Symmetries and Lazy Clause Generation [*]

Geoffrey Chu and **Peter J. Stuckey**
NICTA and University of Melbourne
{gchu,pjs}@csse.unimelb.edu.au

Maria Garcia de la Banda and **Chris Mears**
Monash University
{cmears,mbanda}@infotech.monash.edu.au

Abstract

Lazy clause generation is a powerful approach to reducing search in constraint programming. This is achieved by recording sets of domain restrictions that previously led to failure as new clausal propagators. Symmetry breaking approaches are also powerful methods for reducing search by recognizing that parts of the search tree are symmetric and do not need to be explored. In this paper we show how we can successfully combine symmetry breaking methods with lazy clause generation. Further, we show that the more precise nogoods generated by a lazy clause solver allow our combined approach to exploit redundancies that cannot be exploited via any previous symmetry breaking method, be it static or dynamic.

1 Introduction

Lazy clause generation [Ohrimenko *et al.*, 2009] is a hybrid approach to constraint solving that combines features of finite domain propagation and Boolean satisfiability. In particular, finite domain propagation is instrumented to record the reasons for each propagation step, thus allowing the creation of nogoods that record the reasons for failure, and which can be propagated efficiently using SAT technology. The resulting hybrid system combines the advantages of constraint programming (high level modeling and programmable search) with those of SAT solvers (reduced search by nogood creation and effective search using variable activities), and provides state of the art solutions to a number of combinatorial optimization problems such as Resource Constrained Project Scheduling Problems [Schutt *et al.*, 2009].

Symmetry breaking methods aim at speeding up the execution by pruning parts of the search tree known to be symmetric to those already explored. Static symmetry breaking methods achieve this by adding constraints to the original problem, while dynamic symmetry breaking methods alter the search. As we will see later, combining static symmetry breaking with lazy clause generation is straightforward and quite successful. However, we are also interested in the combination with dynamic symmetry breaking as it can sometimes be more effective. While this combination is more complex, it allows us to exploit certain types of redundancies that cannot be exploited by any other traditional symmetry breaking method alone.

The key to the success of our combination resides in the fact that dynamic symmetry breaking can also be defined in terms of nogoods. In particular, it can be thought of as utilising symmetric versions of nogoods derived at each search node to prune off symmetric portions of the search space. Thus, both lazy clause generation and dynamic symmetry breaking use nogoods to prune the search space. The differences arise in the kind of nogoods used and in the way these nogoods are used. As we will see later, lazy clause solvers [Ohrimenko *et al.*, 2009] use what is called the *first unique implication point* (1UIP) nogood (described in Section 3), which has been empirically found to have stronger pruning than the nogoods used by traditional dynamic symmetry breaking methods. As our theoretical exploration will show, this difference in pruning strength carries over to dynamic symmetry breaking methods. Combining lazy clause generation and dynamic symmetry breaking allows us to take advantage not only of 1UIP nogoods (as lazy evaluation does) but also of symmetric 1UIP nogoods. This leads to strictly more pruning.

2 Finite Domain Propagation

Let $\equiv$ denote syntactic identity and $vars(O)$ denote the set of variables of object O. We use $\Rightarrow$ and $\Leftrightarrow$ to denote logical implication and logical equivalence, respectively.

A *constraint problem* P is a tuple (C, D), where C is a set of constraints and D is a *domain* which maps each variable $x \in vars(C)$ to a finite set of integers $D(x)$. The set C is logically interpreted as the conjunction of its elements, while D is interpreted as $\wedge_{x \in vars(C)} x \in D(x)$. A variable x is said to be Boolean if $D(x) = \{0, 1\}$, where 0 represents *false* and 1 represents *true*.

An *equality literal* of $P \equiv (C, D)$ is of the form $x = d$, where $x \in vars(C)$ and $d \in D(x)$. A *valuation* θ *of* P *over the set of variables* $V \subseteq vars(C)$ is a set of equality literals of P with exactly one literal per variable in V. It can be understood as a mapping of variables to values. The

[*]NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council. This work was partially supported by Asian Office of Aerospace Research and Development 10-4123.

projection of valuation θ over a set of variables $U \subseteq vars(\theta)$ is the valuation $\theta_U = \{x = \theta(x) | x \in U\}$.

A constraint $c \in C$ can be seen as a set of valuations over $vars(c)$. Valuation θ *satisfies* c iff $vars(c) \subseteq vars(\theta)$ and $\theta_{vars(c)} \in c$. A *solution* of P is a valuation over $vars(P)$ that satisfies each constraint in C.

An *inequality literal* of $P \equiv (C, D)$ is of the form $x \le d$ or $x \ge d$ where $x \in vars(C)$ and $d \in D(x)$. A *disequality literal* for x is of the form $x \ne d$ where $d \in D(x)$. The equality, inequality and disequality literals of P, together with the special literal *false* representing failure, are denoted the *literals* of P. Literals represent the basic changes in domain that occur during propagation. Note they are not independent of each other, e.g. $x \le d \wedge x \ge d \Leftrightarrow x = d$.

Each constraint c is implemented by a *propagator*, i.e., a function f_c from domains to domains that ensures that $c \wedge D \Leftrightarrow c \wedge f_c(D)$. We compute the *new* information obtained by running f_c on domain D as the set of literals that are newly implied: $new(f_c, D) = \{\ell \mid D \not\Rightarrow \ell \wedge f_c(D) \Rightarrow \ell\}$. We will assume that we remove from this set literals that are redundant. Note that if the propagator detects failure we assume $new(f_c, D) = \{false\}$.

Example 1 Consider the effect of propagator f_c of constraint $c \equiv \sum_{i=1}^{5} x_i \le 12$ on the domain $D(x_1) = \{1\}, D(x_2) = D(x_3) = D(x_4) = D(x_5) = \{2\,..\,10\}$. Now $D' = f_c(D)$ has $D'(x_2) = D'(x_3) = D'(x_4) = D'(x_5) = \{2\,..\,5\}$. Hence, $new(f_c, D)$ includes $x_2 \ge 2, x_2 \le 5, x_2 \le 6, x_2 \le 7, \ldots$ (for x_2). Since the second literal makes those following redundant, we assume they are not part of the result. □

A constraint programming solver starts from an original problem $P \equiv (C, D)$ and applies propagation to reduce domain D to D' as a fixpoint of all propagators for C. If D' is equivalent to *false*, we say P is failed. If D' fixes all variables, we have found a solution to P (under some reasonable assumptions about propagators). Otherwise, the solver splits P into n subproblems $P_i \equiv (C \wedge c_i, D'), 1 \le i \le n$ where $C \wedge D' \Rightarrow (c_1 \vee c_2 \vee \ldots \vee c_n)$ and where c_i are literals (called *decision literals*), and iteratively searches these. We can identify any subproblem P' appearing in the search tree for P by the set of decision literals $c_{1'}, \ldots, c_{m'}$ taken to reach P'; we call this set $choices(P')$.

3 Lazy Clause Generation

As mentioned before, in lazy clause generation each finite domain propagator is modified so that it explain all its changes to domains in terms of the literals of the problem. An *explanation* for literal ℓ is $S \rightarrow \ell$ where S is a set of literals. A correct explanation for ℓ by f_c propagating on a problem with initial domain D, is an explanation $S \rightarrow \ell$ where $c \wedge S \wedge D \Rightarrow \ell$. For example, the propagator for constraint $x \ne y$ may infer literal $y \ne 3$ given literal $x = 3$ resulting in explanation $\{x = 3\} \rightarrow y \ne 3$.

In a lazy clause generation solver each new literal inferred by a propagator is recorded in a stack in the order of generation and with its explanation attached to it. The *implication graph* is thus a stack of literals each with either an attached explanation or marked as a decision literal. We define the *decision level* for any literal as the number of decision literals pushed in the stack before it.

Example 2 Consider a problem $P \equiv (C, D)$ where $C \equiv \{\sum_{i=1}^{5} x_i \le 12,\ alldiff(\{x_1, x_2, x_3, x_4, x_5\})\}$ and $D(x_i) = \{1\,..\,8\}, 1 \le i \le 5$. If the search chooses $x_1 = 1$ we arrive at subproblem $P_1 = (C \cup \{x_1 = 1\}, D)$. Then, the *alldiff* constraint determines that $x_2 \ne 1$ from set of literals $\{x_1 = 1\}$ (i.e., with explanation $\{x_1 = 1\} \rightarrow x_2 \ne 1$), and similarly for x_3, x_4 and x_5. This builds the second column of the implication graph in Figure 1. Then, the relation between literals for x_2 determine that $x_2 \ge 2$ from $\{x_2 \ne 1\}$ and similarly for the literal relationships of x_3, x_4, and x_5, building the third column. The sum constraint determines that the upper bound of each of x_2, x_3, x_4 and x_5 is 5 from the lower bounds in the third column, thus building the fourth column. The new domain is $D'(x_1) = \{1\}$, $D'(x_i) = \{2\,..\,5\}, 2 \le i \le 5$. If the search now chooses $x_2 = 2$, we arrive at subproblem $P_2 = (C \cup \{x_1 = 1, x_2 = 2\}, D')$. Then, the *alldiff* constraint determines $x_3 \ne 2$, $x_4 \ne 2$, and $x_5 \ne 2$ (the 6th column) from $\{x_2 = 2\}$. The literal relationships determine that $x_3 \ge 3$ from $\{x_3 \ge 2, x_3 \ne 2\}$, similarly for x_4 and x_5. The sum constraint determines that $x_4 \le 3$ from $\{x_2 = 2, x_3 \ge 3, x_5 \ge 3\}$, similarly for $x_5 \le 3$. Then, the literal relationships determine $x_4 = 3$ from $\{x_4 \ge 3, x_4 \le 3\}$, similarly for $x_5 = 3$ and finally the *alldiff* constraint determines unsatisfiability of $x_4 = 3$ and $x_5 = 3$. □

A *nogood* N is a set of literals. A correct nogood N from problem $P \equiv (C, D)$ is one where $C \wedge D \Rightarrow \neg \wedge_{\ell \in N} \ell$, that is, in all solutions of P the conjunction of the literals in N is false. Given an implication graph, we determine a correct nogood that explains $N \rightarrow false$ by eliminating literals from N until only one literal at the current decision level remains. The result is the 1UIP (First Unique Implication Point) nogood, which the search records as a clausal propagator and backtracks to the decision level of the second latest literal in the nogood, where it applies the newly derived nogood.

Example 3 Continuing from Example 2 using the implication graph in Figure 1, we start with the explanation of failure $\{x_4 = 3, x_5 = 3\} \rightarrow false$ which gives us the initial nogood $\{x_4 = 3, x_5 = 3\}$. Since both literals were determined at the current decision level, we replace the last one $x_5 = 3$ by the antecedents in its explanation $\{x_5 \ge 3, x_5 \le 3\} \rightarrow x_5 = 3$ to obtain $\{x_5 \ge 3, x_5 \le 3, x_4 = 3\}$. We keep removing the last literal with the current decision level until only one literal remains at the current decision level. The resulting 1UIP nogood is $\{x_2 \ge 2, x_3 \ge 2, x_4 \ge 2, x_5 \ge 2, x_2 = 2\}$ which can be simplified to $\{x_3 \ge 2, x_4 \ge 2, x_5 \ge 2, x_2 = 2\}$, since the last literal implies the first.

On backtracking to undo $x_2 = 2$, the search arrives at subproblem P_1 and immediately determines that $x_2 \ne 2$ using the new nogood. Importantly, if the search ever reaches a state where $\{x_3 \ge 2, x_4 \ge 2, x_5 \ge 2\}$ holds, it will make the same inference. Indeed, if it reaches a point where $\{x_2 = 2, x_3 \ge 2, x_4 \ge 2\}$, it will infer that $x_5 < 2$. □

4 Symmetries and Nogoods

A *symmetry* of $P \equiv (C, D)$ is a bijection ρ on the equality literals of P such that, for each valuation θ of P, $\rho(\theta) = \{\rho(\ell) \mid \ell \in \theta\}$ is a solution of P iff θ is a solution of P. Variable symmetries, value symmetries and variable-value symmetries are all particular cases of symmetries.

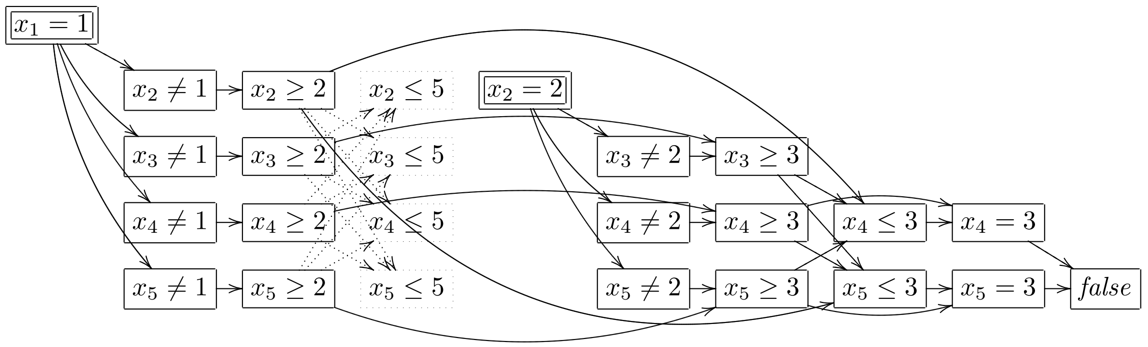

Figure 1: Implication graph of propagation. Decision literals are double boxed.

Static Symmetry Breaking reduces the search by adding to C constraints that remove symmetric solutions. In particular, lexicographical constraints have been used to statically eliminate symmetries (see e.g.[Flener *et al.*, 2002]) with excellent results. This is good news since static symmetry breaking is obviously compatible with lazy clause generation: we only require the new symmetry breaking constraints to have explaining propagators, which are then used just like other propagators. However, static symmetry breaking is not always the best option and, therefore, we are also interested in dynamic symmetry breaking.

Dynamic Symmetry Breaking techniques can be seen as pruning symmetric portions of the search space by using symmetric nogoods. In particular, if the search on subproblem P' fails, then $choices(P')$ is a correct nogood of P (referred to as the *choice nogood*) and, therefore, any symmetric version of $choices(P')$ is also a correct nogood of P. Note that the symmetric version of a nogood N with only equality literals is easy to define: $\rho(N) = \{\rho(\ell) \mid \ell \in N\}$.

Example 4 In problem P of Example 2, the variables $\{x_1, x_2, x_3, x_4, x_5\}$ are interchangeable (i.e., any two can be swapped). Since the subproblem P' with $choices(P') = \{x_1 = 1, x_2 = 2\}$ fails, we know that $\{x_1 = 1, x_2 = 2\}$ is a correct nogood for P. Clearly, any symmetric version, such as $\{x_2 = 1, x_1 = 2\}$ or $\{x_3 = 1, x_5 = 2\}$, is also a correct nogood. $\square$

Such nogoods can be used to prune search in two main ways. Symmetry breaking by dominance detection (SBDD) [Focacci and Milano, 2001; Fahle *et al.*, 2001] keeps a store $\mathbf{N}$ of the non-subsumed choice nogoods derived during search so far. For each subproblem P', it checks whether there exists $N \in \mathbf{N}$ and symmetry ρ, such that $choices(P') \Rightarrow \rho(N)$. If such a pair exists it can immediately fail subproblem P'. Symmetry breaking during search (SBDS) [Gent and Smith, 2000], which is predated by the very similar [Backofen and Will, 1999], works as follows. Whenever a subproblem P' with $choices(P') = \{c_1, c_2, \ldots, c_n, c_{n+1}\}$ fails, SBDS backtracks to the parent subproblem P'' in level n and, for each symmetry ρ, it locally posts in P'' the conditional constraint $(\rho(c_1) \wedge \ldots \wedge \rho(c_n)) \rightarrow \neg\rho(c_{n+1})$. Note that these constraints will only propagate when reaching a subproblem P''' such that $C \cup choices(P''')$ entails the left hand side of the constraint. This will never happen if the symmetry is *broken*, i.e., if $\exists c_i$ s.t. $\neg\rho(c_i)$ is entailed, and that is why SBDS ignores any symmetry ρ which is known to be broken at P''. Still, SBDS can post too many local constraints when the number of symmetries is high. Thus, some incomplete methods ([Mears, 2010; Gent and Smith, 2000]) only post constraints that are known to immediately propagate.

We decided to integrate SBDS, rather than SBDD, with our lazy clause generation because SBDS is much closer to the lazy clause generation approach: they both compute and post nogoods. The main differences being that SBDS only computes decision nogoods and posts their symmetric versions.

5 Symmetries and Lazy Clause Generation

5.1 SBDS-choice

We can naively add SBDS to a lazy clause solver by simply using symmetric versions of the choice nogood at each node to prune off symmetric branches. Hence, we just re-implement standard SBDS in the lazy clause generation solver, but still gain the advantage of reduced search through the lazy clause generation nogoods.

5.2 SBDS-1UIP

Adapting SBDS to use 1UIP nogoods is simple: every time a 1UIP nogood $\{\ell_1, \ldots, \ell_n\} \rightarrow \ell_{n+1}$ is inferred for subproblem P', upon backtracking to parent P'' and for each symmetry ρ, we post the symmetric nogood $\{\rho(\ell_1), \ldots, \rho(\ell_n)\} \rightarrow \neg\rho(\ell_{n+1})$, ignoring those ρ that are known to be broken at P''. We can check this last condition during the construction of the symmetric nogood, as we produce the literals $\rho(\ell_1), \ldots, \rho(\ell_n)$ one at a time. If at any point, one of $\rho(\ell_i)$ is false in P'', we can immediately abort and move on to the next symmetry.

In contrast to SBDS-choice, in SBDS-1UIP we have to post the symmetric nogoods as global rather than local constraints. This is because in SBDS-choice, when you backtrack from parent P'' to grandparent P''', the choice nogood at P''' subsumes that at P'' and, therefore, SBDS-choice will always post a set of symmetric nogoods that subsumes the symmetric nogoods posted below that point. In contrast, there is no guarantee that the 1UIP nogood at P''' subsumes the one at P'' (and in general it does not). This means that many nogoods can be are added at each node. However, this is not a big problem. Nogood propagation in LCG solvers is very

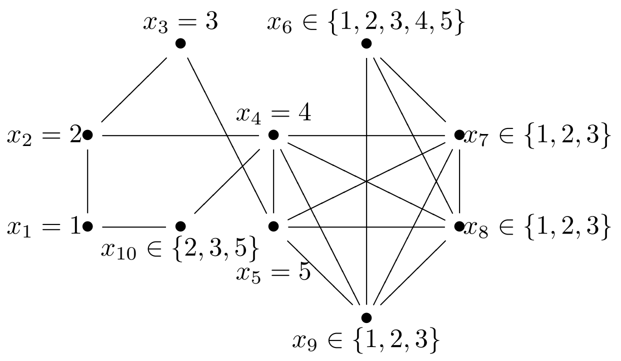

Figure 2: A graph colouring problem where we can exploit additional symmetries

fast since they use the same efficient handling of clauses used in SAT solvers. In addition, activity counts are kept for each nogood and inactive nogoods are periodically pruned just as in SAT solvers.

Example 5 Consider the problem of Example 2. On backtracking to P_1 we infer the nogood $\{x_3 \geq 2, x_4 \geq 2, x_5 \geq 2\} \rightarrow x_2 \neq 2$. With this we not only infer $x_2 \neq 2$ but also the symmetric inferences $x_3 \neq 2$ (from $\{x_2 \geq 2, x_4 \geq 2, x_5 \geq 2\}$), $x_4 \neq 2$ and $x_5 \neq 2$. At this point, a domain consistent *alldiff* will determine failure and generate $\emptyset \rightarrow x_1 \neq 1$, which does not subsume the previously generated nogood. □

We show that SBDS-1UIP exploits strictly more symmetries than SBDS-choice if the asserting literals in the nogoods are the same, and propagation has the following property:

Definition 1 A set of propagators for problem P has *global symmetric monotonicity* iff, for any explanation $\{c_1, \ldots, c_n\} \rightarrow \ell$ produced and any symmetry ρ of P, whenever $\rho(c_1), \ldots, \rho(c_n)$ are entailed, $\rho(\ell)$ is also entailed. □

A sufficient condition for global symmetric monotonicity is the following: all propagators are monotonic, and all symmetries are *propagator symmetric* (i.e., the symmetry maps propagators to propagators). Global symmetric monotonicity is therefore very common, as most propagators are monotonic, and the vast majority of symmetries that are usually exploited are propagator symmetric.

Theorem 1 *Suppose global symmetric monotonicity holds, and we derive choice nogood $\{c_1, \ldots, c_n\} \rightarrow \neg c_{n+1}$, and 1UIP nogood $\{\ell_1, \ldots, \ell_m\} \rightarrow \neg c_{n+1}$ from the same conflict. If $\{\rho(c_1), \ldots, \rho(c_n)\} \rightarrow \neg\rho(c_{n+1})$ propagates then so does $\{\rho(\ell_1), \ldots, \rho(\ell_m)\} \rightarrow \neg\rho(c_{n+1})$.* □

The theorem shows that the symmetric 1UIP nogood subsumes the symmetric choice nogood, since it will always produce any implication that the symmetric choice nogood can, but not vice versa. This means that SBDS-1UIP can exploit strictly more symmetry than SBDS-choice. This result is valid for both complete SBDS, as well as for incomplete SBDS methods which only post nogoods that will immediately produce an implication.

Example 6 Consider the graph colouring problem shown in Figure 2, where we are trying to colour the nodes with at most 5 colours (all of which are interchangeable). After making the decisions $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5$, we have domains as shown in Figure 2. Suppose we label $x_6 = 1$ next. Then propagation gives $x_7 \in \{2,3\}, x_8 \in \{2,3\}, x_9 \in \{2,3\}$. Now, suppose we try $x_7 = 2$. This forces $x_8 = 3, x_9 = 3$, which conflicts. The 1UIP nogood from this conflict is $\{x_8 \neq 1, x_8 \neq 4, x_8 \neq 5, x_9 \neq 1, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_7 \neq 2$. After propagating this nogood, we have $x_7 = 3$, which after further propagation, once again conflicts. At this point, we backtrack to before x_6 is labelled and derive the nogood $\{x_7 \neq 4, x_7 \neq 5, x_8 \neq 4, x_8 \neq 5, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_6 \neq 1$.

Now, let us examine what SBDS-1UIP can do at this point. It is clear that if we apply the value symmetries: 1 is symmetric to 2 ($\ll 1 \gg \leftrightarrows \ll 2 \gg$), or $\ll 1 \gg \leftrightarrows \ll 3 \gg$ to this nogood; the LHS remains unchanged while the RHS changes. Therefore, we can post these two symmetric nogoods and immediately get the inferences $x_6 \neq 2$ and $x_6 \neq 3$. On the other hand, SBDS-choice cannot do anything. The choice nogood is $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5\} \rightarrow x_6 \neq 1$, and it is easy to see that no matter which value symmetry we use on it, the LHS will have a set of literals incompatible with the current set of decisions and, thus, cannot imply the RHS. □

The kind of redundancy we exploit here certainly arises from symmetry. However, it is extremely difficult to exploit. Roughly speaking, we can say that we are exploiting the symmetry that exists in the sub-component of a subproblem which is the actual cause of failure. In this case, they are the variables x_6, x_7, x_8, x_9, their current domains in the subproblem, and the constraints linking them. Even conditional symmetry breaking constraints are powerless to exploit such symmetries, as the subproblem shown in Figure 2 does not have the value symmetries $\ll 1 \gg \leftrightarrows \ll 2 \gg$ or $\ll 1 \gg \leftrightarrows \ll 3 \gg$ due to the existence of x_{10}. It is only because a lazy clause solver gives us such precise information about which variables are involved in failures that we can exploit this kind of redundancy. While the above example might seem contrived, our experiments in Section 7 show that these kinds of redundancies do occur in practice and can yield big speedups.

6 Symmetries on 1UIP nogoods

While SBDS-1UIP is much more powerful than SBDS-choice, having to manipulate 1UIP nogoods has its problems: unlike the choice nogoods which usually only involve equality literals on search variables, 1UIP nogoods can contain virtually any literal in the problem, including disequality and inequality literals, and also literals with intermediate variables.

6.1 Disequality and Inequality Literals

One of the strengths of lazy clause generation is the use of equality, disequality and inequality literals in explanations and nogoods. This can make explanations shorter and is essential for explaining bounds propagation. Thus, we need to extend symmetries to work with all literals.

Extending symmetry ρ to handle disequalities is straightforward: if $\rho(x = d) \equiv x' = d'$, then $\rho(x \neq d) \equiv x' \neq d'$. Extending ρ to handle inequalities is also straightforward for a *variable* symmetry σ: if $\rho(x = d) \equiv x' = d$, then $\rho(x \leq d) \equiv x' \leq d$ and $\rho(x \geq d) \equiv x' \geq d$. However, it is not clear how to extend *value* and *variable-value* symmetries. Thus, before we apply such symmetries we transform nogoods by replacing any of their inequality literals by an equivalent set (conjunction) of disequality literals (e.g. $x \leq 3$ is replaced by $x \neq 4, x \neq 5, x \neq 6$ assuming an initial domain $[1..6]$). Note that, while this transformation is theoretically correct, it may create very unwieldy nogoods.

6.2 Intermediate variables

An important problem for combining dynamic symmetry breaking and lazy clause generation is the fact that intermediate variables may be introduced in the course of converting a high level model to the low level variables and constraints implemented by the solver. If the symmetry declaration was made in the high level model, it may not specify how literals on the intermediate variables map to each other. Thus, we must extend symmetries to include such literals.

In some cases intermediate variables are idempotent under the symmetries, i.e., for each symmetry ρ of P, we can extend ρ to ρ' where $\rho'(\ell) = \rho(\ell)$ if $vars(\ell) \subseteq vars(C)$ and $\rho'(\ell) = \ell$ otherwise. The extended ρ' is a symmetry of P with intermediate variables. We can imagine automating the proof of idempotence of intermediate variables under symmetries.

Example 7 Consider a model for concert hall scheduling. Every two values in $\{1..k\}$ for the k identical concert halls are symmetrical (can be swapped). The model includes the constraint $x_i = k+1 \vee x_j = k+1 \vee x_i \neq x_j$ for all concerts i and j which overlap in time. In a flattened form we introduce new Boolean variables $diff_{i,j} \Leftrightarrow x_i \neq x_j$ and $noton_i \Leftrightarrow x_i = k+1$ (indicating concert i is not scheduled) to represent disjuncts of this constraint. Since each introduced $noton_i$ and $diff_{i,j}$ variable is idempotent under the value symmetries, any symmetry ρ on the original variables can be trivially extended. □

Sometimes we can manually extend our symmetry declarations to take into account the intermediate variables. Other times it is not easy to see how to extend symmetries to all intermediate variables, and indeed quite often intermediate variables are introduced far below the modelling level. In order to handle these cases we modify learning as follows. We extend the model to explicitly mark which literals are allowed to appear in nogoods. We then modify the learning process to always explain unmarked literals. There is a requirement that all literals generated by search are allowed to appear in nogoods. This ensures that the process always terminates and always generates an asserting nogood.

In summary the SBDS-1UIP method is as follows:

- Identify any intermediate variables introduced by flattening or by the solver
- Attempt to extend any symmetry declarations to include these variables
- If the last step is not possible, make a declaration to the solver that literals on these intermediate variables are not allowed to be used in nogoods
- When a 1UIP nogood is derived by standard conflict analysis, examine all symmetric versions of it and post any that can immediately propagate
- Handle these extra nogoods in the same way as normal nogoods, i.e. post as clausal propagators, periodically prune inactive ones

7 Experiments

We now provide experimental evidence that symmetry breaking with 1UIP nogoods is superior to using only choice nogoods and that exploiting the redundancies such as those in Example 6 gives improved performance. The two problem we examine are the Concert Hall Scheduling problem (which has value symmetries as explained in Example 7 and variable symmetries for identical concerts) and the Graph Colouring problem (which has value symmetry on colors, and variable symmetries for nodes with the same set of neighbours). We take the benchmarks originally used by the authors of [Law *et al.*, 2007]. The benchmarks are available at `http://www.cmears.id.au/symmetry/symcache.tar.gz`.

We implemented SBDS in CHUFFED, which is a state of the art lazy clause solver. We run CHUFFED with three different versions of SBDS: choice, where we use symmetric versions of choice nogoods, 1UIP, where we use symmetric versions of 1UIP nogoods, and crippled, where we use symmetric versions of 1UIP nogoods, but only those nogoods derived from symmetries where choice could also exploit the symmetry. We compare against CHUFFED with no symmetry breaking (none) and with standard lexicographical symmetry breaking constraints (static). Finally, we compare against an implementation of SBDS in [Mears, 2010] called Lightweight Dynamic Symmetry Breaking (LDSB), which is implemented in Eclipse and is the fastest dynamic symmetry breaking implementation on the two problems we examine, beating GAP-SBDS and GAP-SBDD by significant margins.

All versions of CHUFFED were run on Xeon Pro 2.4GHz processors. The results for LDSB were run on a Core i7 920 2.67 GHz processor. Times displayed are the average run times for all instances of each size. Instances which timeout (at 600s) are counted as 600 seconds.

The results are shown in Tables 1 and 2. Note that the Eclipse solver reports nodes, the number of internal nodes, but since the tree is binary and there are not many solutions this is comparable with failure counts. Eclipse LDSB fails to solve many instances before timeout, and choice fails to solve a few instances. 1UIP, crippled and static all solve every instance in the benchmarks. In fact, this set of instances, which is of an appropriate size for normal CP solvers, is a bit too easy for lazy clause solvers such as CHUFFED, as is apparent from the run times.

Comparison between choice and 1UIP shows that SBDS-1UIP is superior to SBDS-choice. Comparison between crippled and 1UIP shows that the additional symmetries that we can only exploit with SBDS-1UIP indeed gives us reduced search and additional speedup. Comparison with static shows that dynamic symmetry breaking can be superior to static symmetry breaking on appropriate problems. The comparison with LDSB shows that lazy clause with symmetry breaking explores orders of magnitude less nodes.

References

[Backofen and Will, 1999] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *CP1999*,

Table 1: Comparison of symmetry breaking approaches for the Concert Hall Scheduling problem

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	259.8	686018	0.04	84	0.05	130	0.07	350	0.05	134	0.29	3283
22	381.5	749462	0.07	181	0.08	299	0.17	1207	0.07	183	0.73	7786
24	576.9	1438509	0.10	275	0.11	316	0.78	3426	0.15	486	2.70	12611
26	483.4	1189930	0.10	282	0.19	677	2.26	5605	0.25	685	2.71	12724
28	530.7	1282797	0.68	1611	1.12	2613	3.64	10530	0.42	1041	9.94	57284
30	581.3	1251980	0.27	761	0.53	2042	19.52	48474	0.52	2300	121.50	722668
32	542.4	936019	0.40	1522	1.01	4845	21.48	65157	1.31	5712	97.90	641071
34	600.0	1039051	1.10	2636	3.22	8761	19.86	48837	1.60	4406	72.73	425718
36	600.0	1223864	1.40	3156	5.02	13606	59.70	131142	2.37	5707	171.14	938439
38	600.0	1027778	1.91	5053	12.56	26556	82.77	178170	3.51	10518	268.05	1211086
40	600.0	1447604	2.96	6648	10.27	27028	102.1	219454	6.40	18169	240.84	1220934

Table 2: Comparison of symmetry breaking approaches for the Graph Colouring problems

Uniform

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
30	140.7	282974	0.00	14	0.06	474	0.26	3049	0.02	277	19.63	56577
32	211.4	390392	0.00	17	0.00	146	0.24	3677	0.00	84	14.11	27178
34	213.9	272772	0.00	25	0.29	1182	3.53	11975	0.03	433	22.06	30127
36	195.9	296358	0.00	36	0.04	467	6.91	23842	0.01	200	35.66	85505
38	224.0	297138	0.00	55	0.04	516	23.55	69480	0.03	526	51.18	107574
40	250.9	423326	0.00	83	0.31	1879	21.07	78918	0.06	878	84.16	185707

Biased

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	13.25	39551	0.00	27	0.00	32	0.01	639	0.00	29	0.72	1376
22	11.53	63984	0.00	25	0.00	34	0.02	727	0.00	25	0.16	538
24	66.60	154409	0.00	35	0.00	47	0.07	1992	0.00	32	1.91	2114
26	74.77	277290	0.00	55	0.00	93	0.12	3385	0.00	104	9.56	34210
28	130.5	280649	0.00	62	0.00	84	0.58	6402	0.00	103	9.14	37738
30	267.6	480195	0.00	101	0.01	239	10.48	43835	0.01	359	57.18	215932
32	331.7	600772	0.01	232	0.24	1597	9.98	44216	0.16	1864	110.16	288707
34	219.9	387213	0.20	806	0.40	1946	10.26	47470	0.45	1730	64.14	165943
36	442.6	709888	0.01	317	0.04	857	27.39	113252	0.80	3226	152.62	327472
38	382.5	631403	0.10	798	1.01	5569	31.63	138787	4.12	9413	193.40	508164
40	465.6	531285	0.02	410	0.36	2660	24.68	91847	0.12	2133	196.02	476486

pages 73–87, 1999.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *CP2001*, pages 93–107, 2001.

[Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP2002*, pages 462–476, 2002.

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *CP2001*, pages 77–92, 2001.

[Gent and Smith, 2000] I. Gent and B.M. Smith. Symmetry breaking in constraint programming. In *ECAI 2000*, pages 599–603, 2000.

[Law *et al.*, 2007] Y. C. Law, J. H. M. Lee, T. Walsh, and J. Y. K. Yip. Breaking symmetry of interchangeable variables and values. In *CP2007*, pages 423–437, 2007.

[Mears, 2010] C. Mears. *Automatic Symmetry Detection and Dynamic Symmetry Breaking for Constraint Programming*. PhD thesis, Clayton School of Information Technology, Monash University, 2010.

[Ohrimenko *et al.*, 2009] O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[Schutt *et al.*, 2009] A. Schutt, T. Feydy, P.J. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In *CP2009*, pages 746–761, 2009.