Exploiting Symmetries within Constraint Satisfaction Search *

Pedro Meseguer
Institut d'Investigació en Intel.ligència Artificial
CSIC
Campus UAB, 08193 Bellaterra
Spain
pedro@iiia.csic.es

Carme Torras
Institut de Robòtica i Informàtica Industrial
CSIC-UPC
Gran Capità 2-4, 08034 Barcelona
Spain
ctorras@iri.upc.es

April 10, 2001

Abstract

Symmetry often appears in real-world constraint satisfaction problems, but strategies for exploiting it are only beginning to be developed. Here, a framework for exploiting symmetry within depth-first search is proposed, leading to two heuristics for variable selection and a domain pruning procedure. These strategies are then applied to two highly symmetric combinatorial problems, namely the Ramsey problem and the generation of balanced incomplete block designs. Experimental results show that these general-purpose strategies can compete with, and in some cases outperform, previous more ad hoc procedures.

*This paper is an extended and updated version of [17], presented at the IJCAI-99 conference.

1 Introduction

Symmetry is present in many natural and artificial settings. A symmetry is a transformation of an entity such that the transformed entity is equivalent to and indistinguishable from the original one. We can see symmetries in nature (a specular reflection of a daisy flower), in human artifacts (a central rotation of 180 degrees of a chessboard), and in mathematical theories (inertial changes in classical mechanics). The existence of symmetries in these systems allows us to generalize the properties detected in one state to all its symmetric states.

Regarding constraint satisfaction problems (CSPs), many real problems exhibit some kind of symmetry, embedded in the structure of variables, domains and constraints. This means that their state space is somehow fictitiously enlarged by the presence of many symmetric states. From a search viewpoint, it is advisable to visit only one among those states related by a symmetry, since either all of them lead to a solution or none does. This may cause a drastic decrease in the size of the search space, which would have a very positive impact on the efficiency of the constraint solver.

Previous works on symmetric CSPs have been aimed at eradicating symmetries from either the initial problem state space or the explicit search tree as it is developed. The former approach, advocated by Puget [20], consists in reducing the initial state space by adding symmetry-breaking constraints to the problem formulation. The goal is to turn the symmetric problem into a new problem without symmetries, but keeping the non-symmetric solutions of the original one. Although this ideal goal is seldom reached, the reductions attained are substantial enough to turn some hard combinatorial problems into manageable ones. For generic problem statements, the detection of symmetries and the formulation of the ad hoc symmetry-breaking constraints is performed by hand [20]. Alternatively, in the context of propositional logic, existing symmetries and the corresponding symmetry-breaking predicates can be computed automatically [7], although with a high computational complexity.

The second approach, namely pruning symmetric states from the search tree as it develops, entails modifying the constraint solver to take advantage of symmetries. A modified backtracking algorithm appears in [4], where each expanded node is tested to assess whether it is an appropriate representative of all the states symmetric to it. Concerning specific symmetries, neighborhood interchangeable values of a variable are discussed in [10], while value pruning after failure for strongly permutable variables is proposed in [21]. This last strategy can be seen as a particular case of the symmetry exclusion method introduced in [1] for concurrent constraint programming, and applied to the CSP context in [12].

In this paper, we propose a third approach to exploit symmetries inside CSPs. The idea is to use symmetries to guide the search. More specifically, the search is directed towards subspaces with a high density of non-symmetric states, by breaking as many symmetries as possible with each new variable assignment. This is the rationale for our *symmetry-breaking* heuristic for variable selection, which can be theoretically combined with the popular minimum-domain heuristic. The result of this combination is the new *variety-maximization* heuristic for variable selection, which has been shown more effective than symmetry-breaking or minimum-domain separatedly, and it has speeded up significantly the solving process of CSPs with many symmetries. For problems without a solution, variable selection heuristics can do nothing to avoid revisiting symmetric states along the search. To cope with this shortcoming, we have developed several value pruning strategies (in the spirit of the second approach mentioned above), which allow one to reduce the domain of the current or future variables. These strategies remove symmetric values, without removing non-symmetric solutions. In particular, there is a strategy based on nogoods learned in previous search states. Problem symmetries allow us to keep limited the potentially exponential size of the nogood storage. This strategy has been shown very effective for hard solvable and unsolvable instances. Results for the Ramsey problem and for the generation of balanced incomplete block designs (BIBDs) are provided. Once a set of symmetries is specified, our approach provides a general-purpose mechanism to exploit them within the search. Moreover, it can be combined with the two previous approaches and incorporated into any depth-first search procedure.

The paper is structured as follows. In Section 2, we introduce some basic concepts. Section 3 presents the symmetry-breaking heuristic and its combination with the minimum-domain one, generating the variety-maximization heuristic. Section 4 details several strategies for symmetric value pruning along the search, especially those based on nogood recording. Section 5 is devoted to the Ramsey and BIBD problems. Finally, Section 6 puts forth some conclusions and prospects for future work.

2 Basic Definitions

2.1 Constraint Satisfaction

A finite CSP is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables, $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is a collection of domains, $D(x_i)$ is the finite set of possible values for variable x_i, and $\mathcal{C}$ is a set of constraints among variables. A constraint c_i on the ordered set of variables $var(c_i) = (x_{i_1}, \ldots, x_{i_{r(i)}})$ specifies the relation $rel(c_i)$ of the *allowed* combinations of values for the variables in $var(c_i)$. An element of $rel(c_i)$ is a tuple $(v_{i_1}, \ldots, v_{i_{r(i)}})$, $v_i \in D(x_i)$. An element of $D(x_{i_1}) \times \cdots \times D(x_{i_{r(i)}})$ is called a *valid* tuple on $var(c_i)$. A *solution* of the CSP is an assignment of values to variables which satisfies every constraint. A *nogood* is an assignment of values to a subset of variables which does not belong to any solution. Typically, CSPs are solved by depth-first search algorithms with backtracking. At a point in search, P is the set of assigned or *past* variables, and F is the set of unassigned or *future* variables. The variable to be assigned next is called the *current* variable.

A classical example of CSP is the n-queens problem. It consists in placing n chess queens on a $n \times n$ chessboard in such a way that no pair of queens is attacking one another. Constraints come from chess rules: no pair of queens can occur at the same row, column or diagonal. This problem is taken as running example throughout the paper.

2.2 Symmetries

A *symmetry* on a CSP is a collection of $n+1$ bijective mappings $\{\theta, \theta_1, \ldots, \theta_n\}$ defined as follows,

- θ is a variable mapping, $\qquad \theta : \mathcal{X} \to \mathcal{X}$
- $\{\theta_1, \ldots, \theta_n\}$ are domain mappings, $\qquad \theta_i : D(x_i) \to D(\theta(x_i))$
- constraints are transformed by the adequate combination of variable and domain mappings; a constraint c_i is transformed into c_i^θ, with $var(c_i^\theta) = (\theta(x_{i_1}), \ldots, \theta(x_{i_{r(i)}}))$ and $rel(c_i^\theta) = \{(\theta_{i_1}(v_{i_1}), \ldots, \theta_{i_{r(i)}}(v_{i_{r(i)}}))\}$,

such that the set $\mathcal{C}$ remains invariant by the action of the symmetry, i.e., $\forall c_j \in \mathcal{C}$, the transformed constraint c_j^θ is in $\mathcal{C}$. There exists always a trivial symmetry, that in which the variable mapping and the domain mappings are all the identity. The remaining symmetries, those interesting for our purposes, will be referred to as nontrivial symmetries. Moreover, when no ambiguity may occur, we will denote a symmetry $\{\theta, \theta_1, \ldots, \theta_n\}$ by its variable mapping θ.

Note that the above definition of symmetry applies to CSPs, i.e., to problems formulated in terms of a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, and not to problems in general. To make this point clear, consider the n-queens problem, which admits at least nine different problem formulations as a CSP [18]. These formulations vary in the number of variables, sizes of the domains, and constraint set. They specify different CSPs and, as such, it is not surprising that they have different symmetries.

Let us consider the most widely used formulation, namely that in which variables are chessboard rows and domains are column indices. Figure 1 shows an example of a symmetry using this formulation in the case of 5 queens. It is a central rotation of 180 degrees, which exchanges variables x_1 with x_5 and x_2 with x_4, and maps domains with the function $\theta_i(v) = 6 - v$, $i = 1, \ldots, 5$. This transformation is a symmetry because the mappings on variables and domains are bijective, and the set of constraints is left invariant by the transformation of variables and values. For example, the transformed constraint c_{12}^θ is computed as follows,

$$var(c_{12}^\theta) = (\theta(x_1), \theta(x_2)) = (x_5, x_4) = var(c_{45})$$

$$\begin{aligned} rel(c_{12}^\theta) &= \{(\theta_1(1), \theta_2(3)), (\theta_1(1), \theta_2(4)), (\theta_1(1), \theta_2(5)), (\theta_1(2), \theta_2(4)), (\theta_1(2), \theta_2(5)), (\theta_1(3), \theta_2(1)), \\ &\quad (\theta_1(3), \theta_2(5)), (\theta_1(4), \theta_2(1)), (\theta_1(4), \theta_2(2)), (\theta_1(5), \theta_2(1)), (\theta_1(5), \theta_2(2)), (\theta_1(5), \theta_2(3))\} = \\ &= \{(5,3), (5,2), (5,1), (4,2), (4,1), (3,5), (3,1), (2,5), (2,4), (1,5), (1,4), (1,3)\} = rel(c_{45}) \end{aligned}$$

Thus, $c_{12}^\theta = c_{45}$. Two other nontrivial symmetries of this CSP formulation of 5-queens are the reflections about the horizontal and vertical axes, as depicted in Figure 2. The remaining four symmetries of the chessboard are not symmetries of this formulation.

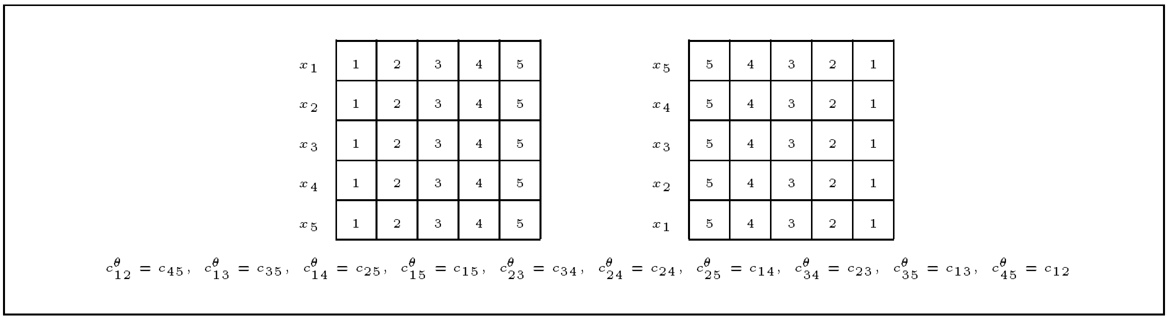


Figure 1: Central rotation of 180 degrees is a symmetry of the 5-queens problem.

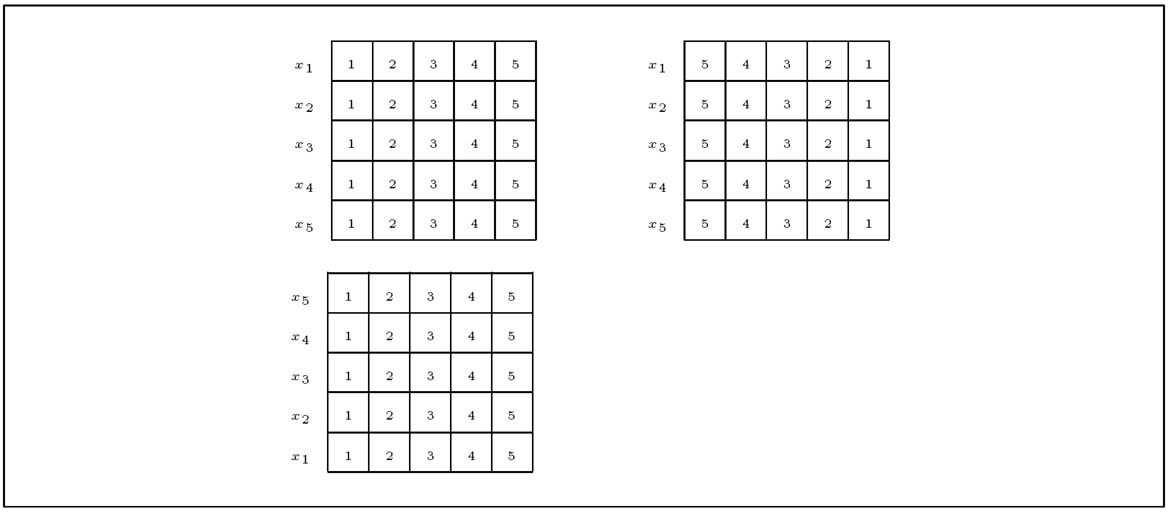


Figure 2: Two other symmetries of the 5-queens problem. Top-right: reflection about the vertical axis. Bottom-left: reflection about the horizontal axis.

Now, let us turn to the formulation of n-queens where each queen is a variable whose domain contains all the squares of the chessboard. The eight symmetries of the chessboard and all permutations of queens are symmetries of this particular CSP formulation.

Taken together, the two examples above illustrate the remark we made that our definition of symmetry applies to CSP formulations and not to problems in general. Such symmetries can be viewed as mapping a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ onto itself, which is needed to stay within the formulation. Thus, transformations that change variables into values and vice versa, as would be required to represent a rotation of 90 degrees under the formulation in Figure 1, are not allowed within our framework.

Following [23], we say that two variables x_i, x_j are *symmetric* if there exists a symmetry θ such that $\theta(x_i) = x_j$. This concept generalizes the previous definition of strong permutability [21]: x_i and x_j are strongly permutable if they play exactly the same role in the problem, i.e., if there exists a symmetry ϕ such that its only action is exchanging x_i with x_j ($\phi(x_i) = x_j$, $\phi(x_j) = x_i$, $\phi(x_k) = x_k$, $\forall k \neq i, j$, $\phi_k = I$, $\forall k$, I being the identity function). We say that two values $a, b \in D(x_i)$ are *symmetric* if there exists a symmetry θ such that $\theta(x_i) = x_i$ and $\theta_i(a) = b$. This concept generalizes previous definition of value interchangeability [10]: a and b are neighbourhood interchangeable if they are consistent with the same set of values, i.e., if there exists a symmetry ϕ such that its only action is exchanging a with b ($\phi = I$, $\phi_i(a) = b$, $\phi_i(b) = a$, $\phi_i(c) = c$, $\forall c \neq a, b$, $\phi_k = I$, $\forall k \neq i$).

The set of symmetries of a problem forms a group with the composition operator [23]. Because of this, it can be shown that the symmetry relation between variables is an equivalence relation. The existence of this equivalence relation divides the set $\mathcal{X}$ in equivalence classes, each class grouping symmetric variables. Domains are also divided into equivalence classes by symmetries acting on values only (with identity

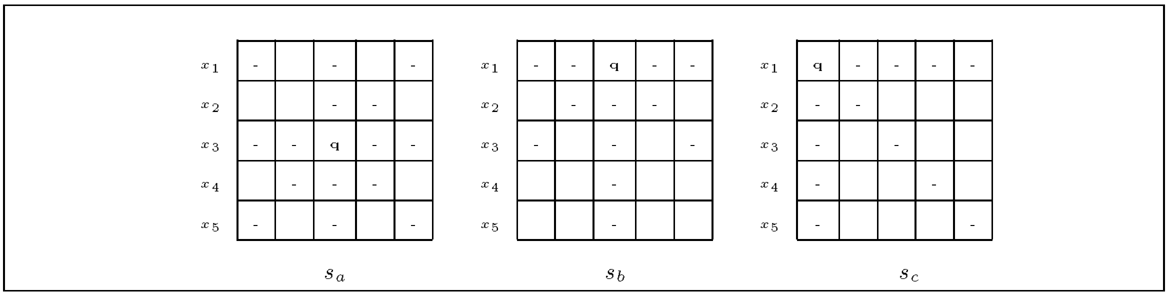


Figure 3: Three states of the 5-queens problem, with different types of local symmetries.

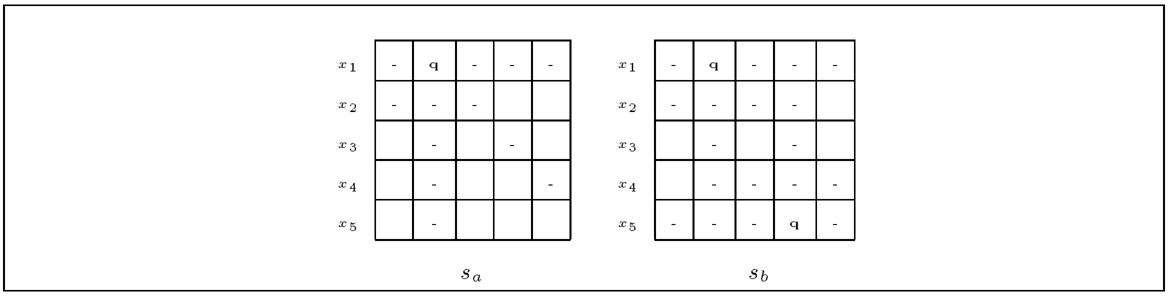


Figure 4: The central rotation symmetry is broken in s_a and restored in s_b.

variable mapping). Regarding the 5-queens problem under the formulation of Fig. 1, there are three equivalence classes of variables: $\{x_1, x_5\}$, $\{x_2, x_4\}$ and $\{x_3\}$. Concerning values, there are also three equivalence classes: $\{1, 5\}$, $\{2, 4\}$ and $\{3\}$. Neither strongly permutable variables nor neighbourhood interchangeable values exist in this problem.

2.3 Symmetries in Search

Symmetries can occur in the initial problem formulation, and also in any search state s, characterized by an assignment of past variables plus the current domains of future variables. State s defines a subproblem of the original problem, where the domain of each past variable is reduced to its assigned value and the relation $rel(c_i)$ of each constraint c_i is reduced to its valid tuples with respect to current domains. A symmetry *holds* at state s if it is a symmetry of the subproblem occurring at s. A symmetry holding at s is said to be *local* to s if it does not change the assignments of past variables [1]. The set of symmetries local to s forms a group with the composition operation. A symmetry holding at the initial state s_0 is called a *global* symmetry of the problem. Any global symmetry is local to s_0, the state where the set of past variables is empty. Symmetries depicted in Figures 1 and 2 are global symmetries of the 5-queens problem. An important property of symmetries is that they are solution-preserving, transforming solutions into solutions.

Let s be a search state with symmetry θ local to it, and s' a successor state. We say that the assignment occurring between s and s' *breaks* symmetry θ if θ is not local to s'. Typically, symmetries local to s are global symmetries that have not been broken by the assignments occurring between s_0 and s. However, this is not always the case. New symmetries may appear in particular states. For the 5-queens problem, some states with local symmetries appear in Figure 3. State s_a keeps as local the three nontrivial global symmetries of the problem, since none is broken by the assignment of x_3. State s_b keeps as local the reflection about the vertical axis only, since the central rotation and the other reflection are broken by the assignment of x_1. In state s_c, all nontrivial global symmetries are broken by the assignment of x_1 but a new symmetry appears: a central rotation of 180 degrees on the 4×4 subboard involving variables from x_2 to x_5 and columns from 2 to 5. A broken symmetry can be restored by another assignment, as

[1]Notice that this definition differs from the one appearing in [17] in that the mapping on past variables is not required to be the identity.

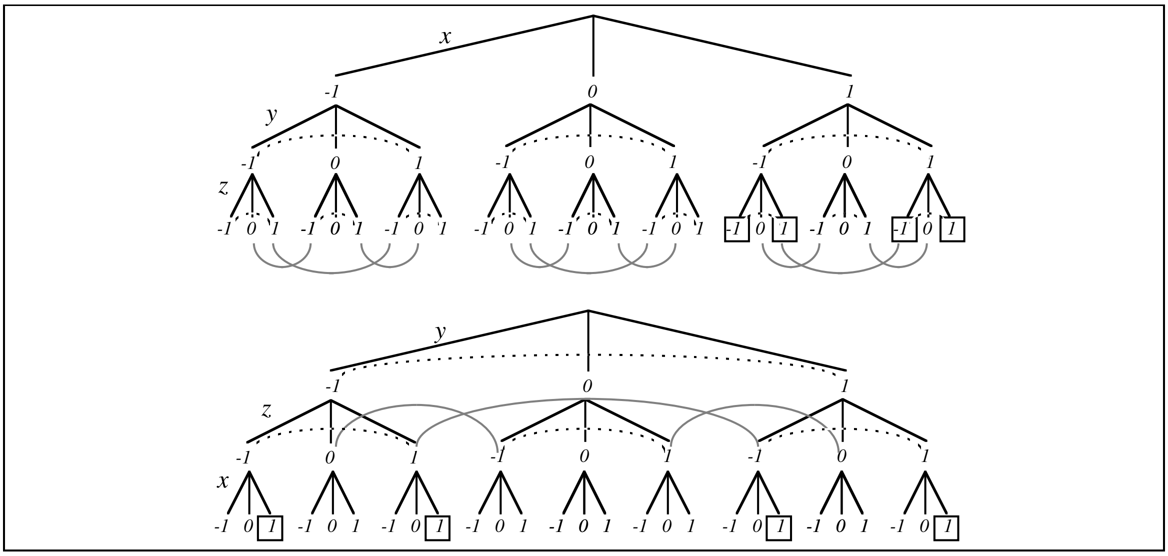


Figure 5: Search tree generated to solve the equation $x+y^2z^2 = 2$ under two variable orderings. Symmetric states originated by permutable variables are connected by shadowed lines, while those arising from interchangeable values are joined by broken lines. Solutions are marked with squares.

it can be seen in Figure 4. In state s_a the assignment of x_1 breaks the central rotation symmetry, which is restored after the assignment of x_5 in state s_b.

3 Heuristics Based on Symmetries

3.1 The Symmetry-Breaking Heuristic

We argue that breaking as many symmetries as possible at each stage is a good strategy to speed up the search. Let us first illustrate some points with a simple example. Consider the equation $x + y^2z^2 = 2$, where all variables take values in $\{-1, 0, 1\}$. There are 5 non trivial symmetries, derived from combining the permutability of y and z, with the sign irrelevance of both y and z. They can be briefly indicated as follows:

1. $\theta(y) = z$, $\theta(z) = y$;
2. $\theta_y = -I$;
3. $\theta_z = -I$;
4. $\theta_y = -I$, $\theta_z = -I$;
5. $\theta(y) = z$, $\theta(z) = y$, $\theta_y = -I$, $\theta_z = -I$;

where I is the identity mapping, and all the entries not specified are also the identity.

Symmetry 1 is a permutation of variables, symmetries 2-4 interchange values, whereas symmetry 5 entails changes in both variables and values. Note that variables y and z are involved in 4 non trivial symmetries each, while variable x is involved in none.

Figure 5 displays two search trees for that equation, following the variable orderings x, y, z and y, z, x. In the upper tree, no symmetry is broken after assigning x, and therefore all symmetries act inside each subtree at the first level, leading to a low density of distinct final states considered whatever the value assigned to x. This can be more easily visualized in Fig. 6, where states symmetric to a previously expanded one have been removed. There are only 3 distinct states among the 9 final states considered in each of the three subtrees resulting from assigning a value to x. For the leftmost subtree, these are (x, y, z)=(-1,-1,-1), (-1,-1,0) and (-1,0,0).

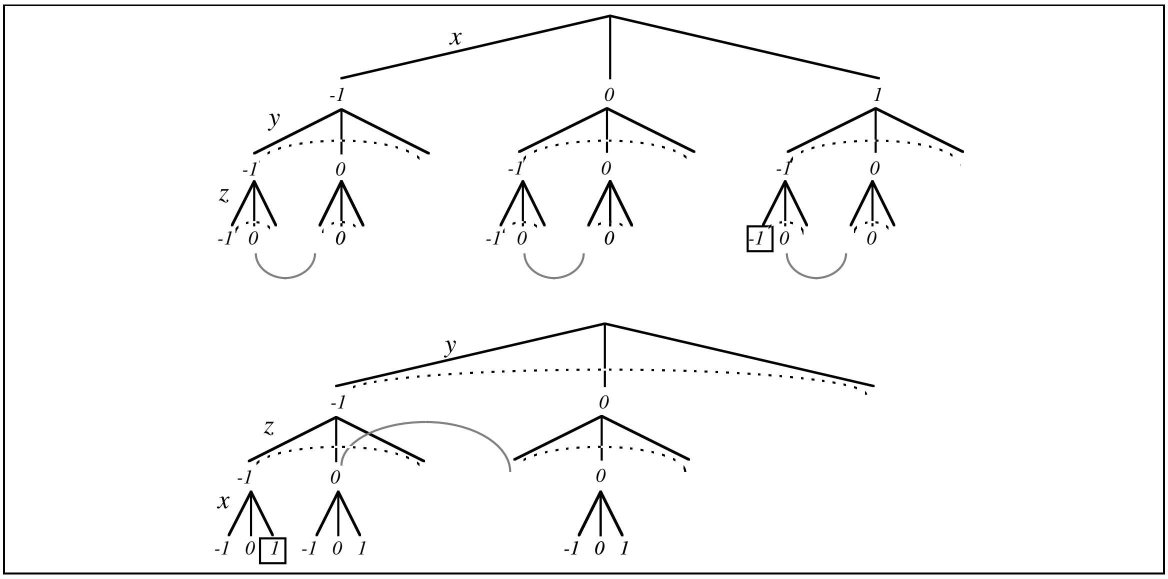


Figure 6: Effect of pruning on the search trees in Fig. 5.

Under the second ordering, represented in the lower tree of Fig. 5, symmetries 1, 2, 4 and 5 are broken after assigning y, and thus only states replicated by symmetry 3 appear inside subtrees at the first level. Concretely, there are 6 distinct states among the 9 final states considered in each of the three subtrees resulting from assigning a value to y. For the leftmost subtree, these are (y, z, x)=(-1,-1,-1), (-1,-1,0), (-1,-1,1), (-1,0,-1), (-1,0,0) and (-1,0,1). The density of distinct final states in each subtree at the first level is thus much higher here (2/3) that under the first ordering (1/3). Again note that this is independent of the value assigned to y. If, in Figure 6, the subtree corresponding to $y = 0$ or $y = 1$ would have been expanded first, instead of that for $y = -1$, then the corresponding subtree would equally have six distinct final states.

When one has no a priori knowledge on the distribution of solutions across the state space, trying to maximize the density of distinct final states considered at each search stage looks like a good strategy. This is the rationale for the following variable selection heuristic.

Symmetry-breaking heuristic: Select for assignment the variable involved in the greatest number of symmetries local to the current state.

The above greedy heuristic, which tries to break as many symmetries as possible at each new variable assignment, produces the following benefits,

1. *Wider distribution of solutions.* Symmetric solutions will spread out under different subtrees instead of grouping together under the same subtree. This increases the likelihood of finding a solution earlier. Take the equation in the example above. It has four solutions, namely $(x = 1, y = -1, z = -1)$, $(x = 1, y = -1, z = 1)$, $(x = 1, y = 1, z = -1)$ and $(x = 1, y = 1, z = 1)$. Under the first variable ordering, they are all grouped below the rightmost subtree, while under the second, they spread two subtrees.

2. *Lookahead of better quality.* A lookahead algorithm prunes future domains taking into account past assignments. When symmetries on future variables are present, some of the lookahead effort is unproductive. If there is a symmetry θ such that $\theta(x_j) = x_k$, with $x_j, x_k \in F$, after lookahead on $D(x_j)$, lookahead on $D(x_k)$ is obviously redundant because it will produce results equivalent (through θ) to lookahead on $D(x_j)$. If no symmetries are present, no lookahead effort will be unproductive. Therefore, the more symmetries are broken, the less unproductive effort lookahead performs. When the number of symmetries is high, savings in unproductive lookahead effort can be substantial.

3. *More effective pruning.* Several techniques to prune symmetric states have been proposed in the literature, such as those based on neigbourhood interchangeable values [10] and on permutable variables [21]. The proposed heuristic amplifies the effect of any pruning technique by moving its operation upwards in the search tree. Figure 6 shows the result of applying the two types of pruning mentioned to the search trees displayed in Fig. 5. The 10 nodes expanded under the variable ordering x, y, z, are reduced to only 6 nodes when the heuristic is in use. Moving pruning upwards tends to produce smaller branching factors in the higher levels of the search tree, resulting in thinner trees.

It is worth noting that points 2 and 3 above apply also to problems without a solution. Empirical results supporting these claims are provided in Section 3.3 for the layout problem.

3.2 The Variety-Maximization Heuristic

Let us return to the example in Figs. 5 and 6. The variable ordering y, z, x suggested by the symmetry-breaking heuristic is the one leading to subtrees with highest density of distinct final states, and, after pruning, it produces the thinnest tree. This is the effect of the heuristic on a problem where all domains have equal sizes. Now consider the same problem but reducing the domain of x to only one value $\{-1\}$. Then, under the variable ordering x, y, z, only the leftmost branch of the upper tree in Fig. 5 would be developed, while under the ordering y, z, x, the whole lower tree in Fig. 5 would be developed, although only for the leaves labelled -1. The effect of pruning could likewise be visualized by looking at Fig. 6. It is clear that, in this case, the best option is the ordering x, y, z since it leads to a thinner tree to start with (13 nodes against 21 for the other ordering) and also after pruning (6 nodes against 14). Thus, in this case, the well-known minimum-domain heuristic would do better than the symmetry-breaking one. And the question arises: When should one or the other heuristic be applied? Even more useful, is there a way of combining both heuristics that outperforms the isolated application of each of them?

To try to answer these questions, let us first recall the interpretations provided for the good performance of the minimum-domain heuristic. The most widespread one is that the heuristic implements the fail-first principle, and thus minimizes the expected depth of each search branch [15]. Smith and Grant [24] tested this interpretation experimentally by comparing the behaviour of several heuristics with increasing fail-first capabilities and concluded that the success of minimum-domain may not necessarily be due to the fact that it implements fail first. Often the effect of shallow branches is counteracted by high branching factors. Thus, another interpretation puts the emphasis on the minimization of the branching factor at the current node [22]: since the minimum-domain heuristic forces the search tree to be as narrow as possible in its upper levels, the expected number of nodes generated is minimized. This holds for problems both with and without a solution. Further along this line, we may view the minimum-domain heuristic as following a least-commitment principle, i.e., it chooses the variable that partitions the state space in less number of subspaces, so that each subspace is larger (contains more states) than if another variable would have been selected. The resulting search trees are, again, as narrow as possible in their upper levels, so the aforementioned node minimization still holds. But now, for problems with a solution, another factor may play a favourable role: in a larger subspace it is more likely to find a solution. A related interpretation was put forth in [11] under the rationale of minimizing the constrainedness of the future subproblem: underconstrained problems tend to have many solutions and be easy to solve.

In dealing with highly symmetric problems, however, the largest subspace does not necessarily contain more distinct final states than a smaller one. Thus, the least-commitment principle has here to be applied in terms of *distinct* final states. What is needed is a strategy that selects the variable leading to consider the highest number of distinct final states, but what we have is,

- the minimum-domain heuristic, which selects the variable that maximizes the *number* of final states considered, and

- the symmetry-breaking heuristic, which chooses the variable that maximizes the *density* of *distinct* final states considered.

In the following, we develop a framework for the combination of both heuristics, based on the two basic types of symmetry, namely interchangeable values and strongly permutable variables. As mentioned in Section 2.2, both types of symmetry induce equivalence classes in the domains and set of variables, respectively. Let $x_1, \ldots x_k$ be the representatives of the equivalence classes of future variables at a given search stage, c_i be the size of the equivalence class to which x_i belongs, and d_i be the number of equivalence classes in $D(x_i)$. In other words, c_i is the number of original variables strongly permutable with x_i, including itself; and d_i is the number of non-interchangeable values that can be assigned to x_i.

Let us calculate the number of distinct final states considered at this search stage, where "distinctiveness" is here taken to mean that no two states can be made equal by interchanging values or permuting variables. For each equivalence class i, we need to assign c_i variables, each of which can take d_i values. If variables were not permutable, the number of joint assignments would be $d_i^{c_i}$. However, since the variables are strongly permutable, two assignments related by a permutation are not distinct. Therefore, the number of distinct joint assignments is given by the combinations with repetition of d_i elements taken c_i at a time. Describing this as an occupancy problem, we need to place c_i balls into d_i buckets (i.e., assign c_i variables, each to one of the possible d_i values). The formula to obtain the number of possible placements (i.e., distinct assignments) is [9][2]: $\binom{d_i + c_i - 1}{c_i}$.

The total number of distinct final states, considering all the equivalence classes of variables, is thus given by the product,

$$\prod_{i=1}^{k} \binom{d_i + c_i - 1}{c_i}.$$

If the next assigned variable belongs to the equivalence class represented by x_{i_0}, then its corresponding term decreases from $\binom{d_{i_0} + c_{i_0} - 1}{c_{i_0}}$ to $\binom{d_{i_0} + c_{i_0} - 2}{c_{i_0} - 1}$, since the equivalence class i_0 loses an element. Thus, the number of distinct final states considered after variable assignment will be,

$$\frac{\binom{d_{i_0} + c_{i_0} - 2}{c_{i_0} - 1}}{\binom{d_{i_0} + c_{i_0} - 1}{c_{i_0}}} \prod_{i=1}^{k} \binom{d_i + c_i - 1}{c_i}.$$

We like to find the i_0 that maximizes this expression, i.e.,

$$\max_i \frac{\binom{d_i + c_i - 2}{c_i - 1}}{\binom{d_i + c_i - 1}{c_i}},$$

which can be developed as,

$$\max_i \frac{\frac{(d_i+c_i-2)!}{(d_i-1)!\ (c_i-1)!}}{\frac{(d_i+c_i-1)!}{(d_i-1)!\ c_i!}},$$

leading to,

$$\max_i \frac{c_i}{d_i + c_i - 1},$$

which is the same as,

$$\min_i \frac{d_i - 1}{c_i}.$$

By taking the index i_0 that realizes this minimum, and assigning a variable in the equivalence class of x_{i_0}, we attain our purpose of considering a subspace with the maximum number of distinct final states, i.e., states containing neither interchangeable values nor strongly permutable variables. This is what the following variable selection heuristic does.

[2]Feller [9, page 38] provides an ingenous and elegant proof: Let us represent the balls by stars and indicate the d_i buckets by the d_i spaces between $d_i + 1$ bars. Thus, $|***|*||||****|$ is used as a symbol for a distribution of $c_i = 8$ balls in $d_i = 6$ buckets with occupancy numbers 3,1,0,0,0,4. Such a symbol necessarily starts and ends with a bar, but the remaining $d_i - 1$ bars and c_i stars can appear in an arbitrary order. In this way it becomes apparent that the number of distinguishable distributions equals the number of ways of selecting c_i places out of $c_i + d_i - 1$, i.e., $\binom{d_i + c_i - 1}{c_i}$.

Variety-maximization heuristic: Select for assignment a variable belonging to the equivalence class for which the ratio $\frac{d_i-1}{c_i}$ is minimum.

When all the equivalence classes of variables are of the same size, then the synthesized heuristic reduces to the minimum-domain one. On the other hand, when all domains have the same number of non-interchangeable values, then the heuristic chooses a variable from the largest equivalence class; this is exactly what the symmetry-breaking heuristic would do. To show this, let us quantify the symmetries broken by a given assignment. Since all permutations inside each class of strongly permutable variables lead to local symmetries, the total number of such symmetries is $c_1!\, c_2! \ldots c_k!$ If we assign a variable from equivalence class i, then the number of remaining symmetries after the assignment will be: $c_1!\, c_2! \ldots (c_i-1)! \ldots c_k!$ Thus, the ratio of remaining symmetries over the total will be $1/c_i$. To maximize symmetry-breaking, we have to determine

$$\min_{1\leq i\leq k} \frac{1}{c_i}$$

which is the same as saying that we have to select a variable from the largest equivalence class.

In sum, by applying the least-commitment principle in terms of maximizing the number of distinct final states considered at each search stage, we have come up with a clean way of combining the minimum-domain and the symmetry-breaking heuristics, so as to extract the best of both along the search.

3.3 An Example: The Layout Problem

To illustrate variety-maximization and its relation with minimum-domain, let us consider the layout problem [13] defined as follows: given a grid, we want to place a number of pieces such that every piece is completely included in the grid and no overlapping occurs between pieces. An example of this problem appears in Figure 7, where three pieces have to be placed inside the proposed grid. As CSP, each piece is represented by one variable whose domain is the set of allowed positions in the grid. There is a symmetry between variables y and z, which are strongly permutable. No symmetry between values exists.

Figure 7 contains two search trees developed by the forward checking algorithm following two variable ordering heuristics. The left tree corresponds to the minimum-domain heuristic, which selects x as first variable ($|D_x| = 3$ while $|D_y| = |D_z| = 4$), and y and z as second and third variables in all the branches. The right tree corresponds to the variety-maximization heuristic. Instead of x, variety-maximization selects y as first variable because $\frac{4-1}{2} < \frac{3-1}{1}$, in agreement with symmetry-breaking. The assignment of y breaks the problem symmetry, so from this point variety-maximization follows minimum-domain. This can be seen in the rightmost branch after assigning y. Variable z is selected as next variable because after forward checking lookahead $|D_z| = 2$ while $|D_x| = 3$. This example shows how variety-maximization combines both symmetry-breaking and minimum-domain heuristics, following at each point the most advisable option (depending on the existing symmetries and domain cardinalities).

To test the benefits that symmetry-breaking (embedded in variety-maximization) brings over minimum-domain, as listed at the end of Section 3.1, we have solved a larger instance of this problem. In a 6×6 square grid, we want to place 4 pieces of size 2×2, plus 4 pieces of size 5×1. As CSP, each piece corresponds to one variable, with domains of cardinalities 25 for 2×2 pieces and 24 for 5×1 pieces. Variables corresponding to equal pieces are strongly permutable. Therefore, there are two equivalence classes of 4 variables each. The minimum-domain heuristic selects two 5×1 pieces as the first two variables of the search tree. At the second level, there are $24^2 = 576$ nodes, 24 of which lead to a solution. The variety-maximization heuristic selects a 5×1 piece as the first variable and a 2×2 piece as second variable. At the second level there are $24 \times 25 = 600$ nodes, 32 of which lead to a solution. The density of nodes leading to a solution at the second level following minimum-domain is $\frac{24}{576} = 0.0417$, and following variety-maximization is $\frac{32}{600} = 0.059$. Thus, variety-maximization yields a better distribution of solutions in the search tree than minimum-domain, increasing the likelihood of finding a solution earlier.

We have solved this problem instance using the standard forward checking algorithm, finding the first solution and all solutions, in successive experiments. Values are selected randomly. Table 1 shows the results averaged over 100 runs, each with a different random seed. We observe that, both in finding one and all solutions, variety-maximization visits less nodes and requires less CPU time than minimum-domain. In addition, Blength records the average length of branches not leading to a solution. We see that variety-maximization generates shorter branches than minimum-domain. Given that the branching factor of both trees is similar, shorter branches suggest a lookahead of better quality. This is also supported

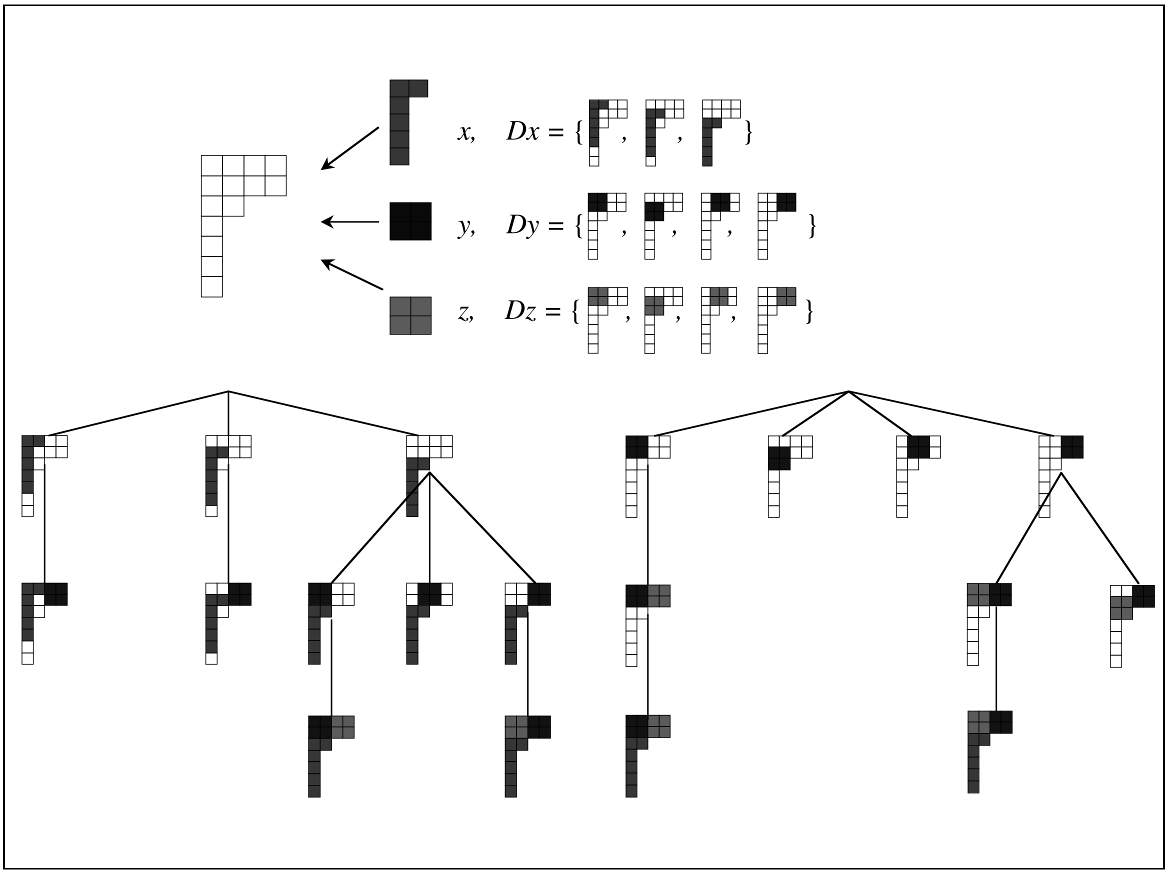


Figure 7: The layout problem and two search trees developed by forward checking with minimum-domain (left) and variety-maximization (right) heuristics.

by the reduction in visited nodes caused by variety-maximization when finding all solutions. We have repeated both experiments including value pruning between strongly permutable variables [21] in the forward checking algorithm. Table 2 shows these results averaged over 100 runs, each with a different random seed. The inclusion of value pruning between strongly permutable variables improves largely the performance of both heuristics. This improvement is higher for variety-maximization when finding one solution. Finding all solutions, the performance of minimum-domain approaches that of variety-maximization. This is because, no matter which variable is selected, all are strongly permutable so they get the benefits of value pruning.

	One solution			All solutions		
heuristic	Nodes	Blength	Time	Nodes	Blength	Time
min-dom	8,906	5.08	0.296	140,656	5.09	4.49
var-max	5,613	4.61	0.239	102,078	4.69	4.29

Table 1: Results of standard forward checking on the layout problem.

	One solution			All solutions		
heuristic	Nodes	Blength	Time	Nodes	Blength	Time
min-dom	1,343	4.52	0.057	14,546	4.69	0.589
var-max	791	3.97	0.045	12,218	4.44	0.489

Table 2: Results of forward checking with value pruning of strongly permutable variables on the layout problem.

4 Value Pruning Based on Symmetries

For problems without a solution, variable selection heuristics can do nothing to avoid revisiting symmetric states along the search. To cope with this shortcoming, we have developed several value pruning strategies, which allow one to reduce the domain of the current and future variables. These strategies remove symmetric values without removing non-symmetric solutions. In the following, we present these strategies and how they are combined, in order to get the maximum profit from symmetric value pruning.

4.1 Domain Reduction

In the particular case that a symmetry θ local to the current state maps the current variable x_k to itself, we can use θ to reduce a priori the current variable domain. Before instantiating x_k, equivalence classes of symmetric values in $D(x_k)$ by θ can be computed, producing $Q_1, Q_2, \ldots, Q_{e_k}$ equivalence classes. A new domain, $D'(x_k)$ is defined as,

$$D'(x_k) = \{w_1, w_2, \ldots, w_{e_k}\}$$

such that each w_i is a representative for the class Q_i. Now, the current variable x_k takes values from $D'(x_k)$ in the following form. If x_k takes value w_i and generates solution S, there is no reason to test other values of Q_i, because they will generate symmetric solutions to S by θ. On the other hand, if value w_i fails, there is no point in testing other values of Q_i because they will fail as well. In this case, all values of Q_i are marked as tested. Once the current variable has been selected, this strategy allows to reduce its domain to non-symmetric values, provided the adequate symmetry θ exists. When backtracking jumps over x_k, equivalence classes are forgotten and the previous $D(x_k)$ is taken as the domain for x_k.

An example of this domain reduction arises in the pigeon-hole problem: locating n pigeons in $n-1$ holes such that each pigeon is in a different hole. This problem is formulated as a CSP by associating a variable x_i to each pigeon, all sharing the domain $\{1, \ldots, n-1\}$, under the constraints $x_i \neq x_j$, $1 \leq i, j \leq n$, $i \neq j$. Among others, this problem has a collection of symmetries in the domains,

$$\forall i, \;\; \forall a, a' \in D(x_i) \; a \neq a', \;\; \exists \theta, \;\; \theta = I, \;\; \theta_i(a) = a', \;\; \theta_i(a') = a$$

where I is the identity mapping. If variables and values are considered lexicographically, before assigning x_1 all values in $D(x_1)$ form a single equivalence class. Then, $D'(x_1) = \{1\}$. Performing search by forward checking, value 1 is removed from all future domains. Considering x_2, all its values form a single equivalence class, $D'(x_2) = \{2\}$. Again, lookahead removes value 2 from all future domains. Considering x_3, all its remaining values form a single equivalence class, $D'(x_3) = \{3\}$, etc. This process goes on until assigning $(x_{n-1}, n-1)$, when lookahead finds an empty domain in $D(x_n)$, so backtracking starts. At that point, all domains of past and current variables have been reduced to a single value, which is currently assigned. Backtracking does not find any other alternative value to test in any previous variable, so it ends with failure when x_1 is reached. Only the leftmost branch of the search tree is generated, and the rest of the tree is pruned.

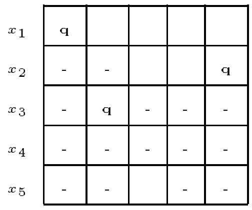
4.2 Value Pruning Through Nogood Recording

A *nogood* is an assignment of values to a subset of variables which does not belong to any solution. Before search, a set of nogoods is determined by the constraints as the set of forbidden value tuples. During search, new nogoods are discovered by the resolution of nogoods responsible of dead-ends. For example, in Fig. 8, the forward checking algorithm finds a dead-end in the 5-queens problem ($D(x_4) = \emptyset$). By the resolution of the nogoods associated with every pruned value of $D(x_4)$, we get the new nogood,

$$(x_1, 1)(x_2, 5)(x_3, 2)$$

which means that variables x_1, x_2 and x_3 cannot simultaneously take the values 1, 5 and 2, respectively. Often nogoods are written in oriented form as,

$$(x_1 = 1) \wedge (x_2 = 5) \Rightarrow (x_3 \neq 2)$$



$$\left.\begin{array}{lcl} (x_1 = 1) & \Rightarrow & (x_4 \neq 1) \\ (x_1 = 1) & \Rightarrow & (x_4 \neq 4) \\ (x_2 = 5) & \Rightarrow & (x_4 \neq 3) \\ (x_2 = 5) & \Rightarrow & (x_4 \neq 5) \\ (x_3 = 2) & \Rightarrow & (x_4 \neq 2) \end{array}\right\} (x_1 = 1) \wedge (x_2 = 5) \Rightarrow (x_3 \neq 2)$$

Figure 8: Nogood resolution in a dead-end for the 5-queens problem.

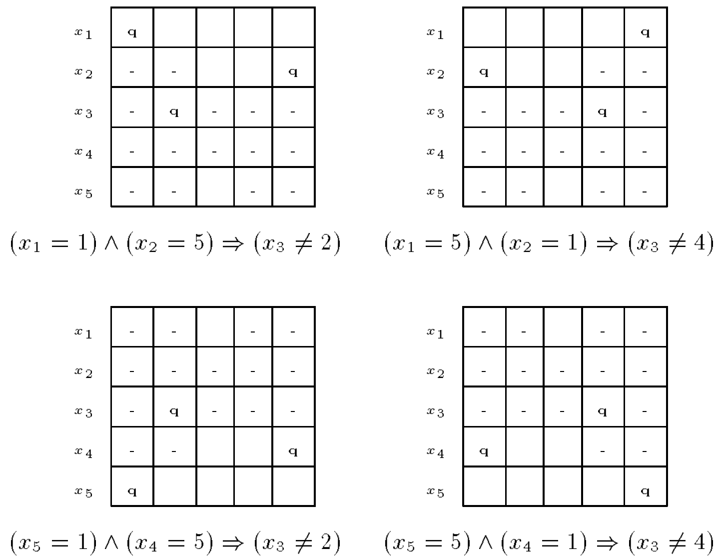


Figure 9: Symmetric nogoods in the 5-queens problem. Left-right symmetry: reflection about the vertical axis. Up-down symmetry: reflection about the horizontal axis.

where the variable at the right-hand side is the last variable among the variables of the nogood that has been instantiated. This variable will be the one changed first when performing backtracking, which is needed to guarantee completeness of tree-search algorithms (see [2] for a detailed explanation of nogood resolution).

4.2.1 Value Pruning due to Symmetric Nogoods

Let $p = (x_1, v_1)(x_2, v_2)\ldots(x_k, v_k)$ be a nogood found during search and θ a global symmetry of the considered problem. It is easy to see that the tuple $\theta(p)$, defined as $(\theta(x_1), \theta_1(v_1))(\theta(x_2), \theta_2(v_2))\ldots(\theta(x_k), \theta_k(v_k))$, is also a nogood. Let us suppose that $\theta(p)$ is not a nogood, that is, it belongs to a solution S. Given that θ^{-1} is also a problem symmetry and problem solutions are invariant through symmetries, $\theta^{-1}(S)$ is also a solution. But $\theta^{-1}(S)$ contains p, in contradiction with the first assumption that p is a nogood. Therefore, $\theta(p)$ is a nogood. Intuitively, $\theta(p)$ is the nogood that we would obtain following a search trajectory symmetric by θ to the current trajectory. An example of this appears in Fig. 9.

Given that we can generate nogoods using previously found nogoods and global symmetries of the problem, we propose to learn nogoods during search in the following form,

1. We store the new nogoods found during search.

2. At each node, we test if the current assignment satisfies some symmetric nogood, obtained by applying a global symmetry to a stored nogood. If it does, the value of the current variable is unfeasible so it can be pruned. Values removed in this way are restored when backtracking jumps above their corresponding variables.

Nogood recording in search presents two main issues: storage size and overhead [8]. Regarding the storage space required, it may be of exponential size which could render the strategy inapplicable in practice. The usual way to overcome this drawback is to store not all but a subset of the nogoods found, following different strategies: storing nogoods of size lower than some limit, fixing in advance the storage capacity and using some policy for nogood replacement, etc. However, this important drawback has been shown to be surmountable in practice due to the following fact: a new nogood is never symmetric to an already stored nogood. Otherwise, the assignment leading to this new nogood would have been found unfeasible, because of the existence of a symmetric nogood, and it would have been pruned before producing the new nogood. If the number of global symmetries is high enough, this may cause a very significant decrement in the number of stored nogoods.

Regarding the overhead caused by nogood recording, it has two main parts: nogood recording and testing against symmetric nogoods. Nogood recording is a simple process performed on a subset of the visited nodes, causing little overhead. However, testing each node against symmetric nogoods could mean checking an exponential number of nogoods per node, which would severely degrade performance, eliminating any possible savings caused by value removal. To prevent this situation, we restrict the number of symmetric nogoods against which the current node is tested, following two criteria,

1. A subset of all global symmetries are used for symmetric nogood generation. The composition of this subset is problem dependent (see Section 5 for further details).

2. A subset of stored nogoods is considered for symmetric nogood generation. If x_i is the current variable and θ is a global symmetry, only nogoods containing $\theta(x_i)$ in its rigth-hand side are considered.

Nevertheless, there are some particular cases where we can prune values without checking stored nogoods, as explained in the following subsection.

4.2.2 Symmetric Nogoods at the Current Branch

Let s be a state defined by the assignment of past variables $\{(x_i, v_i)\}_{i \in P}$, θ a symmetry local to s, and x_k the current variable. If after the assignment of x_k the nogood p is found,

$$p = \bigwedge_{j \in P', P' \subseteq P} (x_j, v_j) \Rightarrow (x_k \neq v_k)$$

it is easy to see that $\theta(p)$ is also a nogood. If p is a nogood, it means that it violates a constraint c. By the definition of symmetry, $\theta(p)$ violates the symmetric constraint c^θ. Therefore, $\theta(p)$ is also a nogood. The interesting point is that $\theta(p)$ also holds at the current state. Effectively,

$$\theta(p) = \bigwedge_{j \in P', P' \subseteq P} (\theta(x_j), \theta_j(v_j)) \Rightarrow (\theta_k(x_k) \neq \theta_k(v_k)) = \bigwedge_{j \in P'', P'' \subseteq P} (x_j, v_j) \Rightarrow (\theta_k(x_k) \neq \theta_k(v_k))$$

since all variables in the left-hand side of p are past variables, so they are mapped to other past variables and their assignments are not changed by θ. Therefore, at this point we can remove $\theta_k(v_k)$ (the value symmetric to v_k) from $D(\theta(x_k))$, because it cannot belong to any solution including the current assignment of past variables. If all values of x_k are tried without success and the algorithm backtracks, all values removed in this way should be restored. If x_k is involved in several symmetries, this reasoning holds for each of them separately. Thus, this strategy can be applied to any variable symmetric to x_k.

This strategy of value removal after failure provides further support to the symmetry-breaking heuristic of Section 3.1. The more local symmetries a variable is involved in, the more opportunities it offers for symmetric value removal in other domains if a failure occurs. This extra pruning is more effective if it is done at early levels of the search tree, since each pruned value represents removing a subtree on the level corresponding to the variable symmetric to the current one.

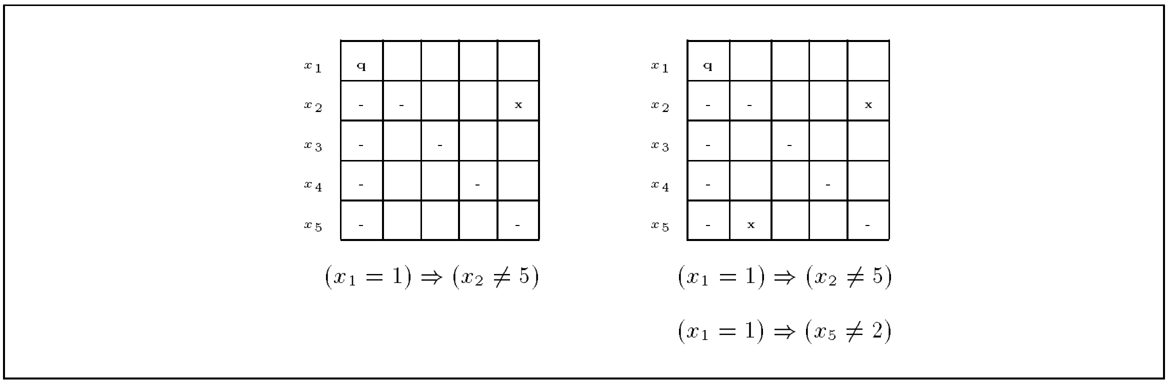


Figure 10: Symmetric nogoods by central rotation of 180 degrees in the subboard including variables x_2 to x_5 and columns 2 to 5.

An example of this pruning capacity appears in Figure 10: further resolution of the nogoods of x_3 in Figure 8 produces the nogood $(x_1 = 1) \Rightarrow (x_2 \neq 5)$. The rotation of 180 degrees of the subboard including variables x_2 to x_5 and columns 2 to 5, is a symmetry local to the state after the assignment $(x_1, 1)$. Therefore, applying this symmetry to the nogood, a new nogood is obtained,

$$(x_1 = 1) \Rightarrow (x_5 \neq 2)$$

which is a justification to prune value 2 from $D(x_5)$.

4.3 Combination of Pruning Strategies

The three pruning strategies mentioned, namely (i) domain reduction, (ii) value pruning due to symmetric nogoods, and (iii) value pruning due to symmetric nogoods at the current branch, can be combined to obtain the maximum profit in future domain reduction. The domain of the current variable is reduced (assuming that the adequate symmetry exists). If, for some reason (lookahead or symmetric nogood existence), its current value is discarded, all values of the same equivalence class are also discarded. If the current variable is symmetric with other future variables, the symmetric images of the discarded values of the current variable can be removed from the domains of the symmetric future variables. This cascade of value removal and symmetry chaining has been shown very effective in the problems tackled (refer to Section 5). In this process, any removed value is labeled with the justification of its removal, computed by applying the corresponding symmetry operators to the nogood which started the pruning sequence. In the following, these strategies are generically named *symmetric value pruning*, and they are implemented by a single procedure called SVP.

5 Experimental Results

5.1 The Ramsey Problem

Aside from the pigeonhole and the n-queens problems, it is hard to find a highly symmetric problem that has been tackled by several researchers following different approaches. The Ramsey problem is one of the rare exceptions. Puget [20] reported results on several instances of this problem obtained by adding ad hoc ordering constraints to its formulation, so as to break symmetries. Gent and Smith [12] followed the alternative approach of pruning symmetric states from the search tree after failure, and compared their results with Puget's. Thus, we think this is a good problem on which to test the efficiency of our symmetry-breaking heuristic and its further enhancements described in the preceding section.

5.1.1 Problem Formulation

Given a complete graph[3] with n nodes, the problem is to colour its edges with c colours, without getting any monochromatic triangle. In other words, for any three nodes n_1, n_2, n_3, the three edges $(n_1, n_2), (n_1, n_3), (n_2, n_3)$ must not have all three the same colour. In the case of 3 colours, it is well known that there are many solutions for $n = 16$, but none for $n = 17$.

This problem can be formulated as a CSP as follows. The variables $x_{ij}, 1 \leq i, j \leq n, i < j$, are the edges of the complete graph, the domains are all equal to the set of three colours $\{c_1, c_2, c_3\}$, and the constraints can be expressed as follows:

$$(x_{ij} \neq x_{ik}) \text{ or } (x_{ij} \neq x_{jk}), \quad \forall i, j, k, \quad i < j < k.$$

All colour permutations and all node permutations are *global* symmetries of the problem. To break them in the problem formulation, Puget [20] added three ordering constraints, one based on values and the remaining two based on cardinalities, as detailed in [12]. Later, Gent and Smith [12] replaced the constraint on values by their procedure of value pruning after failure. A comparison of their results with ours can be found in the next subsection.

Our heuristic does not make use of global symmetries, instead it exploits symmetries *local* to each search state. The latter are determined by the automorphisms of the coloured graph developed so far. Since automorphisms derived from composing colour permutations and general node permutations are very expensive to detect, and we need a simple test that can be applied repeatedly at node expansion, we concentrate on a particular type of node permutation that leaves unchanged the coloured graph developed so far, as described below.

When can two nodes i and j be interchanged without altering the colour graph developed so far? The necessary and sufficient condition is that[4] $x_{ik} = x_{jk}, \forall k$, which can be easily assessed by checking the equality of rows i and j of the adjacency matrix for the graph. Note that this condition requires that x_{ik} and x_{jk} are both either past variables or future variables and, in the former case, they must have the same colour assigned.

Every pair of node interchanges (transpositions) of the type mentioned above defines a symmetry. For instance, if we can interchange nodes i and j, and also nodes k and l, then we have the following symmetry local to the current state,

$$\begin{array}{l} \phi(x_{ij}) = x_{ij}, \quad \phi(x_{kl}) = x_{kl}, \\ \phi(x_{ik}) = x_{jl}, \quad \phi(x_{jl}) = x_{ik}, \quad \phi(x_{il}) = x_{jk}, \quad \phi(x_{jk}) = x_{il}, \\ \phi(x_{ir}) = x_{jr}, \quad \phi(x_{jr}) = x_{ir}, \quad \phi(x_{kr}) = x_{lr}, \quad \phi(x_{lr}) = x_{kr}, \quad \forall r \quad r \neq i \quad r \neq j \quad r \neq k \quad r \neq l, \\ \phi(x_{qr}) = x_{qr}, \quad \forall q, r \quad q, r \neq i \quad q, r \neq j \quad q, r \neq k \quad q, r \neq l, \\ \phi_{qr} = I, \quad \forall q, r. \end{array}$$

We restrict our analysis and experimentation to symmetries resulting from the combination of such node interchanges. They are easy to detect and constitute an important subset of all automorphisms of the coloured graph developed so far. Of course, conditions for progressively more complex subgraph interchangeability, such as those sketched in [21], could be developed for the Ramsey problem, but it is not clear that the effort required to detect more complex symmetries would pay off in terms of search efficiency.

Let us calculate the number of local symmetries of the type mentioned. First note that interchangeability of nodes is an equivalence relation leading to a partition of the set of nodes into equivalence classes. Suppose $e_1, e_2, \ldots e_k$ are the sizes of such classes at the current state. Then, since all permutations inside each class lead to local symmetries, the total number of such symmetries is $e_1! \; e_2! \; \ldots \; e_k!$.

If we assign a variable x_{ij}, with i and j belonging to the same equivalence class, say p, then the number of remaining symmetries after the assignment will be: $e_1! \; e_2! \; \ldots \; 2 \; (e_p - 2)! \; \ldots \; e_k!$, because i and j will now belong to a new class. If, on the contrary, i belongs to class p, and j belongs to class q, $p \neq q$, then the number of remaining symmetries after assigning x_{ij} will be: $e_1! \; e_2! \; \ldots \; (e_p - 1)! \; \ldots \; (e_q - 1)! \; \ldots \; e_k!$ Thus, the ratio of remaining symmetries over the total will be $2/(e_p(e_p - 1))$ in the former case, and $1/(e_p e_q)$ in the latter one.

[3] A graph in which each node is connected to every other node.

[4] For each pair of nodes $(i, j), i \neq j$, there is only one variable, either x_{ij} or x_{ji}, depending on whether $i < j$ or $j < i$. To ease the notation, in what follows, we will not distinguish between the two cases, and thus both x_{ij} and x_{ji} will refer to the same, unique variable.

To maximize symmetry-breaking, we have to determine

$$\min_{1\leq p,q\leq k, p\neq q}\left\{\frac{2}{e_p(e_p-1)}, \frac{1}{e_p e_q}\right\}$$

Now, note that the equivalence relation over nodes induces an equivalence relation over edges, which are the variables of our problem. Two variables x_{ik} and x_{jl} are symmetric if and only if either ($i \equiv j$ and $k \equiv l$) or ($i \equiv l$ and $k \equiv j$), where $\equiv$ denotes node interchangeability. The size c_{ij} of the equivalence class to which x_{ij} belongs is,

$$c_{ij} = \begin{cases} e_p(e_p-1)/2, & \text{if } i \text{ and } j \text{ belong to the same node class } p \\ e_p e_q, & \text{if } i \text{ belongs to class } p, \text{ and } j \text{ belongs to class } q. \end{cases}$$

Therefore, to maximize symmetry-breaking we have to select a variable x_{ij} from the largest equivalence class, in perfect agreement with the case in which we had strongly permutable variables.

5.1.2 Results and Discussion

We aimed at solving the Ramsey problem with 3 colours using the same algorithm and heuristics for solvable and unsolvable cases. As reference algorithm, we take forward checking with conflict-directed backjumping (FC-CBJ) [19], adapted to deal with ternary constraints.

Regarding variable selection heuristics, we tried the following ones (criteria ordering indicates priority),

- DG: minimum domain, maximum degree[5], breaking ties randomly.
- DGS: minimum domain, maximum degree, largest equivalence class, breaking ties randomly.
- VM': we tried the variety-maximization heuristic (VM), which combines minimum-domain and symmetry-breaking. Since VM does not include the degree, which has proved to be quite important for variable selection in this problem, we combined them both in the following way:
  - if the variable selected by VM has a two-valued domain (i.e., minimum-domain dominates symmetry-breaking), use the DG heuristic;
  - if the variable selected by VM has a three-valued domain (i.e., symmetry-breaking dominates minimum-domain), use the following heuristic: maximum degree, largest equivalence class, breaking ties randomly.

  Notice that local symmetries induced by node interchanges do not generate equivalence classes of strongly permutable variables, so the justification for the VM heuristic does not strictly hold in this case. Nevertheless, we take VM as an approximation for the combination of minimum-domain and symmetry-breaking heuristics.

The value selection heuristic is as follows: for variable x_{ij}, select the colour with less occurrences in all triangles including x_{ij} with only one coloured edge, breaking ties randomly.

The FC-CBJ algorithm was unable to find that no solution exists for $n = 17$ within 1 CPU hour, for any of the considered heuristics. Then, we added the symmetric value pruning procedure SVP[6], obtaining the FC-CBJ-SVP algorithm, which has been able to solve the Ramsey problem for n from 14 to 17 with the proposed heuristics. Given that several decisions are taken randomly, we repeated the execution for each dimension 100 times, each with a different random seed. Execution of a single instance was aborted if the algorithm visited more than 100,000 nodes.

Experimental results appear in Table 3, where for each n and heuristic, we give the number of solved instances within the node limit, and for those instances, the average number of visited nodes, the average number of fails and the average CPU time.

[5] In this problem, we take as degree of variable x_{ij} (edge from node i to node j) the number of triangles including x_{ij} with only one edge coloured.

[6] If x_{ij} is the current variable, the subset of symmetries used for symmetric nogood generation is formed by the symmetries exchanging one node (node i or node j) while the other (node j or node i) is kept fixed.

	Fc-cbj-svp											
	dg				dgs				vm'			
n	Sol	Nodes	Fails	Time	Sol	Nodes	Fails	Time	Sol	Nodes	Fails	Time
14	99	4494	1009	3.03	100	2167	384	0.69	100	201	15	0.20
15	59	20673	6627	19.46	100	20706	6168	19.50	100	1732	237	0.57
16	100	17172	5290	13.01	100	17027	5247	13.01	100	906	114	0.35
17	100	7418	3232	1.41	100	7485	3175	1.86	100	2952	1132	0.75

Table 3: Performance results for the Ramsey problem.

	Gent and Smith		Puget	
n	Fails	Time	Fails	Time
16	2030	1.61	2437	1.40
17	161	0.26	636	0.27

Table 4: Performance results of previous approaches on the Ramsey problem (from [12]).

We compare the three variable selection heuristics dg, dgs and vm', within the Fc-cbj-svp algorithm. Of the 400 runs, Fc-cbj-svp with dg solved 358 instances within the node limit, while it was able to solve all instances with dgs or vm'. Considering instances solved within the node limit, there is little difference between dg and dgs, except for $n = 14$ where dgs improves significantly over dg. A main improvement in performance occurs when passing from dgs to vm'. For solvable cases, we observe a decrement of one order of magnitude in visited nodes and number of fails, and of almost two orders of magnitude in CPU time. For $n = 17$, the improvement is not so strong but it is still important.

These results show clearly the importance of exploiting symmetries in the solving process. The svp procedure allowed us to achieve an efficient solution for $n = 17$. The symmetry-breaking heuristic permitted to solve all instances within the node limit, preventing the search process from getting lost in large subspaces without solution. vm' uses the same information as dgs but in a more suitable way, leading to a very substantial improvement for solvable dimensions. Thus, results substantiate the dominance of vm' over dgs, providing experimental support to the theoretically-developed variety-maximization heuristic.

We compare these results with those of Puget [20] and Gent and Smith [12], which are given in Table 4. For $n = 16$, the number of fails for the dgs is higher than Puget's, and Gent and Smith's numbers, while the number of fails for the vm' heuristic is one order of magnitude lower than Puget's, and Gent and Smith's numbers. For dimension 17, results from dgs and vm' are worse than previous approaches. This is not surprising, because our variable selection heuristics have been devised for solvable problems. CPU time cannot be compared because these results come from different machines. From this comparison, we can affirm that our approach, based on a new variable ordering and a pruning procedure, remains competitive with more sophisticated approaches based on a careful problem formulation [20] plus the inclusion of new constraints during search [12], and it is even able to outperform them for solvable dimensions.

5.2 BIBD Generation

Block designs are combinatorial objects satisfying a set of integer constraints [14, 5]. Introduced in the thirties by statisticians working on experiment planning, nowadays they are used in many other fields, such as coding theory, network reliability, and cryptography. The most widely used designs are the Balanced Incomplete Block Designs (BIBDs). Although up to our knowledge, BIBD generation has not been tackled from the CSP viewpoint, it appears to be a wonderful instance of highly symmetric CSP, thus offering the possibility to assess the benefits of different search strategies on such problems.

5.2.1 Problem Formulation

Formally, a (v, b, r, k, λ)-BIBD is a family of b sets (called blocks) of size k, whose elements are from a set of cardinality v, $k < v$, such that every element belongs exactly to r blocks and every pair of elements occurs exactly in λ blocks. v, b, r, k, and λ are called the parameters of the design. Computationally,

$$
\begin{array}{ccccccc}
0&1&1&0&0&1&0\\
1&0&1&0&1&0&0\\
0&0&1&1&0&0&1\\
1&1&0&0&0&0&1\\
0&0&0&0&1&1&1\\
1&0&0&1&0&1&0\\
0&1&0&1&1&0&0
\end{array}
$$

Figure 11: An instance of (7,7,3,3,1)-BIBD.

designs can be represented by a $v \times b$ binary matrix, with exactly r ones per row, k ones per column, and the scalar product of every pair of rows is equal to λ. An example of BIBD appears in Fig. 11.

There are three necessary conditions for the existence of a BIBD:

1. $rv = bk$,
2. $\lambda(v-1) = r(k-1)$, and
3. $b \geq v$.

However, these are not sufficient conditions. The situation is summarized in [16], that lists all parameter sets obeying these conditions, with $r \leq 41$ and $3 \leq k \leq v/2$ (cases with $k \leq 2$ are trivial, while cases with $k > v/2$ are represented by their corresponding complementaries, which are also block designs). For some parameter sets satisfying the above conditions, it has been established that the corresponding design does not exist; for others, the currently known bound on the number of *non-isomorphic* solutions is provided; and finally, some listed cases remain unsettled. The smallest such case is that with parameters (22,33,12,8,4), to whose solution many efforts have been devoted [25, Chapter 11].

Some (infinite) families of block designs (designs whose parameters satisfy particular properties) can be constructed analytically, by direct or recursive methods [14, Chapter 15], and the state of the art in computational methods for design generation is described in [5, 25]. The aforementioned unsettled case, with $vb = 726$ binary entries, shows that exhaustive search is still intractable for designs of this size. In the general case, the algorithmic generation of block designs is an NP problem [6].

Computational methods for BIBD generation, either based on systematic or randomized search procedures, suffer from combinatorial explosion which is partially due to the large number of isomorphic configurations present in the search space. The use of group actions goes precisely in the direction of reducing this isomorphism [25, Chapter 3]. Thus, BIBD generation can be viewed as a large family of highly symmetric CSPs and, as such, constitutes a good testbed on which to test strategies to exploit symmetries within constraint satisfaction search.

The problem of generating a (v, b, r, k, λ)-BIBD admits several CSP formulations. The most direct one would be representing each matrix entry by a binary variable. Then, there would be three types of constraints: (i) v b-ary constraints ensuring that the number of ones per row is exactly r, (ii) b v-ary constraints ensuring that the number of ones per column is exactly k, and (iii) $v(v-1)/2$ $2b$-ary constraints ensuring that the scalar product of each pair of rows is exactly λ. All are high-arity constraints, but especially the last type is very costly to deal with, because of its highest arity and its large number of instances.

We have opted for an alternative formulation that avoids constraints of type (iii), as follows. Two rows i and j of the BIBD should have exactly λ ones in the same columns. We represent this by λ variables $x_{ijp}, 1 \leq p \leq \lambda$, where x_{ijp} contains the column of the pth one common to rows i and j. There are $v(v-1)/2$ row pairs, so there are $\lambda v(v-1)/2$ variables, all sharing the domain $\{1, \ldots, b\}$. From these variables, the BIBD $v \times b$ binary matrix T is computed as follows:

$$T[i,c] = \begin{cases} 1, & \text{if } \exists j, p \text{ s.t. } x_{ijp} = c \text{ or } x_{jip} = c, \\ 0, & \text{otherwise.} \end{cases}$$

Constraints are expressed in the following terms,

$$x_{ijp} \neq x_{ijp'}; \qquad \sum_{c=1}^{b} T[i,c] = r; \qquad \sum_{i=1}^{v} T[i,c] = k$$

where $1 \leq p, p' \leq \lambda$, $1 \leq i, j \leq v$, $1 \leq c \leq b$. Note that the last two types of constraints are exactly the same as the former two in the previous formulation, while we have replaced the costly type (iii) constraints by binary inequality constraints. This reduces considerably the pruning effort.

Turning to symmetries, all row and column permutations are global symmetries of the problem, which are retained in both formulations above. Note, however, that each of these symmetries involves interchanging many variables at once, i.e., they do not yield strongly permutable variables in neither of the two formulations. Moreover, as variables are assigned, many of these global symmetries disappear, because they involve changing past variables. Since we are interested in *local* symmetries that can be *easily detected*, we consider the following ones relating future variables,

1. Variable mapping exchanges x_{ijp} and $x_{ijp'}$, domain mappings are the identity; this symmetry occurs among variables of the same row pair.

2. Variable mapping is the identity, one domain mapping exchanges values c_1 and c_2; this symmetry occurs when $T[l, c_1] = T[l, c_2]$ for $l = 1, \ldots, v$.

3. Variable mapping exchanges x_{ijp} and $x_{i'j'p'}$, domain mappings are the identity; this symmetry occurs when $T[i, c] = T[i', c]$ and $T[j, c] = T[j', c]$ for $c = 1, \ldots, b$.

4. Variable mapping exchanges x_{ij_1p} and $x_{ij_2p'}$, the domain mappings corresponding to these variables exchange values c_1 and c_2; this symmetry occurs when,
   $T[j_1, c_1] = T[j_2, c_2] = 1,\ T[j_1, c_2] = T[j_2, c_1] = 0,$
   $T[j_1, c] = T[j_2, c],\ c = 1, \ldots, b,\ c \neq c_1,\ c \neq c_2,$
   $T[j, c_1] = T[j, c_2],\ j = 1, \ldots, v,\ j \neq j_1,\ j \neq j_2.$

5. Variable mapping exchanges x_{ij_1p} and $x_{ij_2p'}$, the domain mappings corresponding to these variables exchange values c_1 and c_2, and c_3 and c_4; this symmetry occurs when,
   $T[j_1, c_1] = T[j_2, c_2] = 1,\ T[j_1, c_2] = T[j_2, c_1] = 0,$
   $T[j_1, c_3] = T[j_2, c_4] = 1,\ T[j_1, c_4] = T[j_2, c_3] = 0,$
   $T[j_1, c] = T[j_2, c],\ c = 1, \ldots, b,\ c \neq c_1,\ c \neq c_2,\ c \neq c_3,\ c \neq c_4,$
   $T[j, c_1] = T[j, c_2],\ j = 1, \ldots, v,\ j \neq j_1,\ j \neq j_2,$
   $T[j, c_3] = T[j, c_4],\ j = 1, \ldots, v,\ j \neq j_1,\ j \neq j_2.$

These symmetries have a clear interpretation. Symmetry (1) is inherent to the formulation. Symmetry (2) is the local version of column permutability: assigned values must be equal in columns c_1 and c_2, for the values c_1 and c_2 of a variable to be interchangeable. Symmetry (3) is the local version of two pairs of simultaneous row permutations: rows i and i' (respectively, rows j and j') must have the same assigned values for variables x_{ijp} and $x_{i'j'p'}$ to be permutable. The next two symmetries are generalizations of the preceding one. Symmetry (4) relates variables sharing row i, and rows j_1 and j_2 that are equal but for two columns c_1 and c_2. These columns are also equal but for rows j_1 and j_2. Exchanging rows j_1 and j_2, and columns c_1 and c_2, matrix T remains invariant. Symmetry (5) develops the same idea in the case where i is not shared, and thus two rows i_1 and i_2 need to be considered. It occurs when exchanging rows i_1 and i_2, and columns c_1 and c_2, and c_3 and c_4, matrix T remains invariant. It is worth noting that these symmetries keep invariant matrix T because they are local to the current state, that is, they do not change past variables.

Concerning the way symmetries act on variables, symmetry (1) is the only one defining strongly permutable variables. Symmetries (3), (4) and (5) are induced by exchanging rows and columns within the BIBD matrix, leading to equivalence relations of the same type as in the Ramsey problem. Taken together, the symmetries of the latter three types form a subgroup, leading to equivalence classes in which the variables are related by one symmetry type only. In other words, if two variables within a class are related by a given symmetry, all other variables in the class are related by symmetries of the same type. Let us consider a variable x_{ijp} which is strongly permutable with $c_r - 1$ other variables through symmetry (1), and which belongs to a class of size c'_s when the subgroup formed by symmetries (3), (4) and (5) is considered. Then, x_{ijp} belongs to a class of size $c_r c'_s$ when the four variable symmetries are considered together. Now, by combining the reasonings in Sections 3.2 and 5.1.1, we can deduce that, after assigning x_{ijp}, the ratio of remaining symmetries over those before the assignment would be:

$$\min_{r,s} \frac{1}{c_r c'_s}.$$

Therefore, in the case of BIBDs, in order to maximize symmetry-breaking, we also have to select a variable belonging to the largest equivalence class.

5.2.2 Results and Discussion

BIBD generation is a non-binary CSP. We use a forward checking algorithm with conflict-directed back-jumping (FC-CBJ [19]) adapted to deal with non-binary constraints as reference algorithm.

Regarding variable selection heuristics, we tried the following ones (criteria ordering indicates priority),

- DG: minimum-domain, maximum-degree[7], breaking ties randomly.
- SDG: symmetry-breaking, minimum-domain, maximum-degree, breaking ties randomly.
- VM: variety-maximization heuristic, maximum-degree, breaking ties randomly.

Equivalence classes for variables are computed using symmetries 1, 3, 4 and 5, defined in the preceding subsection. Only symmetry 1 generates strongly permutable variables, so justification for the VM heuristic does not strictly hold in this case. Nevertheless, we take VM as an approximation for the combination of minimum-domain and symmetry-breaking heuristics. Equivalence classes for values are computed using symmetry 2. Values are selected as follows,

- if $\lambda = 1$, a value within the largest equivalence class;
- if $\lambda > 1$, randomly.

We compare the performance of these heuristics generating all BIBDs with $vb < 1400$ and $k = 3$, all having solution. Since the performance of the proposed algorithm depends on random choices, we have repeated the generation of each BIBD 50 times, each with a different random seed. Execution of a single instance was aborted if the algorithm visited more than 50,000 nodes.

Empirical results appear in Table 5, where for each heuristic and BIBD, we give the number of solved instances within the node limit, the average number of visited nodes of solved instances, and the average CPU time in seconds for the 50 instances. Of the 2400 instances executed, FC-CBJ with DG solves 940, with SDG solves 2393 and with VM solves 2394. FC-CBJ with DG does not solve any instance for 5 specific BIBDs, while FC-CBJ with both SDG and VM provide solution for all BIBDs tested. Regarding CPU time, SDG dominates DG in 45 classes, and VM dominates SDG in 46 classes, out of the 48 BIBD classes considered. These results show clearly that the inclusion of the symmetry-breaking heuristic is a very significative improvement for BIBD generation, allowing the solution of almost the whole benchmark, while the DG heuristic solved slightly more than one third of it. The VM heuristic means a refinement of SDG: it can solve one more instance, and CPU time decreases for most of the classes tested.

Adding the symmetric value pruning procedure[8] to FC-CBJ, we get the FC-CBJ-SVP algorithm, on which we have tested the heuristics SDG and VM. Empirical results appear in Table 6. FC-CBJ-SVP with SDG can solve 4 more instances than in the previous case, while FC-CBJ-SVP with VM increases in 3 the number of solved instances. In terms of CPU time, the dominance of VM over SDG remains in 42 cases. From this assessment, we conclude that symmetric value pruning does not play an important role in this problem: it produces certain benefits but the main advantadge is provided by the inclusion of symmetries in variable selection, either in the form of symmetry-breaking or in the more elaborated variety-maximization heuristic.

6 Conclusions

In this paper we have analysed how to take symmetry into account to reduce search effort. Two variable selection heuristics and a value pruning procedure have been devised to exploit symmetries inside a depth-first search scheme. We have shown how our symmetry-breaking heuristic can be combined with

[7] The degree of variable x_{ijp} is the number of future variables $x_{klp'}$ such that $i = k$ and $j = l$, or $i \neq k$ and $j \neq l$.

[8] Given that FC-CBJ with SDG or VM solved most of the problem instances, we included a SVP procedure allowing a single form of symmetric value pruning: the one due to symmetric nogoods at the current branch. Therefore, nogoods are not explicitly recorded in this case.

the minimum-domain one to yield a new variable selection heuristic that outperforms them both. This is called variety-maximization heuristic because it selects for assignment the variable leading to a search subspace with the greatest number of distinct final states. Moreover, our value pruning procedure based on nogood recording has proven effective in both solvable and unsolvable problem instances.

These strategies have been tested on two highly symmetric combinatorial problems, namely the Ramsey problem and the generation of BIBDs. For the former, we have compared our results with those obtained in previous works. In the case of solvable instances, i.e., for $n \leq 16$, our general-purpose strategies have been able to outperform the alternative approach of reformulating the original problem by adding new constraints to break problem symmetries. For $n = 17$, our strategies can still compete, although it must be noted that the variable selection heuristics are oriented towards finding solutions nor to prove their inexistence.

BIBD generation is an NP problem that has triggered a considerable amount of research on analytic and computational procedures. Its wide variability in size and difficulty makes it a very appropriate benchmark for algorithms aimed at exploiting symmetries in CSPs. We believe that systematic search procedures are more likely to shed light on the solution of difficult instances of the problem, although randomized algorithms may be quicker at finding solutions in easier cases. The present work has not been aimed at solving a particular such instance, but instead at proposing and evaluating tools to deal with symmetries. In this respect, the proposed strategies have been shown to be effective in reducing search effort.

It is worth mentioning that there is always a trade-off between the effort spent in looking for and exploiting symmetries, and the savings attained. Thus, instead of considering all possible symmetries, it is advisable to establish a hierarchy of them and try to detect the simplest first, as we have done.

Concerning future work, we would like to study whether other variable selection heuristics (such as degree-maximization) can also be integrated with symmetry-breaking and minimum-domain, under a single decision criterion. Along the same line, we would like to extend the variety-maximization heuristic to other variable relations, beyond strong permutability. Moreover, we will try to identify criteria for value selection which complement our heuristics for variable selection. Finally, it would be interesting to assess up to what extent our approach depends on the type and number of symmetries occurring in a particular problem.

Acknowledgements

This work has been partially supported by the Spanish Science and Technology Commission (CICYT) under contract TAP99-1086-C03 (project "Constraint-based computation in robotics and resource allocation"). We thank Thierry Petit, who provided us reference [13]. We also thank Javier Larrosa, Lluìs Ros and three anonymous reviewers for their useful criticisms.

References

[1] R. Backofen, and S. Will. Excluding symmetries in constraint based search. In *Proc. of CP'99*, pp. 73–87, 1999.

[2] A. Baker. The hazards of fancy backtracking. In *Proc. of AAAI'94*, pp. 288–293, 1994.

[3] D. Brelaz. New methods to color the vertices of a graph. *Journal of ACM*, **22**(4), 251–256, 1979.

[4] C.A. Brown, L. Finkelstein, and P.W. Purdom. Backtrack searching in the presence of symmetry. In *Proc. 6th Int. Conf. on Applied Algebra, Algebraic Algorithms and Error Correcting Codes*, pp. 99–110, 1988.

[5] C.H. Colbourn and J.H. Dinitz (Eds.). *The CRC Handbook of Combinatorial Designs*, CRC Press, 1996.

[6] D.G. Corneil and R.A. Mathon. Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations. *Ann. of Discrete Math.*, **2**, 1–32, 1978.

[7] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR-96*, USA, 1996.

[8] R. Dechter. Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition. *Artificial Intelligence*, **41**, 273–312, 1990.

[9] W. Feller. *An Introduction to Probability Theory and its Applications*, Volume I, 3rd edition. New York: Wiley, 1968.

[10] E.G. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. of AAAI'91*, pp. 227–233, 1991.

[11] I.P. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proc. of CP'96*, LNCS 1118, pp. 179–193, 1996.

[12] I.P. Gent and B. Smith. Symmetry breaking in constraint programming. In *Proc. of ECAI-2000*, pp. 599–603, 2000.

[13] ILOG. *Ilog Solver User's Manual 3.1*, 1996.

[14] M. Hall. *Combinatorial Theory*, Ed. John Wiley & Sons, Second Edition, 1986.

[15] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, **14**, 263–313, 1980.

[16] R. Mathon and A. Rosa. Tables of parameters of BIBD with $r \leq 41$ including existence, enumeration and resolvability results: an update. *Ars Combinatoria*, **30**, 1990.

[17] P. Meseguer and C. Torras. Solving strategies for highly-symmetric CSPs. In *Proc. of IJCAI'99*, pp. 400–405, 1999.

[18] B. Nadel. Representation selection for constraint satisfaction: a case study using n-queens. *IEEE Expert*, 16–23, June 1990.

[19] P. Prosser. Hybrid algorithmics for the constraint satisfaction problem. *Computational Intelligence*, **9**(3), 268–299, 1993.

[20] J.F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proc. of ISMIS'93*, pp. 350–361, Norway, 1993.

[21] P. Roy and F. Pachet. Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *Proc. of ECAI'88 Workshop on Non-binary Constraints*, pp. 27–33, Brighton, UK, 1998.

[22] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[23] A. San Miguel. Symmetries and the cardinality operator. In *Proc. of ECAI'92*, pp. 18–22, 1992.

[24] B.M. Smith and S.A. Grant. Trying harder to fail first. In *Proc. of ECAI'98*, pp. 249–253, 1998.

[25] W.D. Wallis. *Computational and Constructive Design Theory*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

	Fc-cbj								
BIBD	DG			SDG			VM		
(v, b, r, k, λ)	Sol	Nodes	Time	Sol	Nodes	Time	Sol	Nodes	Time
7,7,3,3,1	50	21	1.4e-3	50	22	1.4e-3	50	21	2.6e-3
6,10,5,3,2	50	60	3.6e-3	50	31	6.6e-3	50	30	4.6e-3
7,14,6,3,2	50	2152	1.3e-1	50	60	1.9e-2	50	43	1.1e-2
9,12,4,3,1	50	40	1.8e-3	50	80	2.0e-2	50	48	1.0e-2
6,20,10,3,4	18	435	3.7e+0	50	77	5.7e-2	50	61	3.3e-2
7,21,9,3,3	16	2877	4.3e+0	50	65	6.7e-2	50	75	4.5e-2
6,30,15,3,6	6	196	9.9e+0	50	117	2.4e-1	50	95	1.4e-1
7,28,12,3,4	11	195	7.6e+0	50	146	2.2e-1	50	86	1.2e-1
9,24,8,3,2	44	763	1.2e+0	50	75	1.2e-1	50	77	8.2e-2
6,40,20,3,8	3	156	1.7e+1	50	124	6.5e-1	50	128	3.9e-1
7,35,15,3,5	6	230	1.5e+1	50	111	4.3e-1	50	109	2.7e-1
7,42,18,3,6	6	141	1.9e+1	50	131	8.0e-1	50	139	4.8e-1
10,30,9,3,2	38	181	3.4e+0	50	100	2.7e-1	50	120	2.0e-1
6,50,25,3,10	1	1057	3.1e+1	50	467	1.8e+0	50	155	8.1e-1
9,36,12,3,3	29	478	6.8e+0	48	116	2.5e+0	50	202	3.8e-1
13,26,6,3,1	50	1076	3.5e-1	50	151	2.2e-1	50	151	1.7e-1
7,49,21,3,7	2	151	3.3e+1	50	651	2.0e+0	50	164	8.0e-1
6,60,30,3,12	2	139	4.6e+1	50	184	2.7e+0	50	189	1.5e+0
7,56,24,3,8	1	36401	4.6e+1	50	258	2.3e+0	50	179	1.2e+0
6,70,35,3,14	0	0	5.4e+1	50	216	4.9e+0	50	215	2.3e+0
9,48,16,3,4	19	685	1.6e+1	50	151	1.2e+0	50	153	7.3e-1
7,63,27,3,9	0	0	6.0e+1	50	240	3.4e+0	50	196	1.7e+0
8,56,21,3,6	5	285	3.7e+1	49	188	3.9e+0	50	498	1.7e+0
6,80,40,3,6	0	0	7.2e+1	50	243	8.6e+0	50	245	3.6e+0
7,70,30,3,10	1	235	6.7e+1	50	215	5.1e+0	50	215	2.4e+0
15,35,7,3,1	48	395	9.8e-1	50	219	5.3e-1	50	219	4.2e-1
12,44,11,3,2	41	591	5.1e+0	50	166	9.6e-1	50	191	6.5e-1
7,77,33,3,11	0	0	9.3e+1	50	243	7.7e+0	50	246	3.2e+0
9,60,20,3,5	12	386	2.9e+1	49	188	4.8e+0	50	256	1.7e+0
7,84,36,3,12	1	1027	9.2e+1	50	316	1.1e+1	50	254	4.2e+0
10,60,18,3,4	12	613	2.6e+1	50	244	2.8e+0	50	189	1.5e+0
11,55,15,3,3	33	680	1.2e+1	50	180	2.0e+0	50	234	1.2e+0
7,91,39,3,13	0	0	1.3e+2	50	274	1.5e+1	50	280	5.4e+0
9,72,24,3,6	8	671	4.2e+1	49	221	8.4e+0	50	252	2.7e+0
13,52,12,3,2	43	298	4.6e+0	50	583	2.4e+0	49	218	2.9e+0
9,84,28,3,7	8	2054	5.4e+1	50	662	1.5e+1	50	257	4.2e+0
9,96,32,3,8	9	3997	6.6e+1	50	558	2.0e+1	50	296	6.3e+0
10,90,27,3,6	8	3131	5.6e+1	50	279	1.4e+1	50	289	5.3e+0
9,108,36,3,9	3	1193	9.6e+1	50	335	3.0e+1	49	365	1.4e+1
13,78,18,3,3	37	1392	1.6e+1	50	274	7.7e+0	50	282	3.5e+0
15,70,14,3,2	36	1647	2.3e+1	50	615	6.1e+0	49	383	5.5e+0
12,88,22,3,4	33	1271	2.8e+1	50	292	1.3e+1	50	296	5.1e+0
9,120,40,3,10	6	10429	1.1e+2	50	386	4.8e+1	50	268	1.4e+1
19,57,9,3,1	46	778	4.8e+0	48	802	9.1e+0	48	802	8.2e+0
10,120,36,3,8	4	9927	1.1e+2	50	422	5.1e+1	50	377	1.3e+1
11,110,30,3,6	24	2491	4.9e+1	50	353	3.6e+1	49	366	1.6e+1
16,80,15,3,2	40	2275	2.3e+1	50	795	1.1e+1	50	485	4.7e+0
13,104,24,3,4	30	1076	4.9e+1	50	402	2.7e+1	50	344	8.7e+0

Table 5: Performance results of BIBD generation using Fc-cbj with three different variable selection heuristics, on a Sun Ultra 60, 360MHz.

	Fc-cbj-svp					
BIBD	sdg			vm		
(v, b, r, k, λ)	Sol	Nodes	Time	Sol	Nodes	Time
7,7,3,3,1	50	22	3.0e-3	50	21	4.2e-3
6,10,5,3,2	50	31	7.2e-3	50	30	5.6e-3
7,14,6,3,2	50	53	1.9e-2	50	44	1.2e-2
9,12,4,3,1	50	78	1.9e-2	50	48	1.0e-2
6,20,10,3,4	50	62	5.4e-2	50	62	3.5e-2
7,21,9,3,3	50	65	6.8e-2	50	69	4.4e-2
6,30,15,3,6	50	103	2.4e-1	50	95	1.4e-1
7,28,12,3,4	50	111	2.1e-1	50	86	1.2e-1
9,24,8,3,2	50	75	1.2e-1	50	77	8.4e-2
6,40,20,3,8	50	123	6.6e-1	50	126	3.9e-1
7,35,15,3,5	50	111	4.3e-1	50	109	2.7e-1
7,42,18,3,6	50	130	7.9e-1	50	133	4.8e-1
10,30,9,3,2	50	99	2.7e-1	50	123	2.0e-1
6,50,25,3,10	50	239	1.7e+0	50	156	8.3e-1
9,36,12,3,3	50	1896	1.0e+1	50	173	3.8e-1
13,26,6,3,1	50	145	2.1e-1	50	145	1.7e-1
7,49,21,3,7	50	321	1.8e+0	50	163	7.9e-1
6,60,30,3,12	50	184	2.7e+0	50	185	1.5e+0
7,56,24,3,8	50	219	2.2e+0	50	173	1.2e+0
6,70,35,3,14	50	213	4.9e+0	50	217	2.3e+0
9,48,16,3,4	50	152	1.2e+0	50	152	7.3e-1
7,63,27,3,9	50	220	3.3e+0	50	193	1.7e+0
8,56,21,3,6	49	179	1.3e+1	50	323	2.0e+0
6,80,40,3,6	50	242	8.5e+0	50	246	3.6e+0
7,70,30,3,10	50	213	5.0e+0	50	216	2.4e+0
15,35,7,3,1	50	193	5.0e-1	50	193	3.9e-1
12,44,11,3,2	50	166	9.5e-1	50	204	6.7e-1
7,77,33,3,11	50	242	7.6e+0	50	240	3.3e+0
9,60,20,3,5	49	188	1.3e+1	50	238	1.6e+0
7,84,36,3,12	50	270	1.1e+1	50	254	4.2e+0
10,60,18,3,4	50	232	2.8e+0	50	188	1.5e+0
11,55,15,3,3	50	180	2.0e+0	50	229	1.3e+0
7,91,39,3,13	50	274	1.5e+1	50	277	5.4e+0
9,72,24,3,6	50	979	1.4e+1	50	309	3.0e+0
13,52,12,3,2	50	541	2.5e+0	50	1008	3.4e+0
9,84,28,3,7	50	440	1.2e+1	50	257	4.2e+0
9,96,32,3,8	50	418	2.0e+1	50	295	6.4e+0
10,90,27,3,6	50	279	1.4e+1	50	286	5.3e+0
9,108,36,3,9	50	335	3.0e+1	49	341	4.0e+1
13,78,18,3,3	50	273	7.7e+0	50	280	3.5e+0
15,70,14,3,2	50	573	6.1e+0	50	1058	5.7e+0
12,88,22,3,4	50	290	1.3e+1	50	296	5.0e+0
9,120,40,3,10	50	381	4.8e+1	50	461	1.4e+1
19,57,9,3,1	49	745	7.7e+0	49	745	6.9e+0
10,120,36,3,8	50	417	5.1e+1	50	377	1.3e+1
11,110,30,3,6	50	352	3.5e+1	49	366	3.6e+1
16,80,15,3,2	50	643	1.0e+1	50	490	4.7e+0
13,104,24,3,4	50	397	2.8e+1	50	344	8.5e+0

Table 6: Performance results of BIBD generation using Fc-cbj-svp with two different variable selection heuristics, on a Sun Ultra 60, 360MHz.