y \subseteq X$ is a subset of uninstantiated variables. We use functions $left()$, $right()$ and $parent()$ to designate the immediate successors and the parent of a given node. For convenience, these functions will also be applied to problems and sets of uninstantiated variables. Hence, $parent(P)$, will designate the problem available at node $parent(P, Y)$. Processing a node (P, Y) yields two immediate successors, a left child

$$left(P, Y) = (ac(P|_{\mathcal{D}_{i,k}}), Y - \{x_i\}) . \quad (3)$$

and a right child

$$right(P, Y) = (ac(P|_{D_i - \mathcal{D}_{i,k}}), Y) . \quad (4)$$

$left(P, Y)$ is obtained from (P, Y) by reducing D_i to a chain of DS values ($\mathcal{D}_{i,k} \subseteq D_i$), restoring the arc-consistency in $P|_{\mathcal{D}_{i,k}}$ and removing x_i from Y . $right(P, Y)$ is obtained from (P, Y) by removing the elements of $\mathcal{D}_{i,k}$ from D_i , restoring arc-consistency in $P|_{D_i - \mathcal{D}_{i,k}}$ and keeping Y unchanged. It can be easily seen that problems processed at the immediate successors of a given node are reductions of the problem processed at that node. Indeed, we have

$$ac(P|_{\mathcal{D}_{i,k}}) \sqsubseteq P|_{\mathcal{D}_{i,k}} \sqsubseteq P . \quad (5)$$

and

$$ac(P|_{D_i - \mathcal{D}_{i,k}}) \sqsubseteq P|_{D_i - \mathcal{D}_{i,k}} \sqsubseteq P . \quad (6)$$

Therefore, since \sqsubseteq is transitive, given a node (P, Y) , the problems processed at the successors of (P, Y) are all reductions of P and P is a reduction of all problems processed at the predecessors of (P, Y) .

Proposition 8 *Let $P = (X, D, C)$ be a binary CSP and (P^n, Y^n) a node visited by MAC-DS(P, X) such that $Y^n = \emptyset$. Then all value domains in P^n are chains of DS values.*

Proof: For simplicity, we assume that variables are instantiated following the natural order of their indices. Starting

from the root of the search tree (P, X) , which is also denoted by (P^0, Y^0) , the algorithm reaches a node (P^n, Y^n) such that $Y^n = \emptyset$. Then, according to (3) and (4), (P^n, Y^n) is necessarily the last element of a sequence of nodes $(P^0, Y^0), (P^1, Y^1), \dots, (P^n, Y^n)$ with $n = |X|$ such that (P^i, Y^i) , $i : 1, \dots, n$ is a left child and (P^j, Y^j) is a successor of (P^i, Y^i) for all $1 \leq i < j \leq n$. It follows that,

$$\begin{aligned} (P^i, Y^i) &= \text{left}(\text{parent}(P^i, Y^i)) \\ &= (\text{ac}(\text{parent}(P^i)|_{\mathcal{D}_{i,k}}, \text{parent}(Y^i) - \{x_i\}) \end{aligned}$$

The domain of x_i in $\text{parent}(P^i)|_{\mathcal{D}_{i,k}}$ is equal to $\mathcal{D}_{i,k}$ which is a chain of DS values. Since $\text{ac}(\text{parent}(P^i)|_{\mathcal{D}_{i,k}}) \subseteq \text{parent}(P^i)|_{\mathcal{D}_{i,k}}$, by proposition 6, the domain of x_i in $\text{ac}(\text{parent}(P^i)|_{\mathcal{D}_{i,k}})$ is also a chain of DS values. Hence, D_i^i , which denotes the domain of x_i in P^i , is also a chain of DS values, and this is true for $i : 1, \dots, n$.

On the other hand, (P^n, Y^n) is a successor of all (P^i, Y^i) , $i : 1, \dots, n-1$. Then, P^n is a reduction of all (P^i, Y^i) , $i : 1, \dots, n-1$. And since D_i^i is a chain of DS values in P^i , for $i : 1, \dots, n$. By applying proposition 6, we deduce that D_i^n , $i : 1, \dots, n$ is a chain of DS values. Which means that all the value domains in P^n are chains of DS values.

Proposition 8 suggests that if MAC-DS reaches a node of the search tree such that the set of uninstantiated variables is empty then by proposition 4, one can get, in a backtrack-free manner, a solution bundle, that is a set of solution expressed as a Cartesian product of subsets of the value domain of each variable.

3.2 DS Variable Ordering

The variable ordering with regard to which DS is determined has a great impact on the opportunities of DS. It can either be static, a minimum width order for instance, or dynamic, determined during search by any variable ordering heuristic. However, from the practical point of view, we obtained the best results when DS are determined with regard to the order in which the variables are instantiated during search. At a given node of the search tree, the variables that precede the current variable in the ordering are simply the instantiated variables. Hence, the preceding support sets of the values of the current variable can be computed and so the DS partition of its value domain.

This order favors DS at the top of the search tree. Thus, the variable instantiated at first, will have a single chain in its DS partition. This is because the preceding support sets of all its values are empty. This means that the algorithm will not take any decision at the root of the search tree. Branching on large chains of DS values at the top of the search tree is rather an advantageous strategy because at this point, the value ordering heuristic does not have enough information to do good guesses. By taking imprecise decisions, early mistakes could be avoided.

3.3 An Example

Consider the CSP depicted in Fig. 1. All variables have the same value domain which is equal to $\{0, 1, 2, 3, 4\}$.

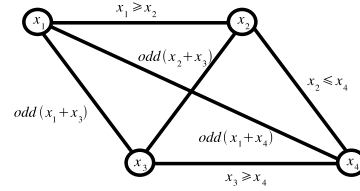


Figure 1: A binary CSP

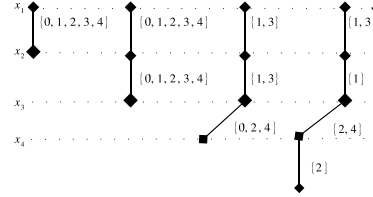


Figure 2: Paths explore by MAC-DS when applied to the CSP of Fig. 1

The application of MAC-DS to this example gives the steps detailed in Fig. 2. For simplicity, we did not use any variable or value ordering heuristic, and the order with regard to which DS is determined is the natural order of variable indices. In step 1, all the values in D_1 are DS since x_1 is the first variable in the ordering and therefore $\vec{N}(x_1, a) = \emptyset$ for all $a \in D_1$. In the next step, the DS based partition of D_2 also gives $\mathcal{D}_2 = \{\{0, 1, 2, 3, 4\}\}$. Indeed, we have $4 \preceq_{DS} 3 \preceq_{DS} 2 \preceq_{DS} 1 \preceq_{DS} 0$. Hence, the algorithm develops a single branch labeled by $\{0, 1, 2, 3, 4\}$. The partition of D_3 yields $\mathcal{D}_3 = \{\{0, 2, 4\}, \{1, 3\}\}$ since we have $\vec{N}(x_3, 0) = \vec{N}(x_3, 2) = \vec{N}(x_3, 4) = \{(x_1, 1), (x_1, 3), (x_2, 1), (x_2, 3)\}$ and $\vec{N}(x_3, 1) = \vec{N}(x_3, 3) = \{(x_1, 0), (x_1, 2), (x_1, 4), (x_2, 0), (x_2, 2), (x_2, 4)\}$. Then, the algorithm attempts the chain $\{0, 2, 4\}$ for x_3 . By constraint propagation, values 0, 2 and 4 are removed from D_1 and D_2 , 0, 1 and 3 are removed from D_4 , and 0 is discarded from D_3 . At this stage, the domain of x_4 is reduced to $\{2, 4\}$. These values are not comparable, which means that $\mathcal{D}_4 = \{\{2\}, \{4\}\}$. The algorithm attempts therefore $\{2\}$ yielding the removal of 3 from x_2 . At that stage, the values remaining in the domains are depicted on the last path of Fig 2. They are all maximum values which yields a bundle containing four solutions.

3.4 DS Partitioning Algorithm

As noticed in the precedent paragraph, the \preceq_{DS} relation defines a partial order on the domain of each variable x_i denoted by (D_i, \preceq_{DS}) . This partial order can be used to partition D_i into chains of DS values. For a given partially ordered set, one can have many chain partitions. In order theory, the optimal chain partition of a partially ordered set is known as the *Dilworth's chain partition* (DCP)[Dilworth,

1950]. It is a chain partition which involves the minimum number of chains over all possible chain partitions. The size of such a partition defines the width of the partial order. The problem of finding a DCP of a partially ordered set can be solved in polynomial time by expressing it as a maximum matching problem in a bipartite graph.

Motivated by the fact that, at each node of the search tree, the solving algorithm will have as many alternatives as there are chains in the DCP of the domain at hand, we propose to compute a DCP at each node of search. This is intended to minimize the size of the search tree to be explored.

The first step in computing a DCP (see Function 9) consists in determining the \preceq_{DS} relation. This is the most time consuming step in the partitioning algorithm. It amounts to performing $d(d-1)/2$ inclusion tests. Each inclusion test can be accomplished by $O(nd)$ consistency checks. Hence, a time complexity of $O(nd^3)$ for this first step. Next, the algorithm constructs a bipartite graph $G = (V, U, E)$ where $V = U = D_i$ and the set of edges E contains an edge (a, b) iff $a \preceq_{DS} b$. The construction of G takes $O(d^2)$ steps. A maximum matching algorithm is therefore applied to G . This can be accomplished by the algorithm described in [Hopcroft and Karp, 1973] which runs in $O(d^{2.5})$. The chains of the partition are extracted from the maximum matching by including a and b in a same chain whenever there is an edge (a, b) in the maximum matching. This takes $O(d^2)$ steps. Hence, an overall time complexity of $O(nd^3)$.

Function 9 DS-Partition(x_i, X, D, C)

1. $\preceq_{DS} \leftarrow \text{PrecSuppSetInclusion}(x_i, X, D, C)$
2. $G \leftarrow \text{BipartiteGraph}(D_i, \preceq_{DS})$
3. $M \leftarrow \text{MaximumMatching}(G)$
4. $\mathcal{D}_i \leftarrow \text{ExtractChains}(D_i, M)$
5. return(\mathcal{D}_i)

4 Directional Interchangeability (DI)

A special case of directional substitutability arises when we replace the inclusion condition in Definition 3 by an equality. This gives rise to the notion of directional interchangeability (DI). The DI condition is of course stronger than the condition imposed by DS. As a consequence, DI may occur less often than DS. Nonetheless, on certain problems, (namely, random binary CSPs), DI based algorithms have experimentally proved to be more efficient than DS based algorithms.

Contrary to DS, DI defines an equivalence relation on the domain of each variable. The domain D_i of a variable x_i can be split into a set of sub-domains $\mathcal{D}_i = \{\mathcal{D}_{i,1}, \dots, \mathcal{D}_{i,s}\}$ such that the elements of each $\mathcal{D}_{i,k}, k : 1, \dots, s$ are pairwise DI.

In the following, we present FC-DI a variation of the Forward-Checking algorithm which exploits directional interchangeability. FC-DI (see Function 10) is typically a k-way-branching depth first search algorithm except that it branches over subsets of DI values. We chose the k-way-branching strategy because it has experimentally proved to be more adequate for FC than a 2-way-branching strategy.

At each choice point, FC-DI selects a non instantiated variable x_i , and computes the partition $\mathcal{D}_i = \{\mathcal{D}_{i,1}, \dots, \mathcal{D}_{i,s}\}$ of D_i based on DI. The DI based partition can be accomplished

in $O(nd^2)$ steps by using the algorithm presented in [Freuder, 1991] or the one proposed in [Naanaa, 2007]. The algorithm iterates over the subsets of the domain partition \mathcal{D}_i and reduces, at each iteration, D_i to $\mathcal{D}_{i,k}$, (see line 7). Then, function Propagate is called. At this point, x_i is considered as an instantiated variable and is discarded from the current sub-problem even if its domain is not reduced to a singleton. If no empty domain is encountered, then the algorithm performs a recursive call to process the future variables. Else, it considers another element of \mathcal{D}_i . If all iterations return no solution, the algorithm backtracks to the immediately preceding variable. If FC-DI succeeds to instantiate all the variables then a solution bundle is found.

Function 10 FC-DI($(X, D, C), Y$)

1. **if** $Y = \emptyset$ **then**
2. return(D)
3. **else**
4. $x_i \leftarrow \text{Select}(Y)$
5. $\mathcal{D}_i \leftarrow \text{DI-Partition}(x_i, X, D, C)$
6. **for each** $\mathcal{D}_{i,k} \in \mathcal{D}_i$ **do**
7. $D_i \leftarrow \mathcal{D}_{i,k}$
8. Propagate(x_i, X, D, C)
9. **if** no empty domain **then**
10. $I \leftarrow \text{FC-DI}((X, D, C), Y - \{x_i\})$
11. **if** $I \neq \emptyset$ **then**
12. return(I)
13. Restore(D)
14. **return**(\emptyset)

To specify the look-ahead scheme performed by FC-DI, we need to distinguish past and future variables. A past variable is a variable which precedes the current variable in the variable ordering used in determining DI. Otherwise, the variable is a future variable. The invariant to be maintained by FC-DI during search is that every arc of the constraint graph whose target node represents a past variable must be arc-consistent. This is ensured by procedure Propagate which is not detailed here. In [Naanaa, 2007], we prove that, if FC-DI succeeds to construct a complete path then we get in D a solution bundle.

5 Experimental Results

The problems investigated in our experiments are random binary CSPs generated according to model RB, radio link frequency assignment problems (RLFAP) [Cabon *et al.*, 1999], modified RLFAP and job-shop problems. We compared FC-DI and MAC-DS with FC and MAC. The arc-consistency algorithm underlying MAC and MAC-DS is AC-3. The variable ordering heuristic used by all algorithms is *min-domain/wdeg* [Boussemart *et al.*, 2004]. For value ordering, we used the min-conflict-first heuristic [Frost and Dechter, 1995]. The evaluation criteria are the number of expanded nodes and CPU time in second. All algorithms were implemented in C++. They were run in a 1.7 GHZ PC having 256 Mb of RAM.

Random Binary CSPs: for random binary CSPs, we experimented on problems involving $n = 30$ variables, a uniform domain size of $d = 25$ and a constraint graph density

p_1 equal to 0.5 and 1. We also experimented on problems involving 40 variables, 20 values per domain and constraint graph density equal to 0.25 and 0.5. The constraint tightness (p_2) is varied so that we obtain instances around the peak of complexity. We used the model RB generator developed by van Hemert which is available at [Van hemert, 2004]. For problem ($n = 40, d = 20, p_1 = 0.25$) and in accordance with [Xu and Li, 2000], we stopped the experimentation at $p_2 = 0.5$ in order to avoid flawed instances. For the ($n = 40, d = 20, p_1 = 0.5$) instances, we did not report the results obtained by MAC because the execution times were prohibitively long. The size of samples is 100 problem instances for each data point. We compared FC-DI against FC and MAC on these problems.

For Fig. 3 and 4 which report results expressed as average values of number of expanded nodes and CPU time, the horizontal axis denotes various tightness. The results indicate that FC-DI outperforms FC and MAC on ($n = 30, d = 25$). This is true even on highly constrained problems ($p_1 = 1$). This advantage is less significant on ($n = 40, d = 20$) problems. For ($n = 40, d = 20, p_1 = 0.5$), FC and FC-DI are very close, nonetheless, FC-DI remains almost always faster than FC. But on problems with more constraints ($n = 40, d = 20, p_1 = 1$) for instance, we expect that FC will be faster than FC-DI. We claim that on random binary CSP, FC-DI would be more efficient than FC and MAC on problems involving a high d/n ratio. because on such problems, DS values would be more frequently encountered whereas FC and MAC are more likely to do early mistakes due to large value domains.

RLFAP and Modified RLFAP: We considered the scen11 instance and instances obtained from scen11 by removing the highest frequencies denoted by scen11-f1 to scen11-f10. We compared MAC-DS against MAC on these instances because MAC outperformed FC. As can be seen in Fig. 5, MAC-DS is clearly better than MAC regarding both criteria, CPU time and of number of expanded nodes. On the hardest instances MAC-DS is up to two times faster.

Job-shop Problems: For the job-shop problems, we investigated the js-taillard-15-by-15-105 instances [Lecoutre, 2007]. The experiment involves ten satisfiable instances generated according to the Taillard model [Taillard, 1993]. In the resulting CSPs, the variables represent the tasks to be scheduled, the domain values represent the possible start-times of the different tasks and the constraints are precedence and resource constraints. To cope with the large domain values, (more than one thousand values), which characterize these instances, we resorted to a limited discrepancy search algorithm [Harvey and Ginsberg, 1995]. Hence, we immersed the MAC and the MAC-DS algorithms in a loop which incrementally varies the number of allowed discrepancies. Discrepancies are defined with regard to the min-conflict value ordering heuristic. The resulting search algorithms are denoted by LDS-MAC and LDS-MAC-DS. On these instances, LDS-MAC-DS is clearly more efficient than LDS-MAC as it can be seen in Fig. 5. Indeed, within three hours per instance, LDS-MAC solved only four instances over ten, while

LDS-MAC-DI solved all the instances. Moreover, on instances solved by both algorithms, LDS-MAC-DI is faster on the hardest instances (instances 8 and 9 in Fig. 5).

6 Related work

Many variations of neighborhood substitutability and interchangeability have been studied. In [Bellicha *et al.*, 1994], neighborhood substitutable values are dynamically detected during search in order to filter value domains.

In [Zhang and Freuder, 2004], the authors proposed conditional interchangeability, which is intended to strengthen the pruning ability of neighborhood interchangeability. A condition is a restriction on the domain of neighboring variables whose role is to capture interchangeably in a limited situation that cannot be detected by neighborhood interchangeability.

Bowen and Likitvivanavong introduced domain transmutation a concept which is closely related to interchangeability [Bowen and Likitvivanavong, 2004]. This approach consists in splitting domain values into several “sub-values” or merging values so that interchangeability is more intensively exploited. The authors reported that their approach is particularly advantageous in finding all solutions.

The concept of neighborhood interchangeability was also applied to non-binary CSPs [Lal *et al.*, 2005]. The authors proposed an algorithm for computing neighborhood interchangeable values which takes into account non-binary constraints. They also described how to interleave their algorithm with search in order to obtain a search which performs dynamic bundling.

All the methods cited above uses special forms of interchangeability or substitutability in order to determine redundant values that can be eliminated from the problem. Hence, these methods can be classified as domain filtering methods. In our case, directional substitutability do not enable the removal of redundant values but in compensation it allow to attempt more than one value at a single branch of the search tree. Directional substitutability is therefore a mean which enables domain decomposition during search.

7 Conclusion

This paper presented directional substitutability (DS), a weak form of neighborhood substitutability that assumes a variable ordering and uses it to focus on a subset of the neighboring variables. We presented an effective algorithm for determining DS values and described how to integrate it into solving algorithms to get search algorithms that assigns to variables subsets of their respective domains at each branch of the search tree.

An experimental study carried on numerous binary CSPs showed that directional substitutability is a concrete technique that repays its overhead especially when considering hard and large problems. Indeed, FC-DS and MAC-DS defeat the standard FC and MAC on modified RLFAP instances and on jobshop instances experimentally proving that exploiting DS is worth the effort on such problems. On random instances, the experiments showed that DI based search is competitive on instances involving a high d/n ratio.

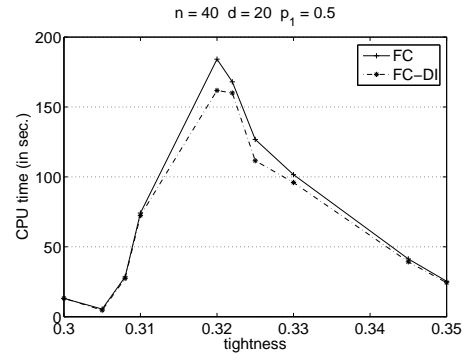
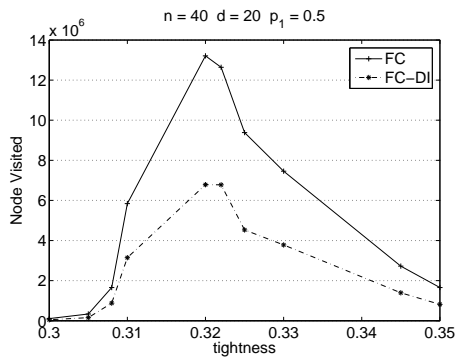
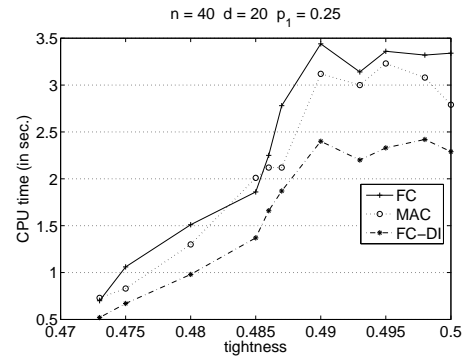
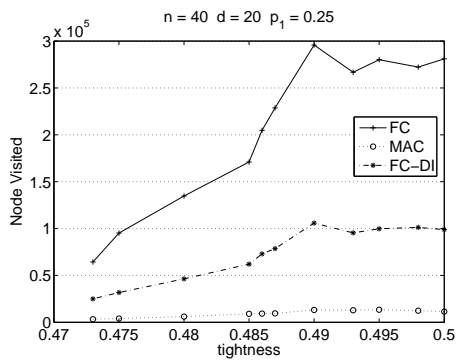
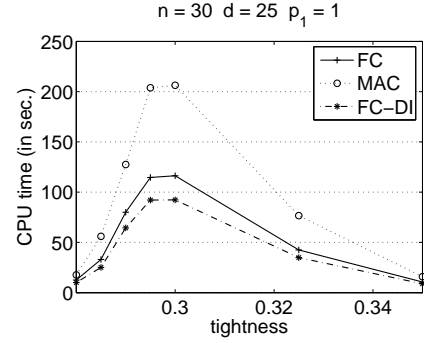
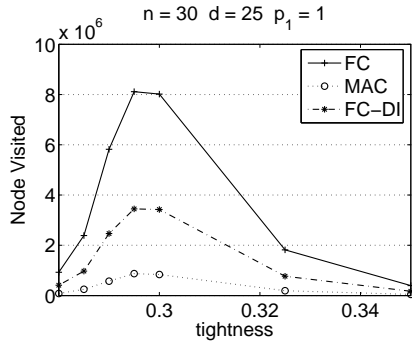
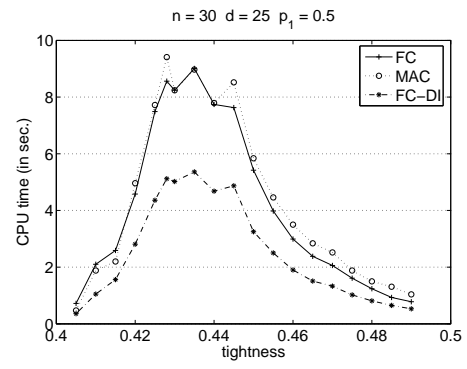
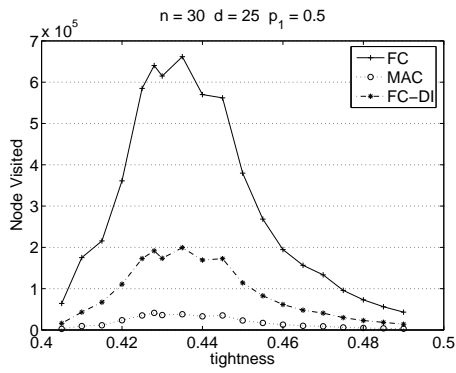


Figure 3: Average search effort for FC, MAC and FC-DI on model RB random binary CSP instances. Mean number of expanded nodes is reported

Figure 4: Average search effort for FC, MAC and FC-DI on model RB random binary CSP instances. Mean CPU time is reported

instance	expanded nodes		time (in sec.)	
	MAC	MAC-DS	MAC	MAC-DS
scen11 (sat)	1201	4480	1	1
scen11-f10	1350	1094	1	1
scen11-f9	21669	13865	11	9
scen11-f8	37935	20724	20	13
scen11-f7	284110	144323	99	63
scenn1-f6	348658	151122	141	72
scen11-f5	2022027	961241	713	402
scen11-f4	11571244	4981978	3564	2092
scen11-f3	37795757	15218681	11571	6277
scen11-f2	–	37116459	> 36000	14615
scen11-f1	–	80698639	> 36000	31014

Figure 5: Search effort for MAC and MAC-DS in terms of expanded nodes and execution time (in second) obtained on RLFAP and modified RLFAP instances. A dash indicates a timeout after 10 hours.

no	expanded nodes		time (in sec.)	
	MAC	MAC-DS	MAC	MAC-DS
0	–	20465	–	618
1	18859	9443	183	302
2	–	39119	–	987
3	–	12583	–	414
4	–	24624	–	612
5	–	13484	–	347
6	27073	31408	213	713
7	–	153424	–	2754
8	1183674	21447	7261	411
9	830122	85732	6308	2226

Figure 6: Search effort for LDS-MAC and LDS-MAC-DS on the js-taillard-15-by-15-105 instances. Number of expanded nodes and CPU time are reported. A dash indicates a timeout after 3 hours.

A natural extension of this work is to exploit DS in the framework of non-binary CSPs. In the case of a k -ary CSP, determining directionally substitutable values amounts to verifying inclusions between sets containing $(k - 1)$ -tuples, which is not a prohibitive task.

References

[Bellicha *et al.*, 1994] Amit Bellicha, Christian Capelle, Tibor Kókény and Marie-Catherine Vilarem. CSP technics using partial orders on domain values. In *Proceedings of the European Conference on Artificial Intelligence Workshop on Constraint Satisfaction*, pages 47–56, Amsterdam, The Netherlands, 1994.

[Boussemart *et al.*, 2004] Frederic Boussemart, Fred Hemery, Christophe Lecoutre and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the European Conference on Artificial Intelligence*, Valencia, Spain, pages 146–150, August 2004.

[Bowen and Likitvivanavong, 2004] James Bowen and Chavalit Likitvivanavong. Domain transmutation in constraint satisfaction problems. In *Proceedings of the American Association for Artificial Intelligence* pages 149–154, 2004.

[Cabon *et al.*, 1999] Bertrand Cabon, Simon De Givry, Lionel Lobjois, Thomas Schiex and Joost P. Warner. Radio link frequency assignment. *Constraints*, 4(1): 79–89, 1999.

[Lecoutre, 2007] Christophe Lecoutre, Benchmarks 2.0, XML representation of CSP instances. <http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/benchmarks.html>, 2007.

[Dilworth, 1950] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[Freuder, 1991] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the American Association for Artificial Intelligence*, pages 227–233, 1991.

[Frost and Dechter, 1995] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 572–578, 1995.

[Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference On Artificial Intelligence*, (1) pages 607–613, 1995.

[Hopcroft and Karp, 1973] John E. Hopcroft and Richard M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[Lal *et al.*, 2005] Lal, A., Berthe Y. Choueiry and Eugene C. Freuder. Neighborhood interchangeability and dynamic bundling for non-binary finite CSPs. In *Proceedings of the American Association for Artificial Intelligence*, pages 397–404, 2005.

[Taillard, 1993] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[Naanaa, 2007] Wady Naanaa. Directional interchangeability for enhancing CSP solving. CPAIOR 2007, Hentenryck and Wolsey Eds. LNCS 4510, 200–213, 2007.

[Van hemert, 2004] Jano I. Van Hemert. RandomCSP Version 1.8.0. <http://www.vanhemert.co.uk/randomcsp.html>, 2004.

[Xu and Li, 2000] Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000

[Zhang and Freuder, 2004] Yuanlin Zhang and Eugene C. Freuder. Conditional interchangeability and substitutability. In *Proceedings of Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, Toronto Canada, pages 95–100, 2004.

Symmetry-Breaking Formulas for Groups with Bounded Orbit Projections

Amitabha Roy*

Boston College

Computer Science Department

140 Commonwealth Avenue

Chestnut Hill, MA 02467

Abstract

This paper considers the problem of generating lex-leader symmetry-breaking formulas for permutation groups whose orbit constituents have bounded size. We prove that one can find an ordering of the permutation domain for which there is a polynomial size lex-leader formula. This generalizes the results of [Luks and Roy, 2004] which considered the problem for permutation groups with orbits of size 2. Our construction is based on writing a Boolean formula that captures the logic of Sims’s methods ([Sims, 1971]) to test membership in permutation groups, a classic result in computational group theory.

1 Introduction

This paper considers the problem of generating lex-leader formulas for a specific class of permutation groups. A lex-leader formula is an example of a *symmetry-breaking formula*, introduced by [Crawford *et al.*, 1996], which is true of exactly one member (in the case of lex-leader formulas, this member is the lexicographic leader) of each set of symmetrical points of the search space. A lex-leader formula when added to the input constraints of a constraint satisfaction problem makes it feasible to exploit symmetries inherent in a problem, in a manner that is agnostic to the search algorithm being used. This approach has been extended and applied to numerous problem domains (see e.g., [Shlyakhter, 2007], [Joslin and Roy, 1997], [Audemard *et al.*, 2007], [Backofen and Will, 1999], [Petrie and Smith, 2003] as a sample).

In [Luks and Roy, 2004], the authors consider the complexity of generating the lex-leader formula for permutation groups with orbits of size 2. These groups are necessarily abelian and are identifiable with subspaces over the 2-element field. They prove that there is an ordering of the permutation domain relative to which there is a lex-leader formula of size $O(n^3 \log^2 n)$. In this paper, we generalize their result by considering a wider class of groups. Specifically, we consider permutation

groups $G \leq \text{Sym}(\Omega)$, with orbits $\Delta_1, \Delta_2, \dots, \Delta_m$ such that the restriction of G in each orbit Δ_i is “small”, i.e. bounded by n^d from some constant d , where $|\Omega| = n$ (we say that G is a \mathcal{P}_d group). A special case of a \mathcal{P}_d group is one with bounded orbits, where each orbit Δ_i is of length $\leq c$ for some constant c . These groups are relevant in constraint satisfaction problems where the number of interchangeable variables are small.

Tools developed in computational group theory are used to find the group of symmetries of a search problem. In this paper, we leverage those same tools to bear on the problem of *generating* the symmetry-breaking formula. In particular, we use the concept of *strong generators* and *membership testing* in finite permutation groups developed by [Sims, 1971] in constructing small lex-leader formulas for a special class of groups.

We summarize our main results.

Theorem 1.1. (i) *Let $G \leq \text{Sym}(\Omega)$ be a \mathcal{P}_d group where $d > 1$. Then there is a canonical ordering of Ω such that there exists a lex-leader formula for G of size $O(dn^{2d+5} \log n)$, where $|\Omega| = n$. Furthermore, such an ordering can be found in polynomial time.*

(ii) *Let $G \leq \text{Sym}(\Omega)$ be a group whose largest orbit has size c (a fixed constant). Then there is a canonical reordering of Ω such that there is a lex-leader formula for G of size $O(n^6)$. Such a reordering can also be found in polynomial time.*

As we already observed, one can consider Theorem 1.1 as a generalization of the results of [Luks and Roy, 2004] (where the groups considered had orbits of size 2) to non-abelian groups. It is no surprise that solving the “bounded orbit” case in such generality in part (ii) of the above theorem gives us much worse results than [Luks and Roy, 2004]: where the authors obtained a formula of size $O(n^3 \log^2 n)$ and now we get a formula of size $O(n^6)$. Another point to note: we need only consider \mathcal{P}_d groups where $d > 1$ otherwise the entire group is polynomially bounded and an exhaustive enumeration would give all the group elements. Any lex-leader formula would work in that situation.

2 Definitions and Notations

*aroy@cs.bc.edu

Let G be a group. We write $H \leq G$ when H is a subgroup of G . If $H \leq G$, then a right transversal of H in G is a complete set of right coset representatives of H in G . The group consisting of all permutations of a set Ω is called the symmetric group, denoted by $\text{Sym}(\Omega)$. G is said to act on Ω if there is a homomorphism $\phi : G \rightarrow \text{Sym}(\Omega)$. Let $\omega \in \Omega$ and $g \in G$, then ω^g is the image of ω under $\phi(g)$. Also $\omega^G = \{\omega^g \mid g \in G\}$ is the orbit of G that contains ω . The point stabilizer of ω is the subgroup $G_\omega = \{g \in G \mid \omega^g = \omega\}$. The pointwise stabilizer of $\Delta \subset \Omega$ is $G_{(\Delta)} = \bigcap_{\delta \in \Delta} G_\delta$. When Ω is ordered as $\omega_1, \omega_2, \dots, \omega_n$, then $\Omega_i = \{\omega_1, \omega_2, \dots, \omega_i\}$ and $G_i = G_{(\Omega_i)}$. Let Δ be an orbit of G on Ω . For $g \in G$, g^Δ is the restriction of g on Δ . The *orbit constituent* $G^\Delta = \{g^\Delta \mid g \in G\}$ is the projection of G onto Δ . A \mathcal{P}_d group is a group $G \leq \text{Sym}(\Omega)$, where $|\Omega| = n$ such that the size of each orbit constituent is at most n^d .

Groups are input (and output) via generators. We write $G = \langle X \rangle$ when the set $X \subset G$ generates G . Subgroups of $\text{Sym}(\Omega)$ have succinct descriptions in terms of generators : they have generating sets of size $O(|\Omega|)$ [Dixon and Mortimer, 1996]. A very useful data structure for permutation groups is a strong generating set (SGS), first introduced by [Sims, 1970]. Given a chain

$$G = G^0 \geq G^1 \geq G^2 \geq \dots \geq G^m = 1$$

of subgroups of G , an SGS with respect to this chain is a set $T \subset G$ such that $\langle T \cap G^i \rangle = G^i$, for each i . We shall use the ‘‘point stabilizer’’ series as our subgroup chain, i.e., $G^i = G_i$ is the subgroup of G that fixes the first i points of Ω . Then an example of an SGS with respect to this chain is the set $R = \bigcup_{i=1}^m R_i$ where R_i is a complete right transversal of G_i in G_{i-1} . A permutation π is said to *sift* through this chain if it can be expressed as product $r_m r_{m-1} \dots r_1$ where $r_i \in R_i$.

Example: An example of an SGS for S_4 , the group of all permutations on 4 letters 1, 2, 3, 4 is the set $S = \{(1, 2, 3, 4), (2, 3, 4), (3, 4)\}$. Clearly $G = \langle S \rangle$ and $G_i = \langle S \cap G_i \rangle$.

We refer to any standard text (e.g [Hall, 1976], [Wielandt, 1964]) for basic facts about groups. For computation in permutation groups, we refer to [Luks, 1993].

A propositional variable can take on two values, true or false (we write 0 for false, 1 for true) . Let L be a set of propositional variables. Literals are variables in L or negations of variables in L . A clause is a disjunction of *distinct* literals in L . A theory is a conjunction of clauses. A truth assignment for a set of variables L is a function $X : L \rightarrow \{0, 1\}$. In the usual way, X extends by the semantics of propositional logic to a function on the set of theories over L and by abuse of notation, we will continue to denote the extended function by X . A truth assignment X of L is called a model of a theory T if $X(T) = 1$.

Let 2^Ω denote the set of functions from Ω to $\{0, 1\}$ (equivalently, 2^Ω is the set of all n -bit strings). Then G acts on 2^Ω via $X \mapsto {}^g X$ for $g \in G$, $X \in 2^\Omega$ where

$({}^g X)(i) = X(i^g)$. There is a natural lexicographic (dictionary) order on 2^Ω : $X < Y$ if $X \neq Y$ and $X(i) < Y(i)$ for the least i such that $X_i \neq Y_i$. The *lex-leader* in an orbit is the lexically *largest* string. A lex-leader formula for G , denoted by $\Lambda_{\text{nat}}(G)$, asserts that

$$\forall g \in G, X \geq {}^g X.$$

3 Lex-Leader Formulas for \mathcal{P}_d Groups

Let $G = \langle R \rangle \leq \text{Sym}(\Omega)$ be a permutation group on n points $\Omega = \{1, 2, \dots, n\}$ where the orbits of Ω are $\Delta_1, \Delta_2, \dots, \Delta_m$ where $\Delta_1 = \{1, 2, \dots, i_1\}$ and $\Delta_j = \{i_{j-1} + 1, i_{j-1} + 2, \dots, i_j\}$ (we reorder Ω if necessary to ensure this). Also assume that the size of the projection $|G^{\Delta_i}| \leq n^d$ ($= \gamma$ say), so that G is a \mathcal{P}_d group.

We assume that the set R is a special set of strong generators of G – these are the coset representatives R_i of G_i in G_{i-1} where G_i is the pointwise stabilizer of the first i points. That is $R = \bigcup_{i=1}^m R_i$ and R_i has size $\leq \delta$ where δ is the size of the largest orbit of G in Ω . Note that $|R| \in O(n^2)$.

We also assume that we have at hand a small set U of generators of G where $|U| = O(\log |G|) = O(\log(\gamma^m)) = O(m \log \gamma)$.

Let $X \in 2^\Omega$ and let X_{Δ_j} denote the projection of X onto Δ_j . The permutation g fixes a string X_{Δ_i} if $X(j) = X(j^g)$ for all $j \in \Delta_i$. For $X \in 2^{|\Omega|}$, let $G_i^X = \{g \in G \mid g \text{ fixes } X_{\Delta_1}, X_{\Delta_2}, \dots, X_{\Delta_i}\}$.

Let $\mathcal{C}(X)$ denote the chain of subgroups

$$\begin{aligned} G &\geq G_1^X \geq G_2^X \dots \geq G_m^X \geq (G_m^X \cap G_1) \\ &\geq (G_m^X \cap G_2) \dots \geq (G_m^X \cap G_{n-1} = 1) \end{aligned} \quad (1)$$

Let S_i for $1 \leq i \leq m$ (T_i for $1 \leq i \leq n-1$) be a complete set of coset representatives of G_i^X in G_{i-1}^X for $1 \leq i \leq m$ ($G_m^X \cap G_i$ in $G_m^X \cap G_{i-1}$ for $1 \leq i \leq n-1$, resp) where $G_0 = G$. Each S_i (T_i) has size at most γ (δ resp).

We now specify the individual pieces that we paste together to build a lex-leader formula.

Notation: We write \vec{x} as a shorthand for the ordered set of variables (x_1, x_2, \dots, x_m) .

Perm(π): $\text{Perm}(\pi)$ is a boolean formula of size $O(n^2)$ whose satisfying assignments correspond to permutations in $\text{Sym}(\Omega)$. A permutation π is represented in our formula by the set of variables $\pi(i, j)$, where $1 \leq i, j \leq n$.

Let x_1, x_2, \dots, x_n be n boolean variables. Let $E_1(x_1, x_2, \dots, x_n)$ be true exactly when only one of the n variables is set true. It is easy to write such a formula:

$$\left(\bigvee_{i=1}^n x_i \right) \wedge \left(\bigwedge_{i \neq j} (\neg x_i \vee \neg x_j) \right)$$

where the first clause specifies that one of the variables is set to true and the remaining clauses enforce that no two variables are simultaneously true. However this is itself a formula of size $O(n^2)$ and thus when applied to

$\text{Perm}(\pi)$ below will yield a formula of size $O(n^3)$. Using extra variables one can write a formula $\phi(\vec{x}, \vec{\mu})$ which is satisfiable iff $E_1(x_1, x_2, \dots, x_n)$ holds:

$$\begin{aligned} & \left(\bigvee_{1 \leq i \leq n} x_i \right) \wedge \\ & (\mu_1 = x_1) \wedge \\ & \bigwedge_{j=2}^n (\mu_j = \mu_{j-1} \oplus x_j) \wedge \\ & \bigwedge_{j=2}^n (\mu_{j-1} \wedge x_j \rightarrow \mu_j) \end{aligned}$$

The least index j such that $x_j = 1$ forces $\mu_j = 1$. Any subsequent k such that $x_k = 1$ forces $\mu_k = 0$ which violates the clause $\mu_{k-1} \wedge x_k \rightarrow \mu_k$. Since the first clause forces at least one x_i to be true, the restriction of a model for the above formula to (x_1, x_2, \dots, x_n) has to be a model of $E_1(x_1, x_2, \dots, x_n)$. We now define $\text{Perm}(\pi)$ to be the formula:

$$\begin{aligned} & \bigwedge_{1 \leq i \leq n} \phi(\pi(i, 1), \pi(i, 2), \dots, \pi(i, n), \vec{\mu}_i) \wedge \\ & \bigwedge_{1 \leq j \leq n} \phi(\pi(1, j), \pi(2, j), \dots, \pi(n, j), \vec{\tau}_j) \end{aligned}$$

where $\vec{\mu}_i, \vec{\tau}_j$ are set of auxiliary variables. The first line of clauses specifies that π is a function. The second line of clauses specifies that π is surjective (which implies that π is injective, and so π is a permutation).

Equal(y, x): We can express that permutations x, y are equal by the formula: $\bigwedge_i \bigwedge_j (x(i, j) = y(i, j))$.

Product(x, y, z): We assert that $x \cdot y = z$ where x, y, z are permutations by the following formula:

$$\bigwedge_{i,j,k} x(i, j) \wedge y(j, k) \rightarrow z(i, k).$$

Member(L, π): $\text{Member}(L, \pi)$ be a boolean formula which is satisfiable when $\pi \in L$. Let $|L| = l$ and let $L = \{\pi_1, \pi_2, \dots, \pi_l\}$, then the desired formula is:

$$\bigvee_{1 \leq i \leq l} \text{Equal}(\pi, \pi_i).$$

Sift($\pi, X, S \cup T$): The formula $\text{Sift}(\pi, X, S \cup T)$ is satisfiable iff π sifts through the chain of coset representatives $S = \cup S_i$ and $T = \cup T_i$ in $\mathcal{C}(X)$,

The formula for $\text{sift}(\pi, X, S, T)$ is:

$$\begin{aligned} & \bigwedge_{2 \leq i \leq m+n-1} \text{Product}(\pi_{i-1}, e_i, \pi_i) \wedge \\ & \text{Member}(S_1, \pi_1) \wedge \\ & \bigwedge_{2 \leq i \leq m} \text{Member}(S_i, e_i) \wedge \\ & \bigwedge_{m+1 \leq i \leq m+n-1} \text{Member}(T_{i-m}, e_i) \wedge \\ & \bigwedge_{1 \leq i \leq m+n-1} \text{Perm}(e_i) \wedge \\ & \text{Equal}(\pi, \pi_{m+n-1}) \end{aligned}$$

The size of sift is $O(m\gamma n^2 + mn^3 + \delta n^3)$. We can similarly define a formula $\text{sift-chain}(\pi, R)$ of size $O(n^4)$ which is satisfiable if the permutation π sifts through the strong generators R .

Now we write a formula $\text{StGen}(R, S, T, X)$ which expresses the fact that the set of generators $S \cup T$ are strong generators for G :

$$\bigwedge_{x \in S \cup T} \text{sift-chain}(x, R) \wedge \bigwedge_{u \in U} \bigwedge_{y \in S \cup T} [\text{Product}(y, u, z) \wedge \text{sift}(z, X, S, T)]$$

The first line ensures that $\langle S \cup T \rangle \leq G$. The second line ensures that $G \leq \langle S \cup T \rangle$ (observe that we took elements from U for sift in this line, because U is of potentially smaller size than R).

Clearly the size of the last line will dominate the size of StGen . The size is $O((\gamma m + \delta n)(m \log \gamma) \times (\text{size of sift})) = O((\gamma m + \delta n)(m \log \gamma)(m\gamma n^2 + mn^3 + \delta n^3))$.

We define a formula $\text{StringLeader}(X, \Delta, s)$ which expresses the fact that $X \geq ({}^s X)$ where the permutation s stabilizes Δ . For $i, j \in \Delta$, let $X_{i,j}$ be 1 iff $X(i) = X(j)$. To express $X \geq ({}^s X)$, we write a formula $\bigwedge_{i \in \Delta} \text{Geq}(i, s)$ where $\text{Geq}(i, s)$ needs to assert the following: if $X_{\{1,2,\dots,i-1\}} = ({}^s X)_{\{1,2,\dots,i-1\}}$ then $X(i) \geq ({}^s X)(i)$. We can assert that $X_{\{1,2,\dots,i-1\}} = ({}^s X)_{\{1,2,\dots,i-1\}}$ by the formula $\text{Stable}(i, s)$:

$$\left[\bigwedge_{j,k < i} (s(j, k) \rightarrow X_{j,k}) \right]$$

which requires that $X(k) = X(j)$ for every $j, k < i$ such that $j^s = k$. To express $X(i) \geq ({}^s X)(i)$, we distinguish the two cases: $i^s = s$ (i.e. $s(i, i) = 1$) and $i^s = l$ where $l > i$ (if $i^s < i$ then $\text{Geq}(i, s)$ is a tautology). If $i^s = l$ then $s(i, l) = 1$ and $X(i) \geq ({}^s X)(i)$ is equivalent to $X(i) \geq X(l)$, which can be written as $X(l) \rightarrow X(i)$. The formula $\text{Improve}(i, s)$ below asserts that $X(i) \geq ({}^s X)(i)$:

$$(s(i, i) = 1) \vee \bigvee_{l \in \Delta} \{s(i, l) \wedge (X(l) \rightarrow X(i))\}$$

Then $\text{Geq}(i, s)$ can now be written as:

$$\text{Stable}(i, s) \rightarrow \text{Improve}(i, s)$$

StringLeader is the following formula of size $O(n^4)$:

$$\bigwedge_{i,j \in \Delta} (X_{i,j} \leftrightarrow (X(i) = X(j))) \wedge \bigwedge_{i \in \Delta} \text{Geq}(i, s)$$

We now write a formula $\text{LL}(X, \Delta, S)$ which expresses the fact that $\forall s \in S, X_\Delta \geq ({}^s X)_\Delta$. We assume that Δ is stabilized by the set of permutations S . This formula of size $O(|S|n^4)$ is

$$\bigwedge_{s \in S} \text{StringLeader}(X, \Delta, s)$$

We can now write a formula $\text{LLG}(X, G)$ which expresses the fact that $X \geq {}^g X$ for all $g \in G$:

$$\text{StGen}(R, S, T, X) \wedge \bigwedge_{1 \leq i \leq m} \Lambda_{\text{nat}}(X, \Delta_i, S_i)$$

Hence the total length of the formula is $O((\gamma m + \delta n)(m \log \gamma)(m\gamma n^2 + mn^3 + \delta n^3) + \gamma mn^4)$. Theorem 1.1 now follows.

Remark: The test for “strong generators” that we encode in the lex-leader formula is expensive, contributing a factor of $O(n^{2d})$ to the size. It is conceivable that through cheaper tests for strong generators, e.g., Sims’s “verify” test (see [Gollan, 2001]), the factor can be reduced to $O(n^d)$.

4 Conclusions

We present a method to generate polynomial size lex-leader formulas for non-abelian groups whose orbit constituents are bounded, assuming that the permutation domain has a certain order (where points in each orbit appear together). An open question for these groups is whether one can remove the restriction on the order and still get a formula of a manageable size. This question is also relevant in the general situation: are there groups which have polynomial size lex-leader formulas in one order and exponential size formulas in another? This is open even for the groups with orbit size 2 considered by [Luks and Roy, 2004]. A limited form of reordering is shown not to affect the size of the lex-leader formula for these groups in [Roy, 2007].

References

- [Audemard *et al.*, 2007] G. Audemard, S. Jabbour, and L. Sais. Symmetry breaking in quantified boolean formulae. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2262–2267, 2007.
- [Backofen and Will, 1999] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *Proceedings of 5th International Conference on Principle and Practice of Constraint Programming (CP’99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 73–87. Springer-Verlag, 1999.
- [Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. M. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning (KR ’96)*, pages 148–159, 1996.
- [Dixon and Mortimer, 1996] J. D. Dixon and B. Mortimer. *Permutation groups*. Springer-Verlag, New York, 1996.
- [Gollan, 2001] H. Gollan. A new existence proof for Ly , the sporadic simple group of r. lyons. *J. Symbolic Computation*, 31:203–209, 2001.
- [Hall, 1976] M. Hall. *The Theory of Groups*. Chelsea Publishing Company, New York, second edition, 1976.
- [Joslin and Roy, 1997] D. Joslin and A. Roy. Exploiting symmetries in lifted csps. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 197–203. American Association for Artificial Intelligence(AAAI), 1997.
- [Luks and Roy, 2004] E. M. Luks and A. Roy. The complexity of symmetry-breaking formulas. *Ann. Math. Artif. Intell.*, 41(1):19–45, 2004.
- [Luks, 1993] E. M. Luks. Permutation groups and polynomial-time computation. In W. M. Kantor L. Finkelstein, editor, *Groups and Computation, Workshop on Groups and Computation*, volume 11 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 139–175, 1993.
- [Petrie and Smith, 2003] K. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.
- [Roy, 2007] Amitabha Roy. The combinatorics of symmetry breaking. Submitted, 2007.
- [Shlyakhter, 2007] Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Appl. Math.*, 155(12):1539–1548, 2007.
- [Sims, 1970] C. C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 169–183. Pergamon, Oxford, 1970.
- [Sims, 1971] C.C. Sims. Computation with permutation groups. In S. R. Petrick, editor, *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*, pages 23–28, Los Angeles, March 23–25 1971. ACM, New York.
- [Wielandt, 1964] H. Wielandt. *Finite Permutation Groups*. Academic Press, New York, 1964.

Breaking Value Symmetry*

Toby Walsh
NICTA and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

Symmetry is an important factor in solving many constraint satisfaction problems. One common type of symmetry is when we have symmetric values. We can eliminate such value symmetry either statically by adding constraints to prune symmetric solutions or dynamically by modifying the search procedure to avoid symmetric branches. We show that either method has computational limitations. With static methods, pruning all symmetric values is NP-hard in general. With dynamic methods, we can take exponential time on problems which static methods solve without search. Finally, we consider a common type of value symmetry, that due to interchangeable values, where polynomial methods have been proposed to break symmetries. We show that despite these theoretical limitations, the methods proposed to break the symmetries introduced by interchangeable values are both effective in theory and in practice.

1 Introduction

Many search problems contain symmetries. Symmetry occurs naturally in many problems (e.g. if we have identical machines to schedule, or identical jobs to process). Symmetry can also be introduced when we model a problem (e.g. if we name the elements in a set, we introduce the possibility of permuting their order). Unfortunately, symmetries increases the size of the search space. We must therefore try to eliminate symmetry or we will waste much time visiting symmetric solutions, as well as those parts of the search tree which are symmetric to already visited states. One common type of symmetry is when values are symmetric. For example, if we are assigning colors (values) to nodes (variables) in a graph coloring problem, we

can uniformly swap the names of the colors throughout a coloring. As a second example if we are assigning nurses (values) to shifts (variables) in a rostering problems, and two nurses have the same skills, we may be able to interchange them uniformly throughout the schedule. For a problem with value symmetries, all symmetric solutions can be eliminated in polynomial time [Roney-Dougal *et al.*, 2004; Puget, 2005]. However, as we show here, pruning *all* symmetric values is NP-hard in general. Nevertheless, methods that have been proposed, like those in [Law and Lee, 2004; Puget, 2005; Aloul, 2006], appear to be effective at dealing with common types of value symmetry.

2 Background

A constraint satisfaction problem consists of a set of variables, each with a domain of values, and a set of constraints specifying allowed combinations of values for given subsets of variables. Variables take one value from a given finite set. A solution is an assignment of values to variables satisfying the constraints. Symmetry occurs in many constraint satisfaction problems. A *value symmetry* is a permutation of the values that preserves solutions. More formally, a value symmetry is a bijective mapping, σ of the values such that if $X_1 = d_1, \dots, X_n = d_n$ is a solution then $X_1 = \sigma(d_1), \dots, X_n = \sigma(d_n)$ is also. For example, suppose we wish to assign colors (values) to nodes (variables) in a graph coloring problem. This problem has a value symmetry that permits us to interchange any two colors uniformly throughout a coloring. A *variable symmetry*, on the other hand, is a permutation of the variables that preserves solutions. More formally, a variable symmetry is a bijective mapping, θ of the indices of variables such that if $X_1 = d_1, \dots, X_n = d_n$ is a solution then $X_{\theta(1)} = d_1, \dots, X_{\theta(n)} = d_n$ is also. For example, suppose we wish to assign times (values) to exams (variables) in an exam scheduling problem and we have two exams taken by the same set of students. This variable symmetry permits us to interchange the two exams.

Symmetries are problematic as they increase the size of the search space. For instance, if we have m interchangeable values, symmetry increases the size of the search space by a factor of $m!$. The set of symmetries of a constraint satisfaction problem form a group under composition. In this work, we place no restrictions on the type of group. In particular, we are not restricted to products of the symmetry group, S_n .

*NICTA is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council DCITA and ARC through Backing Australia's Ability and the ICT Centre of Excellence program. Thanks to Chris Jefferson and Jean-Francois Puget for useful comments. A shorter version of this paper without experimental results appears in the Proceedings of the *13th International Conference on Principles and Practices of Constraint Programming (CP-2007)*.

However, we do assume that the symmetries of the constraint satisfaction problem are known in advance. For instance, if we are coloring a graph and use a straight forward model with variables for nodes and values for colors, we know that the values are fully interchangeable. A number of methods have been developed to find symmetries in a constraint satisfaction problem automatically.

Many constraint solvers explore the space of partial assignments enforcing some local consistency. We consider three local consistencies. Given a constraint C , a *support* is assignment to each variable of a value in its domain which satisfies C . A constraint is *generalized arc consistent (GAC)* iff for each variable, every value in its domain belongs to a support. A set of constraints is GAC iff each constraint is GAC. On binary constraints, GAC is simply called arc consistency (AC). A set of binary constraints is *singleton arc consistent (SAC)* iff we can assign every variable with each value in its domain and make the resulting problem arc consistent (AC). Finally, a set of binary constraint is *k-consistent* iff each $k - 1$ assignment can be consistently extended to a k th variable, and is *strongly k-consistent* iff it is j -consistency for all $j \leq k$. We will compare local consistency properties applied to sets of constraints, c_1 and c_2 which are logically equivalent. As in [Debruyne and Bessière, 1997], a local consistency property Φ on c_1 is as strong as Ψ on c_2 iff, given any domains, if Φ holds on c_1 then Ψ holds on c_2 ; Φ on c_1 is stronger than Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 but not vice versa; Φ on c_1 is equivalent to Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 and vice versa.

3 Static methods

One simple and common mechanism to deal with symmetry is to add constraints which eliminate symmetric solutions [Puget, 1993]. Suppose we have a set Σ of value symmetries. Based on [Crawford *et al.*, 1996], we can eliminate all symmetric solutions by posting a global constraint which ensures that the solution is ordered lexicographically before any of its symmetries. More precisely, we post the global constraint $\text{VALSYMBREAK}(\Sigma, [X_1, \dots, X_n])$ which ensures $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ for all $\sigma \in \Sigma$ where X_1 to X_n is a fixed ordering on the variables and $\sigma(X_i)$ represents the action of the symmetry σ on the value assigned to X_i . Unfortunately, pruning *all* values from such a symmetry breaking constraint is NP-hard.

Theorem 1 *Deciding if $\text{VALSYMBREAK}(\Sigma, [X_1, \dots, X_n])$ is GAC is NP-complete, even when $|\Sigma|$ is linearly bounded.*

Proof: Membership in NP follows by giving a support for every possible assignment. To prove it is NP-hard, we give a reduction from a 3-SAT problem in N Boolean variables and M clauses. We construct a CSP with $N + M + 1$ variables over $4N + 2$ possible values. The first $4N$ values partition into $2N$ interchangeable pairs. The values $4i - 3$ and $4i - 2$ are interchangeable, as are $4i - 1$ and $4i$ for $1 \leq i \leq N$. The values $4i - 3$ and $4i - 2$ represent the SAT variable x_i being true, whilst the values $4i - 1$ and $4i$ represent the SAT variable x_i being false. The final two values, $4N + 1$ and $4N + 2$ are not interchangeable. The first N CSP variables represent a “truth assignment”. We have $X_i \in \{4i - 3, 4i - 2, 4i - 1, 4i\}$

for $1 \leq i \leq N$. The next M CSP variables ensure at least one literal in each clause is true. For example, if the i th clause is $x_j \vee \neg x_k \vee x_l$, then the domain of X_{N+i} is $\{4j - 3, 4j - 2, 4k - 1, 4k, 4l - 3, 4l - 2\}$. The final variable X_{N+M+1} is a “switch” and has the domain $\{4N + 1, 4N + 2\}$. Note that if a value is in the domain of a variable then so is every symmetry of this value.

We have two sets of constraints. First, we have the binary constraints $\text{odd}(X_{N+M+1}) \rightarrow \text{odd}(X_i)$ for $1 \leq i \leq N$ and $\text{odd}(X_{N+M+1}) \rightarrow \text{even}(X_{N+j})$ for $1 \leq j \leq M$. Second, we have the constraints $\text{odd}(X_{N+M+1}) \rightarrow \text{PHP}(N, N + 1)$ and $\text{even}(X_{N+M+1}) \rightarrow \text{PHP}(N, N)$ where $\text{PHP}(i, j)$ is a pigeonhole constraint which holds iff the variables X_1 to X_i take j distinct values. Note that $\text{PHP}(N, N + 1)$ is unsatisfiable and that $\text{PHP}(N, N)$ is satisfiable. Thus, the constructed CSP is unsatisfiable if $X_{N+M+1} = 4N + 1$ and satisfiable if $X_{N+M+1} = 4N + 2$. Note that if we take any solution of the CSP and permute any of the interchangeable values, we still have a solution. Thus, if Σ is the set of symmetries induced by these interchangeable values, it is permissible to add $\text{VALSYMBREAK}(\Sigma, [X_1, \dots, X_n])$ to this constraint satisfaction problem to eliminate value symmetry.

Suppose our branching heuristic sets the switch variable X_{N+M+1} to $4N + 1$. Enforcing AC on the binary constraints prunes the domains of X_i to $\{4i - 3, 4i - 1\}$ for $1 \leq i \leq N$. Similarly, the domain of X_{N+i} is reduced to $\{4j - 2, 4k, 4l - 2\}$. Consider now finding a support for VALSYMBREAK given this particular subproblem. Now, $\text{VALSYMBREAK}(\Sigma, [X_1, \dots, X_n])$ ensures that the interchangeable values first appear in order. In particular, X_{N+i} can only take the value $4j - 2$ if X_j had previously been assigned $4j - 3$. In other words, X_{N+i} can only take the value $4j - 2$ if x_j is set to true in the “truth assignment”. Similarly, X_{N+i} can only take the value $4k$ if X_k had previously been assigned $4k - 1$. In other words, X_{N+i} can only take the value $4k$ if x_k is set to false in the “truth assignment”. Finally, X_{N+i} can only take the value $4l - 2$ if X_j had previously been assigned $4l - 3$. In other words, X_{N+i} can only take the value $4l - 2$ if x_l is set to true in the “truth assignment”. Thus, at least one of the literals in the i th clause must have been set to true in the “truth assignment”. Hence, there is a support for VALSYMBREAK iff the original 3-SAT problem is satisfiable. By Theorem 3, $|\Sigma|$ can be linearly bound. \square

This is a somewhat surprising result. Whilst it is polynomial to eliminate all symmetric solutions either statically [Puget, 2005] or dynamically [Roney-Dougal *et al.*, 2004], it is NP-hard to lookahead and prune all symmetric values. Equivalently, whilst we can avoid visiting symmetric leaves of the search tree in polynomial time, avoiding symmetric subtrees is NP-hard.

4 Dynamic methods

An alternative to static methods which add constraints to eliminate symmetric solutions are dynamic methods which modify the search procedure to ignore symmetric branches. For example, with value symmetries, the GE-tree method dynamically eliminates all symmetric solutions from a back-

tracking search procedure in $O(n^4 \log(n))$ time [Roney-Dougal *et al.*, 2004]. However, as we show now, such dynamic methods may not prune *all* the symmetric values which static methods can do. Suppose we are at a particular node in the search tree explored by the GE-tree method. Consider the current and all past variables seen so far. The GE-tree method can be seen as performing forward checking on a static symmetry breaking constraint over this set of variables. This prunes symmetric assignments from the domain of the *next* branching variable. Unlike static methods, the GE-tree method does not prune *deeper* variables. By comparison, static symmetry breaking constraints can prune deeper variables, resulting in interactions with the problem constraints and additional domain prunings. For this reason, static symmetry breaking methods can solve certain problems exponentially quicker than dynamic methods.

Theorem 2 *There exists a model of the pigeonhole problem in n variables and $n + 1$ interchangeable values such that, given any variable and value ordering, the GE-tree method explores $O(2^n)$ branches, but which static symmetry breaking methods can solve in just $O(n^2)$ time.*

Proof: The $n + 1$ constraints in the CSP are $\bigvee_{i=1}^n X_i = j$ for $1 \leq j \leq n + 1$, and the domains are $X_i \in \{1, \dots, n + 1\}$ for $1 \leq i \leq n$. The problem is unsatisfiable by a simple pigeonhole argument. Any of the static methods for breaking value symmetry presented later in this paper will prune $n + 1$ from every domain in $O(n^2)$ time. Enforcing GAC on the constraint $\bigvee_{i=1}^n X_i = n + 1$ then proves unsatisfiability. On the other hand, the GE-tree method irrespective of the variable and value ordering, will only terminate each branch when $n - 1$ variables have been assigned (and the last variable is forced). A simple calculation shows that the size of the GE-tree more than doubles as we increase n by 1. Hence we will visit $O(2^n)$ branches before declaring the problem is unsatisfiable. \square

This theoretical result supports the experimental results in [Puget, 2005] showing that static methods for breaking value symmetry can outperform dynamic methods. Given the intractability of pruning all symmetric values in general, we focus in the rest of the paper on a common and useful type of value symmetry where polynomial symmetry breaking methods have been proposed: we will suppose that values are ordered into partitions, and values within each partition are uniformly interchangeable.

5 Generator symmetries

One way to propagate VALSYMBREAK is to decompose it into individual lexicographical ordering constraints, $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ and use one of the propagators proposed in [Puget, 2006] or [Walsh, 2006a]. Even if we ignore the fact that such a decomposition may hinder propagation (see Theorem 2 in [Walsh, 2006a]), we have to cope with Σ , the set of symmetries being exponentially large in general. For instance, if we have m interchangeable values, then Σ contains $m!$ symmetries. To deal with large number of symmetries, Aloul *et al.* suggest breaking only those symmetries corresponding to generators of the group

[Aloul *et al.*, 2002]. Consider the generators which interchange adjacent values within each partition. If the m values partition into k classes of interchangeable values, there are just $m - k$ such generators. We prove here that breaking *just* these generator symmetries eliminates *all* symmetric solutions. Thus, we have identified a special class of symmetries where using just generators is complete.

Theorem 3 *If Σ is a set of symmetries induced by interchangeable values, and Σ_g is the set of generators corresponding to interchanging adjacent values then posting VALSYMBREAK($\Sigma_g, [X_1, \dots, X_n]$) eliminates all symmetric solutions.*

Proof: Assume VALSYMBREAK($\Sigma_g, [X_1, \dots, X_n]$). Consider any two interchangeable values, j and k where $j < k$. Let $\sigma_j \in \Sigma_g$ be the symmetry which swaps just j with $j + 1$. To ensure $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma_j(X_1), \dots, \sigma_j(X_n)]$, j must occur before $j + 1$ in X_1 to X_n . By transitivity, j therefore occurs before k . Thus, for the symmetry σ' which swaps just j with k , $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma'(X_1), \dots, \sigma'(X_n)]$. Consider now any symmetry $\sigma \in \Sigma$. The proof proceeds by contradiction. Suppose $[X_1, \dots, X_n] >_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$. Then there exists some j with $X_j > \sigma(X_j)$ and $X_i = \sigma(X_i)$ for all $i < j$. Consider the symmetry σ' which swaps just X_j with $\sigma(X_j)$. As argued before, $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma'(X_1), \dots, \sigma'(X_n)]$. But this contradicts $[X_1, \dots, X_n] >_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ as σ and σ' act identically on the first j variables in X_1 to X_n . Hence, $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$. \square

Not surprisingly, reducing the number of symmetry breaking constraints to linear is not without consequence. Whilst restricting symmetry breaking to just these generators eliminates all symmetric solutions, we may not necessarily prune all symmetric values. Equivalently, whilst restricting symmetry breaking to just these generators eliminates all symmetric leaves of the search tree, we may not necessarily avoid all symmetric subtrees.

Theorem 4 *If Σ is a set of symmetries induced by interchangeable values, and Σ_g is the set of generators corresponding to interchanging adjacent values then GAC on VALSYMBREAK($\Sigma, [X_1, \dots, X_n]$) is stronger than GAC on $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ for all $\sigma \in \Sigma_g$.*

Proof: Clearly it is at least as strong. To show it is stronger, suppose all values are interchangeable with each other. Consider $X_1 = 1$, $X_2 \in \{1, 2\}$, $X_3 \in \{1, 3\}$, $X_4 \in \{1, 4\}$ and $X_5 = 5$. Then enforcing GAC on VALSYMBREAK($\Sigma, [X_1, \dots, X_5]$) prunes 1 from X_2 , X_3 and X_4 . However, $[X_1, \dots, X_5] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_5)]$ is GAC for all $\sigma \in \Sigma_g$. \square

It is not hard to see that there are other sets of generators for the symmetries induced by interchangeable values which do not necessarily eliminate all symmetric solutions (e.g. with the generators which interchange the value 1 with any i , we do not eliminate either the assignment $X_1 = 1$, $X_2 = 2$ or the symmetric assignment $X_1 = 1$, $X_2 = 3$). Thus we have shown that the choice of generators is important.

6 Puget's decomposition

With value symmetries, a second method that eliminates all symmetric solutions is a decomposition due to [Puget, 2005]. Consider a surjection problem (where each value is used at least once) with interchangeable values. We can channel into dual variables, Z_j which record the first index using the value j by posting the binary constraints: $X_i = j \rightarrow Z_j \leq i$ and $Z_j = i \rightarrow X_i = j$ for all $1 \leq i \leq n$, $1 \leq j \leq m$. We can then eliminate all symmetric solutions by insisting that interchangeable values *first* occur in some given order. That is, we place strict ordering constraints on the Z_k within each class of interchangeable values. Puget notes that any problem can be made into a surjection by introducing m additional new variables, X_{n+1} to X_{n+m} where $X_{n+i} = i$. These variables ensure that each value is used at least once. In fact, we don't need additional variables. It is enough to ensure that each Z_j has a dummy value, which means that j is not assigned, and to order (dummy) values appropriately. Unfortunately, Puget's decomposition into binary constraints hinders propagation.

Theorem 5 *If Σ is a set of symmetries induced by interchangeable values, then GAC on VALSYMBREAK($\Sigma, [X_1, \dots, X_n]$) is stronger than AC on Puget's decomposition into binary constraints.*

Proof: It is clearly at least as strong. To show it is stronger, suppose all values are interchangeable with each other. Consider $X_1 = 1$, $X_2 \in \{1, 2\}$, $X_3 \in \{1, 3\}$, $X_4 \in \{3, 4\}$, $X_5 = 2$, $X_6 = 3$, $X_7 = 4$, $Z_1 = 1$, $Z_2 \in \{2, 5\}$, $Z_3 \in \{3, 4, 6\}$, and $Z_4 \in \{4, 7\}$. Then all Puget's symmetry breaking constraints are AC. However, enforcing GAC on VALSYMBREAK($\Sigma, [X_1, \dots, X_5]$) will prune 1 from X_2 . \square

If all values are interchangeable with each other, we only need to enforce a slightly stronger level of local consistency to prune all symmetric values. More precisely, enforcing singleton arc consistency on Puget's binary decomposition will prune all symmetric values.

Theorem 6 *If all values are interchangeable and Σ is the set of symmetries induced by this then GAC on VALSYMBREAK($\Sigma, [X_1, \dots, X_n]$) is equivalent to SAC on Puget's decomposition into binary constraints.*

Proof: Suppose Puget's encoding is AC. We will show that there is at least one support for VALSYMBREAK. We assign Z_1 to Z_m in turn, giving each the smallest remaining value in their domain, and enforcing AC on the encoding after each assignment. This will construct a support without the need for backtracking. At each choice point, we ensure that a new value is used as soon as possible, thus giving us the most freedom to use values in the future. Suppose now that Puget's encoding is SAC. Then, by the definition of SAC, we can assign any variable with any value in its domains and be sure that the problem can be made AC without a domain wipeout. But if the problem can be made AC, it has support. Thus every value in every domain has support. Hence enforcing SAC on Puget's decomposition ensures that VALSYMBREAK is GAC. \square

We might wonder if singleton arc-consistency is enough for arbitrary value symmetries. That is, does enforcing SAC on Puget's encoding prune all symmetric values? We can

prove that no fixed level of local consistency is sufficient. Given the intractability of pruning all symmetric values in general, this result is not surprising.

Theorem 7 *For any given k , there exists a value symmetry and domains for which Puget's encoding is strongly k -consistent but is not $k+1$ -consistent.*

Proof: We construct a CSP problem with $2k+1$ variables over $2(k+1)$ possible values. The $2(k+1)$ values partition into $k+1$ pairs which are interchangeable. More precisely, the values i and $k+1+i$ are interchangeable for $1 \leq i \leq k+1$. The first k variables of the CSP have $k+1$ values between them (hence, one value is not taken). More precisely, $X_i \in \{i, i+1\}$ for $1 \leq i \leq k$. The remaining $k+1$ variables then take the other $k+1$ values. More precisely, $X_{k+i} = k+1+i$ for $1 \leq i \leq k+1$. The values 1 to $k+1$ need to be used by the first k variables, X_1 to X_k so that the last $k+1$ variables, X_{k+1} to $X_{2(k+1)}$ can use the values $k+2$ to $2(k+1)$. But this is impossible by a pigeonhole argument. Puget's encoding of this is strongly k -consistent. since any assignment of $k-1$ or less variables can be extended to an additional variable. On the other hand, enforcing $k+1$ -consistency will discover that the CSP has no solution. \square

Finally, we compare this method with the previous method based on breaking the symmetries corresponding to the generators which interchange adjacent values.

Theorem 8 *If Σ is a set of symmetries induced by interchangeable values, and Σ_g is the set of generators interchanging adjacent values then AC on Puget's decomposition for Σ is stronger than GAC on $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ for all $\sigma \in \Sigma_g$.*

Proof: Suppose Puget's decomposition is AC. Consider the symmetry σ which interchanges j with $j+1$. Consider any variable and any value in its domain. We show how to construct a support for this assignment. We assign every other variable with j if it is in its domain, otherwise any value other than $j+1$ and failing this, $j+1$. Suppose this is not a support for $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$. This means that in the sequence from X_1 to X_n , we had to use the value $j+1$ before the value j . However, as Puget's decomposition is AC, there is a value in the domain of Z_j smaller than Z_{j+1} . This contradicts $j+1$ having to be used before j . Hence, this must be a support. Thus $[X_1, \dots, X_n] \leq_{\text{lex}} [\sigma(X_1), \dots, \sigma(X_n)]$ is GAC for all $\sigma \in \Sigma_g$. To show that AC on Puget's decomposition is stronger consider again the example used in the proof of Theorem 4. The lexicographical ordering constraint for each generator $\sigma \in \Sigma_g$ is GAC without any domain pruning. However, enforcing AC on Puget's decomposition prunes 1 from X_2 , X_3 and X_4 . \square

7 Value precedence

A third method to eliminate all symmetric solutions induced by interchangeable values uses the global *precedence* constraint [Law and Lee, 2004]. The constraint PRECEDENCE($[X_1, \dots, X_n]$) holds iff $\min\{i \mid X_i = j \vee i = n+1\} < \min\{i \mid X_i = k \vee i = n+2\}$ for all $j < k$. That is, the first time we use j is before the first time we use k for all $j < k$. Posting such a precedence constraint

eliminates all symmetric solutions due to interchangeable values. In [Walsh, 2006b], a GAC propagator for such a precedence constraint is proposed which takes $O(nm)$ time. It is not hard to show that $\text{PRECEDENCE}([X_1, \dots, X_n])$ is equivalent to $\text{VALSYMBREAK}(\Sigma, [X_1, \dots, X_n])$. Hence, enforcing GAC on such a precedence constraint prunes all symmetric values in polynomial time.

Precedence constraints can also be defined when values partition into more than one interchangeable class. We just insist that the values within each class first occur in a fixed order. In [Walsh, 2006b], a propagator for such a precedence constraint is proposed which takes $O(n \prod_i m_i)$ time where m_i is the size of the i th class of interchangeable values. Whilst this prunes all symmetric values, it is only polynomial if we can bound the number of classes of interchangeable values. This complexity is now not surprising. We have shown that pruning all symmetric values is NP-hard when the number of classes of interchangeable values is unbounded.

8 Breaking variable and value symmetry

Variable symmetries can also be broken statically by posting constraints. Following [Crawford *et al.*, 1996], we can eliminate all symmetric solutions with a global constraint which ensures that the solution is ordered lexicographically before any of the symmetries of the solution. More precisely, give a set of variable symmetries Θ , we can eliminate all symmetric solutions with a global constraint which ensures $[X_1, \dots, X_n] \leq_{\text{lex}} [X_{\theta(1)}, \dots, X_{\theta(n)}]$ for all $\theta \in \Theta$ where X_1 to X_n is a fixed ordering on the variables. Pruning *all* values from such a symmetry breaking constraint is NP-hard in general [Crawford *et al.*, 1996; Bessiere *et al.*, 2004]. Consider, for example, a model of the rehearsal problem (prob039 in CSPLib) where we have a variable for each time slot whose value is the piece to rehearse. This model has a variable symmetry as we can invert any rehearsal scheduling without violating any constraints. This is equivalent to swapping X_i with X_{n-i+1} . We eliminate this symmetry by posting the constraint: $[X_1, \dots, X_n] \leq_{\text{lex}} [X_n, \dots, X_1]$. Such variable symmetry breaking constraints are consistent with the value symmetry breaking constraints discussed here. We must, however, ensure that all are based on the same fixed variable ordering. Whilst we can consistently post both variable and value symmetry breaking constraints, this may not eliminate all symmetric solutions resulting from the interaction of variable and value symmetry (see, for example, Theorem 3 in [Walsh, 2006a]).

9 Experimental results

We now compare these value symmetry breaking methods experimentally. Puget has shown that his static symmetry breaking method significantly outperforms the dynamic GE-tree method. We therefore look at just the three static methods.

Generator symmetries: we post lexicographical ordering constraints for the generators of the symmetry group that interchange adjacent values and enforce GAC using a linear time propagator [Walsh, 2006a].

Puget’s decomposition: we enforce AC using the solver’s built-in propagators.

Value precedence: we post a single global value precedence constraint and enforce GAC using a linear time propagator [Walsh, 2006b].

Graph coloring

As in an earlier study of interchangeable values [Law and Lee, 2006], we experimented with graph coloring problems from the DIMACS benchmark suite. We use a straight forward model with one decision variable for each vertex. The values represent colors and are completely interchangeable. We coded the graph coloring problems in BProlog and ran them on a PowerPC 1GHz G4 processor with 1.25 GB RAM. Table 1 gives results. There is little to choose between the different symmetry breaking methods. However, the generator symmetry method is never fastest on any problem so can perhaps be excluded on these grounds. Although the differences between Puget’s method and the global value precedence constraint are slight, breaking symmetry using a global value precedence constraint solves the most problems within the 2 hour time-out. It also often has the fewest branches or the shortest runtime.

We will make some additional observations about these results. On the fifth problem DSJC125.1, we actually find the optimal coloring faster without any symmetry breaking constraints. We conjecture that symmetry breaking in this case is conflicting with the branching heuristic. However, this problem is exceptional and symmetry breaking constraints are usually essential to be able to prove optimality within the 2 hour time-out. On the final problem ash331GPIA, we also find the optimal coloring and prove optimality quicker without symmetry breaking constraints. However, when we have an additional color available, the symmetry breaking constraints help keep the branching heuristic on track.

Note that as the number of colors (and thus the number of interchangeable values) increases, we see greater benefits using the global value precedence constraint compared to Puget’s method or using generator symmetries. With the largest number of interchangeable values, the global value precedence constraint can be up to two orders of magnitude faster at reaching its fixed point than the other symmetry breaking methods. Finally, whilst the different symmetry breaking methods in theory prune the search tree differently, this is only seen on a few problem instances.

9.1 n by n queens

As in the first experiment on value symmetry breaking in [Puget, 2005], we used a simple model of the n by n queens problem. The aim is to color each square in a n by n chessboard with one of n colors so that no line (row, column or diagonal) has the same color twice. This is equivalent to finding n non-intersecting solutions to the n -queens problems. This is a difficult combinatorial problem. The existence of a solution for $n = 12$ was open until recently. We model this with n^2 variables, each with n possible values, and an all different constraint along each line. The model has 8 variable symmetries corresponding to the rotations and reflections of the chessboard. We break these by posting the ordering constraints: $X_1 < X_n$, $X_1 < X_{n^2-n+1}$, $X_1 < X_{n^2}$ and $X_2 < X_{n+1}$. The model also has $n!$ value symmetries as

problem vertices/ edges	colors k	value symmetry breaking							
		none		generator symmetries		Puget's method		value precedence	
		b	t	b	t	b	t	b	t
myciel5 47/236	4	6,264	0.31	261	0.08	261	0.04	261	0.06
	5	43,001,880	1,334.71	286,994	53.66	286,994	16.41	286,994	21.45
	6*	1	0.01	1	0.02	1	0.02	1	0.01
GEOM50_5a 50/238	7			5,504	0.89	3,047	0.83	4,928	0.33
	8			61,430	7.81	61,430	10.24	65,398	3.11
	9*	1	0.01	25	0.03	257	0.03	1	0.01
R50_5gb8 50/612	8			3,047	1.57	3,047	0.83	3,061	0.41
	9			92,302	48.59	92,202	19.98	95,685	12.11
	10*	2	0.01	1,901	0.89	1,897	0.43	2,139	0.26
queen8_8 64/1456	7	5,040	0.31	0	0.04	0	0.04	0	0.03
	8			149,573	159.09	149,573	25.89	149,573	15.87
	9*	12,674	0.71	12,674	8.31	12,674	1.83	12,674	1.40
DSJC125.1 125/736	4	1,463	0.15	3,093	0.92	3,093	0.84	3,094	0.89
	5	79,990	5.69	401,777	290.58	401,777	77.40	408,691	110.30
	6*	1	0.03	1	0.06	1	0.07	3	0.04
miles250 128/774	3	6	0.03	11	0.05	11	0.06	11	0.05
	4	26,542,104	1,674.00	770	0.17	770	0.44	770	0.12
	5			73,666	11.68	73,666	34.24	73,666	7.20
mulsol.i.1 197/3925	23			5,040	14.66	5,040	8.58	5,040	1.52
	24			40,320	133.15	40,320	77.73	40,320	14.90
	25					7,869,767	6,971.80	362,880	129.76
	26							3,628,800	886.12
school1 385/19095	13			762	209.83	762	13.82	762	50.46
	14			2,358	1,079.66	2,358	27.14	2,358	56.71
	15*			130	29.95	130	3.81	105	3.16
fpsol2.i.2 451/8691	22			720	4.14	720	71.01	720	0.77
	23			5,040	20.00	5,040	292.47	5,040	2.23
	24			40,320	133.55	40,320	1,517.75	40,320	14.38
	25							362,880	126.86
ash331GPIA 662/4185	3	24	0.43	4	0.47	4	0.77	4	0.49
	4*	1,563	1.01	67,952	26.90	67,952	18.04	67,952	16.71
	5	125,240	47.15	3,815	3.51	3,815	2.86	3,815	2.06
TOTALS									
Instances solved/total		15/32		29/32		30/32		32/32	
Best method/instances		9/32		20/32		22/32		23/32	
Average (solved by all)		4,585,083 204.30		51,952 25.71		51,967 7.95		52,427 10.26	
Average (all solved)		4,585,083 204.30		43,430 77.25		304,231 306.52		175,873 46.24	

Table 1: Graph coloring problems: **branches** and **time** in seconds to find a proper coloring or prove one does not exist. Blank entries are problems not solved in 2 hours. Colorings marked with * are optimal, “Best method/instances” is the fraction of instances on which the method is best, “(solved by all)” is the average over the 15 instances all methods solve before the time-out, whilst “(all solved)” is the average over all instances solved by the given method.

all colors are interchangeable. We break these with one of the three methods mentioned above.

Results are given in Table 2. To look in more detail at the propagation, we ran this experiment using SICSTUS 3.12.7. This also provides statistics on the number of domain prunings performed in propagating constraints. For $n = 5$ and 7, there is a unique solution up to symmetry. For $n = 6$ and 8, there are no solutions. Despite the theoretical differences between the three static symmetric breaking methods identified in Theorems 4 and 5, we see no difference in the size of the search trees explored on these n by n queens problems. The specialized propagator for value precedence is, however, two or so times faster than Puget’s method which itself is two or so time faster than the generator symmetry method. The more detailed statistics suggest that the value precedence constraint often reaches the same fixed point as Puget’s method but with fewer domain prunings.

10 Related work

Puget proved that symmetric solutions can be eliminated by the addition of suitable constraints [Puget, 1993]. Crawford *et al.* presented the first general method for constructing variable symmetry breaking constraints [Crawford *et al.*, 1996]. To deal with large number of symmetries, Aloul *et al.* suggest breaking only those symmetries corresponding to generators of the group [Aloul *et al.*, 2002]. Aloul *et al.* also improved the runtime of this method by reducing the size of a CNF encoding of such a symmetry breaking constraint from quadratic to linear [Aloul *et al.*, 2003]. Petrie and Smith adapted this symmetry breaking method to value symmetries by posting a suitable lexicographical ordering constraint for each value symmetry [Petrie and Smith, 2003]. Puget and Walsh independently proposed propagators for such value symmetry breaking constraints [Puget, 2006; Walsh, 2006a]. To deal with the exponential number of such value symmetry breaking constraints, Puget proposed a global propagator which does forward checking in polynomial time [Puget, 2006].

To eliminate symmetric solutions due to interchangeable values, Law and Lee formally defined value precedence for finite domain and set variables and proposed a specialized propagator for a pair of interchangeable values [Law and Lee, 2004]. Walsh extended this to a propagator for any number of interchangeable values [Walsh, 2006b]. Value precedence enforces the so-called “lowest index color ordering” which eliminates value symmetry in graph coloring problems [Aloul, 2006]. Finally, an alternative way to break value symmetry statically is to convert it into a variable symmetry by channelling into a dual viewpoint and using lexicographical ordering constraints on this dual view [Flener *et al.*, 2002; Law and Lee, 2006].

A number of dynamic methods have been proposed to deal with value symmetry. Van Hentenryck *et al.* gave a labelling schema for eliminating all symmetric solutions due to interchangeable values [Hentenryck *et al.*, 2003]. Inspired by this method, Roney-Dougal *et al.* gave a polynomial method to construct a GE-tree, a search tree without value symmetry [Roney-Dougal *et al.*, 2004]. Finally, Sellmann and van Hen-

tenryck gave a $O(nd^{3.5} + n^2d^2)$ dominance detection algorithm for eliminating all symmetric solutions when both variables and values are interchangeable [Sellmann and Hentenryck, 2005].

11 Conclusion

Value symmetries can be broken either statically (by adding constraints to prune symmetric solutions) or dynamically (by modifying the search procedure to avoid symmetric branches). We have shown that both approaches have computational limitations. With static methods, we can eliminate all symmetric solutions in polynomial time but pruning all symmetric values is NP-hard in general (or equivalently, we can avoid visiting symmetric leaves of the search tree in polynomial time but avoiding symmetric subtrees is NP-hard). With dynamic methods, we typically only perform forward checking and can take exponential time on problems which static methods solve without search. We have studied a common type of value symmetry where values are interchangeable and static methods are polynomial. We considered three different symmetry breaking constraints for interchangeable values: lexicographical ordering constraints based on generators of the symmetry group, symmetry breaking constraints proposed by Puget [Puget, 2005], and a specialized precedence constraint [Law and Lee, 2004; Walsh, 2006b]. We have shown that despite theoretical differences in their ability to prune symmetric values, the three methods appear to explore very similar search spaces in practice. However, the specialized propagator offers runtime savings by reaching its fixed point quicker. There are many open questions raised by this research. For example, are there other types of symmetry where all symmetric values can be pruned tractably? Are there other types of symmetry where it is enough to use just generators?

References

- [Aloul *et al.*, 2002] F.A. Aloul, A. Ramani, I. Markov, and K.A. Sakallah. Solving difficult SAT instances in the presence of symmetries. In *Proceedings of the Design Automation Conference*, pages 731–736, 2002.
- [Aloul *et al.*, 2003] F.A. Aloul, K.A. Sakallah, and I.L. Markov. Efficient symmetry breaking for Boolean satisfiability. In *Proceedings of the 18th International Joint Conference on AI*, pages 271–276. International Joint Conference on Artificial Intelligence, 2003.
- [Aloul, 2006] F.A. Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26:289–322, 2006.
- [Bessiere *et al.*, 2004] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proceedings of the 19th National Conference on AI*. American Association for Artificial Intelligence, 2004.
- [Crawford *et al.*, 1996] J. Crawford, G. Luks, M. Ginsberg, and A. Roy. Symmetry breaking predicates for search problems. In *Proceedings of the 5th International Conference on Knowledge Representation and Reasoning (KR ’96)*, pages 148–159, 1996.

problem <i>n</i>	none				generator symmetries				value symmetry breaking				Puget's method				value precedence			
	<i>c</i>	<i>b</i>	<i>p</i>	<i>t</i>	<i>c</i>	<i>b</i>	<i>p</i>	<i>t</i>	<i>c</i>	<i>b</i>	<i>p</i>	<i>t</i>	<i>c</i>	<i>b</i>	<i>p</i>	<i>t</i>	<i>c</i>	<i>b</i>	<i>p</i>	<i>t</i>
4	22	7	219	0.01	444	1	628	0.02	399	1	591	0.02	156	1	317	0.00				
5	28	59	2,781	0.02	928	2	1,651	0.02	782	2	1,251	0.03	253	2	601	0.02				
6	34	3949	200,395	0.65	1,654	30	9,624	0.07	1,335	30	7,245	0.07	358	30	3,611	0.02				
7	40	882,813	53,528,368	170.75	2,686	838	278,678	1.20	2,104	838	193,901	0.67	481	838	130,695	0.28				
8					4,078	148,564	54,091,553	238.52	3,125	148,564	36,865,615	119.83	622	148,564	19,899,573	50.12				
9																				

Table 2: n by n queens problem: constraints posted, branches, domain prunings and time to find all solutions in secs using the fail first heuristic. Blank entries are problems not solved in 1 hour. Results are similar to find first solution.

- [Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th IJCAI*, pages 412–417. International Joint Conference on Artificial Intelligence, 1997.
- [Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetry in matrix models. In *8th International Conference on Principles and Practices of Constraint Programming (CP-2002)*. Springer, 2002.
- [Hentenryck *et al.*, 2003] P. Van Hentenryck, M. Agren, P. Flener, and J. Pearson. Tractable symmetry breaking for CSPs with interchangeable values. In *Proceedings of the 18th International Conference on AI*. International Joint Conference on Artificial Intelligence, 2003.
- [Law and Lee, 2004] Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pages 362–376. Springer, 2004.
- [Law and Lee, 2006] Y.C. Law and J.M.H. Lee. Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction. *Constraints*, 11(2–3):221–267, 2006.
- [Petrie and Smith, 2003] Karen E. Petrie and Barbara M. Smith. Symmetry Breaking in Graceful Graphs. Technical Report APES-56a-2003, APES Research Group, June 2003.
- [Puget, 1993] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In J. Komorowski and Z.W. Ras, editors, *Proceedings of ISMIS'93*, LNAI 689, pages 350–361. Springer-Verlag, 1993.
- [Puget, 2005] J-F. Puget. Breaking all value symmetries in surjection problems. In P. van Beek, editor, *Proceedings of 11th International Conference on Principles and Practice of Constraint Programming (CP2005)*. Springer, 2005.
- [Puget, 2006] J-F. Puget. An efficient way of breaking value symmetries. In *Proceedings of the 21st National Conference on AI*. American Association for Artificial Intelligence, 2006.
- [Roney-Dougal *et al.*, 2004] C. Roney-Dougal, I. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In *Proceedings of ECAI-2004*. IOS Press, 2004.
- [Sellmann and Hentenryck, 2005] M. Sellmann and P. Van Hentenryck. Structural symmetry breaking. In *Proceedings of 19th IJCAI*. International Joint Conference on Artificial Intelligence, 2005.
- [Walsh, 2006a] T. Walsh. General symmetry breaking constraints. In *12th International Conference on Principles and Practices of Constraint Programming (CP-2006)*. Springer-Verlag, 2006.
- [Walsh, 2006b] T. Walsh. Symmetry breaking using value precedence. In *Proceedings of the 17th ECAI*. European Conference on Artificial Intelligence, IOS Press, 2006.

Symmetry in Constraint Optimization*

Toby Walsh
NICTA and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

Symmetry is an important feature in constraint programming. Whilst there has been considerable progress in dealing with symmetry in constraint satisfaction problems, there has been less attention to symmetry in constraint optimization problems. In this paper, we propose some general methods for dealing with symmetry in constraint optimization problems.

1 Introduction

Many search problems contain symmetries. Symmetry occurs naturally in many problems (e.g. if we have identical machines to schedule, or identical jobs to process). Symmetry can also be introduced when we model a problem. Unfortunately, symmetries increases the size of the search space. We must take into account symmetry or we will waste much time visiting symmetric solutions, as well as those parts of the search tree which are symmetric to already visited states. Sophisticated methods have been developed in constraint satisfaction problems to eliminate symmetry either statically by the addition of symmetry breaking constraints [Puget, 1993; Crawford *et al.*, 1996; Aloul *et al.*, 2002; Flener *et al.*, 2002; Aloul *et al.*, 2003; Law and Lee, 2004; Puget, 2005; 2006; Walsh, 2006] or dynamically by modifying the search method to avoid symmetric states [Gent and Smith, 2000; Fahle *et al.*, 2001; Roney-Dougal *et al.*, 2004; Hentenryck *et al.*, 2003]. In this paper, we outline methods to extend such symmetry breaking methods to work in the context of constraint optimization.

2 Formal Background

A constraint satisfaction problem consists of a set of n variables, each with a domain of m possible values, and a set of constraints specifying allowed combinations of values for given subsets of variables. A solution of a constraint satisfaction problem is an assignment of a value to each variable satisfying the constraints. We will use upper case letters like

*NICTA is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council DCITA and ARC through Backing Australia's Ability and the ICT Centre of Excellence program.

X_i and B_j for variables and lower case letters like d_k for domain values. Without loss of generality, we can assume all variables take values from the same universal domain of values (and unary constraints are used to restrict these domains as appropriate).

A constraint optimization problem is a constraint satisfaction problem with an additional objective function f . A solution of a constraint optimization problem is a complete assignment that satisfies the constraints (a *feasible* assignment) that minimizes or maximizes the objective function as appropriate. In the rest of this paper, we shall consider just minimization as any maximization problem can be turned into a minimization problem by negating the objective function f . To solve such a constraint optimization problem, we consider branch and bound style methods that solve a sequence of constraint satisfaction problems with tightening bounds on the objective function.

3 Symmetry in Satisfaction

Symmetry occurs in many constraint satisfaction problems. A *variable symmetry* of a constraint satisfaction problem is a permutation of the variables that preserves solutions. More formally, a variable symmetry is a bijection σ on the indices of variables such that if $X_1 = d_1, \dots, X_n = d_n$ is a solution then $X_{\sigma(1)} = d_1, \dots, X_{\sigma(n)} = d_n$ is also. For example, suppose we wish to assign times (values) to exams (variables) in an exam scheduling problem and we have two exams taken by the same students. As we can interchange these two exams, the problem has a variable symmetry. A *value symmetry* of a constraint satisfaction problem, on the other hand, is a permutation of the values that preserves solutions. More formally, a value symmetry is a bijection θ on the values such that if $X_1 = d_1, \dots, X_n = d_n$ is a solution then $X_1 = \theta(d_1), \dots, X_n = \theta(d_n)$ is also. For example, suppose we wish to assign colours (values) to nodes (variables) in a graph colouring problem. Value symmetry permits us to interchange any two colours uniformly throughout a colouring. We can further distinguish between *solution symmetries* which preserve solutions, and *constraint symmetries* which are those solution symmetries that also preserve the set of constraints [Cohen *et al.*, 2006]. As similar results hold for both, we will consider here just solution symmetries.

The set of symmetries of a constraint satisfaction or optimization problem form a group under composition. In this

work, we place no restrictions on the type of group. In particular, we are not restricted to products of the symmetry group, S_n . However, we do assume that the symmetries are known in advance. For instance, if we are coloring a graph and use a straight forward model with variables for nodes and values for colors, we know that the values are fully interchangeable. A number of methods have been developed to find symmetries in a constraint satisfaction problem automatically. It would be interesting to adapt these methods to finding symmetries in constraint optimization problems automatically.

Symmetries are problematic as they increase the size of the search space. For instance, if we have m interchangeable values, symmetry increases the size of the search space by a factor of $m!$. One simple and common mechanism to deal with symmetry in constraint satisfaction problems is to add constraints which eliminate symmetric solutions [Puget, 1993]. Suppose we have a set Σ of variable symmetries. We can eliminate all symmetric solutions due to these symmetries by posting “lex leader” constraints which ensure that the solution is ordered lexicographically before any of its symmetries [Crawford *et al.*, 1996]. More precisely, we post:

$$[X_1, \dots, X_n] \leq_{\text{lex}} [X_{\sigma(1)}, \dots, X_{\sigma(n)}]$$

For each $\sigma \in \Sigma$ where X_1 to X_n is a given fixed ordering on the variables. Similar lex leader constraints can be posted to eliminate value symmetries, as well as symmetries which act simultaneously on variables and values [Puget, 2006; Walsh, 2006].

4 Variable Symmetry in Optimization

We now lift both the definition of symmetry and the idea of eliminating symmetries by posting lex leader constraints to constraint optimization problems. We define a *variable symmetry* of a constraint optimization problem as a bijection σ on the indices of the variables that preserves feasibility. Note that such a symmetry may *not* preserve the value of the objective function. For example, consider the constraint optimization on two integer variables with constraints $0 \leq X_1 \leq 4$, $0 \leq X_2 \leq 4$, $X_1 + X_2 \leq 6$ and the objective function $-2X_1 - X_2$. Then X_1 and X_2 are symmetric. Given any assignment that is feasible (e.g. $X_1 = 1$ and $X_2 = 3$), we can swap X_1 and X_2 and construct another feasible assignment (in this case, $X_1 = 3$ and $X_2 = 1$). The objective function does not have this symmetry as the value of the objective changes (from -5 to -7) under this mapping. One appealing feature of this definition of symmetry of a constraint optimization problem is that we can use any of the existing methods for finding symmetries that have been developed for constraint satisfaction problems.

We cannot break variable symmetry in constraint optimization using exactly the same static symmetry breaking methods as in constraint satisfaction. Returning to our running example, suppose we post the lex leader constraint: $[X_1, X_2] \leq_{\text{lex}} [X_2, X_1]$. This gives a new constraint optimization problem with solution -9, whilst the original constraint optimization problem had a solution of -10. The added symmetry breaking constraint conflicts with minimizing the objective. Fortunately, we only need a small modification to

the lex leader method to ensure that the symmetry breaking constraints do not conflict. To eliminate symmetric solutions from a constraint optimization problem, we can post a symmetry breaking constraint for each variable symmetry which ensures:

$$[f(X_1, \dots, X_n), X_1, \dots, X_n] \leq_{\text{lex}} [f(X_{\sigma(1)}, \dots, X_{\sigma(n)}), X_{\sigma(1)}, \dots, X_{\sigma(n)}]$$

Such generalized lex leader constraints limit search to those assignments within each symmetry class to those with the smallest objective value, and between those with an equally small objective, to those that are lexicographically least. Note that if f is constant, this degenerates to the lex leader constraint used to break symmetry in constraint satisfaction problems (as would be expected). We can therefore use generalized lex leader constraints within a branch and bound algorithm to find the optimal solution. To propagate such generalized lex leader constraints, we propose the following simple encoding which introduces Boolean variables, B_i to record where the sequence is lex ordered:

$$\begin{aligned} f(X_1, \dots, X_n) &\leq f(X_{\sigma(1)}, \dots, X_{\sigma(n)}) \\ f(X_1, \dots, X_n) &\geq f(X_{\sigma(1)}, \dots, X_{\sigma(n)}) \quad \equiv \quad \neg B_1 \\ B_i &\vee (X_i \leq X_{\sigma(i)}) \\ B_i &\vee (X_i < X_{\sigma(i)}) \quad \equiv \quad B_{i+1} \end{aligned}$$

Note that, since $f(X_1, \dots, X_n) \leq f(X_{\sigma(1)}, \dots, X_{\sigma(n)})$, we could simply have posted $f(X_1, \dots, X_n) = f(X_{\sigma(1)}, \dots, X_{\sigma(n)}) \equiv \neg B_1$. However, such an equality is typically more difficult to propagate. For instance, if f is linear, it is NP-hard to enforce GAC on $f(X_1, \dots, X_n) = f(X_{\sigma(1)}, \dots, X_{\sigma(n)})$, but polynomial to enforce GAC on $f(X_1, \dots, X_n) \geq f(X_{\sigma(1)}, \dots, X_{\sigma(n)})$. In addition, we can get more pruning by using a more general hypothesis. We now consider this generalized lex leader method for three common classes of objective functions.

4.1 Symmetry-invariant Objective

Consider a constraint optimization problem with variable symmetry in which the objective function is invariant to symmetry. That is, $f(X_1, \dots, X_n) = f(X_{\sigma(1)}, \dots, X_{\sigma(n)})$ for every symmetry σ . Then, the generalized lex leader constraints simplify to the previous lex leader constraints:

$$[X_1, \dots, X_n] \leq_{\text{lex}} [X_{\sigma(1)}, \dots, X_{\sigma(n)}]$$

4.2 Linear Objective

Consider a constraint optimization problem with variable symmetry in which the objective function is linear. That is, $f(x_1, \dots, x_n) = \sum_{i=1}^n a_i \cdot x_i$. Then, we post:

$$\begin{aligned} \sum_{i=1}^n (a_i - a_{\sigma(i)}) \cdot X_i &\leq 0 \\ \sum_{i=1}^n (a_i - a_{\sigma(i)}) \cdot X_i &\geq 0 \quad \equiv \quad \neg B_1 \\ B_i &\vee (X_i \leq X_{\sigma(i)}) \\ B_i &\vee (X_i < X_{\sigma(i)}) \quad \equiv \quad B_{i+1} \end{aligned}$$

Returning to our running example, we get:

$$\begin{aligned} X_2 &\leq X_1 \\ X_2 &\geq X_1 &\equiv \neg B_1 \\ B_1 \vee (X_1 &\leq X_2) \\ B_1 \vee (X_1 < X_2) &\equiv B_2 \\ B_1 \vee (X_2 &\leq X_1) \\ B_2 \vee (X_2 < X_1) &\equiv B_3 \end{aligned}$$

These symmetry breaking constraints reduce the 22 feasible solutions down to the optimal solution ($X_1 = 4, X_2 = 2$ with objective value -10) and 9 other feasible solutions, each representative of an equivalence class of solutions with a different objective value.

4.3 Value-based Objective

Consider a constraint optimization problem with variable symmetry in which the objective function is just a function of the set or multi-set of values used by the feasible assignment. One such objective is $\sum_{i=1}^n g(X_i)$ where $g(X)$ represents the cost of using the value X . A second example is the objective which counts the number of values used, $|\{X_i \mid 1 \leq i \leq n\}|$. Such objective functions are invariant to variable symmetries: $f(X_1, \dots, X_n) = f(X_{\sigma(1)}, \dots, X_{\sigma(n)})$. In such cases, the generalized lex leader method returns the simple lex leader constraint:

$$[X_1, \dots, X_n] \leq_{\text{lex}} [X_{\sigma(1)}, \dots, X_{\sigma(n)}]$$

5 Value Symmetry in Optimization

In a similar fashion, we define a *value symmetry* of a constraint optimization problem as a bijection θ on values that preserves feasibility. Again, such a symmetry may *not* preserve the value of the objective function. For example, consider again the constraint optimization problem on two integer variables with constraints $0 \leq X_1 \leq 4, 0 \leq X_2 \leq 4, X_1 + X_2 \leq 6$ and the objective function $-2.X_1 - X_2$. The values 1 and 2 are symmetric. Given any assignment that is feasible (e.g. $X_1 = 1$ and $X_2 = 3$), we can swap 1 and 2 and construct another feasible assignment (in this case, $X_1 = 2$ and $X_2 = 3$). The objective function does not have this symmetry as the value of the objective changes (from -5 to -7) under this mapping.

As with variable symmetries, we cannot use the same static symmetry breaking methods in constraint optimization as in constraint satisfaction. Returning to our running example, suppose we post the lex leader constraint: $[X_1, X_2] \leq_{\text{lex}} [\theta(X_1), \theta(X_2)]$ where $\theta(1) = 2, \theta(2) = 1$ and $\theta(i) = i$ otherwise. This gives a new constraint optimization problem with solution -9, whilst the original constraint optimization problem had a solution of -10. The added symmetry breaking constraint again conflicts with minimizing the objective. We can, however, modify the lex leader method to ensure that the symmetry breaking constraints do not conflict with the objective function. To eliminate symmetric solutions from a constraint optimization problem, we can post a symmetry breaking constraint for each value symmetry which ensures:

$$[f(X_1, \dots, X_n), X_1, \dots, X_n] \leq_{\text{lex}}$$

$$[f(\theta(X_1), \dots, \theta(X_n)), \theta(X_1), \dots, \theta(X_n)]$$

Again, this limits search to those assignments within each symmetry class with the smallest objective value, and between those with an equally small objective, to those that are lexicographically least. To propagate such generalized lex leader constraints, we propose the following simple encoding which introduces Boolean variables, B_i to record where the sequence is lex ordered:

$$\begin{aligned} f(X_1, \dots, X_n) &\leq f(\theta(X_1), \dots, \theta(X_n)) \\ f(X_1, \dots, X_n) &\geq f(\theta(X_1), \dots, \theta(X_n)) &\equiv \neg B_1 \\ B_i \vee (X_i &\leq \theta(X_i)) \\ B_i \vee (X_i < \theta(X_i)) &\equiv B_{i+1} \end{aligned}$$

Returning to our running example, these symmetry breaking constraints reduce the 22 feasible solutions down to the optimal solution ($X_1 = 4, X_2 = 2$ with objective value -10) and 13 other feasible solutions. They eliminate, for instance, the feasible assignment $X_1 = 1, X_2 = 0$ as this has an objective value of -2 which is larger than the objective value of -4 corresponding to the symmetric assignment $X_1 = 2, X_2 = 0$. By contrast, the usual lex leader constraint, $[X_1, X_2] \leq_{\text{lex}} [\theta(X_1), \theta(X_2)]$ would eliminate $X_1 = 2, X_2 = 0$ in favour of the symmetric and lexicographically smaller assignment $X_1 = 1, X_2 = 0$.

6 Generator symmetries

One difficulty with the generalized lex leader method is that the set of symmetries, Σ can be exponentially large in general. For instance, if we have m interchangeable values, then Σ contains $m!$ symmetries. To deal with large number of symmetries in propositional satisfiability problems, Aloul *et al.* suggest we might break only a subset of the symmetries [Aloul *et al.*, 2002]. For example, we might only break those symmetries which are generators of the symmetry group. This idea lifts immediately to symmetries in constraint optimization problems. We can post generalized lex leader constraints corresponding to just a subset of the symmetries. In the case of interchangeable values, we have shown that posting just those symmetry breaking constraints corresponding to the generators which swap neighbouring values is enough to eliminate all symmetric solutions.

7 Variable and Value Symmetry

If a constraint optimization problem contains both variable and value symmetries, the symmetry breaking constraints for the variable and value symmetries can be combined. Note that each symmetry breaking constraint has to order the variables within an assignment in the same way. Symmetries can also act simultaneously on both the variables and values. Consider, for instance, a standard model of the n -queens problem with a variable, X_i representing the position of the queen along the i th row. The rotational symmetry of the chessboard maps a solution with $X_i = j$ onto a solution with $X_j = n - i + 1$. We can adapt the generalized lex leader method to deal with such symmetries in a straight forward way.

We define a *variable/value symmetry* of a constraint optimization problem as a bijection σ on variable assignments that preserves feasibility. For example, in the n -queens problem, the variable assignment $X_i = j$ maps onto $X_j = n - i + 1$. Let $\sigma(X_1, \dots, X_n)$ be the mapping of the complete assignment X_1, \dots, X_n by the variable/value symmetry σ . Then, we can break symmetry by posting the generalized lex leader constraints:

$$[f(X_1, \dots, X_n), X_1, \dots, X_n] \leq_{\text{lex}} [f(\sigma(X_1, \dots, X_n)), \sigma(X_1, \dots, X_n)]$$

For each σ in the set of symmetries, where X_1 to X_n is some fixed order on the variables.

8 Case Studies

To illustrate these methods for breaking symmetry in constraint optimization, we present two case studies.

8.1 Bin packing

Consider a simple model of bin packing problems where we have a decision variable for each item, and the value taken is the number of the bin into which the item is packed. The goal is to minimize the number of values used. If all bins are the same size, then we can swap the items in any two bins. Hence, we have a value symmetry in which values are interchangeable. The objective function counts the number of values used. This is invariant to the symmetry of interchanging values. Hence, the generalized lex leader constraints simply ensure that we construct the assignment which is lexicographically least. In [Walsh, 2006], I prove that this is equivalent to the global PRECEDENCE constraint.

Suppose we have two items of the same size. Then we can swap the two items in any packing. This corresponds to a variable symmetry which interchanges the corresponding two decision variables. The objective function is invariant to this type of variable symmetry. The generalized lex leader constraints therefore again ensure that we construct the assignment which is lexicographically least. In [Flener *et al.*, 2006], it is shown that this can be ensured using lexicographical ordering constraints on the “signatures” of interchangeable values. All such variable and value symmetry in a bin packing optimization problem can therefore be broken using existing methods.

8.2 Car sequencing

We consider an optimization version of car sequencing (prob001 in CSPLib) in which each car has a due date, and the objective is to minimize the sum of the tardiness. Production capacity constraints remain the same (e.g. only 1 in 2 cars at any point along the production line can have the sun roof option). A simple model of this problem has a decision variable for each position on the production line, and the value taken is the car being produced. Suppose we have two cars, j and k with the same options. Without loss of generality, suppose that car j is due before car k where $j < k$. This gives a value symmetry since we can interchange the corresponding two values and preserve feasibility of the solution.

However, the objective function is not, in general, invariant to this symmetry.

Consider the generalized lex leader constraints corresponding to this value symmetry. There are three cases. In the first case, both cars are produced before j 's due date. Then the objective function is invariant to symmetry. Hence, the generalized lex leader constraints ensure that we find the assignment which is lexicographically smaller than its symmetry. That is, the assignment where j is used before k . In the second case, both cars are produced after k 's due date. The objective function is again invariant to symmetry, and the generalized lex leader constraints ensure that we find the assignment which is lexicographically smaller than its symmetry. That is, the assignment where j is used before k . In the third case, at least one car is produced between j and k 's due dates. There are three subcases. In the first, both cars are produced between j and k 's due dates. Here, the generalized lex leader constraints ensure that we find the assignment whose objective value is less than its symmetry. This requires j to be used before k . The other two subcases are similar. In each, the value j must be used before k . Thus, in every case, the generalized lex leader constraints ensure that the value j must be used before k . This implied constraint breaks the symmetry of these interchangeable values.

9 Dynamic Methods

Another way to deal with symmetry is to adapt the search method to avoid exploring symmetric branches. For instance, after exploring a branch, the Symmetry Breaking During Search method (SBDS) adds a symmetry breaking constraint to avoid visiting any branches which are symmetric [Gent and Smith, 2000]. More precisely, suppose the current partial assignment is A and extending this with the variable assignment $X_i = j$ results in backtracking, then we can add the implied constraint, $\sigma(A) \Rightarrow \neg\sigma(X_i = j)$ where σ is any symmetry not already broken by A .

The advantage of such a method is that it does not conflict with the branching heuristic. Static symmetry breaking constraints, on the other hand, eliminate particular symmetric solutions, and these might be the precise solutions which the branching heuristic is directing the backtrack search procedure towards. To adapt SBDS to deal with symmetries in constraint optimization, we observe that we only want to eliminate those symmetric states with an equal or higher objective value¹. Hence, suppose the current partial assignment is A and extending this with the variable assignment $X_i = j$ results in backtracking, then we can add the implied constraint, $(\sigma(A) \wedge f_{\text{lb}}(\sigma(A)) \geq f_{\text{ub}}(A)) \Rightarrow \neg\sigma(X_i = j)$ where f_{lb} and f_{ub} are lower and upper bounds on the objective function respectively, and σ is some symmetry not already broken by A .

Another way to adapt the search method is Symmetry Breaking by Dominance Detection (SBDD) [Fahle *et al.*, 2001]. In SBDD, check is performed at each node to see if it is symmetric to one already explored. The key idea in SBDD is the definition of *dominance*. In constraint satisfaction problems, a node in the search tree is dominated by another iff the

¹Recall that we consider just minimization problems.

variable assignments of the second are symmetric to a subset of the first. SBDD uses a dominance test to ensure only undominated nodes are explored. This definition of dominance needs to be weakened for constraint optimization as we may want to visit symmetric nodes provided they have a smaller objective value. More precisely, in constraint optimization, a node in the search tree is dominated by another iff the variable assignments of the second are symmetric to a subset of the first *and* a lower bound on the objective value for the first is greater than or equal to an upper bound on the objective value for the second. With this weakened definition of dominance, SBDD can then be used in branch and bound style algorithms.

10 Related Work

Puget proved that symmetric solutions can be eliminated from constraint satisfaction problems by the addition of suitable constraints [Puget, 1993]. Crawford *et al.* presented the first general method for constructing variable symmetry breaking constraints within constraint satisfaction problems [Crawford *et al.*, 1996]. Petrie and Smith adapted this method to value symmetries by posting a lexicographical ordering constraint for each value symmetry [Petrie and Smith, 2003]. Puget and Walsh independently proposed propagators for such value symmetry breaking constraints [Puget, 2006; Walsh, 2006]. Aloul *et al.* have adapted such symmetry breaking methods to Boolean optimization [Aloul *et al.*, 2005]. When the constraints and objective function have similar sets of symmetries, they statically break symmetries in the intersection of these two sets. When the intersection of these two sets of symmetries is small, they use a dynamic method to break symmetries of any infeasible assignments that are discovered. This permits any symmetry of the constraints to be used.

Within branch and bound and dynamic programming algorithms, *dominance relations* are often used to prune symmetric search states. Let S be the set of search states (typically, partial assignments), and $f(s)$ for $s \in S$ be the minimum cost feasible solution that is an extension of s . A dominance relation, \prec is a partial ordering² over search states satisfying $s_i \prec s_j$ implies $f(s_i) \leq f(s_j)$. Thus, if we have already expanded s_i , we can prune s_j . The formal definition of dominance relation given in [Kohler and Steiglitz, 1974] also includes consistency with the lower-bound function (namely, $s_i \prec s_j$ implies $L(s_i) \leq L(s_j)$ where $L(s)$ is a lower bound on the cost of any feasible solution beneath s). Ibaraki proved conditions under which a stronger dominance relation is guaranteed to give a smaller branch-and-bound search tree [Ibaraki, 1977]. Finally, Yu and Wah used machine learning techniques to propose new dominance relations for 0/1 knapsack, scheduling and related problems [Yu and Wah, 1988].

One way to use dominance relations is to keep track of some or all previous states and only explore new states that are not dominated. Dominance relations are thus a generalization of lower-bound pruning. Another way to use dominance relations is to construct constraints that prune domi-

nated search states [Prestwich and Beck, 2004]. Let S_d be some subset of dominated states. That is, $s \in S_d$ implies there exists $s_j \in S$ such that $s_j \prec s_i$. We need a constraint C such that $C(s)$ holds iff $s \in S_d$. However, there is no general method to construct such constraints. Such dominance constraints can, however, do more pruning than the symmetry breaking constraints proposed here. They can prune all feasible states with an equivalence class of assignments if the minimum value of the objective function in this equivalence class is larger than the value taken by some other feasible assignment.

11 Conclusion and Future Work

We have shown how the lex leader method for symmetry breaking can be adapted to work with constraint optimization problems. The key idea is to post symmetry breaking constraints which eliminate within each equivalence class of assignments, all but those assignments with the best objective value, and between those assignments with the best objective value, the single assignment which is lex ordered the least. There are many directions for future research. First, we should consider special classes of symmetries like interchangeable values, or row and column symmetries, where symmetry breaking may be more tractable. Second, we might identify other situations where the generalized lex leader constraints can be simplified. Third, we should consider how these generalized lex leader constraints interact with the problem constraints. For instance, how do they interact with an all-different constraint over the decision variables? Fourth, we might develop specialized propagators for reasoning about the symmetries of the objective function. Fifth, we should test how these methods work in practice on constraint optimization problems.

References

- [Aloul *et al.*, 2002] F.A. Aloul, A. Ramani, I. Markov, and K.A. Sakallah. Solving difficult SAT instances in the presence of symmetries. In *Proceedings of the Design Automation Conference*, pages 731–736, 2002.
- [Aloul *et al.*, 2003] F.A. Aloul, K.A. Sakallah, and I.L. Markov. Efficient symmetry breaking for Boolean satisfiability. In *Proceedings of the 18th International Joint Conference on AI*, pages 271–276. International Joint Conference on Artificial Intelligence, 2003.
- [Aloul *et al.*, 2005] F.A. Aloul, A. Ramani, I. Markov, and K.A. Sakallah. Dynamic symmetry-breaking for improved boolean optimization. In *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC)*, pages 445–450, 2005.
- [Cohen *et al.*, 2006] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B.M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2–3):115–137, 2006.
- [Crawford *et al.*, 1996] J. Crawford, G. Luks, M. Ginsberg, and A. Roy. Symmetry breaking predicates for search

²A partial order is transitive, reflexive ($s \prec s$) and antisymmetric ($s_i \prec s_j$ and $s_j \prec s_i$ implies $s_i = s_j$).

- problems. In *Proceedings of the 5th International Conference on Knowledge Representation and Reasoning, (KR '96)*, pages 148–159, 1996.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Proceedings of 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*, pages 93–107. Springer, 2001.
- [Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetry in matrix models. In *8th International Conference on Principles and Practices of Constraint Programming (CP-2002)*. Springer, 2002.
- [Flener *et al.*, 2006] P. Flener, J. Pearson, M. Sellmann, and P. Van Hentenryck. Static and dynamic structural symmetry breaking. In *Proceedings of 12th International Conference on Principles and Practice of Constraint Programming (CP2006)*. Springer, 2006.
- [Gent and Smith, 2000] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.
- [Hentenryck *et al.*, 2003] P. Van Hentenryck, M. Agren, P. Flener, and J. Pearson. Tractable symmetry breaking for CSPs with interchangeable values. In *Proceedings of the 18th International Conference on AI*. International Joint Conference on Artificial Intelligence, 2003.
- [Ibaraki, 1977] T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the Association for Computing Machinery*, 24(2):264–279, 1977.
- [Kohler and Steiglitz, 1974] W.H. Kohler and K. Steiglitz. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *Journal of the Association for Computing Machinery*, 21(1):140–156, 1974.
- [Law and Lee, 2004] Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pages 362–376. Springer, 2004.
- [Petrie and Smith, 2003] Karen E. Petrie and Barbara M. Smith. Symmetry Breaking in Graceful Graphs. Technical Report APES-56a-2003, APES Research Group, June 2003.
- [Prestwich and Beck, 2004] S.D. Prestwich and C.J. Beck. Exploiting dominance in three symmetric problems. In *Proceedings of 4th International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon-04)*, pages 63–70, 2004. Held alongside 10th International Conference on Principles and Practice of Constraint Programming (CP2004).
- [Puget, 1993] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In J. Komorowski and Z.W. Ras, editors, *Proceedings of ISMIS'93*, LNAI 689, pages 350–361. Springer-Verlag, 1993.
- [Puget, 2005] J-F. Puget. Breaking all value symmetries in surjection problems. In P. van Beek, editor, *Proceedings of 11th International Conference on Principles and Practice of Constraint Programming (CP2005)*. Springer, 2005.
- [Puget, 2006] J-F. Puget. An efficient way of breaking value symmetries. In *Proceedings of the 21st National Conference on AI*. American Association for Artificial Intelligence, 2006.
- [Roney-Dougal *et al.*, 2004] C. Roney-Dougal, I. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In *Proceedings of ECAI-2004*. IOS Press, 2004.
- [Walsh, 2006] T. Walsh. General symmetry breaking constraints. In *12th International Conference on Principles and Practices of Constraint Programming (CP-2006)*. Springer-Verlag, 2006.
- [Yu and Wah, 1988] C.-F. Yu and B.W. Wah. Learning dominance relations in combined search problems. *IEEE Transactions on Software Engineering*, 14(8):1155–1175, 1988.

Symmetry Breaking in Subgraph Isomorphism

Stéphane Zampelli⁽¹⁾, Yves Deville⁽¹⁾, Mohamed Réda Saïdi⁽²⁾, Belaïd Benhamou⁽²⁾

⁽¹⁾ Université catholique de Louvain,
 Department of Computing Science and Engineering,
 2, Place Sainte-Barbe 1348 Louvain-la-Neuve (Belgium)
⁽²⁾ Centre de Mathématique et d'Informatique
 39, rue Joliot Curie - 13453 Marseille cedex 13, France

Abstract

The present work studies symmetry breaking for the subgraph isomorphism problem. This NP-Complete problem decides if a pattern graph is isomorphic to a subgraph of a target graph. The first part of the paper shows how to detect and break all variable and value global symmetries. The second part studies local symmetries, and shows that subgraphs of the initial instance allow to efficiently compute local variable and value symmetries. Experiments show that global symmetries are an efficient technique for subgraph isomorphism, and that limited local symmetries may be useful for difficult instances.

1 Introduction

A symmetry in a Constraint Satisfaction Problem (CSP) is a bijective function that preserves CSP structure and solutions. Symmetries are important because they induce symmetric subtrees in the search tree. If the instance has no solution, failure has to be proved for equivalent subtrees regarding symmetries. If the instance has solutions, many symmetric solutions will have to be enumerated in symmetric subtrees. The detection and breaking of symmetries can thus speed up the solving of a CSP. Symmetries arise naturally in graphs as automorphisms. However, although many graph problems have been tackled [Beldiceanu *et al.*, 2005] [Cambazard and Bourreau, 2004] [Sellman, 2003] and a computation domain for graphs has been defined [Dooms *et al.*, 2005], and despite the fact that symmetries and graphs are related, little has been done to investigate the use of symmetry breaking for graph problems in constraint programming.

This work aims at applying and extending symmetry techniques for subgraph isomorphism. We show how to detect and handle global variable and value symmetries as well as local symmetries.

2 Background and Definitions

Basic definitions for subgraph isomorphism and symmetries are introduced.

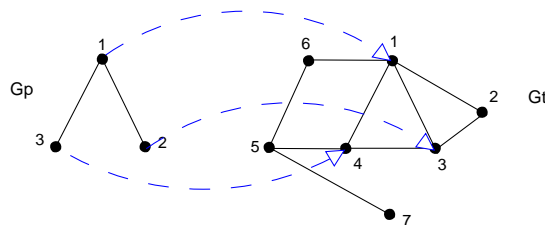


Figure 1: Example solution for an isomorphism problem instance.

A graph $G = (N, E)$ consists of a node set N and an edge set $E \subseteq N \times N$, where an edge (u, v) is a pair of nodes. The nodes u and v are the endpoints of the edge (u, v) . We consider directed and undirected graphs. A subgraph of a graph $G = (N, E)$ is a graph $S = (N', E')$ where N' is a subset of N and E' is a subset of E such that for all $(u, v) \in E'$, $u, v \in N'$.

A subgraph isomorphism problem between a pattern graph $G_p = (N_p, E_p)$ and a target graph $G_t = (N_t, E_t)$ consists in deciding whether G_p is isomorphic to some subgraph of G_t . More precisely, one should find an injective function $f : N_p \rightarrow N_t$ such that $\forall (u, v) \in N_p \times N_p, (u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$. This NP-Hard problem is also called subgraph monomorphism problem or subgraph matching in the literature. The function f is called a subgraph matching function.

The CSP model of subgraph isomorphism should represent a total function $f : N_p \rightarrow N_t$. This total function can be modeled with $X = x_1, \dots, x_n$ with x_i a FD variable corresponding to the i^{th} node of G_p and $D(x_i) = N_t$. The injective condition is modeled with the global constraint $\text{alldiff}(x_1, \dots, x_n)$. The isomorphism condition is translated into a set of constraints $MC_l(x_i, x_j) \equiv (x_i, x_j) \in E_t$ for all $(i, j) \in E_p$. This set of constraints can be turned into a global constraint $\text{MC}(x_1, \dots, x_n) \equiv \bigwedge_{(i,j) \in E_p} MC_l(x_i, x_j)$. Implementation, comparison with dedicated algorithms, and extension to subgraph isomorphism and to graph and function computation domains can be found in [Zampelli *et al.*, 2005; Deville *et al.*, 2005].

A CSP instance is a triple $\langle X, D, C \rangle$ where X is the set of variables, D is the universal domain spec-

ifying the possible values for those variables, and C is the set of constraints. In the sequel, $n = |N_p|$, $d = |D|$, and $D(x_i)$ is the domain of x_i . A symmetry over a CSP instance P is a bijection σ mapping solutions to solutions, and hence non solutions to non solutions [Puget, 2005b]. Since a symmetry is a bijection where domain and target sets are the same, a symmetry is a permutation. A *variable symmetry* is a bijective function $\sigma : X \rightarrow X$ permuting a (non) solution $s = ((x_1, d_1), \dots, (x_n, d_n))$ to a (non) solution $\sigma s = ((\sigma(x_1), d_1), \dots, (\sigma(x_n), d_n))$. A *value symmetry* is a bijective function $\sigma : D \rightarrow D$ permuting a (non) solution $s = ((x_1, d_1), \dots, (x_n, d_n))$ to a (non) solution $\sigma s = ((x_1, \sigma(d_1)), \dots, (x_n, \sigma(d_n)))$. A *value and variable symmetry* is a bijective function $\sigma : X \times D \rightarrow X \times D$ permuting a (non) solution $s = ((x_1, d_1), \dots, (x_n, d_n))$ to a (non) solution $\sigma s = (\sigma(x_1, d_1), \dots, \sigma(x_n, d_n))$. A *global symmetry* of a CSP is a symmetry holding on the initial problem. A *local symmetry* of a CSP P is a symmetry holding only in a sub-problem P' of P . The conditions of the symmetry are the constraints necessary to generate P' from P [Gent *et al.*, 2005] [Benhamou, 1994]. A *group* is a finite or infinite set of elements together with a binary operation (called the group operation) that satisfies the four fundamental properties of closure, associativity, the identity property, and the inverse property. An *automorphism of a graph* is a graph isomorphism with itself. The set of automorphisms $Aut(G)$ defines a finite group of permutations.

3 Variable Symmetries

In this section, we show that the set of global variable symmetries of a subgraph isomorphism CSP is the set of automorphisms of the pattern graph. Moreover, we show how existing techniques can be used to break all global variable symmetries.

3.1 Detection

This subsection shows that, in subgraph isomorphism, global variable symmetries are the automorphisms of the pattern graph and do not depend on the target graph. It has been shown that the set of variable symmetries of the CSP is the automorphism group of a *symbolic graph* [Puget, 2005b]. The pattern G_p is transformed into a symbolic graph $S(G_p)$ where $Aut(S(G_p))$ is the set of variable symmetries of the CSP.

A CSP P modeling a subgraph isomorphism instance (G_p, G_t) can be transformed into the following symbolic graph $S(P)$:

1. Each variable x_i is a distinct node labelled i .
2. If there exists a constraint $MC(x_i, x_j)$, then there exists an arc between i and j in the symbolic graph.
3. The constraint `alldiff` is transformed into a node typed with label 'a'; an arc (a, x_i) is added to the symbolic graph for each x_i .

Figure 2 shows a pattern transformed into its symbolic graph. If we do not consider the extra node and

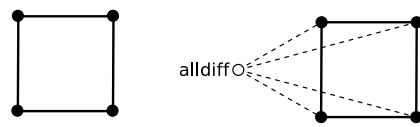


Figure 2: Example of symbolic graph for a square pattern.

arcs introduced by the `alldiff` constraint, then the symbolic graph $S(P)$ and G_p are isomorphic by construction. Given the labelling of nodes representing constraints, an automorphism in $S(P)$ maps the `alldiff` node to itself and the nodes corresponding to the variables to another node corresponding to the variables. Each automorphism in $Aut(G_p)$ will thus be a restriction of an automorphism in $Aut(S(P))$, and an element in $Aut(S(P))$ will be an extension of an element in $Aut(G_p)$. Hence the two following theorems.

Let (G_p, G_t) be a subgraph isomorphism instance, P its associated CSP. We have :

- $\forall \sigma \in Aut(G_p) \exists \sigma' \in Aut(S(P)) : \forall n \in N_p : \sigma(n) = \sigma'(n)$
- $\forall \sigma' \in Aut(S(P)) \exists \sigma \in Aut(G_p) : \forall n \in N_p : \sigma(n) = \sigma'(n)$

Let (G_p, G_t) be a subgraph isomorphism instance, P its associated CSP. The set of variable symmetries of P is the set of bijective functions $Aut(S(P))$ restricted to N_p , which is equal to $Aut(G_p)$.

The above theorem states that only $Aut(G_p)$ has to be computed in order to get all variable symmetries.

3.2 Breaking

Two existing techniques are relevant to our particular problem. The first technique is an approximation and consists in breaking only the generators of the symmetry group [Crawford *et al.*, 1996]. Those generators are obtained by an automorphism detection software such as NAUTY [McKay, 1981]. For each generator σ , an ordering constraint $s \leq \sigma s$ is posted.

The second technique breaks all variable symmetries of an injective problem by using a Schreier-Sims algorithm, provided that the generators of the variable symmetry group are known [Puget, 2005a]. Puget showed that the number of constraints to be posted is linear with the number of variables. The Schreier-Sims algorithm computes a base and a strong generating set of a permutation group. Let G be the group, S_g the symmetry group of g elements containing G , and t the number of generators, then its complexity is in $O(g^2 \log^3 |G| + t.g.\log |G|)$.

4 Value Symmetries

In this section we show how all global value symmetries can be detected and how existing techniques can be extended to break them.

4.1 Detection

In subgraph isomorphism, global value symmetries are automorphisms of the target graph and do not depend on the pattern graph.

Let (G_p, G_t) be a subgraph isomorphism instance and P be its associated CSP. Then each $\sigma \in \text{Aut}(G_t)$ is a value symmetry of P .

Proof Suppose that f is a subgraph isomorphism between G_p and G_t , and $f(i) = v_i$ for $i \in N_p$. Consider the subgraph $G = (N, E)$ of G_t , where $N = \{v_1, \dots, v_n\}$ and $E = \{(i, j) \in E_t \mid (f^{-1}(i), f^{-1}(j)) \in E_p\}$. This means that there exists a isomorphic function f' matching G_p to σG . Hence $((x_1, \sigma(v_1)), \dots, (x_n, \sigma(v_n)))$ is a solution. ■

4.2 Breaking

Breaking global value symmetries can be performed by using the GE-Tree technique [Ronay-Dougal *et al.*, 2004]. The idea is to modify the distribution by avoiding symmetrical value assignments. Suppose a state S is reached, where x_1, \dots, x_k are assigned to v_1, \dots, v_k respectively, and x_{k+1}, \dots, x_n are not assigned yet. The variable x_{k+1} should not be assigned to two symmetrical values, since two symmetric subtrees would be searched. For each value $v_i \in D(x_{k+1})$ that is symmetric to a value $v_j \in D(x_{k+1})$, only one state S_1 should be generated with the new constraint $x_{k+1} = v_i$.

A convenient way to compute those symmetrical values uses the Schreier-Sims algorithm. Algorithm Schreier-Sims outputs the sets $U_i = \{k \mid \exists \sigma \in \text{Aut}(G_t) : \sigma(i) = k \wedge \sigma(j) = j \forall j < i\}$. A set U_i gives the images of i by the automorphisms of G mapping $0, \dots, i-1$ to themselves. If values are assigned in an increasing order, assigning symmetrical values can be avoided by using those sets U_i . Using symmetry breaking constraints together with GE-Tree is complete and correct as shown in [Puget, 2005a].

5 Local Symmetries

Global symmetries may hide symmetries arising during search. During search, variables are assigned and new variable symmetries arise. As values are removed from domains, new value symmetries are created and can be exploited. In this section, we focus on detecting those symmetries for the subgraph isomorphism problem.

Local symmetries for subgraph isomorphism can be found through local graphs of the initial problem. During the search, subgraphs of the pattern and target graph define variable and value local symmetries. We first show how to define those subgraphs, and then we explain local variable symmetry detection and local value symmetry detection.

5.1 Partial dynamic graphs

We first introduce partial dynamic graphs. Those graphs are associated to a state in the search and correspond to the unsolved part of the problem. This can be viewed as a new local problem to the current state.

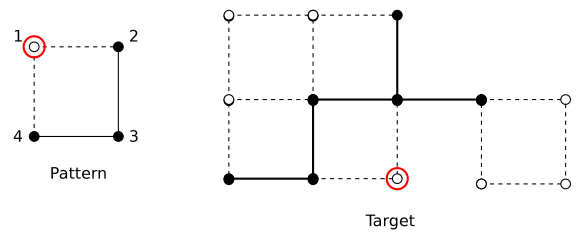


Figure 3: Example of local subgraphs.

Let S be a state in the search.

The **partial dynamic pattern graph** $G_p^- = (N_p^-, E_p^-)$ induced by S is a subgraph of G_p such that :

- $N_p^- = \{i \in N_p \mid \exists j : (i, j) \in E_p \wedge \exists a \in D(x_i) \wedge \exists b \in D(x_j) \wedge (a, b) \notin E_t\}$
- $E_p^- = \{(i, j) \in E_p \mid i \in N_p^- \wedge j \in N_p^-\}$

The **partial dynamic target graph** $G_t^- = (N_t^-, E_t^-)$ is a subgraph of G_t such that :

- $N_t^- = \cup_{i \in N_p^-} D(x_i)$
- $E_t^- = \{(a, b) \in E_t \mid a \in N_t^- \wedge b \in N_t^-\}$

Those partial dynamic graphs define the local CSP corresponding to the local state.

Figure 3 shows an example where circled nodes are assigned to each other. In the pattern graph, plain nodes and edges represent G_p^- . Regarding morphism constraints, dashed edges are entailed MC_i constraints and plain edges are non entailed MC_i constraints. In the target graph, plain nodes and edges represent G_t^- assuming a forward checking propagation for the MC_i constraints.

One general way to compute local symmetries is to use the microstructure of the CSP [Cohen *et al.*, 2006]. The set of nodes of the microstructure graph is the product set of the variables and the domain. In our particular problem of subgraph isomorphism, the variables are the nodes of G_p^- and the domain is the set of nodes of G_t^- . Hence the size of the microstructure is $|G_p^- \times G_t^-|$ and can be very large. But in subgraph isomorphism, local symmetries can be computed directly in the graphs G_p^- and G_t^- , without using the microstructure.

5.2 Local variable symmetries

Local variable symmetries must map variables having the same domain. This fact follows directly from the definition of a variable symmetry. This problem was not present for global variable symmetries as the initial domains are N_t . The set of automorphisms of the partial dynamic pattern graph has to be redefined.

Given a partial dynamic pattern graph G_p^- , $\text{Aut}'(G_p^-)$ is the set of automorphisms mapping a node i to a node j if and only if $D(x_i) = D(x_j)$. The following theorem states that local variable symmetries can be obtained by computing $\text{Aut}'(G_p^-)$.

Let (G_p, G_t) be a subgraph isomorphism instance, L be a state in the search space, G_p^- the partial dynamic

pattern graph associated with L , and P' be the CSP associated with L . Then each $\sigma \in \text{Aut}'(G_p^-)$ is a variable symmetry of P' .

Proof Let $\sigma \in \text{Aut}'(G_p^-)$. Consider the symbolic graph $\text{Aut}(S(P'))$ of P' . Recall that the alldiff constraint has no influence on $\text{Aut}(S(P'))$. All automorphisms β of $\text{Aut}(S(P'))$ are not variable symmetry of P' since domains of variables may be different in the local subproblem P' . Since $\sigma \in \text{Aut}(S(P'))$ and is restricted to map only variables with the same domains, σ is a variable symmetry of P' . ■

Computing $\text{Aut}'(G_p^-)$ can be done as usual by using automorphism detection software. The initial partition is refined into ordered sets containing variables having the same domain.

Breaking

Local variable symmetries can be broken by using the same technique for global variable symmetries (Section 3.2). This ensures that all detected local variable symmetries are broken. However, adding breaking constraint of the form $x_i < x_j$ modify the local symbolic graph $S(P)$. This may introduce or remove new local variable symmetries. The detection presented in the previous section is however valid. Indeed, the additional constraints $x_i < x_j$ ensure that $D(x_i) \neq D(x_j)$. Any automorphism between x_i and x_j is excluded from $\text{Aut}'(G_p^-)$.

5.3 Local value symmetries

The following theorem states that value symmetries of the local CSP P' can be obtained by computing $\text{Aut}(G_t^-)$ and that these symmetries can be exploited without losing or adding solutions to the initial problem.

Let (G_p, G_t) be a subgraph isomorphism instance, P' be the local CSP associated with a state during the search. Then each $\sigma \in \text{Aut}(G_t^-)$ is a value symmetry of P' . **Proof** This follows directly from Theorem 4.1 and the fact that (G_p^-, G_t^-) is a subgraph isomorphism instance. ■

The dynamic target graph G can be computed dynamically. In [Deville *et al.*, 2005], we showed how subgraph isomorphism can be modeled and implemented in CP(Graph), an extension of CP with graph domain variables [Dooms *et al.*, 2005]. The domain of a graph variable is modeled by a lower bound and an upper bound graph, and represents all the graphs between the lower and upper bound. In this setting, a graph domain variable T represents the matched target subgraph. The initial lower bound of T is the empty graph, and the initial upper bound is G_t . When a solution is found, T is instantiated to the matched subgraph of G_t . Hence, during the search, the dynamic target graph G_t^- will be the upper bound of variable T and can be obtained in $O(1)$.

Speeding up detection

Computing directly $\text{Aut}(G_t^-)$ is correct but this computation can be fasten. Actually, all value symmetries are not possible in a local instance (G_p^-, G_t^-) . Only nodes

that are all present in at least one domain can be mapped to each other in a value symmetry of P' . The search tree of the automorphism algorithm can be pruned when such nodes are mapped together.

Breaking

In this subsection, we show how to modify the GE-Tree method to handle local value symmetries. Before distribution, the following actions are triggered :

1. Compute the partial dynamic target graph G_t^- .
2. The NAUTY and Schreier-Sims algorithms are called to produce the new U'_i sets.
3. Given a state S , a new variable and value selection can be used such that local value symmetries are broken :
 - (a) a new state S_1 with a constraint $x_k = v_k$
 - (b) a new state S_2 with constraints : $x_k \neq v_k$ and $x_k \neq v_j \forall j \in U_{k-1} \cup U'_{k-1}$.

The only difference with the original GE-Tree method is the addition of the U'_{k-1} during the creation of the second branch corresponding to the state S_2 .

An issue is how to handle the global and local structures U . In the Gecode system (<http://www.gecode.org>), in which the actual implementation is made, the states are copied and trailing is not needed. Thus the global structure U must not be updated because of backtracking. A single global copy is kept during the whole search process. In a state S where local values symmetries are discovered, structure U is copied into a new structure U'' and merged with U' . This structure U'' shall be used for all states S' having S in its predecessors.

6 Experimental results

The objectives in this section are to assess performances of global symmetries, and performance of local symmetries against global symmetries. For local symmetries, we study the overhead of computing local symmetry information and their ability to solve more difficult instances. Moreover, we would like to know whether local symmetries can be applied on the whole search space.

The CSP model for subgraph isomorphism has been implemented in Gecode, using CP(Graph) and CP(Map) [Dooms *et al.*, 2005] [Deville *et al.*, 2005]. The CP(Graph) framework provides graph domain variables and CP(Map) provides function domain variables. All the software is implemented in C++. The standard implementation of NAUTY [McKay, 1981] algorithm is used. We also implemented Schreier-Sims algorithm. The computation of the constraints for breaking injective problems is implemented, and GE-Tree method is also incorporated. All local symmetry techniques presented are also implemented.

Instances - The data graphs used to generate instances are from the GraphBase database containing different topologies and has been used in [Larrosa and Valiente, 2002]. Experiments are performed on the first

50 undirected graphs from GraphBase. The undirected set was selected because it holds potentially more symmetries than the directed graphs. This undirected set contains graphs ranging from 10 nodes to 138 nodes. All those graphs are tested for isomorphism with one another. Only subgraph isomorphism instances with a pattern graph smaller than the target graph are kept. There are 1225 instances.

Setup - All runs were performed on a dual Intel(R) Xeon(TM) CPU 2.66GHz with 2 Go of RAM. In our tests, we look for all solutions. This ensures that we measure the whole tree search reduction, and we avoid strong influence of the heuristic. As shown later in this section, the number of solved instances stabilizes for all instances after a couple of minutes. Hence a run time limit is set. A run is solved if it finishes in less than 5 minutes, unsolved otherwise. Detecting the local symmetries on the whole search space tends to be time-consuming. Hence local symmetry detection is seen as an extension of global symmetries. No detection is made when 3 variables are instantiated. Breaking is performed over the whole search space.

Automorphism detection time - A main concern is how much time it takes to compute the symmetries of the graphs. Regarding global symmetries, NAUTY processed each undirected graph in less than 0.02 second. All undirected graphs were processed by Schreier-Sims in less than one second, except two of them, with 4 seconds and 8 seconds. This shows a negligible time regarding symmetry detection on this set of instances.

Models - Depending on the symmetry breaking techniques, various models are selected for these experiments :

- vflib : state of the art dedicated C++ algorithm [Cordella *et al.*, 2001]
- light : simple CP model
 - Forward checking constraints
 - No redundant constraint
- heavy : advanced CP model
 - Arc consistency
 - Redundant constraint [Larrosa and Valiente, 2002]
- global var : heavy + global variable symmetry
- global value : heavy + global value symmetry
- global varvalue : heavy + global variable and value symmetry
- local var : heavy + local variable symmetry
- local value : heavy + local value symmetry
- glocal var : heavy + global *and* local variable symmetry
- glocal value : heavy + global *and* local value symmetry

Vflib and the light model are considered as basic models since they perform only forward checking. We call

easy instances those instances that are quickly solved by vflib and the light model. Those instances do not require any arc consistent or redundant constraint.

Detailed results - We study first experimental results for global symmetries. Figure 4 shows the number of solved instances against time. This Figure justifies the choice of a time limit of 5 minutes, as most of the solved instances are solved during the first 100 seconds. Hence only the percentage of solved instances is relevant. Figure 5 shows the detailed results. The total time is the time to solve all instances, the mean time is the mean time over all solved instances, the common mean time(memory) is the mean time(memory) over instances solved by vflib. Global symmetries clearly outperforms light, heavy and vflib and improve time on easy instances and all instances. Thanks to global variable and value symmetries, 18% more instances are solved compared to vflib and all instances are solved much more efficiently.

We now study the experimental results for local symmetries. Figure 6 shows the detailed results. The common time is still reduced, but local symmetries achieve the same performance as the heavy model without any symmetry technique, with the exception of local and global variable symmetries. Those results for local symmetries are due to the time needed to compute local symmetries. Actually, some easy instances are not solved with local symmetries.

In order to assess efficiency of local symmetries for difficult problems, we performed the following experiment. The light model is ran for 30 seconds, and if the instance is not solved, local symmetry models are used for 270 seconds. This coroutinging setup ensures that easy instances are solved. Results for this new setup are shown in Figure 7. We compare the results of local symmetries against global symmetries. Local symmetries slightly outperform global symmetries. Inside local symmetry models, the models combining global symmetries outperform pure local symmetry models. This is mainly because some instances contains a lot of symmetries that disappear during search. Local symmetry has a high cost but reduces time for difficult instances. Not surprisingly, local symmetries performance are poor on easy instances, but outperform global symmetry on difficult instances.

To the best of our knowledge, subgraph isomorphism with symmetry breaking achieves the best percentage of solved instances over the GraphBase benchmark proposed by [Larrosa and Valiente, 2002].

7 Conclusion

In subgraph isomorphism, both global variable and value symmetries can be computed on the initial instance. Indeed, this computation can be made directly on the pattern graph and the target graph. Moreover, all variable and value symmetries can be broken by computing a base and a strong generating set of the permutation groups thanks to the Schreier-Sims algorithm. Local variable and value symmetries can be found in a similar way. A

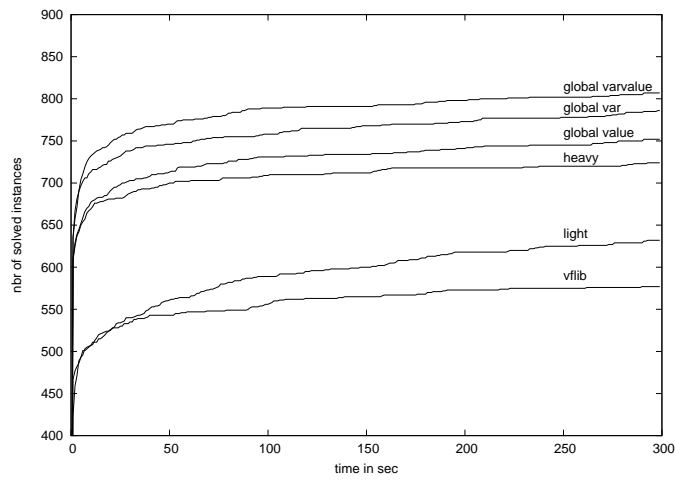


Figure 4: Results for global symmetries

	solved	total time	mean time	mean mem	c. mean time	c. mean mem.
vflib	47.1%	3329 min.	9.35 sec.	115 kb	9.35 sec.	92 kb
light	51.4%	3162 min.	17.91 sec.	2028 kb	14.89 sec.	1391 kb
heavy	58.94%	2584 min.	6.57 sec.	7892 kb	5.81 sec.	3103 kb
global var	64%	2318 min.	8.72 sec.	7744 kb	2.75 sec.	2933 kb
global value	61.2%	2479 min.	8.45 sec.	7820 kb	3.06 sec.	3014 kb
global varvalue	65.7%	2197 min.	7.35 sec.	7545 kb	2.50 sec.	2983 kb

Figure 5: Detailed results for global symmetries.

	solved	total time	mean time	mean mem	c. mean time	c. mean mem.
heavy	58.94%	2584 min.	6.57 sec.	7892 kb	5.81 sec.	3103 kb
glocal var	60.82%	2473 min.	5.93 sec.	19201 kb	4.40 sec.	3182 kb
local var	58.45%	2615 min.	5.88 sec.	18666 kb	3.30 sec.	3096 kb
glocal value	58.78%	2601 min.	6.37 sec.	17839 kb	3.88 sec.	3684 kb
local value	57.14%	2682 min.	4.96 sec.	7812 kb	3.29 sec.	3243 kb

Figure 6: Detailed results for local symmetries over all instances.

	solved	total time	mean time	mean mem
global var	64,73%	2203 min.	4.27 sec.	11371 kb
glocal var	65,96%	2150 min.	3.26 sec.	11503 kb
local var	63,10%	2300 min.	3.15 sec.	4105 kb
global value	63,92%	2256 min.	2.94 sec.	11475 kb
glocal value	64,49%	2234 min.	3.78 sec.	28610 kb
local value	63,18%	2301 min.	3.77 sec.	4330 kb

Figure 7: Detailed results for local symmetries with corouting.

suitable definition of the local pattern and target graphs makes the computation of local symmetries as direct as for global symmetries.

Experimental results suggest that breaking all variable and value symmetries is an efficient way to solve difficult instances. Global symmetries together with arc consistency and redundant constraints were able to solve 65% of the instances, which makes constraint programming the most efficient technique for this data set. Local symmetries achieve also good results on difficult instances. However, computing local symmetries may not be the good tradeoff between search and symmetries, especially for easy instances. Computing local symmetries during the whole search is inefficient.

Interesting directions include experiments on faster but weaker detection methods. One could search for and break generators, as the Schreier-Sims tends to be time consuming. Other weaker forms of detection could also be used. Finally, experiments should be conducted on real-world class of graphs such as scale-free networks.

References

- [Beldiceanu *et al.*, 2005] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. The tree constraint. In Roman Bartak, editor, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, Proceedings*, volume 3524 of *Lecture Notes in Computer Science*, pages 64–78. Springer, June 2005.
- [Benhamou, 1994] Belaid Benhamou. Study of symmetry in constraint satisfaction problems. In Alan Borning, editor, *Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 246–254, Orcas Island, Seattle, USA, may 1994. Springer.
- [Cambazard and Bourreau, 2004] Hadrien Cambazard and Eric Bourreau. Conception d'une contrainte globale de chemin. In *10e Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'04)*, pages 107–121, Angers, France, June 2004.
- [Cohen *et al.*, 2006] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3):115–137, 2006.
- [Cordella *et al.*, 2001] Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159. Cuen, 2001.
- [Crawford *et al.*, 1996] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, November 1996.
- [Deville *et al.*, 2005] Yves Deville, Grégoire Doms, Stéphane Zampelli, and Pierre Dupont. Cp(graph+map) for approximate graph matching. In Francisco Azevedo, Carmen Gervet, and Enrico Pontelli, editors, *1st International Workshop on Constraint Programming Beyond Finite Integer Domains (in conjunction with the Eleventh International Conference on Principles and Practice of Constraint Programming), CP2005, Sitges, Spain*, pages 33–47, October 2005.
- [Doms *et al.*, 2005] Grégoire Doms, Yves Deville, and Pierre Dupont. Cp(graph): Introducing a graph computation domain in constraint programming. In van Beek [2005], pages 211–215.
- [Gent *et al.*, 2005] Ian .P. Gent, Tom Kelsey, Steve A. Linton, Iain McDonald, Ian Miguel, and Barbara M. Smith. Conditional symmetry breaking. In van Beek [2005], pages 256–270.
- [Larrosa and Valiente, 2002] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Comp. Sci.*, 12(4):403–422, 2002.
- [McKay, 1981] B. D. McKay. Practical graph isomorphism. *Congressum Numerantium*, 30:35–87, 1981.
- [Puget, 2005a] Jean-Francois Puget. Breaking symmetries in all different problems. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 272–277. Professional Book Center, 2005.
- [Puget, 2005b] Jean-François Puget. Automatic detection of variable and value symmetries. In van Beek [2005], pages 477–489.
- [Ronay-Dougal *et al.*, 2004] C.M. Ronay-Dougal, I.P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking in using restricted search trees. In Ramon López de Mántaras and Lorenza Saitta, editors, *16th European Conference on Artificial Intelligence, Proceedings, Valencia, Spain*, volume 110, pages 211–215. IOS Press, 2004.
- [Sellman, 2003] M. Sellman. Cost-based filtering for shorter path constraints. In Fransceca Rossi, editor, *Ninth International Conference on Principles and Practice of Constraint Programming, CP 2003, Kinsale, Ireland, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 694–708. Springer-Verlag, october 2003.
- [van Beek, 2005] Peter van Beek, editor. *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, Lecture Notes in Computer Science, Barcelona, Spain, October 2005. Springer.
- [Zampelli *et al.*, 2005] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Approximate constrained subgraph matching. In van Beek [2005], pages 832–836.

Author Index

Aloul, F., 9
Audemard, G., 14

Benhamou, B., 22, 90

Deville, Y., 56, 90
Domoen, B., 30
Dupont, P., 56

Fukunaga, A.S., 39

Garcia de la Banda, M., 30
Grayland, A., 47

Jabbour, S., 14

Lynce, I., 9

Mears, C., 30
Miguel, I., 47

Monette, J.N., 56

Naanaa, W., 64

Prestwich, S., 9

Rouney-Dougal, C., 47
Roy, A., 72

Saïs, L., 14
Saidi, M.R., 22, 90
Schaus, P., 56
Szeider, S., 5

Wallace, M., 30
Walsh, T., 84
Walsh, T., 76

Zampelli, S., 56, 90