Detecting and breaking symmetries on specifications

Marco Cadoli and Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli|tmancini@dis.uniroma1.it

Abstract. In this paper we address symmetry on combinatorial problems by following the approach of imposing additional symmetry-breaking constraints. Differently from other works in the literature, we attack the problem at the *specification* level. In fact, sometimes symmetries on specifications can be very easily detected, and symmetry-breaking formulae generated. We give formal definitions of symmetries and symmetry-breaking formulae on specifications written in existential second-order logic, clarifying the new definitions on two specifications: the graph 3-coloring and the social golfer problems. Finally, we show the results of a preliminary experimentation of our techiques on examples using state-of-the-art linear and constraint programming solvers.

1 Introduction

In order to speed up the solving process of combinatorial and constraint satisfaction problems, much work has been done on detecting and breaking symmetries, with the aim of greatly reducing the size of the search space. In the literature, three approaches have been followed to deal with symmetries:

1. Imposing additional constraints to the model of the problem, which are satisfied only for one of the symmetrical points in the search space, cf., e.g., [12–14];
2. Introducing additional constraints during the search process, to avoid the traversal of symmetrical points, cf., e.g., [8, 6];
3. Defining a search strategy able to break symmetries as soon as possible (e.g., by first selecting variables involved in the greatest number of local symmetries), cf., e.g., [10].

In this paper we follow the first approach, but, differently from other works in the literature, we aim to attack the problem at the *specification* level. Here are our motivations:

– Many systems and languages for the solution of constraint problems (e.g., AMPL [7], OPL [15], GAMS [2], DLV [4], SMODELS [11], and NP-SPEC [1]) clearly separate the *specification* of a problem, e.g., graph three-coloring, and its

instance, e.g., *a* graph, using a two-level architecture for finding solutions: the specification is *instantiated* against the instance, and then an appropriate solver is invoked.
- In some cases, symmetries on specifications can be very easily detected by humans (cf. Sec. 3), and convenient symmetry breaking formulas can be added to the specification itself. Since specifications are logical formulas, in principle, a theorem prover can be used for detecting and breaking symmetries in complex cases.
- Detecting and breaking symmetries at the specification level does not rule out the possibility to use symmetry-breaking techniques at the instance level. As an example, since some systems generate a SAT instance, e.g., [1], or an instance of integer linear programming, e.g., [15], it is possible to do symmetry breaking on such instances, using existing techniques, cf. e.g., [3].

In this work we show some preliminary results of our research. The outline of the paper is as follows: in Section 2, after recalling basic definitions of CSPs and symmetries, we lift to the specification level, defining the concepts of symmetry on a specification and of symmetry-breaking formula. In Section 3 we clarify the new definitions by showing two specifications: the graph 3-coloring and the social golfer problems. For both specifications we detect and break symmetries; notably, some of our symmetries have not been, to the best of our knowledge, proposed before. Finally, in Section 4 we show the results of a preliminary experimentation of our symmetry-breaking techiques on the first example of Section 3, using state-of-the-art linear and constraint programming solvers CPLEX and SOLVER, provided by OPL.

2 Basic definitions

In this section we briefly recall the standard definitions of Constraint Satisfaction Problem (CSP) and symmetry. Afterwards, we lift up to the definition of problem specification, which is independent on a particular instance, and give the definition of symmetry in the new context.

2.1 CSPs and their symmetries

Definition 1 (Constraint satisfaction problem (CSP)). *A CSP is a tuple* $\langle \boldsymbol{V}, \boldsymbol{D}, \boldsymbol{C} \rangle$*, where* $\boldsymbol{V} = \{X_1, \ldots, X_n\}$ *is a set of variables,* $\boldsymbol{D} = \{D_1, \ldots, D_n\}$ *is a set of finite domains, one for each variable,* $\boldsymbol{C} = \{C_1, \ldots, C_m\}$ *is a set of constraints on* $\boldsymbol{V}$ *and* $\boldsymbol{D}$*, i.e., a set of relations* $C_i(X_{i_1}, \ldots, X_{i_k}\} \subseteq D_{i_1} \times \cdots \times D_{i_k}$*, for every* $i \in [1..m]$*, containing those configurations of assignments allowed by the constraint.*

Given a CSP π, we can transform it in a new CSP π' by exchanging variables and/or domains order. A CSP transformation is defined in the following way:

Definition 2 (CSP transformation). *Given a CSP* $\pi = \langle \boldsymbol{V}, \boldsymbol{D}, \boldsymbol{C} \rangle$*, with* $\boldsymbol{V} = \{X_1, \ldots, X_n\}$ *and* $\boldsymbol{D} = \{D_1, \ldots, D_n\}$*, a transformation of* π *is a set of bijections* $\sigma_0, \sigma_1, \ldots, \sigma_n$ *such that:*

$$\begin{aligned} \sigma_0 \ &: \boldsymbol{V} \to \boldsymbol{V} \\ \textit{for every } i \in [1,n],\ \sigma_i \ &: D_i \to D_{index(\sigma_0(X_i))}, \end{aligned}$$

where index() is a function that returns the index of the argument variable. Is has to be observed that the transformation is defined on $\boldsymbol{V}$ *and* $\boldsymbol{D}$ *only (not on* $\boldsymbol{C}$*).*

Here are interesting specializations of Definition 2 (which apply also to forthcoming Definition 3):

Variable transformation: $\sigma_1, \ldots, \sigma_n$ are the identity function;
Value transformation: σ_0 is the identity function;
Uniform value transformation: σ_0 is the identity function, $D_1 = D_2 = \cdots = D_n$, and $\sigma_1 = \sigma_2 = \cdots = \sigma_n$.

Intuitively, a variable transformation exchanges only the order of variables, leaving the domains unchanged. On the other hand, a value transformation does not modify the variable order. In what follows, we focus mainly on uniform value transformations, in which domain values for each variable are swapped *uniformly*.

As widely described in the literature, symmetries of a CSP are transformations that map solutions into solutions:

Definition 3 (CSP symmetry [9]). *Given a CSP* $\pi = \langle \boldsymbol{V}, \boldsymbol{D}, \boldsymbol{C} \rangle$*, a symmetry on* π *is a CSP transformation on* $\langle \boldsymbol{V}, \boldsymbol{D} \rangle$ *which preserves constraints: an assigment* $\tau = \{X_1 = v_1, \ldots, X_n = v_n\}$ *to variables in* $\boldsymbol{V}$ *satisfies every constraint in* $\boldsymbol{C}$ *iff* $\sigma(\tau) = \{\sigma_0(X_1) = \sigma_{index(\sigma_0(X_1))}(v_1), \ldots, \sigma_0(X_n) = \sigma_{index(\sigma_0(X_1))}(v_n)\}$ *does.*

2.2 Specifications and their symmetries

Several languages for specifying problems exist. For the sake of simplicity, in this paper we focus on the most basic one, the existential fragment of second-order logic (ESO). ESO is able to specify all problems in the complexity class NP [5], where the instance is represented as a relational database. An ESO specification is a formula

$$\exists \boldsymbol{S}\ \phi(\boldsymbol{S}, \boldsymbol{R}), \tag{1}$$

where $\boldsymbol{R}$ is the relational schema for every input instance, and ϕ is a quantified first-order formula on the relational vocabulary $\boldsymbol{S} \cup \boldsymbol{R} \cup \{=\}$ ("=" is always interpreted as identity). Predicates in $\boldsymbol{S}$ are called *guessed*.

Example 1 (Graph 3-Coloring).

Input: A graph, according to the input relational schema $\boldsymbol{R} = \{edge(\cdot, \cdot)\}$.

Question: Is there a way to assign each node of the input graph one out of three colors such that every edge links nodes with different colors?

An ESO specification for the problem is as follows:

$$
\begin{aligned}
&\exists RGB \\
&\quad \forall X \;\; R(X) \vee G(X) \vee B(X) \;\; \wedge && (2)\\
&\quad \forall X \;\; R(X) \rightarrow \neg G(X) \;\; \wedge && (3)\\
&\quad \forall X \;\; R(X) \rightarrow \neg B(X) \;\; \wedge && (4)\\
&\quad \forall X \;\; B(X) \rightarrow \neg G(X) \;\; \wedge && (5)\\
&\forall XY \;\; X \neq Y \wedge R(X) \wedge R(Y) \rightarrow \neg edge(X,Y) \wedge && (6)\\
&\forall XY \;\; X \neq Y \wedge G(X) \wedge G(Y) \rightarrow \neg edge(X,Y) \wedge && (7)\\
&\forall XY \;\; X \neq Y \wedge B(X) \wedge B(Y) \rightarrow \neg edge(X,Y). && (8)
\end{aligned}
$$

Clauses (2–8) denote the first order part ϕ of the specification. Clause (2) imposes that every node is assigned at least one color; clauses (3–5) impose that every node is assigned at most one color; clauses (6–8) impose every edge to link nodes with different colors. □

It is worthwhile to note that, when a specification is instantiated against an input database, a CSP, in the sense of Definition 1, is obtained.

In what follows we focus initially on problem specifications which, like the one of Example 1, have only *monadic* guessed predicates. In this way we have a neat conceptual correspondence between the predicates and the *values* of a CSP. In Example 5 we show how our definitions can be used for non-monadic predicates, essentially by unfolding non-unary predicates and exploiting finiteness of the input database.

In this new context we can give the following definition:

Definition 4 (Uniform value transformation (UVT) of a specification). *Given a problem specification* $\psi \doteq \exists \boldsymbol{S} \;\; \phi(\boldsymbol{S}, \boldsymbol{R})$*, with* $\boldsymbol{S} = \{S_1, \ldots S_n\}$*,* S_i *monadic for every* i*, and input schema* $\boldsymbol{R}$*, a* uniform value transformation (UVT) *for* ψ *is a mapping* $\sigma : \boldsymbol{S} \rightarrow \boldsymbol{S}$*, which is total and onto, i.e., defines a permutation of guessed predicates in* $\boldsymbol{S}$*.*

The term "uniform value transformation" in Definition 4 is used because swapping monadic guessed predicates is conceptually the same as uniformly exchanging domain values in a CSP. We now define when a UVT is a symmetry for a given specification.

Definition 5 (Uniform value symmetry (UVS) of a specification). *Let* $\psi \doteq \exists \boldsymbol{S} \;\; \phi(\boldsymbol{S}, \boldsymbol{R})$*, be a specification, with* $\boldsymbol{S} = \{S_1, \ldots S_n\}$*,* S_i *monadic for every* $i \in [1, n]$*, and input schema* $\boldsymbol{R}$*, and let* σ *be a UVT for* ψ*. Tranformation* σ *is a* uniform value symmetry (UVS) *for* ψ *if the following holds:*

$$\forall \boldsymbol{S} \;\forall \mathcal{I} \;\; \phi(\boldsymbol{S}, \mathcal{I}) \;\leftrightarrow\; \phi^{\sigma}(\boldsymbol{S}, \mathcal{I}), \tag{9}$$

where $\phi^\sigma \doteq \phi[S_1/\sigma(S_1), \ldots, S_n/\sigma(S_n)]$*, i.e.,* ϕ^σ *is obtained from* ϕ *by uniformly substituing every occurrence of each guessed predicate with the one given by the transformation* σ*, and* $\mathcal{I}$ *is an instance on the input schema* $\boldsymbol{R}$*.*

Intuitively, formula (9) says that σ is a UVS for ψ if every extension for $\boldsymbol{S}$ which satisfies ϕ, satisfies also ϕ^σ and viceversa, for every input instance $\mathcal{I}$. Note that every CSP obtained by instantiating a specification with σ has at least the corresponding uniform value simmetry.

The following proposition (proof is omitted for lack of space) shows that checking whether a UVT is a UVS reduces to checking equivalence of two first-order formulae. As a consequence this task can in principle be performed by a first-order theorem prover. In the full paper we prove the converse reduction, i.e., that checking equivalence of two first-order formulae can be translated to checking whether a UVT is a UVS, thus proving that the problem is not decidable.

Proposition 1. *Let* ψ *be a problem specification of the kind* (1)*, with only monadic guessed predicates, and* σ *a UVT for* ψ*. Tranformation* σ *is a UVS for* ψ *if and only if* $\phi \equiv \phi^\sigma$*.*

Example 2 (Graph 3-Coloring: Example 1, continued). Three uniform value symmetries can be very easily obtained:

- $\sigma^{R,G}$ s.t. $\sigma^{R,G}(R) = G$, $\sigma^{R,G}(G) = R$, and $\sigma^{RG}(B) = B$;
- $\sigma^{R,B}$ s.t. $\sigma^{R,B}(R) = B$, $\sigma^{R,B}(G) = G$, and $\sigma^{RB}(B) = R$;
- $\sigma^{G,B}$ s.t. $\sigma^{G,B}(R) = R$, $\sigma^{G,B}(G) = B$ and $\sigma^{G,B}(B) = G$.

It is easy to observe that, for each of the above tranformations, formulae $\phi^{\sigma^{R,G}}$, $\phi^{\sigma^{R,B}}$, and $\phi^{\sigma^{G,B}}$ are all equivalent to ϕ, because clauses of the formers are syntactically equivalent to clauses of ϕ and viceversa. This implies, by Proposition 1, that they are all UVSs. □

Example 3 (Graph 3-Coloring with red self-loops). To play the devil's advocate, we consider a modification of the problem of Example 1, and show that only one of the UVTs in Example 2 is indeed a UVS. The problem is obtained by adding one more constraint.

Question: Is there a way to assign each node of the input graph one out of three colors such that: *(i)* every edge links nodes with different colors, and *(ii)* every self loop insists on a red node?

In ESO, one more clause (which forces the nodes with self loops to be colored in red) must be added.

$$\begin{aligned} &\exists RGB \\ &\quad \ldots \\ &\quad \forall X \;\; edge(X, X) \rightarrow R(X). \end{aligned} \tag{10}$$

- $\sigma^{G,B}$: it is a UVS, because the argument of previous example applies.

– $\sigma^{R,G}$: in this case, $\phi^{\sigma^{R,G}}$ is not equivalent to ϕ: as an example, for the input instance $edge = \{(v, v)\}$, the color assignment $\overline{R}, \overline{G}, \overline{B}$ such that $\overline{R} = v, G = B = \emptyset$ is a model for the original problem, i.e., $\overline{R}, \overline{G}, \overline{B} \models \phi(R, G, B, edge)$. It is however easy to observe that $\overline{R}, \overline{G}, \overline{B} \not\models \phi^{\sigma^{R,G}}(R, G, B, edge)$, because $\phi^{\sigma^{R,G}}$ is verified only by color assigments for which $\overline{G}(v)$ holds. This implies, by Proposition 1, that σ is not a UVS.
– $\sigma^{R,B}$: it is not a UVS, because the same argument of the previous point applies. □

Once symmetries of a specification have been detected, additional constraints can be added in order to *break* them, i.e., to wipe out from the solution space (some of) the symmetrical points. These kind of constraints are called *symmetry-breaking formulae*, and are defined as follows.

Definition 6 (Symmetry-breaking formula). *Let $\psi \doteq \exists \boldsymbol{S}\ \phi(\boldsymbol{S}, \boldsymbol{R})$, be a specification, with $\boldsymbol{S} = \{S_1, \ldots S_n\}$, S_i monadic for every $i \in [1, n]$, and input schema $\boldsymbol{R}$, and let σ be a UVS for ψ. A* symmetry-breaking formula *for ψ wrt symmetry σ is a closed (except for $\boldsymbol{S}$) formula $\beta(\boldsymbol{S})$ such that the following two conditions hold:*

1. *Transformation σ is no longer a symmetry for $\phi \wedge \beta$:*
$$(\phi \wedge \beta(\boldsymbol{S})) \not\equiv (\phi \wedge \beta(\boldsymbol{S}))^{\sigma}\ ;$$
2. *Every model of ϕ can be obtained by those of $\phi \wedge \beta(\boldsymbol{S})$ and their symmetric wrt σ:*
$$\phi\ \equiv\ (\phi \wedge \beta(\boldsymbol{S}))\ \vee\ (\phi \wedge \beta(\boldsymbol{S}))^{\sigma}\ .$$

If $\beta(\boldsymbol{S})$ matches the above definition, then we are entitled to solve the problem $\exists \boldsymbol{S}\ \phi(\boldsymbol{S}, \boldsymbol{R}) \wedge \beta(\boldsymbol{S})$ instead of the original one $\exists \boldsymbol{S}\ \phi(\boldsymbol{S}, \boldsymbol{R})$. In fact, point 1 in the above definition states that formula $\beta(\boldsymbol{S})$ actually breakes σ, since, by Proposition 1, σ is not a symmetry of the rewritten problem. Point 2 guarantees that all solutions are preserved in the rewritten problem, up to symmetric ones, which can be readily obtained by applying σ.

Intuitively, the "quality" of a symmetry-breaking formula is higher when $(\phi \wedge \beta(\boldsymbol{S}))$ and $(\phi \wedge \beta(\boldsymbol{S}))^{\sigma}$ have few common models, i.e., the less models in the conjunction $(\phi \wedge \beta(\boldsymbol{S}))\ \wedge\ (\phi \wedge \beta(\boldsymbol{S}))^{\sigma}$, the better. In next section we see several examples of UVSs and different breaking formulas for them.

The above definition deals with breaking a single symmetry, and it can be generalized with breaking *multiple* symmetries simultaneously. The definition, omitted for lack of space, deals with a set $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ of UVSs. The first condition must be repeated for all symmetries in Σ, and the the second one becomes $\phi\ \equiv\ (\phi \wedge \beta(\boldsymbol{S}))\ \vee\ (\phi \wedge \beta(\boldsymbol{S}))^{\sigma_1} \vee\ \cdots\ \vee (\phi \wedge \beta(\boldsymbol{S}))^{\sigma_n}$.

3 Examples of symmetry-breaking formulas

In this section we discuss two examples of well-known combinatorial problems which have many symmetries, show some of them by using Proposition 1, and present breaking formulae respecting conditions of Definition 6.

Example 4 (Graph 3-Coloring: Example 1, continued).
In the following we present different symmetry-breaking formulae for this problem, referring to $\sigma^{R,G}$, $\sigma^{R,B}$, and $\sigma^{G,B}$.

1. Let us consider $\sigma^{G,B}$: a symmetry-breaking formula is:

$$\beta^{G,B}(R,G,B) \doteq |G| \leq |B|, \tag{11}$$

   i.e., the number of green nodes is less than or equal to the number of blue ones (this constraint can be written in ESO using standard techniques). Formula (11) matches Definition 6. In fact we have that:
   - Symmetry $\sigma^{G,B}$ is broken, since:

$$\big(\phi \wedge \beta^{G,B}(R,G,B)\big) \not\equiv \big(\phi \wedge \beta^{G,B}(R,G,B)\big)^{\sigma^{G,B}};$$

   - All solutions –up to symmetric ones– are preserved, since:

$$\phi \equiv \big(\phi \wedge \beta^{G,B}(R,G,B)\big) \vee \big(\phi \wedge \beta^{G,B}(R,G,B)\big)^{\sigma^{G,B}}.$$

2. Let us now consider the two symmetries $\sigma^{R,G}$ and $\sigma^{R,B}$. A multiple-symmetry breaking formula for them is the following one:

$$\beta^{R}(R,G,B) \doteq |R| \leq |G| \wedge |R| \leq |B|, \tag{12}$$

   which uses a partial order among the sets of nodes with the same color, R being the minimal element. For lack of space, we don't formally prove here that β^R is a multiple-symmetry breaking formula.
3. The next formula breakes all three symmetries:

$$\beta(R,G,B) \doteq |R| \leq |G| \wedge |R| \leq |B| \wedge |G| \leq |B|, \tag{13}$$

   which uses a total order among colors, i.e., $|R| \leq |G| \leq |B|$. It is interesting to note that $\beta(R,G,B)$ can be obtained by composing the above formulae. Such a composition can be extended to the k-coloring problem for handling any number of colors. □

Example 5 (The social golfer problem (Problem 10 at `www.csplib.org`*)).*

Input: Positive integers ng, nw and ngr, denoting respectively the number of golfers, weeks, and groups.
Question: Is there a way to arrange, for w weeks, ng golfers in ngr groups of the same size, such that any two golfers play in the same group at most once?

A specification for this problem (supposing the ratio ng/ngr, i.e., the group size, integral) is the following one ($ASSIGN(G,W,GR)$ states that golfer G plays in group GR on week W):

$$
\begin{aligned}
&\exists ASSIGN \\
&\quad \forall G, W\ \exists GR\ \ ASSIGN(G, W, GR)\ \wedge && (14)\\
&\quad \forall G, W, GR, GR'\ \ ASSIGN(G, W, GR) \wedge ASSIGN(G, W, GR') \rightarrow && (15)\\
&\qquad\qquad GR = GR'\ \wedge \\
&\quad \forall G, G', W, W'\ \ (G \neq G' \wedge W \neq W') \rightarrow \\
&\qquad\qquad \big\{\forall GR, GR'\ \big(GR \neq GR' \wedge ASSIGN(G, W, GR) \wedge && (16)\\
&\qquad\qquad\quad ASSIGN(G', W, GR)\big) \rightarrow \\
&\qquad\qquad\quad \neg\left[ASSIGN(G, W', GR') \wedge ASSIGN(G', W', GR')\right]\big\}\ \wedge \\
&\quad \forall GR, GR', W, W'\ \ |\{G : ASSIGN(G, W, GR)\}| = && (17)\\
&\qquad\qquad |\{G : ASSIGN(G, W', GR')\}|.
\end{aligned}
$$

Constraints (14–15) force $ASSIGN$ to be a total function assigning a group to each golfer on each week; moreover, (16) is the *meet only once* constraint, while (17) forces groups to be of the same size.

In order to highlight UVSs according to Definition 5, we need to substitute the ternary guessed predicate $ASSIGN$ by means of $nw \times ngr$ monadic predicates $ASSIGN_{W,GR}$ (each one listing golfers playing in group GR on week W). The above specification must be unfolded accordingly. In this way, UVTs $\sigma_W^{GR,GR'}$, swapping $ASSIGN_{W,GR}$ and $ASSIGN_{W,GR'}$, i.e., given a week W, and two groups GR and GR', assign to group GR' on week W all golfers assigned to group GR on week W, and viceversa, are symmetries for the social golfer problem. Intuitively, UVTs $\sigma_W^{GR,GR'}$ are UVSs for the unfolded specification because group renamings have no effect.

After fixing arbitrarily linear orders '$<$' on golfers and groups, each symmetry $\sigma_W^{GR,GR'}$ (with $GR \neq GR'$) can be broken by, e.g., the following formula:

$$
\begin{aligned}
&\beta_W^{GR,GR'}(ASSIGN_{W,GR}, ASSIGN_{W,GR'})\ \doteq \\
&\qquad\qquad \forall G\ least(G, ASSIGN_{W,GR} \cup ASSIGN_{W,GR'}) \rightarrow && (18)\\
&\qquad\qquad\quad \rightarrow \begin{cases} ASSIGN_{W,GR}(G) \text{ if } GR < GR' \\ ASSIGN_{W,GR'}(G) \text{ otherwise,} \end{cases}
\end{aligned}
$$

which forces the least golfer in $ASSIGN_{W,GR} \cup ASSIGN_{W,GR'}$ to play in the least group between GR and GR'.

Similarly to Example 4, we can simultaneously break several of the above symmetries, by adding the following constraints, one for each week W and each pair GR, GR' of groups such that GR' is the successor of GR in the given linear order '$<$':

$$
\forall G\ least(G, ASSIGN_{W,GR}) < least(G, ASSIGN_{W,GR'}). \qquad (19)
$$

The formulae say that, for each week, the least golfer in group GR precedes the least golfer in group GR'. An implication of the above formulae is that the first golfer always plays in the first group.

As another example, the following family of transformations (which we call $\sigma^{W,W'}$) are also symmetries for the original social golfer specification: given two weeks W and W', swap uniformly all groups in week W with the ones in week W', and viceversa.

Each symmetry $\sigma^{W,W'}$ can be broken: in the following, we give only an intuitive description of a possible symmetry-breaking formula $\beta^{W,W'}$, once total orders '$<$' on weeks and on weekly assigments, i.e., on sets $\{ASSIGN_{W,GR_1}, \ldots, ASSIGN_{W,GR_{ngr}}\}$ have been fixed: given two assignments of golfers to groups $\{ASSIGN_{W,GR_1}, \ldots, ASSIGN_{W,GR_{ngr}}\}$ and $\{ASSIGN_{W',GR_1}, \ldots, ASSIGN_{W',GR_{ngr}}\}$, on weeks W and W', they are swapped iff the first is greater than the second one.

By breaking several of the aforementioned symmetries, we can add the following constraints, one for each pair W, W' of weeks such that W' is the successor of W in the given linear order '$<$':

$$\{ASSIGN_{W,GR_1}, \ldots ASSIGN_{W,GR_{ngr}}\} < \{ASSIGN_{W',GR_1}, \ldots ASSIGN_{W',GR_{ngr}}\}, \tag{20}$$

saying that weekly assigments are in increasing order wrt weeks.

It is worth noting that, by combining the symmetry-breaking formulae for the two families of symmetries described above, we are able to get some of the symmetry-breaking constraints described in [14], like the one which arbitrarily fixes the assigment in the first week. Moreover, we can further fix one golfer (the one having the least order number) to be assigned always to the same group (the one having the least order number). □

4 Experiments

In this section we present a preliminary experimentation of the previously described techniques for the graph coloring problem shown in Section 3. Actually, rather than 3-coloring, we focused on the k-coloring problem. We used state-of-the-art linear and constraint programming solvers, and wrote both a linear and a non linear specification, solved by using CPLEX and SOLVER, respectively.

The symmetry breaking formulae we added are the generalization of (12) to the case of k colors. In fact, using the generalization of (13) resulted in poorer performances: intuitively, solutions of coloring problems are rarely "color unbalanced", therefore adding too many constraints may not be effective. As shown in Table 1 for some of the graphs in the DIMACS benchmark repository (`ftp://dimacs.rutgers.edu/pub/challenge`), adding our symmetry-breaking formulae seems to be effective especially when CPLEX is used on negative instances. SOLVER seems to be negatively affected, although in some cases is outperformed by CPLEX. Both SOLVER and CPLEX have built-in features for symmetry cuts; all experiments have been done both enabling and disabling such a

Table 1. Solving times (seconds) for k-coloring

			CPLEX			SOLVER		
Instance	Colors	Solvable?	W/o s.b.	W s.b.	% saving	W/o s.b.	W s.b.	% saving
anna	11	Y	3.2	1.3	59.4	0.8	–	$-\infty$
DSJC125.9	21	N	1388.7	1206.2	13.1	–	–	–
DSJR500.1	12	Y	22.5	32.1	-42.7	0.7	0.6	14.3
fpsol2.i.2	21	N	139.8	102.2	26.9	–	–	–
le450_15a	13	N	8.6	4.2	51.2	–	–	–
le450_25a	21	N	85.4	23.3	72.7	–	–	–
queen9_9	10	Y	–	–	–	122.9	205.6	-67.3
queen12_12	15	Y	453.6	357.1	21.3	8.7	9.4	-8.0

('–' means that the solver did not terminate in one hour)

feature (Table 1 shows times with no symmetry cuts). In general, effectiveness of our technique is independent on using that feature; the built-in feature results anyway in further speed-up when used along with our formulas (thus confirming the intuition of Section 1).

Acknowledgements The authors are grateful to Igor Markov for interesting discussions on symmetries and for pointing out references.

References

1. Marco Cadoli and Andrea Schaerf. Compiling problem specifications into SAT. In *Proceedings of the European Symposium On Programming (ESOP 2001)*, volume 2028 of *LNCS*, pages 387–401. Springer, 2001.
2. Enrique Castillo, Antonio J. Conejo, Pablo Pedregal, and Natalia Alguacil Ricardo Garca. *Building and Solving Mathematical Programming Models in Engineering and Science*. Wiley, 2001.
3. J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR'96*, pages 148–159, 1996.
4. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system `dlv`: Progress report, Comparisons and Benchmarks. In *Proc. of KR'98*, pages 406–417, 1998.
5. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.
6. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. of CP 2001*, volume 2239 of *LNCS*, page 77 ff. Springer, 2001.
7. Robert Fourer, David M. Gay, and Brian W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming*. International Thomson Publishing, 1993.
8. I. P. Gent and B. Smith. Symmetry breaking during search in constraint programming. In *Proc. of ECAI 2000*, pages 599–603, 2000.
9. P. Meseguer and C. Torras. Solving strategies for highly symmetric CSPs. In *Proc. of IJCAI'99*, pages 400–405, 1999.
10. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Journal of Artificial Intelligence*, 129:133–163, 2001.

11. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
12. E. Rothberg. Using cuts to remove symmetry. In *Proc. of ISMP'00*, 2000.
13. H. D. Sherali and J. Cole Smith. Improving discrete model representation via symmetry considerations. In *Proc. of ISMP'00*, 2000.
14. B. Smith. Reducing symmetry in a combinatorial design problem. In *Proc. of CPAIOR'01*, pages 351–360, 2001.
15. Pascal Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.