Systematically Identifying and Exploiting Dominance Relations

Geoffrey Chu and Peter J. Stuckey

National ICT Australia, Victoria Laboratory,
Department of Computer Science and Software Engineering,
University of Melbourne, Australia
{gchu,pjs}@csse.unimelb.edu.au

Abstract. Many constraint problems exhibit *dominance relations* which can be exploited for dramatic reductions in search space. Dominance relations are a generalization of symmetry and conditional symmetry. Unlike symmetry breaking which is relatively well studied, dominance breaking techniques are not very well understood and are not commonly applied. In this paper, we present formal definitions of dominance breaking and a systematic method for identifying and exploiting dominance relations via *dominance breaking constraints*. We also give a generic proof of the correctness and compatibility of symmetry breaking constraints, conditional symmetry breaking constraints and dominance breaking constraints.

1 Introduction

In a constraint satisfaction or optimization problem, dominance relations describe pairs of assignments where one is known to be at least as good as the other with respect to satisfiability or the objective function. When such dominance relations are known, we can often prune off many of the solutions without changing the satisfiability or the optimal value of the problem. Many constraint problems exhibit dominance relations which can be exploited for significant speedups (e.g., [13, 19, 3, 18, 6, 9, 16, 18]).

Dominance relations are a generalization of symmetry and conditional symmetry and offer similar or greater potential for reductions in search space. Unlike symmetries however, dominance relations are not very widely exploited. Dominance relations can be hard to identify, and there are few standard methods for exploiting them. It it also often hard to prove that a particular method is correct, especially when multiple dominance relations are being exploited simultaneously. These issues have been overcome in the case of symmetry, which is why symmetry breaking is now standard and widely used. Dominance relations have been successfully applied in a number of problems, but their treatment is often very problem specific and yields little insight as to how they can be generalized. In this paper, we seek to advance the usage of dominance relations by making the following contributions:

- We describe a systematic way of identifying and exploiting a certain class of dominance relations using *dominance breaking constraints*.

– We show that our method naturally produces symmetry breaking and conditional symmetry breaking constraints as well (since they are simply special cases of dominance breaking).
– We give a generic theorem proving the correctness and compatibility of all symmetry breaking, conditional symmetry breaking and dominance breaking constraints generated by our method.

The layout of the paper is as follows. In Section 2, we give our definitions. In Section 3, we describe our method of identifying and exploiting dominance relations using dominance breaking constraints. In Section 4, we describe how our method can be extended to generate symmetry and conditional symmetry breaking constraints as well. In Section 5, we discuss related work. In Section 6, we provide experimental results. In Section 7, we discuss future work. In Section 8, we conclude.

2 Definitions

To facilitate rigorous proofs in the later sections, we will give our own definitions of variables, domains, constraints, constraint problems and dominance relations. These are be slightly different from the standard definitions but are equivalent to them in practice.

Let $\equiv$ denote syntactical identity, $\Rightarrow$ denote logical implication and $\Leftrightarrow$ denote logical equivalence. We define variables and constraints in a problem independent way. A variable v is a mathematical quantity capable of assuming any value from a set of values called the *default domain* of v. Each variable is typed, e.g., Boolean or Integer, and its type determines its default domain, e.g., $\{0, 1\}$ for Boolean variables and $\mathbb{Z}$ for Integer variables. Given a set of variables V, let Θ_V denote the set of valuations over V where each variable in V is assigned to a value in its default domain. A constraint c over a set of variables V is defined by a set of valuations $solns(c) \subseteq \Theta_V$. Given a valuation θ over $V' \supset V$, we say θ satisfies c if the restriction of θ onto V is in $solns(c)$. Otherwise, we say that θ violates c. A domain D over variables V is a set of *unary constraints*, one for each variable in V. In an abuse of notation, if a symbol A refers to a set of constraints $\{c_1, \ldots, c_n\}$, we will often also use the symbol A to refer to the constraint $c_1 \wedge \ldots \wedge c_n$. This allows us to avoid repetitive use of conjunction symbols.

A *Constraint Satisfaction Problem* (CSP) is a tuple $P \equiv (V, D, C)$, where V is a set of variables, D is a domain over V, and C is a set of n-ary constraints. A valuation θ over V is a *solution* of P if it satisfies every constraint in D and C. The aim of a CSP is to find a solution or to prove that none exist. In a *Constraint Optimization Problem* (CSOP) $P \equiv (V, D, C, f)$, we also have an objective function f mapping Θ_V to an ordered set, e.g., $\mathbb{Z}$ or $\mathbb{R}$, and we wish to minimize or maximize f over the solutions of P. In this paper, we deal with finite domain problems only, i.e., where the initial domain D constrains each variable to take values from a finite set of values.

We define dominance relations over full valuations. We assume that all objective functions are to be minimized, and consider constraint satisfaction problems as constraint optimization problems with $f(\theta) = 0$ for all valuations θ.

Definition 1. *A* dominance relation $\prec$ *for CSOP* $P \equiv (V, D, C, f)$ *is a* transitive *and* irreflexive *binary relation on* Θ_V *such that if* $\theta_1 \prec \theta_2$*, then either: 1)* θ_1 *is a solution and* θ_2 *is a non-solution, or 2) they are both solutions or both non-solutions and* $f(\theta_1) \leq f(\theta_2)$*.*

If $\theta_1 \prec \theta_2$, we say that θ_1 *dominates* θ_2. Note that we require our dominance relations to be irreflexive. This means that no loops can exist in the dominance relation, and makes it much easier to ensure the correctness of the method. The following theorem states that it is correct to prune all dominated assignments.

Theorem 1. *Given a finite domain CSOP* $P \equiv (V, D, C, f)$*, and a dominance relation* $\prec$ *for* P*, we can prune all assignments* θ *such that* $\exists \theta'$ *s.t.* $\theta' \prec \theta$*, without changing the satisfiability or optimal value of* P*.*

Proof. Let θ_0 be an optimal solution. If θ_0 is pruned, then there exists some solution θ_1 s.t. $\theta_1 \prec \theta_0$. Then θ_1 must be a solution with $f(\theta_1) \leq f(\theta_0)$, so θ_1 is also an optimal solution. In general, if θ_i is pruned, then there must exist some θ_{i+1} s.t. $\theta_{i+1} \prec \theta_i$ and θ_{i+1} is also an optimal solution. Since $\prec$ is transitive and irreflexive, it is impossible for the sequence $\theta_0, \theta_1, \ldots$ to repeat. Then since there are finitely many solutions, the sequence must terminate in some θ_k which is an optimal solution and which is not pruned. □

We can extend $\prec$ to relate search nodes in the obvious way.

Definition 2. *Let* D_1 *and* D_2 *be the domains from two different search nodes. If* $\forall \theta_2 \in \mathit{solns}(D_2), \exists \theta_1 \in \mathit{solns}(D_1)$ *s.t.* $\theta_1 \prec \theta_2$*, then we define* $D_1 \prec D_2$*.*

Clearly if $D_1 \prec D_2$, Theorem 1 tells us that we can safely prune the search node with D_2. We call the pruning allowed by Theorem 1 *dominance breaking* in keeping with *symmetry breaking* for symmetries.

Dominance relations can be derived either statically before search or dynamically during search in order to prune the search space. It is easy to see that the lex-leader method of symmetry breaking is a special case of static dominance breaking. Suppose S is a symmetry group of problem P. Suppose $lex(\theta)$ is the lexicographical function being used in the lex-leader method. We can define a dominance relation: $\forall \sigma \in S, \forall \theta, \sigma(\theta) \prec \theta$ if $lex(\sigma(\theta)) < lex(\theta)$. Then applying Theorem 1 to $\prec$ gives the lex-leader symmetry breaking constraint (i.e., prune all solutions which are not the lex-leader in their equivalence class). Similarly, dynamic symmetry breaking techniques such as Symmetry Breaking During Search [12] and Symmetry Breaking by Dominance Detection [7] are special cases of dynamic dominance breaking. Nogood learning techniques such as Lazy Clause Generation [17, 8] and Automatic Caching via Constraint Projection [5] are also examples of dynamic dominance breaking. We will discuss these two methods in more detail in Section 5.

Just as in the case of symmetry breaking, it is generally *incorrect* to simultaneously post dominance breaking constraints for multiple dominance relations. This is because dominance relations only ensure that one assignment is at least as good as the other (not strictly better than), thus when we have multiple dominance relations, we could have loops such as $\theta_1 \prec_1 \theta_2$ and $\theta_2 \prec_2 \theta_1$, and posting the dominance breaking constraint for both $\prec_1$ and $\prec_2$ would be wrong. We

have to take care when breaking symmetries, conditional symmetries and dominances that all the pruning we perform are compatible with each other. As we shall show below, one of the advantages of our method is that all the symmetry breaking, conditional symmetry breaking and dominance breaking constraints generated by our method are provably compatible.

Dominance breaking constraints can be particularly useful in optimization problems, because they provide a completely different and complementary kind of pruning to the branch and bound paradigm. In the branch and bound paradigm, the only way to show that a partial assignment is suboptimal is to prove a sufficiently strong bound on its objective value. Proving such bounds can be very expensive, especially if the model does not propagate strong bounds on the objective. In the worse case, further search is required, which can take an exponential amount of time. On the other hand, dominance breaking can prune a partial assignment without having to prove any bounds on its objective value at all, since it only needs to know that the partial assignment is suboptimal. Once dominance relations expressing conditions for suboptimality are found and proved, the only cost in the search is to check whether a partial assignment is dominated, which can often be much lower than the cost required to prove a sufficiently strong bound to prune the partial assignment.

3 Systematically Identifying and Exploiting Dominance Relations

We now describe a method for systematically identifying and exploiting a fairly large class of dominance relations using dominance breaking constraints. The idea is to use mappings σ from valuations to valuations to construct dominance relations. Given a mapping σ, we ask: under what conditions does σ map a solution to a better solution? If we can find these conditions, then we can build a dominance relation using these conditions and exploit it by posting a dominance breaking constraint. More formally:

Step 1 Find mappings $\sigma : \Theta_V \rightarrow \Theta_V$ which are likely to map solutions to better solutions.
Step 2 For each σ, find a constraint $scond(\sigma)$ s.t. if $\theta \in solns(C \wedge D \wedge scond(\sigma))$, then $\sigma(\theta) \in solns(C \wedge D)$.
Step 3 For each σ, find a constraint $ocond(\sigma)$ s.t. if $\theta \in solns(C \wedge D \wedge ocond(\sigma))$, then $f(\sigma(\theta)) < f(\theta)$.
Step 4 For each σ, post the dominance breaking constraint $db(\sigma) \equiv \neg(scond(\sigma) \wedge ocond(\sigma))$ if it is simple and efficient enough to propagate.

The following theorem proves the correctness of this method.

Theorem 2. *Given a finite domain CSOP $P \equiv (V, D, C, f)$, a set of mappings S, and for each mapping $\sigma \in S$ constraints $scond(\sigma)$ and $ocond(\sigma)$ satisfying: $\forall \sigma \in S$, if $\theta \in solns(C \wedge D \wedge scond(\sigma))$, then $\sigma(\theta) \in solns(C \wedge D)$, and: $\forall \sigma \in S$, if $\theta \in solns(C \wedge D \wedge ocond(\sigma))$, then $f(\sigma(\theta)) < f(\theta)$, we can add all of the dominance breaking constraints $db(\sigma) \equiv \neg(scond(\sigma) \wedge ocond(\sigma))$ to P without changing its satisfiability or optimal value.*

Proof. Construct a binary relation $\prec$ as follows. For each σ, for each $\theta \in solns(C \wedge D \wedge scond(\sigma) \wedge ocond(\sigma))$, define $\sigma(\theta) \prec \theta$. Now, take the transitive closure of $\prec$. We claim that $\prec$ is a dominance relation. It is transitive by construction. Also, by construction, $\theta \in solns(C \wedge D \wedge scond(\sigma) \wedge ocond(\sigma))$ guarantees that $\sigma(\theta)$ is a solution and that $f(\sigma(\theta)) < f(\theta)$. Thus $\forall \theta_1, \theta_2, \theta_1 \prec \theta_2$ implies that θ_1 and θ_2 are solutions, and that $f(\theta_1) < f(\theta_2)$. This means that $\prec$ is irreflexive and satisfies all the properties of a dominance relation, thus by Theorem 1, we can prune any $\theta \in solns(C \wedge D \wedge scond(\sigma) \wedge ocond(\sigma))$ for any σ without changing the satisfiability or optimality of P. Thus it is correct to add $db(\sigma)$ for any σ to P. □

Note that there are no restrictions on σ. It does not have to be injective or surjective. The $db(\sigma)$ are guaranteed to be compatible because they all obey the same strict ordering imposed by the objective function f, i.e., they prune a solution only if a solution with strictly better f value exists. We illustrate the method with two simple examples before we go into more details.

Example 1. Consider the Photo problem. A group of people wants to take a group photo where they stand in one line. Each person has preferences regarding who they want to stand next to. We want to find the arrangement which satisfies the most preferences.

We can model this as follows. Let $x_i \in \{1, \ldots, n\}$ for $i = 1, \ldots, n$ be variables where x_i represent the person in the ith place. Let p be a 2d integer array where $p[i][j] = p[j][i] = 2$ if person i and j both want to stand next to each other, $p[i][j] = p[j][i] = 1$ if only one of them wants to stand next to the other, and $p[i][j] = p[j][i] = 0$ if neither want to stand next to each other. The only constraint is: $alldiff(x_1, \ldots, x_n)$. The objective function is given by: $f = p[x_1][x_2] + \ldots + p[x_{n-1}][x_n]$.

Step 1 Since this is a sequence type problem, mappings which are likely to map solutions to solutions are things that permute the sequence in some way. For simplicity, consider the set of mappings which flip a subsequence of the sequence, i.e., $\forall i < j, \sigma_{i,j}$ maps x_i to x_j, x_{i+1} to x_{j-1}, …, x_j to x_i.

Step 2 We want to find the conditions under which σ maps solutions to solutions. Since all of these σ are symmetries of $C \wedge D$, we do not need any conditions and it is sufficient to set $scond(\sigma_{i,j}) \equiv true$.

Step 3 We want to find the conditions under which $f(\sigma_{i,j}(\theta)) < f(\theta)$. If we compare the LHS and RHS, it is clear that the only difference is the terms $p[x_{i-1}][x_j] + p[x_i][p_{j+1}]$ on the LHS and the terms $p[x_{i-1}][x_i] + p[x_j][p_{j+1}]$ on the RHS. So it is sufficient to set $ocond(\sigma_{i,j}) \equiv p[x_{i-1}][x_j] + p[x_i][p_{j+1}] < p[x_{i-1}][x_i] + p[x_j][p_{j+1}]$.

Step 4 For each $\sigma_{i,j}$, we can post the dominance breaking constraint: $\neg(p[x_{i-1}][x_j] + p[x_i][p_{j+1}] < p[x_{i-1}][x_i] + p[x_j][p_{j+1}])$.
These dominance breaking constraints ensure that if some subsequence of the assignment can be flipped to improve the objective, then the assignment is pruned. □

Example 2. Consider the 0-1 knapsack problem where x_i are 0-1 variables, we have constraint $\sum w_i x_i \leq W$ and we have objective $f = -v_i x_i$, where w_i and v_i are constants.

Step 1 Consider mappings which swap the the values of two variables, i.e., $\forall i < j, \sigma_{i,j}$ swaps x_i and x_j.

Step 2 A sufficient condition for $\sigma_{i,j}$ to map the current solution to another solution is: $scond(\sigma_{i,j}) \equiv w_i x_j + w_j x_i \leq w_i x_i + w_j x_j$. Rearranging, we get: $(w_i - w_j)(x_i - x_j) \geq 0$.

Step 3 A sufficient condition for $\sigma_{i,j}$ to map the current solution to an assignment with a better objective function is: $ocond(\sigma_{i,j}) \equiv v_i x_j + v_j x_i > v_i x_i + v_j x_j$. Rearranging, we get: $(v_i - v_j)(x_i - x_j) < 0$.

Step 4 For each $\sigma_{i,j}$, we can post the dominance breaking constraint: $db(\sigma_{i,j}) \equiv \neg(scond(\sigma_{i,j}) \wedge ocond(\sigma_{i,j}))$. After simplifying, we have $db(\sigma_{i,j}) \equiv x_i \leq x_j$ if $w_i \geq w_j$ and $v_i < v_j$, $db(\sigma_{i,j}) \equiv x_i \geq x_j$ if $w_i \leq w_j$ and $v_i > v_j$, and $db(\sigma_{i,j}) \equiv true$ for all other cases.

These dominance breaking constraints ensure that if one item has worse value and greater or equal weight to another, then it cannot be chosen without choosing the other also. □

3.1 Step 1: Finding Appropriate Mappings σ

In general, we want to find σ's such that $scond(\sigma)$ and $ocond(\sigma)$ are as small and simple as possible, as this will lead to dominance breaking constraints that are easier to propagate and prune more. So we want σ such that it often maps a solution to a better solution. σ's which are symmetries or almost symmetries of the problem make good candidates, since their $scond(\sigma)$ will be simple, and all else being equal, there is around a 50% chance that it will map the solution to one with a better objective value. In general, we can try all the common candidates for symmetries such as swapping two variables, swapping two values, swapping two rows/columns in matrix type problems, flipping/moving a subsequence in a sequence type problem, etc. In scheduling type problems, we can also try shifting items forwards or backwards in the schedule. There may also be problem specific σ's that we can try.

3.2 Step 2: Finding $scond(\sigma)$

We can calculate $scond(\sigma)$ straightforwardly with the help of the following definition.

Definition 3. *Given a mapping $\sigma : \Theta_V \to \Theta_V$, we can extend σ to map constraints to constraints as follows. Given a constraint c, $\sigma(c)$ is defined as a constraint over V such that θ satisfies $\sigma(c)$ iff $\sigma(\theta)$ satisfies c.*

For example, if $c \equiv x_1 + 2x_2 + 3x_3 \geq 10$, and σ swaps x_1 and x_3, then $\sigma(c) \equiv x_3 + 2x_2 + 3x_1 \geq 10$. Or if $c \equiv (x_1, x_2) \in \{(1,1),(2,3),(3,1)\}$, and σ permutes the values (1, 2, 3) to (2, 3, 1) on x_1 and x_2, then $\sigma(c) \equiv (x_1, x_2) \in \{(3,3),(1,2),(2,3)\}$.

It is easy to define $\sigma(c)$, however, $\sigma(c)$ may or may not be a simple logical expression. For example, if $c \equiv x_1 + 2x_2 \geq 5$ and σ swaps the values 1 and 2, then $\sigma(c) \equiv (x_1 = 1 \wedge x_2 = 1) \vee (x_1 = 2 \wedge x_2 = 1) \vee \vee (x_1 \neq 1 \wedge x_1 \neq 2 \wedge x_2 \neq 1 \wedge x_2 \neq 2 \wedge x_1 + 2x_2 \geq 5)$ which does not simplify at all.

For each σ, we want to find a sufficient condition $scond(\sigma)$ so that if a solution satisfied $scond(\sigma)$, then σ maps it to another solution. A necessary and sufficient condition is: $C \wedge D \wedge scond(\sigma) \Rightarrow \sigma(C \wedge D)$, i.e., C and D together with $scond(\sigma)$ must imply the mapped versions of every constraint in C and D.

We can construct $scond(\sigma)$ as follows. We calculate $\sigma(c)$ for each $c \in C \cup D$. If it is not implied by $C \wedge D$, then we add a constraint c' to $scond(\sigma)$ such that $C \wedge D \wedge c' \rightarrow \sigma(c)$. For example, in the knapsack problem in Example 2, $\sigma_{i,j}(C) \equiv w_1x_1 + \ldots + w_ix_j + \ldots + w_jx_i + \ldots w_nx_n \leq W$. Its easy to see that $\sum w_ix_i \leq W \wedge w_ix_j + w_jx_i \leq w_ix_i + w_jx_j \Rightarrow \sigma_{i,j}(C)$, hence we could set $scond(\sigma_{i,j}) \equiv w_ix_j + w_jx_i \leq w_ix_i + w_jx_j$.

If σ is a symmetry of $C \wedge D$ then $scond(\sigma) \equiv true$. If σ is almost a symmetry of $C \wedge D$, then $scond(\sigma)$ is usually fairly small and simple, because most of the $\sigma(c)$ are already implied by $C \wedge D$.

3.3 Step 3: Finding *ocond*(σ)

We assume that the objective function $f(\theta)$ is defined over all assignments (not just solutions). We first give a few definitions.

Definition 4. *Given a function σ mapping assignments to assignments, we extend σ to map functions to functions as follows: $\forall \theta,\ \sigma(f)(\theta) = f(\sigma(\theta))$.*

Definition 5. *Given two functions mapping assignments to the reals f and g, we use $f < g$ to denote a constraint such that: θ satisfies $f < g$ iff $f(\theta) < g(\theta)$.*

For each σ, we want to find a sufficient condition $ocond(\sigma)$ so that if a solution satisfied $ocond(\sigma)$, then σ maps it a assignment with a strictly better objective value. A necessary and sufficient condition is: $C \wedge ocond(\sigma) \Rightarrow \sigma(f) < f$. We can typically just set $ocond(\sigma) \equiv \sigma(f) < f$. For example, in both the Photo and Knapsack examples above, we simply calculated $\sigma(f) < f$, eliminated equal terms from each side, and used that as $ocond(\sigma)$.

3.4 Step 4: Posting the Dominance Breaking Constraint

Once we have found $scond(\sigma)$ and $ocond(\sigma)$, we can construct the dominance breaking constraint $db(\sigma) \equiv \neg(scond(\sigma) \wedge ocond(\sigma))$ and simplify it as much as possible. If it is simple enough to implement efficiently, we can add it to the problem. It is quite common that the dominance breaking constraint for different σ's will have common subexpressions. We can take advantage of this to make the implementation of the dominance breaking constraints more efficient.

4 Generating Symmetry and Conditional Symmetry Breaking Constraints

The method described so far only finds dominance breaking constraints which prune a solution when its objective value is *strictly* worse than another. We can do better than this, as there are often pairs of solutions which have equally good objective value and we may be able to prune many of them. Exploiting such sets of equally good pairs of solution is called symmetry breaking and conditional symmetry breaking. We show that with a slight alteration, our method will generate dominance breaking constraints that will also break symmetries and conditional symmetries.

We modify the method as follows. We add in a Step 0, and alter Step 3 slightly.

Step 0 Choose a refinement of the objective function f' with the property that $\forall \theta_1, \theta_2, f'(\theta_1) \leq f'(\theta_2)$ implies $f(\theta_1) \leq f(\theta_2)$.

Step 3* For each σ, find a constraint $ocond(\sigma)$ s.t. if $\theta \in solns(C \wedge D \wedge ocond(\sigma))$, then $f'(\sigma(\theta)) < f'(\theta)$.

We have the following theorem concerning the correctness of the altered method.

Theorem 3. *Given a finite domain CSOP $P \equiv (V, D, C, f)$, a refinement of the objective function f' satisfying $\forall \theta_1, \theta_2, f'(\theta_1) \leq f'(\theta_2)$ implies $f(\theta_1) \leq f(\theta_2)$, a set of mappings S, and for each mapping $\sigma \in S$ constraints $scond(\sigma)$ and $ocond(\sigma)$ satisfying: $\forall \sigma \in S$, if $\theta \in solns(C \wedge D \wedge scond(\sigma))$, then $\sigma(\theta) \in solns(C \wedge D)$, and: $\forall \sigma \in S$, if $\theta \in solns(C \wedge D \wedge ocond(\sigma))$, then $f'(\sigma(\theta)) < f'(\theta)$, we can add all of the dominance breaking constraints $db(\sigma) \equiv \neg(scond(\sigma) \wedge ocond(\sigma))$ to P without changing its satisfiability or optimal value.*

Proof. The proof is analogous to that of Theorem 2.

The $db(\sigma)$ are guaranteed to be compatible because they all obey the same strict ordering imposed by the refined objective function f', i.e., they prune a solution only if a solution with strictly better f' value exists. Theorem 3 is a very useful result as it is generally quite difficult to tell whether different symmetry, conditional symmetry or dominance breaking constraints are compatible. There are lots of examples in literature where individual dominance breaking constraints are proved correct, but no rigorous proof is given that they are correct when used together (e.g., [9, 6, 11]). The symmetry, conditional symmetry or dominance breaking constraints generated by our method are guaranteed to be compatible by Theorem 3, thus the user of the method does not need to prove anything themselves. We now show with some examples how the altered method can generate symmetry and conditional symmetry breaking constraints.

Example 3. Consider the Photo problem from Example 1. Suppose that in Step 0, instead of setting $f' = f$, we set $f' = lex(f, x_1, \ldots, x_n)$, the lexicographic least vector $(f, x_1, \ldots, x_n)$. That is, we order the solutions by their objective value, and then tie break by the value of x_1, then by x_2, etc. Clearly, $f'(\theta_1) \leq f'(\theta_2)$

implies $f(\theta_1) \leq f(\theta_2)$ so we are fine. Now, consider what happens in Step 3. In general, we have $\sigma(lex(f_1, \ldots, f_n)) < lex(f_1, \ldots, f_n) \Leftrightarrow lex(\sigma(f_1), \ldots, \sigma(f_n)) < lex(f_1, \ldots, f_n) \Leftrightarrow (\sigma(f_1) < f_1) \vee (\sigma(f_1) = f_1 \wedge \sigma(f_2) < f_2) \vee \ldots \vee (\sigma(f_1) = f_1 \wedge \ldots \wedge \sigma(f_{n-1}) = f_{n-1} \wedge \sigma(f_n) < f_n)$.

In this problem, we have: $\forall i < j, ocond(\sigma_{i,j}) \equiv \sigma(f') < f' \equiv (p[x_{i-1}][x_j] + p[x_i][p_{j+1}] < p[x_{i-1}][x_i] + p[x_j][p_{j+1}]) \vee (p[x_{i-1}][x_j] + p[x_i][p_{j+1}] = p[x_{i-1}][x_i] + p[x_j][p_{j+1}] \wedge x[j] < x[i])$. There is an additional term in $ocond(\sigma_{i,j})$ which says that we can also prune the current assignment if the flipped version has *equal* objective value but a better lexicographical value for $\{x_1, \ldots, x_n\}$. Thus $db(\sigma_{i,j})$ not only breaks dominances but also includes a conditional symmetry breaking constraint. Similarly, consider $\sigma_{1,n}$. Because it is a boundary case, the terms in $\sigma(f)$ and f all cancel and we have $ocond(\sigma_{1,n}) \equiv x[n] < x[1]$, so $db(\sigma_{1,n}) \equiv x[1] \leq x[n]$ which is simply a symmetry breaking constraint. □

Example 4. Consider the Knapsack problem from Example 2. In Step 0, we can tie break solutions with equal objective value by the weight used, and then lexicographically, i.e., $f' = lex(f, \sum w_i x_i, x_1, \ldots, x_n)$. In Step 3, we have: $\forall i < j, ocond(\sigma_{i,j}) \equiv \sigma(f') < f' \equiv ((v_i - v_j)(x_i - x_j) < 0) \vee ((v_i - v_j)(x_i - x_j) = 0 \wedge (w_i - w_j)(x_i - x_j) > 0) \vee ((v_i - v_j)(x_i - x_j) = 0 \wedge (w_i - w_j)(x_i - x_j) = 0 \wedge x_j < x_i)$. In Step 4, after simplifying, in addition to the dominance breaking constraints we had before, we would also have: $db(\sigma_{i,j}) \equiv x_i \leq x_j$ if $w_i > w_j$ and $v_i = v_j$, $db(\sigma_{i,j}) \equiv x_i \geq x_j$ if $w_i < w_j$ and $v_i = v_j$, and $db(\sigma_{i,j}) \equiv x_i \leq x_j$ if $w_i = w_j$ and $v_i = v_j$ which is a symmetry breaking constraint. □

We can also apply the altered method to satisfaction problems to generate symmetry and conditional symmetry breaking constraints.

Example 5. The Black Hole Problem [11] seeks to find a solution to the Black Hole patience game. In this game the 52 cards of a standard deck are laid out in 17 piles of 3, with the Ace of spades starting in a "black hole". Each turn, a card at the top of one of the piles can be played into the black hole if it is ± 1 from the card that was played previously, with king wrapping back around to ace. The aim is to play all 52 cards. We can model the problem as follows. Let the suits be numbered from 1 to 4 in the order spades, hearts, clubs, diamonds. Let the cards be numbered from 1 to 52 so that card i has suit $(i-1)/13 + 1$ and number $(i-1)\%13 + 1$, where 11 is jack, 12 is queen and 13 is king. Let $l_{i,j}$ be the jth card in the ith pile in the initial layout. Let x_i be the turn in which card i was played. Let y_i be the card which was played in turn i. We have:

$$x_1 = 1 \tag{1}$$

$$inverse(x, y) \tag{2}$$

$$x_{l_{i,j}} < x_{l_{i,j+1}} \qquad \forall 1 \leq i \leq 17, 1 \leq j \leq 2 \tag{3}$$

$$(y_{i+1} - y_i)\%13 \in \{-1, 1\} \qquad \forall 1 \leq i \leq 51 \tag{4}$$

We now apply our method. It is well known that the lexicographical ordering used to generate lex-leader symmetry breaking constraints should be consistent with the search strategy, otherwise they may conflict and result in a slow down. An obvious strategy for the Black Hole problem is to prefer to play

cards at the top of piles, since that will uncover other cards to play. We define $f' = lex(x_{l_{1,1}}, \ldots, x_{l_{17,1}}, \ldots, x_{l_{1,3}}, \ldots, x_{l_{17,3}})$. An obvious set of mapping that are likely to map solutions to solutions is to swap cards of the same number in the sequence of cards to be played. Consider $\sigma_{i,j}$ for $i - j\%13 = 0, i \neq 1, j \neq 1$ where $\sigma_{i,j}$ swaps x_i and x_j, and swaps the values of i and j among $\{y_1, \ldots, y_{52}\}$.

Now we construct $scond(\sigma_{i,j})$. For each constraint c in the problem, we need to find a c' such that $C \wedge D \wedge c' \Rightarrow \sigma_{i,j}(c)$ and add it to $scond(\sigma_{i,j})$. Clearly, the domain constraints and the constraints in 1, 2 and 4 are all symmetric in $\sigma_{i,j}$, so we do not need to add anything for them. However, there will be some constraints in 3 which are not symmetric in $\sigma_{i,j}$. For example, suppose we wished to swap $3\spadesuit$ and $3\heartsuit$, and they were in piles: $(2\spadesuit, 3\spadesuit, 5\clubsuit)$ and $(1\diamondsuit, 3\heartsuit, 6\diamondsuit)$, where $3\spadesuit$ is in lexicographically earlier pile than $3\heartsuit$. The constraints in 3 which are not symmetric in $\sigma_{i,j}$ are those involving $3\spadesuit$ or $3\heartsuit$, i.e., $2\spadesuit < 3\spadesuit$, $3\spadesuit < 5\clubsuit$, $1\diamondsuit < 3\heartsuit$ and $3\heartsuit < 6\diamondsuit$. Their symmetric versions are $2\spadesuit < 3\heartsuit$, $3\heartsuit < 5\clubsuit$, $1\diamondsuit < 3\spadesuit$ and $3\spadesuit < 6\diamondsuit$ respectively, so we can set $scond(\sigma_{2,15}) \equiv 2\spadesuit < 3\heartsuit \wedge 3\heartsuit < 5\clubsuit \wedge 1\diamondsuit < 3\spadesuit \wedge 3\spadesuit < 6\diamondsuit$. To construct $ocond(\sigma_{i,j})$, we can set $ocond(\sigma_{i,j}) \equiv \sigma_{i,j}(f') < f'$. For this example, we have $ocond(\sigma_{2,15}) \equiv 3\heartsuit < 3\spadesuit$. Combining, we have $db(\sigma_{2,15}) \equiv \neg(2\spadesuit < 3\heartsuit \wedge 3\heartsuit < 5\clubsuit \wedge 1\diamondsuit < 3\spadesuit \wedge 3\spadesuit < 6\diamondsuit \wedge 3\heartsuit < 3\spadesuit)$. We can use the constraints in the original problem to simplify this further. Since $3\spadesuit < 5\clubsuit$ is an original constraint and $3\heartsuit < 3\spadesuit \wedge 3\spadesuit < 5\clubsuit \Rightarrow 3\heartsuit < 5\clubsuit$, we can eliminate the second term in $db(\sigma_{2,15})$. Since $1\diamondsuit < 3\heartsuit$ is an original constraint and $1\diamondsuit < 3\heartsuit \wedge 3\heartsuit < 3\spadesuit \Rightarrow 1\diamondsuit < 3\spadesuit$, we can eliminate the third term in $db(\sigma_{2,15})$. The result is $db(\sigma_{2,15}) \equiv \neg(2\spadesuit < 3\heartsuit \wedge 3\spadesuit < 6\diamondsuit \wedge 3\heartsuit < 3\spadesuit)$. The other cases are similar. □

Although the conditional symmetry breaking constraints derived in Example 5 are identical to those derived in [11], our method is much more generic and can be systematically applied to other problems as well. Also, no rigorous proof of correctness is given in [11], whereas Theorem 3 shows that these conditional symmetry breaking constraints are compatible. In this problem it is quite possible to derive multiple incompatible conditional symmetry breaking constraints which are individually correct. For example, suppose in addition to $(2\spadesuit, 3\spadesuit, 5\clubsuit)$ and $(1\diamondsuit, 3\heartsuit, 6\diamondsuit)$, we had a third pile $(2\heartsuit, 3\diamondsuit, 7\spadesuit)$, then the following conditional symmetry breaking constraints are all individually correct: $\neg(2\spadesuit < 3\heartsuit \wedge 3\spadesuit < 6\diamondsuit \wedge 3\heartsuit < 3\spadesuit)$, $\neg(2\heartsuit < 3\spadesuit \wedge 3\diamondsuit < 5\clubsuit \wedge 3\spadesuit < 3\diamondsuit)$, $\neg(1\diamondsuit < 3\diamondsuit \wedge 3\heartsuit < 7\spadesuit \wedge 3\diamondsuit < 3\heartsuit)$, but they are incompatible. For example, no matter which permutation of $3\spadesuit$, $3\heartsuit$, and $3\diamondsuit$ is applied, the partial solution $1\spadesuit, 2\heartsuit, 1\diamondsuit, 2\heartsuit, 3\spadesuit, 4\spadesuit, 3\heartsuit, 4\diamondsuit, 3\diamondsuit$ is pruned by one of the three conditional symmetry breaking constraints. Our method will never produce such incompatible sets of dominance breaking constraints.

Example 6. The Nurse Scheduling Problem (NSP) is to schedule a set of nurses over a time period such that work and hospital regulations are all met, and as many of the nurses' preferences are satisfied. There are many variants of this problem in the literature (e.g., [15, 2]). We pick a simple variant to illustrate our method. Each day has three shifts: day, evening, and overnight. On each day, each nurse should be scheduled into one of the three shifts or scheduled a day off. For simplicity, we can consider a day off to be a shift as well. We number the shifts as day: 1, evening: 2, over-night: 3, day-off: 4. Each shift besides day-off

requires a minimum number r_i of nurses to be rostered. Nurses cannot work for more than 6 days in a row, and must work at least 10 shifts per 14 days. Each nurse i has a preference $p_{i,j}$ for which of the four shift they wish to take on day j. The objective is to maximize the number of satisfied preferences. Let n be the number of nurses and m be the number of days. Let $x_{i,j}$ be the shift that nurse i is assigned to on day j. Then the problem can be stated as follows:

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{i,j} = p_{i,j})$$

Subject to

$$\mathit{among}(r_i, \infty, [x_{k,j} \mid 1 \leq k \leq n], i) \qquad \forall 1 \leq i \leq 3, 1 \leq j \leq m \qquad (5)$$

$$\mathit{among}(1, \infty, [x_{i,j} \mid k \leq j < k+7], 4) \qquad \forall 1 \leq i \leq n, 1 \leq k \leq n-6 \qquad (6)$$

$$\mathit{among}(-\infty, 4, [x_{i,j} \mid k \leq j < k+14], 4) \qquad \forall 1 \leq i \leq n, 1 \leq k \leq n-13 \qquad (7)$$

Where $\mathit{among}(l, u, [x_1, \ldots, x_n], v)$ means that there are at least l and at most u variables from among $[x_1, \ldots, x_n]$ which take the value v. We now apply our dominance breaking method. Firstly, we can potentially get some symmetry or conditional symmetry breaking in by refining the objective function to $f' = \mathit{lex}(f, x_{1,1}, x_{2,1}, \ldots, x_{n,m})$. Let us consider mappings which are likely to map solutions to solutions. An obvious set of candidates are mappings which swap the shifts of two nurses on the same day, i.e., mappings $\sigma_{i_1,i_2,j}$ which swap $x_{i_1,j}$ and $x_{i_2,j}$.

We wish to calculate $\mathit{scond}(\sigma_{i_1,i_2,j})$. For each $c \in C \cup D$, we need to find c' such that $C \wedge D \wedge c' \Rightarrow \sigma_{i_1,i_2,j}(c)$. It is easy to see that the constraints in (5) are all symmetric in $\sigma_{i_1,i_2,j}$, so we do not need to add anything to $\mathit{scond}(\sigma_{i_1,i_2,j})$. The constraints $\mathit{among}(1, \infty, [x_{i,j} \mid k \leq j < k+7], 4)$ in (6) will be satisfied by $\sigma(\theta)$ iff: $x_{i_1,j} \neq 4 \vee x_{i_2,j} = 4 \vee \mathit{among}(1, \infty, [x_{i,j} \mid k \leq j < k+7, j \neq i_1], 4)$. Similarly, the constraints $\mathit{among}(-\infty, 4, [x_{i,j} \mid k \leq j < k+14], 4)$ in (7) will be satisfied by $\sigma(\theta)$ iff: $x_{i_1,j} = 4 \vee x_{i_2,j} \neq 4 \vee \mathit{among}(-\infty, 3, [x_{i,j} \mid k \leq j < k+14, j \neq i_1], 4)$. Since the *among* conditions are probably too expensive to check, we can simply throw them away. We lose some potential pruning, but it is still correct, since we had a disjunction of conditions. So we add $x_{i_1,j} \neq 4 \vee x_{i_2,j} = 4$ and $x_{i_1,j} = 4 \vee x_{i_2,j} \neq 4$ to $\mathit{scond}(\sigma_{i_1,i_2,j})$. Calculating $\sigma(f') > f'$ is straight forward. We simply try each pair of values for $x_{i_1,j}$ and $x_{i_2,j}$ and see if swapping them improves the refined objective function. For example, suppose $i_1 < i_2$, $p_{i_1,j} = 4$, $p_{i_2,j} = 2$, then $\mathit{ocond}(\sigma_{i_1,i_2,j}) \equiv \sigma(f') > f' \equiv (x_{i_1,j}, x_{i_2,j}) \in \{(1,4),(2,1),(2,3),(2,4),(3,1),(3,4)\}$. Then $\mathit{db}(\sigma_{i_1,i_2,j}) \equiv \neg(\mathit{scond}(\sigma_{i_1,i_2,j}) \wedge \mathit{ocond}(\sigma_{i_1,i_2,j})) \equiv (x_{i_1,j}, x_{i_2,j}) \notin \{(2,1),(2,3),(3,1)\}$, which is a table constraint encapsulating both dominance breaking and symmetry breaking constraints. □

5 Related Work

There have been many works on problem specific applications of dominance relations, e.g., the template design problem [19], online scheduling problems [13], the

Maximum Density Still Life problem, Steel Mill Design problem and Peaceable Armies of Queens problem [18], the Minimization of Open Stacks problem [6], and the Talent Scheduling Problem [9]. However, the methods used are typically very problem specific and offer little insight as to how they can be generalized and applied to other problems. The implementations of these methods are also often quite ad-hoc (e.g., pruning values from domains even though they do not explicitly violate any constraint), and it is not clear whether they can be correctly combined with other constraint programming techniques. In contrast, our new method rests on a much stronger theoretical foundation and is completely rigorous. Since our method simply adds constraints to the problem, the modified problem is a perfectly normal constraint problem and it is correct to use any other constraint programming technique on it. Another important advantage of our method is that we are able to use any search strategy we want on the modified problem. This is not the case with many of the problem specific dominance breaking methods as they rely on specific labeling strategies.

There are a small number of works on generic methods for detecting and exploiting dominance relations. Machine learning techniques have been proposed as a method for finding candidate dominance relations [20]. This method works by encoding problems and candidate dominance relations into forms amenable to machine learning. Machine learning techniques such as experimentation, deduction and analogy are then used to identify potential dominance relations. This method was able to identify dominance relations for the 0/1 knapsack problem and a number of scheduling problems. However, the main weakness of this method is that it only generates candidate dominance relations and does not prove their correctness. Each candidates has to be analyzed to see if they are in fact a dominance relation. Then the dominance relation has to be manually proved and exploited.

Recently, several generic and automatic methods have been developed for exploiting certain classes of dominance relations. These include nogood learning techniques such as Lazy Clause Generation [17, 8] and Automatic Caching via Constraint Projection [5]. Both of these can be thought of as dynamic dominance breaking, where after some domain D_1 is found to fail, a nogood (constraint) n is found which guarantees that if D_2 violates n, then D_2 is dominated by D_1 and must also fail. The nogood n is posted as an additional redundant constraint to the problem. Lazy Clause Generation derives this n by resolving together clauses which explain the inferences which led to the failure. Automatic Caching via Constraint Projection derives n by finding conditions such that projection of the subproblem onto the subset of unfixed variables yield a more constrained problem. These methods are to a large extent complimentary to the method presented in this paper. None of these methods exhausts all possible dominances occurring in a problem, and there are dominances which can be exploited by one method but not another. Thus we can often use them simultaneously to gain an even greater reduction in search space.

6 Experimental Results

We now give some experimental results for our method on a variety of problems. We have already discussed how our method applies to the Photo Problem, Knap-

sack Problem, Black Hole Problem, and Nurse Scheduling. For these problems, we generate random instances of several different sizes, with 10 instances of each size. We also give experimental results for four further problems:

RCPSP The resource constrained project scheduling problem (RCPCP) [4] schedules n tasks using m renewable resources so that ordering constraints among tasks hold and resource usage limits are respected. A standard dominance rule for this problem, used in search, is that each task must start at time 0 or when another task ends, since any schedule not following this rule is domainated by one constructed by shifting tasks earlier until the rule holds. We use the instances from the standard J60 benchmark set [1] which are non-trivial (not solved by root propagation) and solvable by at least one of the methods.

Talent Scheduling Problem In the Talent Scheduling Problem [9], we have a set of scenes and a set of actors. Each actor appears in a number of scenes and is paid a certain amount per day they are on location. They must stay on location from the first scene they are in till the last scene they are in. The aim is to find the schedule of scenes $x_1, \ldots, x_n$ which minimize the cost of the actors. We set $f' = lex(f, x_1, \ldots, x_n)$. We consider mappings which take one scene and move it to another position in the sequence. We generate 10 random instances of size 14, 16, and 18.

Steel Mill Problem In the Steel Mill Problem [10], we have a set of orders to be fulfilled and the aim is to minimize the amount of wasted steel. Each order i has a size and a color (representing which path it takes in the mill) and is to be assigned to a slab x_i. Each slab can only be used for orders of two different colors. Depending on the sum of the sizes of the orders on each slab, a certain amount of steel will be wasted. We set $f' = lex(f, x_1, \ldots, x_n)$ and try mappings where we take all orders of a certain color from one slab, and all orders of a certain color from another slab, and swap the slabs they are assigned to. We generate 10 random instances of size 40 and 50.

PC Board Problem In the PC Board Problem [14], we have $n \times m$ components of various types which need to be assigned to m machines. Each machine must be assigned exactly n components and there are restrictions on the sets of components that can go on the same machine. Each type of component gains a certain utility depending on which machine it is assigned to and the goal is to maximize the overall utility. We set $f' = lex(f, x_{1,1}, x_{1,2}, \ldots, x_{n,m})$ where $x_{i,j}$ is the type of component assigned to the jth spot on the ith machine. We consider mappings which swap two components on different machines. We generate 20 random instances of size 6×8.

The experiments were performed on Xeon Pro 2.4GHz processors using the CP solver Chuffed. For each set of benchmarks, we report the geometric mean of time taken in seconds and the number of failed nodes for: the original model with no dominance breaking of any form (base), with dominance breaking constraints generated by our method (db), with Lazy Clause Generation (lcg), and with

dominance breaking constraints and Lazy Clause Generation (db+lcg) . A time-out of 900 seconds was used. Fastest times and lowest node counts are shown in bold.

Table 1. Comparison of the original model and the model augmented with dominance breaking constraints

Problem	base		db		lcg		db+lcg	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
Photo-14	1.09	57773	0.90	10967	0.30	5791	**0.25**	**1962**
Photo-16	8.38	441574	4.00	43373	6.49	44325	**1.40**	**8960**
Photo-18	60.68	2828622	22.09	206507	19.73	138926	**6.25**	**24523**
Knapsack-20	**0.01**	215	**0.01**	9	**0.01**	212	**0.01**	**7**
Knapsack-30	0.17	46422	**0.01**	91	0.85	45733	**0.01**	**65**
Knapsack-50	602	1×10^8	**0.01**	684	900	1×10^7	**0.01**	**507**
Knapsack-100	900	1×10^8	**0.40**	54705	900	1×10^7	1.05	**37571**
Black-hole	5.18	77542	**0.08**	607	0.97	2767	0.09	**347**
Nurse-7	900	9×10^7	900	8×10^7	1.72	55217	**0.91**	**24258**
Nurse-14	900	8×10^7	900	8×10^7	483.29	7×10^6	**140.95**	$\mathbf{1 \times 10^6}$
RCPSP	358.95	2779652	279.74	781399	**4.07**	**7890**	32.84	32770
Talent-Sched-14	1.66	39479	0.42	10122	0.45	4983	**0.27**	**3189**
Talent-Sched-16	16.08	349704	2.33	51993	3.71	27186	**1.28**	**12336**
Talent-Sched-18	252.05	5557959	13.88	299043	26.25	128810	**4.28**	**31829**
Steel-Mill-40	60.64	1×10^6	22.00	451636	16.31	75293	**4.53**	**27225**
Steel-Mill-50	379.21	7×10^6	231.95	3×10^6	249.39	714451	**32.24**	**129788**
PC-board	547.93	4×10^7	412.29	1×10^7	20.28	156933	**7.51**	**64320**

As Table 1 shows, adding dominance breaking constraints can significantly reduce the search space on a variety of problems, leading to large speedups which tend to grow exponentially with problem size. Our method can also often be combined with Lazy Clause Generation for additional speedup (e.g., Photo, Steel Mill, Talent Scheduling, Nurse Scheduling, PC Board). In some cases (e.g., Knapsack, Black Hole), even though adding LCG on top of our method can reduce the node count further, the extra overhead of LCG swamps out any benefit. In other cases (e.g., RCPSP), adding our dominance breaking constraints on top of LCG actually increases the run time and node count. In this problem, the dynamically derived dominances from LCG are stronger than the static ones that our method derives. Adding the dominance breaking constraints interferes with and reduces the benefit of LCG. In general however, our method appears to provide significant speedups over a wide range of problems for both non-learning and nogood learning solvers.

7 Future Work

Although we have developed this method in the context of Constraint Programming, the dominance relations we find can be applied to other kinds of search as

well. For example, MIP solvers, which use branch and bound, can also benefit from the power of dominance relations, as a MIP solver can encounter suboptimal partial assignments which nevertheless do not produce an LP bound strong enough to prune the subproblem. Similarly, local search can benefit tremendously from dominance relations, as they can show when a solution is suboptimal and map it to another solution which is better. Exploring how our method could be adapted for use in other kinds of search is an interesting avenue of future work.

It may also be possible to automate many or all of the steps involved in our method. Such automation would provide a great benefit for system users as they will be able to feed in a relatively "dumb" model and have the system automatically identify and exploit the dominances. Step 0 typically requires augmenting the objective function with an appropriate lexicographical ordering of the variables. For Step 1, there already exist automated methods for detecting symmetries in problem instances. Such methods can be adapted to look for good candidates for σ. Step 2 and 3 involves algebraic manipulations which are not difficult for a computer to do. The difficulty lies in Step 4, where we need to simplify the logical expressions and determine whether the dominance breaking constraint is sufficiently simple, efficient and powerful that it is worth adding to to problem. Automating our method is another interesting avenue of future work.

8 Conclusion

We have described a general method for systematically identifying and exploiting dominance relations in constraint problems. The method generates a set of dominance breaking constraints which are provably correct and compatible with each other. The method also generates symmetry and conditional symmetry breaking constraints as a special case, thus it unifies symmetry breaking, conditional symmetry breaking and dominance breaking under one method. Experimental results show that the dominance breaking constraints so generated can lead to significant reductions in search space and run time on a variety of problems, and that they can be effectively combined with other dominance breaking techniques such as Lazy Clause Generation.

References

1. PSPLib - project scheduling problem library. `http://129.187.106.231/psplib/`. Accessed on 1 March 2012.
2. Slim Abdennadher and Hans Schlenker. Nurse scheduling using constraint logic programming. In *AAAI/IAAI*, pages 838–843, 1999.
3. Tariq A. Aldowaisan. A new heuristic and dominance relations for no-wait flowshops with setups. *Computers & OR*, 28(6):563–584, 2001.
4. Peter Brucker, Andreas Drexl, Rolf H. Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.
5. Geoffrey Chu, Maria Garcia de la Banda, and Peter J. Stuckey. Automatically Exploiting Subproblem Equivalence in Constraint Programming. In Andrea Lodi,

Michela Milano, and Paolo Toth, editors, *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2010.

6. Geoffrey Chu and Peter J. Stuckey. Minimizing the Maximum Number of Open Stacks by Customer Search. In Ian P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 2009.
7. Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry Breaking. In Toby Walsh, editor, *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2001.
8. Thibaut Feydy and Peter J. Stuckey. Lazy Clause Generation Reengineered. In Ian P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 352–366. Springer, 2009.
9. Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. Solving Talent Scheduling with Dynamic Programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011.
10. Antoine Gargani and Philippe Refalo. An Efficient Model and Strategy for the Steel Mill Slab Design Problem. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 77–89. Springer, 2007.
11. Ian P. Gent, Tom Kelsey, Steve Linton, Iain McDonald, Ian Miguel, and Barbara M. Smith. Conditional Symmetry Breaking. In Peter van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2005.
12. Ian P. Gent and Barbara M. Smith. Symmetry Breaking in Constraint Programming. In Werner Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 599–603. IOS Press, 2000.
13. Lise Getoor, Greger Ottosson, Markus P. J. Fromherz, and Björn Carlson. Effective redundant constraints for online scheduling. In *AAAI/IAAI*, pages 302–307, 1997.
14. Roland Martin. The Challenge of Exploiting Weak Symmetries. In Brahim Hnich, Mats Carlsson, François Fages, and Francesca Rossi, editors, *Proceedings of the International Workshop on Constraint Solving and Constraint Logic Programming*, volume 3978 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2005.
15. H.E. Miller, W.P. Pierskalla, and G.J. Rath. Nurse scheduling using mathematical programming. *Operations Research*, pages 857–870, 1976.
16. J.N. Monette, P. Schaus, S. Zampelli, Y. Deville, and P. Dupont. A CP Approach to the Balanced Academic Curriculum Problem. In *Seventh International Workshop on Symmetry and Constraint Satisfaction Problems*, volume 7, 2007. `http://www.info.ucl.ac.be/~yde/Papers/SymCon2007_bacp.pdf`.
17. Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = Lazy Clause Generation. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer, 2007.
18. S. Prestwich and J.C. Beck. Exploiting dominance in three symmetric problems. In *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, pages 63–70, 2004. `http://zeynep.web.cs.unibo.it/SymCon04/SymCon04.pdf`.

19. Les G. Proll and Barbara Smith. Integer linear programming and constraint programming approaches to a template design problem. *INFORMS Journal on Computing*, 10(3):265–275, 1998.
20. Chee Fen Yu and Benjamin W. Wah. Learning Dominance Relations in Combinatorial Search Problems. *IEEE Trans. Software Eng.*, 14(8):1155–1175, 1988.