Solving Strategies for Highly Symmetric CSPs

Pedro Meseguer
Institut d'Investigació en Intel·ligència Artificial
CSIC
Campus UAB, 08193 Bellaterra
Spain

Carme Torras
Institut de Robòtica i Informàtica Industrial
CSIC-UPC
Gran Capità 2-4, 08034 Barcelona
Spain

Abstract
Symmetry often appears in real-world constraint satisfaction problems, but strategies for exploiting it are only beginning to be developed. Here, a rationale for exploiting symmetry within depth-first search is proposed, leading to an heuristic for variable selection and a domain pruning procedure. These strategies are then applied to a highly symmetric combinatorial problem, namely the generation of balanced incomplete block designs. Experimental results show that these strategies achieve a reduction of up to two orders of magnitude in computational effort. Interestingly, two previously developed strategies are shown to be particular instances of this approach.

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. During search, if two or more states of a problem are related by a symmetry, it means that all of them represent the same state, so it is enough to visit only one of them. This causes a drastic decrease in the size of the search space, which has a very positive impact on the efficiency of the constraint solver.

In this paper, we propose two strategies for symmetry exploitation, which can speed-up significantly the solving process of CSPs with many symmetries. We have used these strategies to solve the problem of generating balanced incomplete block designs (BIBD from now on), a combinatorial problem of interest in statistics, coding theory and computer science. With them, we are able to solve the BIBD generation problem with a simple algorithm (FC-CBJ) for a wide set of designs.

This paper is organized as follows. In Section 2, we introduce some basic concepts. In Section 3, we explain two strategies for symmetry exploitation during search. In Section 4, we present the problem of BIBD generation. In Section 5, we formulate the BIBD generation as a CSP and give empirical results. In Section 6, we revise previous approaches to this topic. Section 7 contains conclusions and future work.

2 Basic Definitions
Constraint satisfaction. A finite CSP is defined by a triple \( \langle X, D, C \rangle \), where \( X = \{ x_1, \ldots, x_n \} \) is a set of \( n \) variables, \( D = \{ D(x_1), \ldots, D(x_n) \} \) is a collection of current domains where \( D(x_i) \) is the finite set of possible values for variable \( x_i \), and \( C \) is a set of constraints among variables. A constraint \( c_i \) on the ordered set of variables \( \text{var}(c_i) = (x_{i_1}, \ldots, x_{i_m}) \) specifies the relation \( \text{rel}(c_i) \) of the allowed combinations of values for the variables in \( \text{var}(c_i) \). An element of \( \text{rel}(c_i) \) is a tuple \( (v_{i_1}, \ldots, v_{i_m}) \), \( v_i \in D_i(x_i) \), where \( D(x_i) \) represents the initial domain of \( x_i \). An element of \( \text{rel}(c_i) \) is called a valid tuple on \( \text{var}(c_i) \).

A solution of the CSP is an assignment of values to variables which satisfies every constraint. A value \( a \) is good for a variable \( x_i \) if a solution includes the assignment \( (x_i, a) \). Typically, CSPs are solved by depth-first search algorithms with backtracking. At a point in search, \( P \) is the set of assigned or post variables, and \( F \) is the set of unassigned or future variables. The variable to be assigned next is called the current variable.

Symmetries. A symmetry on a CSP is a collection of bijective mappings \( \{ \theta, \theta_1, \ldots, \theta_n \} \).

\[ \theta : X \to X \]
\[ \theta_i : D(x_i) \to D(\theta_i(x_i)) \]
that preserve the set of constraints, i.e., \( \forall c_j \in C \) with \( var(c_j) = (x_{j_1}, \ldots, x_{j_{|c_j|}}) \) and \( rel(c_j) = (\{(x_{j_1}, \ldots, x_{j_{|c_j|}})\}) \), the transformed constraint \( c_j^\theta \) with \( var(c_j^\theta) = (\theta(x_{j_1}), \ldots, \theta(x_{j_{|c_j|}})) \) and \( rel(c_j^\theta) = (\{(\theta_1(x_{j_1}), \ldots, \theta_{|c_j|}(x_{j_{|c_j|}}))\}) \), is in \( C \).

An example of a symmetry on the 4-queens problem appears in Figure 1. Domains are:

\[ D_0(x_1) = \{1, 2, 3, 4\}, D_0(x_2) = \{5, 6, 7, 8\}, D_0(x_3) = \{9, 10, 11, 12\}, D_0(x_4) = \{13, 14, 15, 16\} \]

A central rotation of 180 degrees exchanges variables \( x_1 \) with \( x_4 \) and \( x_2 \) with \( x_3 \), and value domains are mapped as indicated. This transformation is a symmetry because all 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 \( c_{12}^\theta \) is computed as follows:

\[
\begin{align*}
\text{var}(c_{12}^\theta) &= (\theta(x_1), \theta(x_2)) = (x_4, x_3) = \text{var}(c_{34}), \\
\text{rel}(c_{12}^\theta) &= (\{(\theta_1, \theta_2(7)), (\theta_1, \theta_2(8)), (\theta_1, 2), (\theta_2(8)), (\theta_1(3), \theta_2(5)), (\theta_1(4), \theta_2(5)), (\theta_1(4), \theta_2(6))\}) = (\{16, 10\}, \{16, 9\}, \{15, 9\}, \{14, 12\}, \{13, 12\}, \{13, 11\}) = \text{rel}(c_{34}).
\end{align*}
\]

Thus, \( c_{12}^\theta = c_{34} \). Some symmetries leave subsets of variables unchanged. They are of special interest, as we will see in the next paragraph.

Symmetries and depth-first search. A search state \( s \) is characterized by an assignment of past variables, plus the current domains of future variables. It 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 neither past variables nor their assigned values. A symmetry local to the initial state is a global symmetry of the problem.

The notion of local symmetry is important because of the use of symmetries during search. If a state reports failure, all the states symmetric to it can be removed.

Since constraint satisfaction algorithms are based on depth-first search with backtracking, they may only remove states that are in the subtree below the current node, but never above it. Therefore, we are only interested in symmetries connecting states below the current node, that is, leaving the set of past variables unchanged. These symmetries are local to the current node. In the rest of the paper, we will consider local symmetries only.

3 Solving Strategies

In the following subsections, we describe two practical strategies for highly symmetrical CSPs, which can be embedded in any constraint satisfaction algorithm. Both are based on the detection of local symmetries at each search state. Automatic discovery of symmetries is a too complex task to be carried out at run time. Instead, we take a simpler approach. From an initial analysis of the considered problem, we identify a set of symmetries which may appear along the search. When a new state is generated, we check which of these previously identified symmetries are local to this state.

3.1 Breaking Symmetries While Searching

Let \( s \) be a search state where the symmetry \( \theta \) local to \( s \) involves a future variable \( x_i \) in the following form,

\[ x_i \in F, \theta(x_i) \neq x_i \quad \text{or} \quad \forall a \in D(x_i), \theta(a) \neq a \]

If \( x_i \) is assigned in the next step, symmetry \( \theta \) no longer holds in the current subproblem after \( x_i \) assignment. To see this, it is enough to realize that \( x_i \) is now a past variable (which cannot be changed) and \( \theta \) will change it. In this case, we say that the assignment of \( x_i \) breaks symmetry \( \theta \). If at state \( s \) several symmetries \( \theta, \phi, \psi \) . . . are local to \( s \), all involving variable \( x_i \), assigning \( x_i \) will break all these symmetries: no state in the current subproblem will be “repeated” by the action of these symmetries. This positive effect is only due to the assignment of \( x_i \), taken as the current variable. This is the rationale for our variable selection heuristic.

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

This greedy heuristic tries to break as many symmetries as possible in the next assignment. When it is applied consistently throughout the search tree, its positive effects accumulate. If \( x_1 \) is the variable selected at the first tree level, no matter which value is assigned to it, all symmetries involving \( x_1 \) are broken below level 1. If \( x_2 \) is the variable selected at the second tree level, no matter which value is assigned to it, all symmetries involving \( x_1 \) and \( x_2 \) are broken below level 2, and so on. This heuristic tries to maximize the total number of broken symmetries at each level of the search tree. It causes the following benefits.

\footnote{It could be argued that the assignment of \( x_2 \) may restore a symmetry \( \theta \) broken by the assignment of \( x_1 \), if \( \theta \) exchanges both variables and their values. But now \( \theta \) is no longer a local symmetry, given that it acts on past variables.}
exactly in $\lambda$ blocks. $v, b, r, k,$ and $\lambda$ are called the parameters of the design. Computationally, 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 $\lambda$. An example of BIBD appears in Figure 3.

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

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

However, these are not sufficient conditions. The situation is summarized in [Mathon and Rosa, 1990], that lists all parameter sets obeying these conditions, with $r \leq 41$ and $3 \leq k \leq v/2$ (cases with $k < 2$ 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 [Wallis, 1996, Chapter 11].

Some (infinite) families of block designs (designs whose parameters satisfy particular properties) can be constructed analytically, by direct or recursive methods [Hall, 1986, Chapter 15], and the state of the art in computational methods for design generation is described in [Colbourn and Dinitz, 1996; Wallis, 1996]. 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 [Cornell and Mathon, 1978].

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. 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 Experimental Results

The problem of generating a $(v, b, r, k, \lambda)$-BIBD can be formulated as a CSP as follows. Two rows $i$ and $j$ of the BIBD should have exactly $\lambda$ ones in the same columns. We represent this by $\lambda$ variables $x_{ijp}$, $1 \leq p \leq \lambda$, where $x_{ijp}$ contains the column of the $p$th 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 table $T$, a $v \times b$ binary matrix, is computed as follows,

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

Constraints are expressed in the following terms,

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

where $1 \leq p, p' \leq \lambda$, $1 \leq i, j \leq v$, $1 \leq c \leq b$. This problem presents many local symmetries. 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 variables $c_1$ 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 $x_{ijp'}$, domain mappings are the identity; this symmetry occurs when $T[i, c] = T[i', c] = T[j, c] = T[j', c]$ for $c = 1, \ldots, b$.
4. Variable mapping exchanges $x_{ijp}$ and $x_{ijp'}$, domain mappings corresponding to these variables exchange values $c_1$ and $c_2$; this symmetry occurs when,

$$T[i, c_1] = T[j, c_2], T[i, c_2] = T[j, c_1], T[i, c_1] = T[j, c_2] = 0$$

5. Variable mapping exchanges $x_{ijp}$ and $x_{ijp'}$, the domain mappings corresponding to these variables exchange values $c_1$ and $c_2$; this symmetry occurs when,

$$T[i, c_1] = T[j, c_2], T[i, c_2] = T[j, c_1], T[i, c_1] = T[j, c_2] = 0$$

These symmetries have a clear graphical interpretation. Symmetry (1) is inherent to the formulation. Symmetry (2) relates values of the same variable corresponding to equal columns. Symmetry (3) relates variables corresponding to equal rows. Symmetry (4) relates variables sharing row $i$, and rows $j_1$ and $j_2$, and that are equal 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) follows the same idea although it is more complex. It occurs when exchanging rows $i_1$ and $i_2$, rows $j_1$ and $j_2$, and columns $c_1$ and $c_2$, 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.

Symmetries are detected dynamically at each visited node. The specific implementation of the symmetry-breaking heuristic performs a weighted sum of the number of symmetries involving each future variable, where symmetries (4) and (5) are considered of less importance than the others.

**BIBD generation is a non-binary CSP.** We use a forward checking algorithm with conflict-directed backjumping (FC-cub) [Prosser, 1993] adapted to deal with non-binary constraints, with Brelaz heuristic [Brelaz, 1979] for variable selection and random value selection, as reference algorithm. This algorithm is modified to include the symmetry-breaking heuristic for variable selection, with Brelaz as tie-breaker, producing FC-CBJ-SB. Adding to this algorithm the strategy of value removal after failure, we obtain FC-CBJ-SB-VR. We compare the performance of these algorithms generating all BIBDs with $v < 1400$ and $k = 3$, all having solution. Since the performance of the proposed algorithms 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 1, where for each algorithm and BIBD, we give the number of solved problems within the node limit and the average CPU time in seconds for the 50 instances. Comparing FC-cub and FC-CBJ-SB-VR, we see that FC-cub solves 899 instances while FC-CBJ-SB-VR solves 2382 out of the 2400 instances executed. FC-CBJ-SB-VR does not solve any instance for 8 specific BIBDs, while FC-CBJ-SB-VR provides solution for all BIBDs tested. Regarding CPU time, FC-CBJ-SB-VR dominates FC-cub in 44 out of the 48 BIBDs considered, and this dominance is of one or two orders of magnitude in 39 cases. These results show clearly that the proposed strategies improve greatly the efficiency of the FC-CBJ algorithm for BIBD generation.

FC-CBJ-SB-VR results show that this algorithm almost achieves FC-CBJ-SB-VR performance. FC-CBJ-SB solves 2362 instances, 20 less than FC-CBJ-SB-VR, requiring slightly more time on the average. So, for BIBD generation, the symmetry-breaking heuristic is the main responsible for the savings in search effort, while value removal plays a very secondary role.

We also reimplemented FC-cub adding constraints $x_{ijp} < x_{ijp'}$ if $p < p'$, to break type (1) symmetries. The resulting algorithm, which included the extra pruning capacities caused by these new constraints, ran significantly slower than the original FC-cub in all BIBDs with $\lambda > 1$.

### 6 Related Work

Previous work on symmetries and CSPs can be classified in two general approaches. An approach, where our work fits in, consists in modifying the constraint solver to take advantage of symmetries. A modified backtracking algorithm appears in [Brown et al., 1988], testing each node to see whether it is the appropriate representative of those states symmetric to it. Considering specific symmetries, [Freuder, 1991] discusses the pruning of neighborhood interchangeable values of a variable. Another strategy [Roy and Pachet, 1998] considers value pruning between permutable variables. Interestingly, these two strategies are particular cases of the more general strategy presented in Section 3.2. It is easy to show that,

1. Let $x_i \in F, a, b \in D(x_i)$. Values $a, b$ are neighborhood interchangeable iff there exists a symmetry $\theta$ such that $\theta(x_i) = x_i, \theta(a) = b$.

2. Let $x_i, x_j \in F$. Variables $x_i, x_j$ are permutable iff there exists a symmetry $\theta$ such that $\theta(x_i) = x_j, \theta(a) = a, a \in D(x_i)$.

So, if $a$ is assigned to $x_i$ and fails, (1) all values neighborhood interchangeable with $a$ in $D(x_i)$, and (2) all values $a$ appearing in future domains of variables permutable with $x_i$, can be removed. Although developed
independently, our strategy of value removal after failure can be seen as a particular case of the symmetry exclusion method introduced by [Backofen and Will, 1998] for concurrent constraint programming, and applied to the CSP context by [Gent and Smith, 1999].

Another approach consists in modifying the symmetric problem to obtain a new problem without symmetries, but keeping the non-symmetric solutions of the original one. To do this, new constraints are added to the original problem in order to break the symmetries. Detecting symmetries and computing the new constraints is performed by hand in [Puget, 1993]. Alternatively, existing symmetries and the corresponding symmetry-breaking predicates (in the context of propositional logic) are computed automatically in [Crawford et al., 1996].

7 Conclusions

In this paper we have analysed how to take symmetry into account to reduce search effort. We have presented two strategies to exploit symmetries inside a depth-first search scheme. These strategies have been tested on a highly symmetric combinatorial problem, namely the generation of BIBDs, an NP problem which 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 procedures are more likely to shed light on the solution of difficult instances of the problem, whereas 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 plan to compare our strategies with the alternative approach of reformulating the original problem by adding new constraints to break problem symmetries. We also want to assess to what extent our approach depends on the type and number of symmetries occurring in a particular problem. We would like to identify criteria for value selection which complement our symmetry-breaking heuristic for variable selection. Moreover, the experimentation should be extended to other BIBD families, and the benefits obtained validated by applying these strategies to other domains.

Acknowledgements

We thank Javier Larrosa and the anonymous reviewers for their constructive criticisms.

References


