Solving Strategies for Highly Symmetric CSPs *

Pedro Meseguer
Inst. Invest. Intel.ligencia Artificial
CSIC
Campus UAB, 08193 Bellaterra
Spain

Carme Torras
Inst. Robotica i Informatica Industrial
CSIC-UPC
Gran Capita 2-4, 08034 Barcelona
Spain

Solving Strategies for
Pedro Meseguer
Inst. Invest. Intel.ligencia Artificial
CSIC
Campus UAB, 08193 Bellaterra
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.

*This research is supported by the Spanish CICYT proyect TIC96-0721-C02.

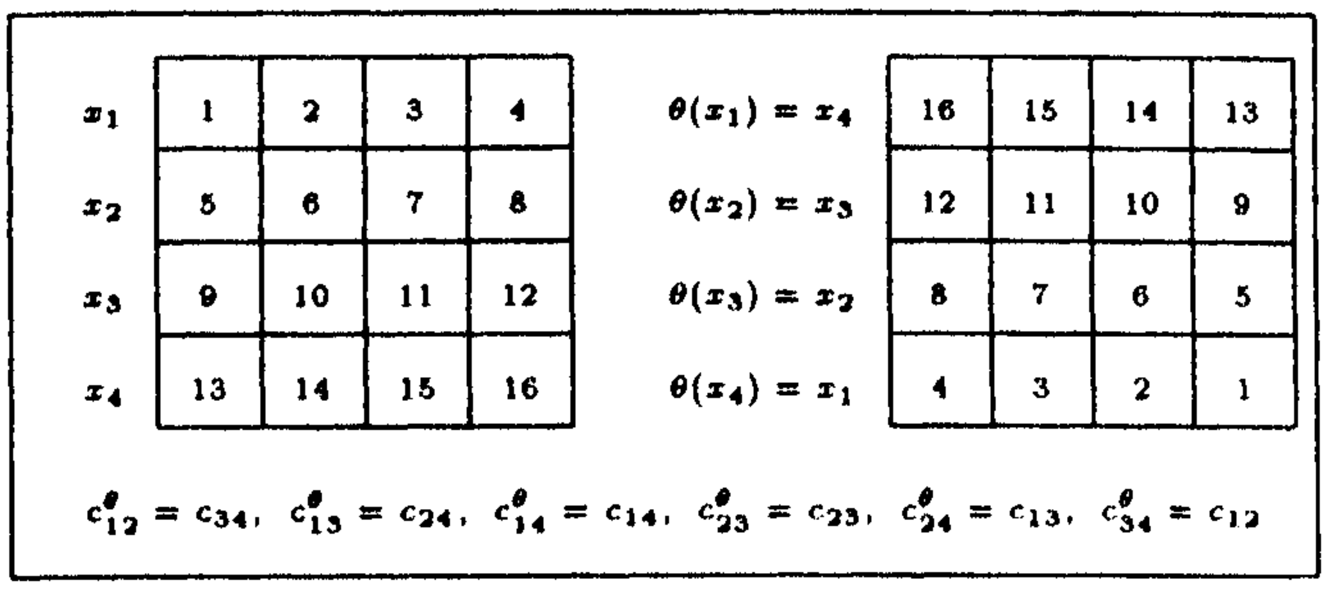
2 Basic Definitions

Constraint satisfaction. A finite CSP is defined by a triple (X',D,C), where ,V = $\{x_1,\ldots,x_n\}$ is a set of *n* variables, V = $\{D(x_1),\ldots,D(x_n)\}$ 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 $var(c_i) = (x_{i_1},\ldots,x_{i_{r(i)}})$ 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 $(v_{i_1},\ldots,v_{i_{r(i)}})$, $v_i \in D_0(x_i)$, where $D_0(x_i)$ represents the initial domain of 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 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 *past* 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 $\{\theta,\theta_1,\ldots,\theta_n\}$,

$$\theta : \mathcal{X} \to \mathcal{X}$$

$$\theta_i : D(x_i) \to D(\theta(x_i))$$



x_1 1 2 3 4
x_2 5 6 7 8
x_3 9 10 11 12
x_4 13 14 15 16
$\theta(x_1) = x_4$ 16 15 14 13
$\theta(x_2) = x_3$ 12 11 10 9
$\theta(x_3) = x_2$ 8 7 6 5
$\theta(x_4) = x_1$ 4 3 2 1
$c^\theta_{12} = c_{34},\ c^\theta_{13} = c_{24},\ c^\theta_{14} = c_{14},\ c^\theta_{23} = c_{23},\ c^\theta_{24} = c_{13},\ c^\theta_{34} = c_{12}$

Figure 1: Central rotation of 180 degrees is a symmetry for the 4-queens problem.

that preserve the set of constraints, i.e., $\forall c_j \in \mathcal{C}$ with $var(c_j) = (x_{j_1}, \ldots, x_{j_{r(j)}})$ and $rel(c_j) = \{(v_{j_1}, \ldots, v_{j_{r(j)}})\}$, the transformed constraint c_j^θ with $var(c_j^\theta) = (\theta(x_{j_1}), \ldots, \theta(x_{j_{r(j)}}))$ and $rel(c_j^\theta) = \{(\theta_{j_1}(v_{j_1}), \ldots, \theta_{j_{r(j)}}(v_{j_{r(j)}}))\}$, is in $\mathcal{C}$.[1]

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^θ_{12} is computed as follows,

$$var(c^\theta_{12}) = (\theta(x_1), \theta(x_2)) = (x_4, x_3) = var(c_{34}),$$

$$\begin{aligned} rel(c^\theta_{12}) &= \{(\theta_1(1), \theta_2(7)), (\theta_1(1), \theta_2(8)), (\theta_1(2), \theta_2(8)), \\ &\quad (\theta_1(3), \theta_2(5)), (\theta_1(4), \theta_2(5)), (\theta_1(4), \theta_2(6))\} \\ &= \{(16,10), (16,9), (15,9), (14,12), (13,12), (13,11)\} \\ &= rel(c_{34}). \end{aligned}$$

Thus, $c^\theta_{12} = c_{34}$. 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(ci)* of each constraint c, 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.

[1]Through an abuse of notation, we denote a symmetry $\{\theta, \theta_1, \ldots, \theta_n\}$ by its variable mapping θ.

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 *0* local to *s involves a future variable x_t in the following form,

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

If *X{* is assigned in the next step, symmetry θ 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 θ will change it. In this case, we say that the assignment of z, *breaks* symmetry θ. If at state *s* several symmetries $\theta, \phi, \psi, \ldots$ 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.

[2]It could be argued that the assignment of x_2 may restore a symmetry θ broken by the assignment of x_1, if θ exchanges both variables and their values. But now θ is no longer a local symmetry, given that it acts on past variables.

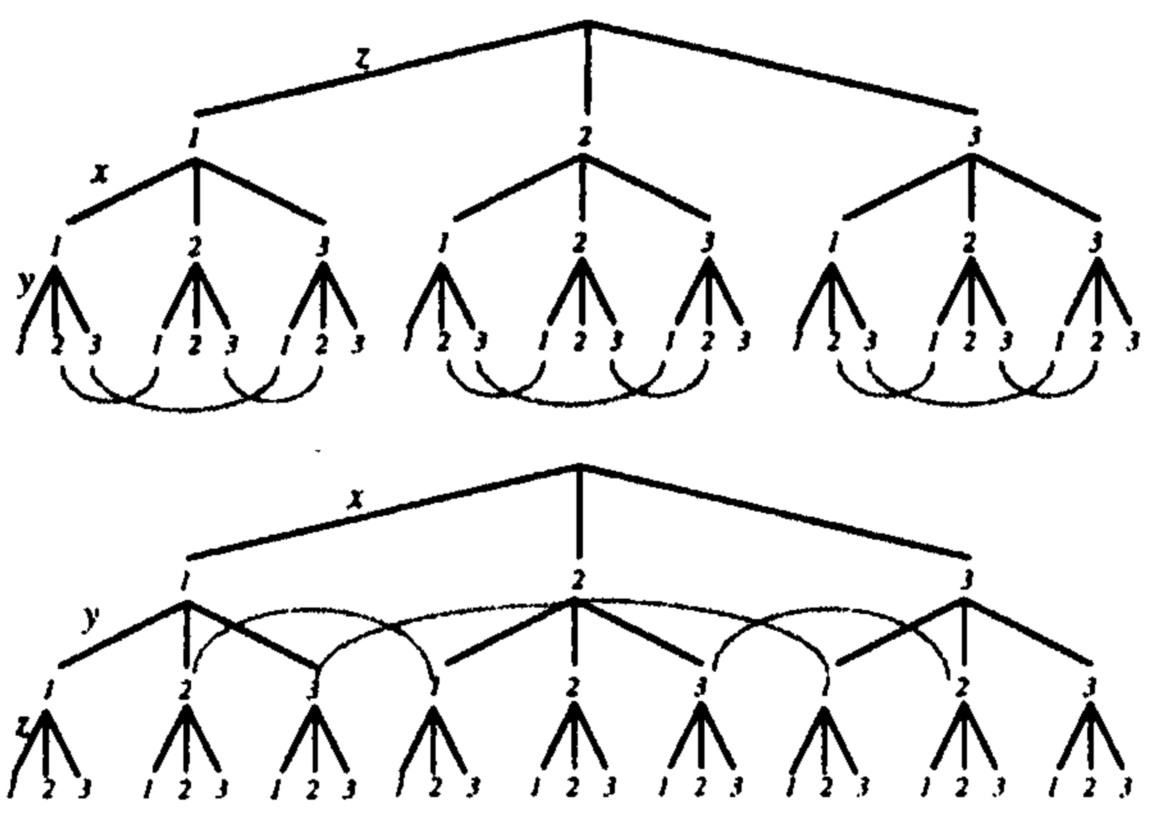


Figure 2: Search tree to solve the equation $xy - z = 4$ with different variable orderings. Symmetric states are connected by shadowed lines.

1. 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 could be substantial.

2. Better value selection. Let us suppose a problem with solution. At state s, there is a symmetry $\theta(x_j) = x_k$, with $x_j, x_k \in F$. Let $sol(x_j) = a$ and $sol(x_k) = b$ be the solution values for these two variables. Because of the symmetry, there is another solution with $sol(x_j) = b$ and $sol(x_k) = a$. Therefore, if x_j is selected as current variable, both values a, b are good to bring search to a solution. In a more general setting, this argument supports the fact that a variable involved in many symmetries will have many good values in its domain.

Some of these facts are illustrated in the following example. Let us consider the equation, $xy - z = 4$, where all variables take values in $\{1,2,3\}$. There is a symmetry, $\theta(x) = y$, and two solutions $(x = 2, y = 3, z = 2)$ and $(x = 3, y = 2, z = 2)$. Figure 2 displays two search trees for this equation, one following the variable ordering z, x, y and the other x, y, z. In the first tree, symmetry θ is not broken after assigning z, so symmetric states appear inside subtrees at the first level. In the second tree, symmetry θ is broken after assigning x, and symmetric states only appear between subtrees at the first level but not inside them. In addition, $D_0(x)$ has two good values (2 and 3), but $D_0(z)$ has only one (2)

3.2 Value Removal After Failure

Let s be a search state, where x_i is selected as the current variable and value a is tried without success. At this point, a backtracking-based algorithm will try another value for x_i. If θ is a symmetry local to s, we can remove $\theta_i(a)$ (the value symmetric to a) from $D(\theta(x_i))$, because it cannot belong to any solution including the current assignment of past variables. If all values of x_i are tried without success and the algorithm backtracks, all values removed in this way should be restored. If x_i, is involved in several symmetries, this argument holds for each of them separately.

This method 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.

An example of this value removal arises in the pigeonhole 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 domains,

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

(I is the identity mapping). If search is performed by forward checking and variables and values are assigned lexicographically, the first dead-end occurs when $(x_1, 1), (x_2, 2), \ldots, (x_{n-1}, n - 1)$, causing $D(x_n)$ to be empty. $D(x_{n-1})$ has no more values, so backtracking goes back to x_{n-2}. There, it finds that the only remaining value, $n - 1$, is symmetric to $n - 2$ which failed, so $n - 1$ can be pruned and no more values remain in $D(x_{n-2})$. Backtracking goes back to x_{n-3} where, by the same argument, the two remaining values are pruned. This process goes on up to reach x_1, where all its remaining values are pruned and search terminates with failure. Only the leftmost branch of the search tree is generated, and the rest of the tree is pruned.

4 BIBDs

Block designs are combinatorial objects satisfying a set of integer constraints [Hall, 1986; Colbourn and Dinitz, 1996). 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).

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

$$\begin{matrix} 0&1&1&0&0&1&0 \\ 1&0&1&0&1&0&0 \\ 0&0&1&1&0&0&1 \\ 1&1&0&0&0&0&1 \\ 0&0&0&0&1&1&1 \\ 1&0&0&1&0&1&0 \\ 0&1&0&1&1&0&0 \end{matrix}$$

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

exactly in λ blocks. t;,b,r, *k,* and λ are called the parameters of the design. Computationally, designs can be represented by a *v* x 6 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 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 $k \leq 2$ 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-xsomorphic* 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 II].

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 [Corneil 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,λ)-BIBD can be formulated as a CSP 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* — l)/2 row pairs, so there are $\lambda v(v-1)/2$ variables, all sharing the domain {1, . . ., *b}.* From these variables, the BIBD tableT," " binary matrix, 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 $1 \leq p,p' \leq \lambda$, $1 \leq i,j \leq v$, $1 \leq c \leq b$. 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 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 $x_{i_1j_1p}$ and $x_{i_2j_2p'}$, the domain mappings corresponding to these variables exchange values c_1 and c_2; this symmetry occurs when,
$T[i_1,c_1] = T[i_2,c_2] = 1, T[i_1,c_2] = T[i_2,c_1] = 0$
$T[i_1,c] = T[i_2,c], c = 1,\ldots,b, c \neq c_1, c \neq c_2,$
$T[j_1,c_1] = T[j_2,c_2] = 1, T[j_1,c_2] = T[j_2,c_1] = 0$
$T[j_1,c] = T[j_2,c], c = 1,\ldots,b, c \neq c_1, c \neq c_2,$
$T[j,c_1] = T[j,c_2], j = 1,\ldots,v, j \neq j_1, j \neq j_2.$

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 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

BIBD	Fc-CBJ		Fc-CBJ-SB		Fc-CBJ-SB-VR	
(v, b, r, k, λ)	Sol	Time	Sol	Time	Sol	Time
7,7,3,3,1	50	1.8e-3	50	3.2e-3	50	3.1e-3
6,10,5,3,2	50	6.8e-3	50	7.1e-3	50	6.9e-3
7,14,6,3,2	49	2.8e-1	50	2.1e-2	50	1.9e-2
9,12,4,3,1	50	5.7e-3	50	1.2e-2	50	1.2e-2
6,20,10,3,4	18	7.0e+0	50	1.5e-1	50	7.5e-2
7,21,9,3,3	19	6.8e+0	50	8.2e-2	50	8.1e-2
6,30,15,3,6	5	1.7e+1	50	5.4e-1	50	2.6e-1
7,28,12,3,4	20	1.0e+1	50	2.1e-1	50	2.0e-1
9,24,8,3,2	42	2.5e+0	50	1.4e-1	50	1.3e-1
6,40,20,3,8	1	3.1e+1	49	2.5e+0	49	1.5e+0
7,35,15,3,5	8	2.5e+1	49	1.3e+0	49	1.2e+0
7,42,18,3,6	5	3.5e+1	49	1.5e+0	50	1.1e+0
10,30,9,3,2	38	5.5e+0	50	3.3e-1	50	2.9e-1
6,50,25,3,10	2	6.1e+1	47	5.1e+0	47	4.6e+0
9,36,12,3,3	26	1.4e+1	50	5.4e-1	50	5.4e-1
13,26,6,3,1	49	5.2e-1	49	1.1e+0	50	3.0e-1
7,49,21,3,7	0	6.8e+1	50	2.2e+0	50	1.3e+0
6,60,30,3,12	0	9.6e+1	48	5.0e+0	50	3.1e+0
7,56,24,3,8	2	8.8e+1	48	5.1e+0	50	2.8e+0
6,70,35,3,14	0	1.2e+2	48	7.3e+0	50	4.4e+0
9,48,16,3,4	16	3.3e+1	50	1.4e+0	50	1.3e+0
7,63,27,3,9	1	1.2e+2	49	5.1e+0	49	4.3e+0
8,56,21,3,6	1	8.1e+1	48	5.3e+0	49	4.4e+0
6,80,40,3,6	0	1.7e+2	48	1.0e+1	48	1.0e+1
7,70,30,3,10	0	1.8e+2	47	9.6e+0	49	7.4e+0
15,35,7,3,1	50	1.0e+0	49	2.7e+0	49	2.6e+0
12,44,11,3,2	45	5.4e+0	50	1.6e+0	50	1.3e+0
7,77,33,3,11	0	2.1e+2	49	9.1e+0	50	6.1e+0
9,60,20,3,5	17	5.8e+1	50	3.2e+0	50	3.2e+0
7,84,36,3,12	0	2.2e+2	48	1.2e+1	49	9.9e+0
10,60,18,3,4	13	5.6e+1	50	3.5e+0	50	3.2e+0
11,55,15,3,3	35	1.9e+1	50	2.8e+0	50	3.2e+0
7,91,39,3,13	0	2.8e+2	48	1.8e+1	50	9.7e+0
9,72,24,3,6	9	8.8e+1	50	5.7e+0	50	4.9e+0
13,52,12,3,2	38	1.5e+1	50	3.5e+0	50	2.6e+0
9,84,28,3,7	6	1.2e+2	49	1.5e+1	50	1.0e+1
9,96,32,3,8	3	1.6e+2	50	1.3e+1	50	1.2e+1
10,90,27,3,6	7	1.3e+2	50	1.4e+1	50	1.4e+1
9,108,36,3,9	2	2 3e+2	49	3.0e+1	50	1.8e+1
13,78,18,3,3	36	3.3e+1	50	1.1e+1	50	8.6e+0
15,70,14,3,2	38	2.5e+1	50	9.1e+0	50	6.4e+0
12,88,22,3,4	35	5.9e+1	50	1.3e+1	50	1.2e+1
9,120,40,3,10	2	3.0e+2	49	3.0e+1	50	2.4e+1
19,57,9,3,1	47	1.0e+1	45	2.2e+1	46	2.1e+1
10,120,36,3,8	3	2.4e+2	50	2.9e+1	50	2.7e+1
11,110,30,3,6	17	1.4e+2	48	3.4e+1	49	3.0e+1
16,80,15,3,2	37	4.4e+1	49	2.2e+1	49	1.8e+1
13,104,24,3,4	27	8.9e+1	50	2.1e+1	50	1.9e+1

Table 1: Performance results of the proposed algorithms.

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-CBJ {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 *vb* < 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-CBJ and Fc-CBJ-SB-VR, we see that Fc-CBJ solves 899 instances while FC-CBJ-SB-VR solves 2382 out of the 2400 instances executed. Fc-CBJ 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-CBJ 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 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-CBJ 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, runned significantly slower than the original Fc-CBJ 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 advantadge 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 θ such that $\theta(x_i) = x_i, \theta_i(a) = b$.
2. Let $x_i, x_j \in F$. Variables x_i, x_j are permutable iff there exists a symmetry θ such that $\theta(x_i) = x_j, \theta_i(a) = a, a \in D(x_i)$.

o, if a is assigned to x_i and fails, (1) all values neighorhood interchangeable with a in $D(x_i)$, and (2) all alues a appearing in future domains of variables perıutable 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

[Backofen and Will, 1998] R. Backofen, and S. Will. Excluding symmetries in concurrent constraint programming. In *Workshop on Modeling and Computing with Concurrent Constraint Programming,* 1998.

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

[Brown *et al.,* 1988] 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,* 99 110, 1988.

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

[Corneil and Mathon, 1978] 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.

[Crawford *et al,* 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-Breaking Predicates for Search Problems. In *Proc. of KR-96,* USA, 1996.

[Freuder, 199l] E.G. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. of AAAI'91,* pages 227-233, 1991.

[Gent and Smith, 1999] i.P. Gent and B. Smith. Symmetry breaking during search in constraint programming. Research report 99.02, School of Computer Studies, University of Leeds.

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

[Mathon and Rosa, 1990] 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.

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

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

[Roy and Pachet, 1998] P. Roy and F. Pachet, Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *Proc. of ECAT88 workshop on Non-binary constraints,* pages 27-33, Brighton, UK, 1998.

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