Solving the Kirkman's Schoolgirl Problem in a Few Seconds

Nicolas Barnier[1] and Pascal Brisset[2]

[1] Centre d'Études de la Navigation Aérienne, Toulouse, France
[2] École Nationale de l'Aviation Civile, Toulouse, France
{barnier,brisset}@recherche.enac.fr

Abstract. The Social Golfer Problem has been extensively used in recent years by the constraint community as an example of highly symmetric problem. It is an excellent problem for benchmarking symmetry breaking mechanisms such as SBDS or SBDD and for demonstrating the importance of the choice of the right model for one problem. We address in this paper a specific instance of the Golfer Problem well known as the Kirkman's Schoolgirl Problem and list a collection of techniques and tricks to find efficiently all its unique solutions. In particular, we propose SBDD+, an generic improvement over SBDD which allows a deep pruning when a symmetry is detected during the search. Our implementation of the presented techniques allows us to improve previous published results by an order of magnitude for CPU time as well as number of backtracks, and to compute the seven unique solutions of the Kirkman's problem in a few seconds.

Keywords: Symmetry Breaking, Social Golfer Problem, Resolvable Steiner Systems.

1 Introduction

Highly symmetric problems are always challenging for Constraint Programming and breaking, removing, discarding symmetries among solutions has been the subject of much interest among researchers of the CP community in recent years. We focus in this paper on one particular symmetric problem: the Social Golfer Problem[7], also known as resolvable Steiner system in the combinatorial area[12]. Except for small instances, this problem is open and Constraint Programming gives balanced results: formulation is straightforward but far from being sufficiently efficient to solve all instances.

We are interested in breaking as much symmetry as possible, as well as combining and improving previously proposed techniques to find all solutions of one specific instance of the problem. The first important choice concerns the model. We naturally choose a set model [9] which automatically removes one kind of symmetry. The next step is to statically remove symmetries by adding constraints. Additional redundant constraints may be added to detect failures

as soon as possible. The crucial point is then to be able to find an isomorphism relating two solutions quickly; we propose a "lazy" approach which splits the detection into two phases, building and checking, that provides the required efficiency.

One of the key ideas of the paper is to exploit an isomorphism found between two solutions, or partial solutions, to prune as much as possible of the subsequent search tree. We show that this is possible if the structure of the search tree is intrinsically related to the symmetries; in this case an isomorphism which maps a solution to another one similarly maps also some ancestor nodes. Combined with SBDD [5], we call the technique SBDD+.

Our experiments show that results presented in previous papers[17] can be greatly improved both in number of failures and in CPU time. The problem of finding all solutions to the Kirkman's Problem, which might have been considered hard to solve one year ago using Constraint Programming (two hours of CPU mentioned in [5]) can be solved in few seconds using our approach.

The remainder of the paper is structured as follows: we first define the Social Golfer Problem and model it, giving numerous possible redundant constraints. In the next section, we present our algorithm used to check symmetry and explain extensively our deep pruning technique associated with symmetry finding. Section 4 displays results of our experiments, confirming the approach. We conclude by recalling that many challenges remain.

2 Model

The classical instance of the *Social Golfer Problem* is described in the following terms:

> 32 golfers want to play in 8 groups of 4 each week, in such way that any two golfers play in the same group at most once. How many weeks can they do this for?

The problem may be generalized to w weeks of g groups, each one containing s golfers. This instance will be denoted g-s-w in the sequel of the paper. We note $n = g \times s$ the total number of golfers. The most famous and historical instance is the 5-3-7 for which all 7 unique (non-symmetric) solutions were already computed by Kirkman in the early 1850's [11]. In the combinatorics area, solutions for $s = 3$ are known as *Kirkman triple systems* or *resolvable Steiner systems*. Such systems have been extensively investigated (see for example [2]).

In the context of constraint programming, different models have been proposed in [18] to the Golfer Problem. The integer set model[1] which automatically removes symmetries inside groups is the one we chose for our experiments. In this model, the variables are the groups themselves and constraints are expressed as operations on sets.

The variables $G_{i,j}$, with i index of weeks and j index of groups, are sets and their associated domain is a lattice of sets defined by its greatest lower bound

[1] `http://www.icparc.ic.ac.uk/eclipse/examples`

(the necessary elements) and its lowest upper bound (the possible elements) [9]. The $G_{i,j}$'s are subsets of the set of golfers. Each of them contains exactly s elements. All the groups of a week are disjoint and every pair of groups from different weeks share at most one element. All these properties are expressed with the following constraints:

$$1 \leq i \leq w,\ 1 \leq j \leq g \quad G_{i,j} \subset \{1, 2, ..., n\}$$
$$1 \leq i \leq w,\ 1 \leq j \leq g \quad |G_{i,j}| = s \tag{1}$$
$$1 \leq i \leq w,\ 1 \leq j < j' \leq g \quad G_{i,j} \cap G_{i',j} = \emptyset \tag{2}$$
$$1 \leq i < i' \leq w,\ 1 \leq j, j' \leq g \quad |G_{i,j} \cap G_{i',j'}| \leq 1 \tag{3}$$

The constraints (2) may be basically implemented as w `all_disjoint` global constraints instead of the $wg(g-1)/2$ binary disjoint constraints. Note that no global consistency is achieved for this global constraint. The number of constraints of type (3) grows quadratically with the number of groups. It may prevent to solve large instances with this model.

According to [18] experiments, the naive set model is not the best one. However, we choose it for several reasons. First it is the simplest one and it uses the highest abstraction level. Second, redundant constraints described in the following section are easy to express with this model. Third, this is the model used in [5, 17] and it allows us to compare our approach with this previous ones.

2.1 Redundant constraints

Several constraints can be added to the original model. While they may help to solve the hardest instances, the induced overhead is sometimes too large for small instances like Kirkman's one.

The fact that a player plays only once per week is not explicit in the original model, but only entailed by the constraints (1) and (2). The corresponding constraint is written using reified membership constraints:

$$1 \leq i \leq w, 1 \leq p \leq n \quad \sum_{1 \leq j \leq g} (p \in G_{i,j}) = 1 \tag{4}$$

Warwick Harvey (`www.icparc.ic.ac.uk/~wh/golf`) proposes to express the fact that the players of a group appear in exactly s groups in other weeks:

$$1 \leq i \neq i' \leq w,\ 1 \leq j \leq g \quad \sum_{1 \leq j' \leq g} (G_{i,j} \cap G_{i',j'} \neq \emptyset) = s \tag{5}$$

Taking into account the size of the groups, the global constraint `atmost1` proposed by [16] may also be set on the list of all groups.

$$\texttt{atmost1}(\{G_{i,j} / 1 \leq i \leq w,\ 1 \leq j \leq g\}, s) \tag{6}$$

where $\texttt{atmost1}(S, c)$ states that sets of S must have cardinal c and must intersect pairwise in atmost one element. The propagation associated with this constraint

basically ensures that the possible number of partners of a player p is large enough, i.e. greater or equal to $(c-1)N_p$ where N_p is the number of occurrences of p. In our case, N_p is statically known (equal to w) so the propagation rule of the constraint can be efficiently customized.

2.2 Breaking Symmetries Statically

Our first goal is to compute all the unique non-symmetric solutions to the problem. As described in previous papers[18, 5, 17], the Social Golfer Problem is highly symmetric:

- Players can be exchanged inside groups (ϕ_P);
- Groups can be exchanged inside weeks (ϕ_G);
- Weeks can be ordered arbitrarily (ϕ_W);
- Players can be renamed among $n!$ permutations (ϕ_X).

The symmetry inside groups is inherently removed by modelling groups as sets. The symmetry inside weeks may be handled by ordering the g groups. Because these groups are disjoint, a total order can be achieved by sorting the smallest element of the groups.

$$1 \leq i \leq w,\ 1 \leq j \leq g-1 \quad \min G_{i,j} < \min G_{i,j+1}$$

Note that this implies that the first player is in the first group for each week.

Following the same idea, weeks can be ordered with the first group as key, which can be easily done with the second smallest element:

$$1 \leq i \leq w-1 \quad \min(G_{i,1} \setminus \{1\}) < \min(G_{i+1,1} \setminus \{1\})$$

Symmetries among players are more difficult to handle and only dynamic checks will be able to remove them completely. Statically:

- First week is fixed;
- First group of second week is fixed with smallest possible players
- "Small" players are put in "small" groups: for every week, p^{th} player is in a smaller group than the p^{th} group

  $$1 \leq i \leq w, 1 \leq p \leq g \quad \bar{G}_{i,p} \leq p$$

  where $\bar{G}_{i,p}$ is the number of the group of player p in week i, i.e. the $\bar{G}_{i,p}$ are dual variables defined by

  $$1 \leq i \leq w, 1 \leq p \leq n \quad \bar{G}_{i,p} = j \text{ iff } p \in G_{i,j}$$

- Players together in a same group in the first week are placed in ordered groups in the second week

  $$1 \leq j \leq g,\ p_1, p_2 \in G_{1,j},\ p_1 < p_2 \quad \bar{G}_{2,p_1} < \bar{G}_{2,p_2}$$

– Groups of the first week are ordered in the second week:

$$1 \leq j < j' \leq g \quad \bar{G}_{2,G_{1,j}} <_{lexico} \bar{G}_{2,G_{1,j'}}$$

where $\bar{G}_{i,\{x_1,x_2,...\}}$ is the tuple $(\bar{G}_{i,x_1}, \bar{G}_{i,x_2}, ...)$ and $<_{lexico}$ stands for the lexicographic order on integer tuples.

Unfortunately, the conjunction of all these constraints does not remove all the symmetries among players. For example, for the 5-2-2 instance, the two following solutions are found:

1	2	3	4	5	6	7	8	9	10
1	3	2	4	5	7	6	9	8	10

1	2	3	4	5	6	7	8	9	10
1	3	2	5	4	6	7	9	8	10

Both solutions satisfy all aforementioned breaking symmetry constraints but the second one is isomorphic to the first one through the functions (ϕ_G is the same for the two rounds):

$$\phi_X = \{1 \to 7, 2 \to 8, 3 \to 9, 4 \to 10, 5 \to 1, 6 \to 2, 7 \to 3, 8 \to 4, 9 \to 5, 10 \to 6\}$$
$$\phi_G = \{1 \to 4, 2 \to 5, 3 \to 1, 4 \to 2, 5 \to 3\}$$
$$\phi_W = \{1 \to 1, 2 \to 2\}$$

We notice that even if permutations within weeks and groups may be statically removed by constraints when considered alone, it is still necessary to take them into account when the permutation within players is handled.

Remaining symmetries must be dynamically discarded; we discuss an efficient way to do it in the next section.

2.3 Integer Model with a Cardinality Constraint

We give here another model which allows us to solve efficiently the problem. Its originality comes from the use of a global cardinality constraint [15]. To the best of our knowledge, it is the first time it is proposed.

Decision variables in this model are $\bar{G}_{i,p}$, the number of the group of player p in week i.

$$1 \leq p \leq n \quad \bar{G}_{i,p} \in [1..g]$$
$$1 \leq i \leq w \quad \texttt{gcc}(\{\bar{G}_{i,p} / 1 \leq p \leq n\}, < (1,s), ..., (g,s) >)$$

where $\texttt{gcc}(S, < (v_1, c_1), ..., (v_k, c_k), ... >)$ constrains the number of occurrences of elements of S equal to v_k to be equal to c_k.

The "not with the same golfer more than once" is straighforward with reified constraints (whose number is increasing quadratically with the number of golfers):

$$1 \leq p_1 < p_2 \leq n \quad \sum_{1 \leq i \leq w} (\bar{G}_{i,p_1} = \bar{G}_{i,p_2}) \leq 1$$

This model allows for example a short program[2] to solve the classic 8-4-9 instance in half a second and 32 backtracks to find the fist solution. However, as explained earlier, it is not the one used for the experiments in this paper.

3 Handling Symmetries During Search

In this section we present our adaptation to the Social Golfer Problem of a generic symmetry breaking mechanism proposed in [5].

3.1 Generic Techniques for Breaking Symmetries

Symmetry breaking constraints fail to remove statically all symmetries among the players in the Social Golfer problem. Therefore, several solutions have been proposed to prune the search tree taking into account these symmetries dynamically.

Using SBDS [8], which needs to list explicitly the symmetries to remove, Barbara Smith in [18] was able to break most of the symmetries and obtain new results. Later, two generic and similar approaches were proposed in the same time [6, 5]. In the second one, the technique called SBDD (for *Symmetry Breaking via Dominance Detection*) was applied with success to the Social Golfer Problem and allows us to compute all the non-symmetric solutions of small instances.

In SBDD, states during the search (i.e. nodes of the search tree) are compared to previously explored ones modulo a symmetry mapping function. A (new) state P' is *dominated* by an (old) state P if P' is subsumed by $\phi(P)$ where ϕ is a symmetry mapping function. When searching only for non-symmetric solutions, a state which is dominated by an already explored ones is discarded. Then, it requires to store all explored nodes. However, it can be noticed that if P' is dominated by P then it is dominated by the father node of P. It means that when all the sons of a state have been explored, one can remove them from the store and keep only the father. Concretely, in case of depth first search, the store for SBDD can be handled as a stack where states are associated to their depth. When a new state must be added to the store, all the states on top of the stack which have a greater depth may be removed. We will see later that it is worthwhile to store states in a compiled form in order to ease future dominance checks against this state.

One issue of the technique is its efficiency because checking dominance may be very expensive: $w(s!)^g g!$ symmetries to check in the Social Golfer Problem. Some restrictions are necessary to get an effective procedure:

- storage of explored states may be limited;
- checking of dominance may be restricted to some depths.

We propose to specialize the SBDD technique for the Social Golfer Problem, first to be able to check for dominance quickly, second, to better exploit symmetries found.

[2] This solution for the Golfer Problem is an example provided in the constraint library we use.

3.2 Filtering Efficiently Symmetric Solutions

It is shown in experiments of [5] that it is not worth to check dominance for every node during search for golfer solutions. For the 4-4-4 instance, authors conclude that checks every 8-th depth give the best result. Results given in the next section show that lazy dominance checking is effective when solving small instances of the Golfer Problem.

General dominance check for the Golfer Problem as described in [5] requires to compute a matching in a bipartite-graph, for which the best algorithm is in $O(n^{5/2})$ [10]. However checking that there exists a symmetric mapping function which maps an old solution (a leaf in the search tree) to a new one is significantly easier.

Actually, it can be noticed that a solution to the Golfer Problem is fully specified by the week number of the pairs of players[3]. Precisely, a solution can be described by the following mapping:

$$(p_1, p_2) \rightarrow i \text{ such that } \bar{G}_{i,p_1} = \bar{G}_{i,p_2} \tag{7}$$

where (p_1, p_2) is a pair of golfers. Note however that the mapping is not total for instances where a player does not play with all others.

A check must be done for each possible symmetry. [5] remarks that the possible symmetries may be easily enumerated looking for a matching from the first week of the first solution (or partial solution) P to any week of the second solution P'. The key idea to compute symmetry checking efficiently is to compute it lazily: instead of choosing a complete matching and checking the other weeks afterward, it is worth checking them *while* the matching is built.

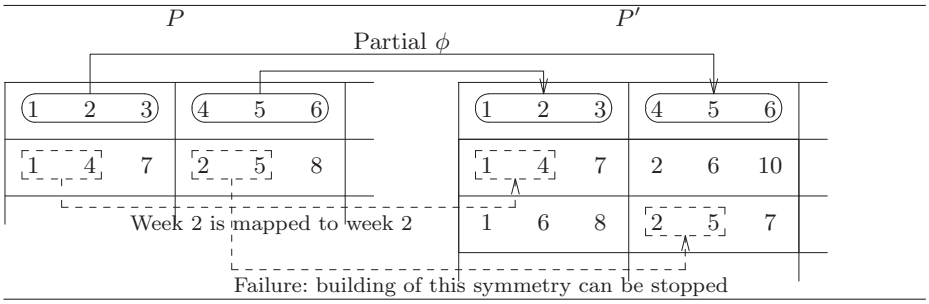

Fig. 1. Partial symmetry between solutions. Solid arrows show isomorphism building while dashed lines show isomorphism checking.

Figure 1 illustrates this principle: Suppose a partial isomorphism is built by mapping an exchange of the first two groups of the first week ($1 \leftrightarrow 4$, $2 \leftrightarrow 5$, $3 \leftrightarrow 6$). There is enough information in this partial matching to check all pairs

[3] Is is the integer model proposed by Barbara Smith in [18].

among other weeks with index of players less than 6. In this example, one can check that pairs $(1,4)$ and $(2,5)$ of the second week would be mapped in the same week of the second solution. Because it is not the case (pair $(1,4)$ is mapped to $(4,1)$ which appears in second week, $(2,5)$ is mapped to $(5,2)$ which does not appear at all), it is not necessary to try to complete this partial matching and we can "backtrack" to consider another mapping for the first two groups of the first week.

The check for pair mapping may be easily performed at a low cost by a precomputation phase:

- Derive the set of pairs of each week i of the first solution P, sorted according to the greatest element of the pair, denoted $C_i(P)$. Note that this precomputation may be done only once when the solution is found and stored.
- Build a table of the week numbers indexed by the pairs of the second solution P' (mapping of equation (7), noted $W_P(c)$).

The complete procedure is detailed in figure 2. The worst case complexity is the same as a naïve approach but experiments show that this checking algorithm, while only applied on leaves of the search tree, is efficient enough to compute all solutions of the Kirkman's problem in a reasonable time (c.f. section 4).

```
Procedure CheckSymmetry(P, P')
  Compute C_i(P) for i ∈ 2..w                          Usually already computed with P
  Compute W_P'(c) for all pairs c of P'
  for i' ∈ 1..w                                        Map first week of P to week i' of P'
    φ_W[1] ← i
    for φ_G ∈ P_g                                      Permute groups within the week
      for j ∈ 1..g
        for φ_P ∈ P_s                                  Permute players within the group
          Set φ_X such that P'_{i',φ_G(j)} = φ_X(φ_P(P_{1,j}))
          try
            for i ∈ 2..w                               For all other weeks of P
              for c ∈ C_i(P) s.t. c ≤ j s               For all mapable pairs
                if c is the first encountered pair of C_i(P)
                  φ_W[i] ← W_P'(φ_X(c))                Store image week of week i
                else
                  if W_P'(φ_X(c)) <> φ_W[i]            Check if pairs of week i in P are
                    continue                           mapped to the same week in P'
          return φ_X                                   Symmetry found is returned
  return NoSymmetry
```

Fig. 2. Search for symmetry ϕ_X mapping a solution P to P'

3.3 Pruning Deeply

An efficient symmetry checking procedure applied on leaves allows us to compute all the unique solutions but does not improve the search itself: no subtrees are removed. However, following an idea of [13] used in an algorithm to compute graph isomorphisms[4], symmetry checking on leaves may be used to prune large subtrees.

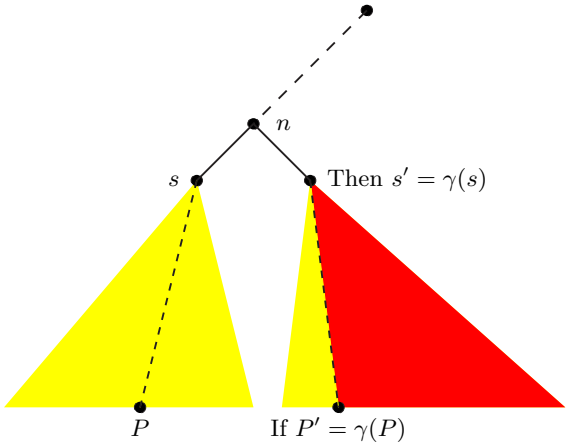

Fig. 3. Illustration of McKay's idea: The dark subtree can be pruned.

The idea is illustrated in figure 3. Let P be a first solution (the tree is explored in depth first search, from left to right) and P' a second solution proved to be isomorphic to P ($P' = \gamma(P)$). We note $n = P \setminus P'$ the lowest node in the search tree which is common to paths from the root to P and P', and s (resp. s') its immediate successor (we suppose that we do only binary branching) leading to P (resp. its immediate successor leading to P'). Under some conditions (which we call "McKay condition" in the sequel), it can be shown that node s' is the image of s by the isomorphism γ (more precisely the canonical extension of γ over partial solutions). In this case, the remaining unexplored subtree of s' is itself the image of an already explored subtree starting from s. Then it can be pruned because it would lead only to solutions that are images by γ of already found solutions.

In order to be able to apply this idea to a search tree, the structure of the tree (i.e. the labelling procedure) must be *compatible* with the symmetric mapping function we consider. For the Golfer Problem, all symmetries but the

[4] The `nauty` software based on McKay ideas is used by combinatorics people to find resolvable Steiner sytems[2].

one on players permutation may be removed simply with constraints. So the symmetries discovered at the leaves concern only ϕ_X. Following these remarks, the right choice is to label golfer by golfer to be able to apply ϕ_X extension on a node of the search tree. Note that a full choice for one golfer p amounts to labelling the w $\bar{G}_{i,p}$ variables.

Unfortunately, this labelling does not ensure the McKay condition if the set of golfers above node n (c.f. figure 4) is not stable through the found isomorphism γ: suppose node n concerns golfer 3 and γ is such that $\gamma(1) = 4$; we clearly cannot have $s' = \gamma(s)$ in this case. The problem in this example is that the choice on golfer 1 for P is mapped to the choice on golfer 4 in P', the latter being *under* the node n. Hence, node n cannot be pruned.

The necessary adaptation of McKay idea for our search tree is to consider two nodes s, s' which are descendants of n, leading respectively to P and P', such that the set of choices above s is mapped by γ to the set of choices above s'. In this case s' can be pruned. Of course, it is better to choose the highest such node in the search tree to prune as much as possible. Figure 4 illustrates the idea; the smallest set of golfers stable for γ which contains 3 (node n) is $\{1, 2, 3, 4, 5, 6\}$.

Results given in next section show that this deep pruning is highly effective.

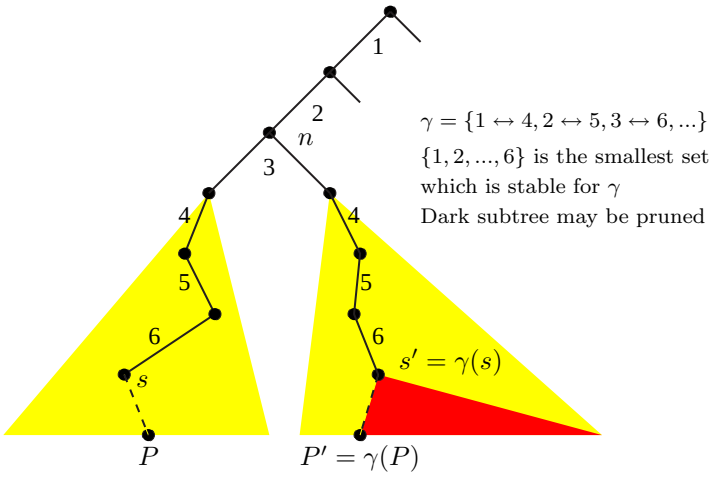

Fig. 4. Deep Pruning for the Golfer Problem.

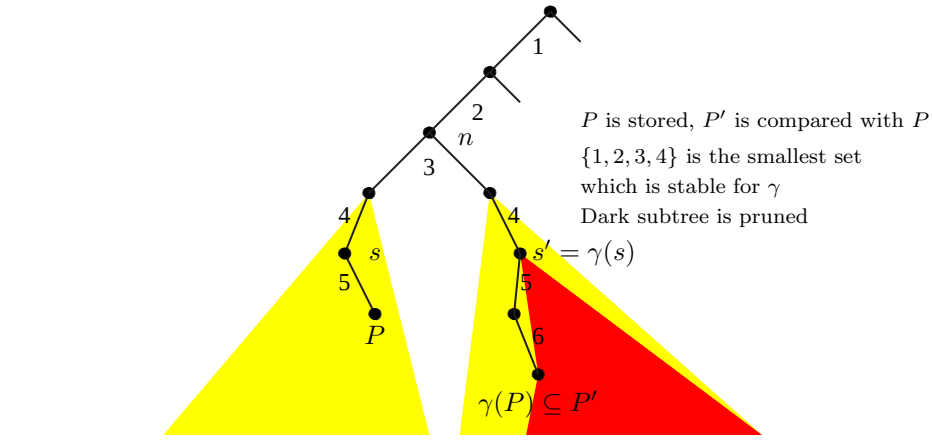

Fig. 5. SBDD+: Deep pruning integrated to SBDD.

3.4 SBDD+

We propose in this section an integration of the idea of deep pruning presented in the last section with the symmetry breaking mechanism SBDD. This mechanism computes isomorphism not only on solutions (leaves of the search tree) but on states of the search tree which can be described with the current domains of the variables. In SBDD+, we will exploit such isomorphism and try to show that it is applicable on ancestors, leading then to a best pruning.

Based on SBDD, the approach is generic: no hypothesis are required on the nature of the variables (integer or set variables) or the nature of the isomorphism. However, we present here the application of the method only to the Golfer Problem and we did not try to implement it in a generic way.

Fine symmetry checking on leaves improved with deep pruning allows us to solve our problem in a reasonable time. However, better pruning may be obtained if the method is integrated with SBDD approach. It requires first to be able to apply a refinement of the algorithm given in figure 2 to incomplete solutions (nodes of the search tree), second to call the procedure only at the appropriate times.

Experiments show that most of the symmetries found between two complete solutions involve a mapping from the first week to itself[5]. Moreover, incomplete solutions always get a complete first week. This means that the previous symmetry checking algorithm may be easily modified to be applied to incomplete solution if only this kind of symmetry is considered; building of the symmetry can be kept. However, the checking phase of the algorithm remains the same: a

[5] For 7 non-symmetric solutions, 927 solutions are found and 603 of them have this property.

symmetry is found if all pairs of P get an image in P'. But it does not mean that the symmetry is necessarily found when it exists, i.e. we possibly do not find all symmetries.

Dominance checking remains expensive and it must not be done too often. The check frequency must of course be related to the structure of the problem. A good compromise for the Golfer Problem is to

- Store nodes at every depth of the search tree;
- Check dominance for nodes only against stored nodes of smaller depth;
- Check dominance only for nodes at depth multiple of s.

The maximum size of the node store may be estimated: the depth of the search tree is the number of golfer gs; at each level, there may be g^w (for one golfer, g choices for each week) nodes to store. So the size of the store is bound by gsg^w (12 890 625 for Kirkman's Problem). This bound is a bad upper bound due to numerous symmetries removed by constraints (first week fixed, small golfers fixed, ...). For the Kirkman's Problem, with the detailed previous choices, only 15 nodes are effectively stored in the same time.

SBDD is compatible with the deep pruning mechanism: when a dominance is found, it is usually possible to prune more than the dominated node, just looking for the highest ancestor of the node for which the McKay condition is verified. We call the method SBDD+. It is illustrated in figure 5.

4 Results

We give in this section results of our experiments with the different techniques described in previous sections. The implementation has been done using FaCiLe [1], a functional constraint library written in Objective Caml (`caml.org`). This library provides usual finite domain variables over integers and sets, arithmetic constraints, set constraints[9], global constraints and an extensive mechanism to build search goals. It also includes basic mechanisms to maintain *invariant* expressions. This library, under LGPL license, and its documentation are available at `www.recherche.enac.fr/opti/facile`. CPU times are given for a Pentium 700 MHz running Linux 2.4.5.

The set model has been used with the labelling described earlier, golfer by golfer, choosing a group for each week, in increasing order. Additionally, some simple symmetries are dynamically removed during the search: when a golfer is placed in a group which does not contain any other golfers, this choice is fixed[6]. The refinement of the redundant `atmost1` (c.f. equation (6)) constraint is set in every experiment.

Labeling is slightly improved according to the following remark: the unique solutions of 5-3-7 are extensions of unique solutions of 5-3-2. Let S and S' be isomorphic solutions of 5-3-2 such that $S' = \gamma(S)$. If S is extended into a solution

[6] A similar idea is used in graph coloring algorithm: if a "new" unused color is tried for a node, there is no need to consider later other new colors for this node.

P of 5-3-7, then it is clear that $\gamma(P)$ is an isomorphic solution of 5-3-7 and also is an extension of 5-3-2. Then, our labeling first computes all unique solutions of 5-3-2 and extend them to 7 weeks. There are only 2 unique solutions for 5-3-2 and it takes 0.1 seconds to compute them.

Table 1 shows the number of created choice points, the number of backtracks, the total number of found solutions, the number of dominance checks and the CPU time for different combinations of methods and tricks presented in this paper to compute all 7 unique solutions to the schoolgirl problem (then 11 found solutions means that isomorphism has been detected only at the leaf of search the tree for 4 of them). First column (Leaves) gives the results for the straightforward search with simple discarding of symmetric solutions at leaves. This time can be compared with the one annouced in [5] (two hours). Next column (McKay) corresponds to the symmetry detection at leaves with deep pruning. We see that the number of failures and the time are smaller by an order of magnitude from the previous ones. Column SBDD uses our incomplete dominance checking; SBDD+ adds deep pruning. It is the best time achieved with our experiments. It can be compared with the result of [17] (400s on a similar CPU).

However, the number of backtracks can still be reduced with redundant constraints: in column "+(4)", the redundant constraint (4) stating that a golfer plays only once per week allows us to further reduce the search but the overhead is too large (with our implementation) and CPU time is not improved. The last column adds redundant constraint (5) which expresses that players of a group are spread among s different groups in other weeks. The overhead is here dissuasive but the number of backtracks is 5 times smaller than what was done in [17].

Table 1. Computing the 7 solutions of the Kirkman's Problem

	Leaves	McKay	SBDD	SBDD+	+(4)	+(5)
Choice points	20062206	1845543	107567	29954	18705	18470
Fails	19491448	1803492	104134	28777	16370	16169
Solutions	20640	934	11	11	11	11
Dominance checks			5373	456	456	443
CPU(s)	5925	484	24	7.8	9.4	36

Our combination of tricks to solve the Golfer Problem allowed us to solve open (at least for constraint programming) instances (6-4-6, 7-3-9, 8-3-7, 7-4-6, 6-5-7, ...). Some of these instances, at the time of writing this paper, are no longer open [14] and last updates of Warwick Harvey's web page include all these results (`http://www.icparc.ic.ac.uk/~wh/golf/`).

5 Conclusion

We have presented in this paper a combination of techniques which allows us to find efficiently all solutions to the Golfer Problem. The main contribution of the paper is an improvement of the SBDD principle that we call SBDD+. The key idea of the improvement is, while breaking symmetries, to exploit the symmetry function to be able to prune higher in the search tree. Extensive experiments show that this new mechanism can reduce by an order of magnitude the CPU time as well as the number of backtracks on the considered problem.

The deep pruning technique has been applied only to the Golfer Problem but is general. The only restriction is to have a relative compatibility between the structure of the search tree and the considered symmetry mappings in order to be able to prove that a symmetry found between two nodes is also true for ancestors of these two nodes. We believe that the notion of stability through the isomorphism of ancestor nodes, a necessary and sufficient condition for our problem, should be a general property. Further work is needed in this direction.

For the Golfer Problem itself, some instances remain open to constraint programming approaches (even if they are well known by combinatorics, for example 7-3-10 and 7-4-9 are extensively studied). Our model may be improved using the incomplete propagation techniques proposed by [17] in order to attack these instances. Note that SBDD+ must be refined and tuned for larger instances to avoid an explosion of node store and an extreme time overhead due to node dominance checking.

In spite of many efforts from the constraint community, the 8-4-10 instance is still open. This challenge is fascinating and it can be considered with the highest priority to show that constraint technology is really suited for this kind of combinatorial problem.

References

1. Nicolas Barnier and Pascal Brisset. Facile: a functional constraint library. In *Proceeding of CICLOPS2001*, Paphos, 2001.
2. M.B. Cohen, C.J. Colbourn, L.A. Ives, and A.C.H. Ling. Kirkman triple systems of order 21 with nontrivial automorphism group. *Mathematics of Computation*, 2001.
3. *CP'01: 7th International Conference on Principle and Practice of Constraint Programming*, Paphos, Cyprus, 2001.
4. *CPAIOR'02: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, Le Croisic, France, 2002.
5. Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In CP'01 [3], pages 93–107.
6. Filippo Focacci and Michaela Milano. Global cut framework for removing symmetries. In CP'01 [3], pages 77–92.
7. Ian Gent, Toby Walsh, and Bart Selman. CSPlib: a problem library for constraints. `csplib.org`.

8. I.P. Gent and Barbara Smith. Symmetry breaking during search in contraint programming. In W. Horn, editor, *EACI'2000*, pages 599–603, 2000.
9. Carmen Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997. `www.icparc.ic.ac.uk/~cg6`.
10. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal of Computing*, 2(4):225–231, 1973.
11. T.P. Kirkman. Note on an unanswered prize question. *Cambridge and Dublin Mathematics Journal*, 5:255–262, 1850.
12. JR Marshall Hall. *Combinatorial Theory*. Wiley Classics Library, second edition edition, 1983.
13. Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.
14. Steven Prestwich. Randomised backtracking for linear pseudo-boolean constraint problems. In CPAIOR'02 [4], pages 7–19.
15. Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.
16. Andre Sadler and Carmen Gervet. Global reasoning on sets. In *Formul'01, Workshop Modelling and Problem Formulation*, 2001.
17. Meinolf Sellmann and Warwick Harvey. Heuristic constraint propagation. In CPAIOR'02 [4], pages 191–204.
18. Barbara Smith. Reducing symmetry in a combinatorial design problem. In *CPAIOR'01*, pages 351–359, April 2001. `www.icparc.ic.ac.uk/cpAIOR01`.