An Attempt to Dynamically Break Symmetries in the Social Golfers Problem

Francisco Azevedo

CENTRIA, Universidade Nova de Lisboa, Portugal
fa@di.fct.unl.pt

Abstract. A number of different satisfaction and optimisation combinatorial problems have recently been approached with constraint programming over the domain of finite sets, for increased declarativity and efficiency. Such problems where one tries to find sets of values that satisfy some conditions, often present much symmetry on variables and values. In particular, the social golfers problem encompasses many possible symmetries. Allowing symmetric solutions increases search space unnecessarily, thus multiplying solution time. Therefore, ordering constraints have been proposed and incorporated in set solvers. However, such constraints are imposed statically in the global problem model and are unable to detect symmetries that still occur in sub-problems after a partial labelling. In this paper we discuss how to overcome this and present an approach that sequentially labels variables avoiding such symmetries by dynamically disallowing the assignment of other values from the same equivalence class in the golfers problem. Experimental results show that this approach outperforms previous ones, recently achieved by the constraint programming community, namely over sets. Unfortunately, the current method is incomplete and may loose solutions. Nevertheless, results are correct and show that similar techniques can be used efficiently to obtain faster solutions.

1 Introduction

A number of different satisfaction and optimisation combinatorial problems have recently been approached with constraint logic programming (CLP) over the domain of finite sets (CLP(*Sets*)). In fact, such problems are naturally expressed with set constraints, and recent advances on constraint propagation over this particular domain allowed an efficient solving where other techniques, such as integer linear programming, were inadequate or could not obtain good results.

This kind of problems, where one tries to find sets of values that satisfy some conditions, often presents much possible symmetry on variables and values, which can be swapped and still maintain the solution valid. Allowing symmetric solutions increases search space unnecessarily, thus multiplying solution time, possibly by orders of magnitude. The mere use of set variables automatically allows the removal of some symmetry as opposed to other finite domains. Nevertheless, much symmetry is still possible and, therefore, ordering constraints have been proposed and incorporated in set solvers. However, such constraints are imposed statically in the

F. Azevedo et al. (Eds.): CSCLP 2006, LNAI 4651, pp. 33–47, 2007.

global problem model and are unable to detect important symmetries that still occur in sub-problems after a partial labelling. To overcome this source of inefficiency much research has been recently devoted to (static and dynamic) symmetry breaking in general and, in particular, to the social golfers problem [5,7,9,10-12,14,22,24,25,27].

In this paper we present and discuss an example approach that treats the golfers problem globally and sequentially labels variables avoiding such symmetries by disallowing the assignment of considered equivalent values, which are calculated dynamically, in a similar way to GE-trees [26].

We start by briefly discussing set reasoning approaches and evolution of CLP(*Sets*) solvers in the next section. Then, in section 3, we show two example benchmark applications and present results, discussing the applicability of such solvers and how symmetry is handled. In section 4 we show how equivalence classes of values can be applied to break symmetry in the golfers problem, and present compared results that show the effectiveness of this technique. Finally, we conclude in the last section.

2 Set Reasoning

Set constraint solving has been proposed in [23] and formalised in [13] with ECLiPSe (http://eclipse.crosscoreop.com/eclipse/) library *Conjunto*, specifying set domains by intervals whose lower and upper bounds are known sets ordered by set inclusion. Such bounds are denoted as *glb* (for greatest lower bound) and *lub* (least upper bound). The *glb* of a set variable *S* can be seen as the set of elements that are known to belong to set *S*, while its *lub* is the set of all elements that can belong to *S*. Local consistency techniques are then applied using interval reasoning to handle set constraints (e.g. equality, disjointness, containment, together with set operations such as union and intersection). *Conjunto* proved its usefulness in declarativeness and efficiency for NP-complete combinatorial search problems dealing with sets, compared to constraint solving over finite integer domains.

Afterwards, another ECLiPSe set constraints solver library, *Cardinal* [2-4], improved on *Conjunto* by extending propagation on set functions such as cardinality.

Recently, a set solver based on reduced ordered binary decision diagrams (ROBDD) was proposed [15,19], that may compactly represent a set domain considering only the effective set instances that it may assume (instead of a set interval, that can include an exponentially larger than desired number of possible values). ROBDD operations then allow building efficient domain propagators eliminating intermediate variables, being able to reason more globally. This led to significant improvements over other CP approaches on a comparison that their authors performed.

In the next sections we present models and results using such solvers. As usual, these solvers are not complete, which means that generally a search phase must still occur after posting all constraints, in order to find a solution or prove its impossibility. This search process of labelling set domain variables is usually performed differently from labelling integer variables, where each possible value is directly assigned. With set domains, it is possible to achieve less commitment, by trying to successively include or exclude set elements, instead of assigning a definite set at once, which may lead to more failures. So, when instantiating set variable *S* (with domain $[glb_S,lub_S]$),

each value of $lub_S \backslash glb_S$ will be tried for inclusion or exclusion, until S is ground. This can be done with 2 different strategies: 1) trying inclusion first (which we call "element in set" strategy as in [19]); or 2) trying exclusion first ("element not in set").

Often, variables to label are chosen in increasing order of their domain sizes (*first-fail* heuristic). With sets, this procedure may recalculate the next variable to refine, after each element inclusion or exclusion from the current set variable (and respective propagation), according to the new domains.

3 Example Applications

In this section we discuss modelling and solving of CSPlib (www.csplib.org) problems 44 (Steiner triples) and 10 (social golfers) with set reasoning. These typical applications for set constraint solving are the motivating examples to the introduction of the equivalence classes approach to remove symmetries, described in section 4, where we apply it to the social golfers problem.

3.1 Steiner Triples

The ternary Steiner problem of order n consists of finding a set of $n.(n\text{-}1)/6$ triples of distinct integer elements in $U = \{1,\ldots,n\}$ such that any two triples have at most one common element. It is a hyper-graph problem coming from combinatorial mathematics [21] where n modulo 6 has to be equal to 1 or 3 [17,20]. One possible solution for n=7 is {{1, 2, 3}, {1, 4, 5}, {1, 6, 7}, {2, 4, 6}, {2, 5, 7}, {3, 4, 7}, {3, 5, 6}}. The solution contains 7*(7-1)/6 = 7 triples. Steiner Triple Systems (STS) are a special case of Balanced Incomplete Block Designs (BIBD). In fact, a STS with n elements is a BIBD(n, $n.(n\text{-}1)/6$, $(n\text{-}1)/2$, 3, 1).

A CP(*Sets*) approach directly models this problem by representing each triple as a set variable with upper bound U and cardinality 3, and constraining the cardinality of each intersection of a pair of triples to be not greater than 1. Furthermore, since set elements are automatically ordered, much symmetry is naturally eliminated. Nevertheless, symmetry still occurs, since any 2 sets (triples) in a solution can be swapped without affecting it. For that reason, usually CP(*Sets*) models add ordering constraints on the sets variables.

Since these solvers are not complete, a search phase must still be used to find a completely instantiated solution, and the way to do this may be critical on the computation time. Good heuristics for the order in which variables are picked for assignment and for the order in which values are assigned are then essential. For that, one should study the mathematical properties of the problem to find the best strategy. For example, each element must belong to exactly $(n\text{-}1)/2$ triples [8]. With this information, a labelling strategy of deciding, for each element in turn, which are the $(n\text{-}1)/2$ triples containing it may drastically reduce computation times. Different simple CP approaches (e.g. [6,13,15,19]) often do not go beyond n=15, but may go up until n=31, in times under 10 minutes, even using such heuristics and global constraints. In addition, best results are obtained with different labelling strategies, and some smaller instances may remain unsolved. All this shows how labelling strategies are crucial in this sort of problems, according to the propagation used. This

is so much so, that, in fact, a backtrack-free labelling for this problem is possible, and the solution is available on the internet (perso.wanadoo.fr/colin.barker/lpa/sts.htm and [1])! A labelling method is described based on the mathematical properties of the problem and the fact that Steiner triple systems can be constructed from a Latin square. One can then easily solve instances for n in the order of 1000 and more. We included this example in this paper not only to warn about this fact, which seems to be mostly unknown to the CP community, but also to show the importance of labelling based on the mathematical study of the problem to be solved.

The labelling phase is especially crucial for satisfaction problems where one tries to find a single solution, which is the case that is already solved. Constraints may still be useful in optimisation where one has to explore the whole search space. For that, the *ROBDD* approach of [19] finds all (30) solutions for n=7 in 0.1 seconds, and all (840) solutions for n=9 in 22.5 seconds. But, in fact, all solutions for each of these 2 instances are basically the same: a solution can be obtained from another simply by renumbering some values. It has been shown that (up to isomorphism) there is a unique solution for these 2 instances [8], which shows the importance of breaking symmetries.

3.2 Social Golfers

As taken from CSPlib: "The coordinator of a local golf club has come to you with the following problem. In her club, there are 32 social golfers, each of whom play golf once a week, and always in groups of 4. She would like you to come up with a schedule of play for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion."

The problem generalizes to that of scheduling g groups of s golfers over w weeks, such that no golfer plays in the same group as any other golfer twice (i.e. maximum socialization is achieved). Thus, the golfers problem, given values g-s-w, can be formally defined as finding a set, *Weeks*, of sets, for the golfers in *Golfers*, such that:

$$\#Weeks = w \wedge \#Golfers = g \quad s \wedge \forall_{wk \in Weeks} \bigcup_{grp \in wk} grp = Golfers\,. \tag{1}$$

$$\forall_{wk \in Weeks}(\#wk = g \wedge \forall_{g1,\, g2 \in wk,\, g1 \neq g2}(\#g_1 = s \wedge g_1 \cap g_2 = \varnothing))\,. \tag{2}$$

$$\forall_{w1,\, w2 \in Weeks,\, w1 \approx w2} \forall_{g1 \in w1} \forall_{g2 \in w2} \#(g_1 \cap g_2) \leq 1\,. \tag{3}$$

Actually, the disjointness condition ($g1 \cap g2 = \varnothing$) in (**2**) is redundant in face of the others.

The optimisation problem corresponds to maximising w, given g and s.

A CP(*Sets*) approach can model the g-s-w satisfaction problem with $w*g$ set variables of cardinality s, and if a solution is found try to increment w iteratively, until there are no more solutions.

Again, much symmetry occurs in this problem since any 2 week variables can be swapped, as well as any 2 groups in the same week, and even 2 values in the whole solution (e.g replacing value 1 with 2, and vice-versa, using integers to represent golfers).

In this section we compare 4 CP approaches to this golfers problem: *Cardinal, ic_sets* and *ROBDD*, together with a more elaborate symmetry breaking approach [18] using ILOG integer solver (www.ilog.com/products/solver), which we refer to as *Symm*. *ROBDD* represents the ROBDD-based set domain solver with merged constraints and no intermediate variables; *ic_sets* is the ECLiPSe library for a solver over sets of integers (cooperating with lib(ic), a hybrid integer/real interval arithmetic constraint solver), using a set ordering constraint that had to be implemented using reified constraints. Let us see how each one copes with the mentioned symmetries.

All 4 approaches order groups in each week with appropriate constraints. In particular, the *ROBDD* model uses a global constraint on the whole week, which partitions it from the set of golfers and simultaneously orders the groups. *Cardinal* just uses #< on the minimum of 2 sets, cooperating with the integer solver, thanks to the *Cardinal* facility to constrain set functions, and making use of the knowledge that sets to be ordered (groups) have each 4 elements and are disjoint. Thus, in each week, for 2 groups, G_1 and G_2, G_1 comes before G_2 iff $min(G_1) < min(G_2)$.

Weeks can be ordered by comparisons on the first group. *Cardinal* did not adopt these ordering constraints since, in general, it led to poorer execution times.

Symm ordering constraints apply globally to row and column symmetries on a Boolean matrix model, achieving more propagation.

As for value symmetry, and as mentioned in [18], some values can be fixed a priori. E.g. for problems with 4 groups of 3 golfers, since initially there is no difference between the possible values, the first week can be fixed as:

{ {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }.

Then, values of one group (which cannot play together anymore) can be fixed in other weeks as e.g.:

{ {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }
{ {1,?,?}, {2,?,?}, {3,?,?}, {?,?,?} }
{ {1,?,?}, {2,?,?}, {3,?,?}, {?,?,?} }

and so on, for the remaining weeks. *Symm* forces these values and refers to such constraints as basic symmetry breaking constraints. *Cardinal* also adopts this initial assignment of values. This makes no difference in finding a first solution since these values would come up naturally as the first values assigned, due to the modelled constraints and the usual ascending value labelling order. But in optimisation problems, or when proving that there is no solution, a lot of backtracking is avoided.

Table 1 shows the results obtained with these 4 approaches for the same instances of [18] and [19] (*Cardinal*: Pentium 4, 2.4 GHz, 480 Mb RAM; *ic_sets* and *ROBDD*: Pentium-M 1.5 GHz, all 3 with a 10 minutes time limit, whereas *Symm* ran on a 1 GHz Pentium III, 256Mb RAM, for a maximum of one hour on each instance). Shaded rows indicate instances with no possible solution. Times on these rows represent the time needed to prove it.

All 3 CLP(*Sets*) approaches use the "element in set" labelling. *Cardinal* and *ROBDD* obtained the best results with a first-fail heuristic, while *ic_sets* and *Symm* used sequential labelling.

Here, *ROBDD* obtains the best results due to its higher propagation having no intermediate variables, which greatly reduces search space. *ROBDD* is the only approach to solve all these instances, although *Symm* has alternative models (based on

Table 1. Time results (in seconds) of different CP approaches for social golfers

g-s-w	Cardinal	ic_sets	ROBDD	Symm
4-3-5	165.63	—	44.4	0.27
4-3-6	94.67	—	29.6	0.30
5-3-6	—	—	2.0	367.00
5-3-7	—	—	28.4	—
5-4-2	0.83	5.3	0.1	0.07
5-4-3	1.89	9.3	0.5	0.12
5-4-4	3.13	10.5	1.3	0.20
5-4-5	28.65	267.3	4.4	0.84
5-5-7	—	—	0.4	19.30
6-3-6	1.20	2.7	2.5	—
6-4-2	1.75	35.5	0.2	0.20
6-4-3	4.62	59.2	2.3	0.34
6-5-4	—	—	171.5	2.01
7-4-2	2.82	70.3	0.6	7.70
7-4-3	6.37	113.6	3.5	0.50
7-4-4	12.46	135.8	21.8	—
7-4-5	17.18	—	54.7	—
8-3-5	1.01	4.1	6.6	—
8-5-2	—	—	3.1	24.70
9-4-4	42.45	22.7	338.4	—

different orderings) [18] that can solve the remaining instances. The presented *Symm* results correspond to its best model (i.e. the one that can solve more instances). Other models can fill the gaps but then leave other instances unsolved, which shows the importance of using different strategies. Thus, in general, results are very dependent on "luck", according to each specific instance. Even with the more consistent *ROBDD*, results fluctuate a bit, and are still far from corresponding to each problem model size. When applying sequential labelling, *ROBDD* does not obtain a solution in 4 instances.

The fact is that much symmetry is still hidden and not taken into account, as we will see in the next section.

4 Breaking Symmetries with Equivalence Classes of Values

Inspired in the examples above, showing the importance of labelling and symmetry according to the properties of the problem, in this section we discuss and present an approach using equivalence classes of values to break symmetries, which we apply to the social golfers problem with a sequential labelling.

4.1 Examples and Definitions

In the example of the previous section for problems with 4 groups of 3 golfers, we have seen that we could start from an initial fixed configuration as:

{ {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }
{ {1,?,?}, {2,?,?}, {3,?,?}, {?,?,?} }
{ {1,?,?}, {2,?,?}, {3,?,?}, {?,?,?} }
…

We could fix the first week because weeks are indistinguishable; we could fix values 1, 2 and 3 in the first 3 groups of other weeks because groups within a week are indistinguishable; and we could fix values using integers from 1 to 12 because values are indistinguishable. Furthermore, this was done in a way that the imposed ordering constraints remain valid.

We can now ask: is that all we can do? Could we fix more values? In fact, the first group of the second week can be fixed as {1,4,7} so that we now have:

{ {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }
{ {1,4,7}, {2,?,?}, {3,?,?}, {?,?,?} }
{ {1,?,?}, {2,?,?}, {3,?,?}, {?,?,?} }
…

The reason why this is possible is because, after assigning value 1 to the group, values 2 and 3 (former partners of 1) become impossible, and we need to assign two other values that must come from two other different groups of the first week. Since there is no reason to differentiate between the 3 groups {4,5,6}, {7,8,9}, and {10,11,12}, we may pick the first 2, and from these, pick the first element, since, at this point, there is no reason to differentiate values inside these groups. The first possible groups and first (smallest) values are chosen to make it the lexicographically first group among the remaining possible combinations of values. (The fixing of this group was actually already performed in the *Cardinal* model of the previous section.)

From this point on, the situation is not that simple, since if we continue to sequentially assign to each group the first possible value, we would first assign 5 and 8 to group 2, to obtain {2,5,8}, and then {3,6,9} to the third group, which would lead to having only 10,11, and 12 as possible values to the last group, which is impossible. Since there is a solution to the problem, what went wrong?

What happens is that value 5 cannot be forced to be part of the second group, since now not all values in this set domain are indistinguishable. Although it is easy to see that 5 is indistinguishable from its former partner 6, it is not so in relation to value 10, for instance. The reason is that a former partner of 5 has already been assigned in this week (value 4). So, assigning value 5 now results in having assigned already 2 elements of a former group (all of which must be distributed throughout the whole week), while values from group {10,11,12} still have no assignment in this week, which may make it harder to obtain a solution, since 3 different groups must be found for them, and the week is getting "shorter".

But we have seen that 5 is indistinguishable from 6, so, if value 5 is tried and no solution is obtained, there should be no backtracking to value 6, since it will lead to a failure for the same basic reasons. The same happens with values 10, 11 and 12 – they are indistinguishable since they belong to the same group and have not been used anymore. We can also see that value 8 is indistinguishable from 9 and, in fact, these are also indistinguishable from 5 (and consequently from 6), since they share the same characteristics: they are partners of a golfer which has already been assigned in the current week.

There are thus 2 equivalence classes of indistinguishable values at this point: {5,6,8,9} and {10,11,12}. Therefore, only 2 values (at most) need to be tried to check whether a solution exists. When trying each class, the first value can be tried so that eventual ordering constraints remain valid.

What is it then that, in general, makes two values indistinguishable in the social golfers problem? What we have seen so far, takes into account values already assigned to previous weeks and previous groups (if we consider a week to be a sequence of groups), as well as values already assigned to the current group. It thus makes sense to consider classes of values in a sequential labelling context. For that, we model the *g-s-w* social golfers problem with a sequence of groups S_{ij} as $<S_{1,1},S_{1,2},\ldots S_{1,g},S_{2,1}, S_{2,2},\ldots S_{2,g},\ldots S_{w,1},S_{w,2},\ldots S_{w,g}>$, where i is the week index and j the group index, and each S_{ij} is a set, also seen as a sequence, of s golfers. We refer to the n^{th} element of S_{ij} as $S_{ij}(n)$.

In a sequential labelling of weeks, groups and group elements, we define *Dom(i,j,n)* to be a domain of possible values for $S_{ij}(n)$ in position given by *i,j* and *n*, as below, according to: values already assigned in the week, *Assigned(i,j,n)*; current group partners, *GPartners(i,j,n)*; and all former partners of these, *Ps(i,j,n)*:

$$Assigned(i,j,n) = \{x : \exists_{j1,\, n1}((j1 \quad s+n1 < j \quad s+n) \wedge S_{i,\, j1}(n1) = x)\}. \tag{4}$$

$$GPartners(i,j,n) = \{x : \exists_{n1 \,<\, n} S_{i,\, j}(n1) = x\}. \tag{5}$$

$$Partners(v,i) = \{x : \exists_{i1 \,<\, i} \exists j (v \in S_{i1,\, j} \wedge x \in S_{i1,\, j} \wedge x \neq v)\}. \tag{6}$$

$$Ps(i,j,n) = \bigcup_{v \in GPartners(i,j,n)} Partners(v,i). \tag{7}$$

$$Dom(i,j,n) = Golfers \setminus (Assigned(i,j,n) \cup Ps(i,j,n)). \tag{8}$$

where we represent *Golfers* as the set of integers ranging from 1 to *g*s*, and *Partners(v,i)* is the set of partners of golfer *v* in all weeks before *i*.

We consider two values of a domain of golfers to belong to the same class, when adding an element to a group, when they have the following 2 counts in common:

1- number of partners from those in the domain
2- number of partners from those already assigned to the week

which we may now formally define as follows.

Two values, v_1 and v_2, of *Dom(i,j,n)*, are said to be equivalent, or symmetrical, on assignment to the n^{th} element of group $S_{i,j}$ when both (**9**) and (**10**) below apply:

$$\#(Partners(v1,i) \cap Dom(i,j,n)) = \#(Partners(v2,i) \cap Dom(i,j,n)). \tag{9}$$

$$\#(Partners(v1,i) \cap Assigned(i,j,n)) = \#(Partners(v2,i) \cap Assigned(i,j,n)). \tag{10}$$

Condition (**9**) differentiates class {5,6,8,9} from {10,11,12} of the example above. We explain the necessity of condition (**10**) with the following example: consider the 3-2-3 problem instance when trying to label $S_{3,2}(2)$ as:

{ {1,2}, {3,4}, {5,6} }
{ {1,3}, {2,5}, {4,6} }
{ {1,4}, {2,?}, {?,?} }

At this point, only values 3 and 6 are possible, but only value 6 leads to a solution. Condition (**9**) does not differentiate them since their partners are, respectively, {1,4} and {4,5}, none of these sets sharing a partner with the current domain. The reason

why they are not indistinguishable is that one (value 3) has already 2 partners (1 and 4) assigned in this week, whereas the other (value 6) has only one partner already assigned (4), thus leaving more possibilities to the rest of the groups.

The two conditions are both thus necessary conditions of indistinguishability, although their conjunction is not assured to be a sufficient condition, for this particular problem, as we confirmed later.

We thus consider equivalence classes of symmetrical values in a similar fashion as GE-trees [26], but with our own dynamic *ad-hoc* notion of *symmetrical* values, which may include two non-symmetrical (but *similar*) values in the same equivalence class.

4.2 Algorithm

The basic algorithm to solve a *g-s-w* golfers problem using equivalence classes (EC) of values can thus be defined by the 2 procedures in pseudo-code below (with *GolfersEC(g,s,w,S)* as the top goal), where *S* is the desired solution as a 3-dimensional array with the complete schedule, if successful:

```
Procedure GolfersEC (In: g,s,w, Out: S)
  for i from 1 to w do
      for j from 1 to g do
          S_{i,j}(1) ← first_value(Dom(S_{i,j}(1)))
          for n from 2 to s do
              AssignEC(S,i,j,n,Dom(i,j,n),∅)
          end for
      end for
  end for
end Procedure

Procedure AssignEC (In/Out:S, In: i,j,n,Vals,Tried)
  remove_first v from Vals yielding RestVals
  N1 ← #(Dom(i,j,n) ∩ Partners(v,i))
  N2 ← #(Assigned(i,j,n) ∩ Partners(v,i))
  ChoicePoint
      (<N1,N2> ∉ Tried
       S_{i,j}(n) ← v
      ) or
      AssignEC(S,i,j,n, RestVals, Tried ∪ {<N1,N2>})
  end ChoicePoint
end Procedure
```

The algorithm consists of a sequential labelling, as explained, with possible choice-points in assignments when there are differentiated values. This is performed on procedure *AssignEC*, by verifying on backtracking whether a new value (from the initial domain) is of a class already tested (in which case, it will not be tried again). Hence, equivalence classes need not be computed at each instantiation – only if necessary will a value be checked for indistinguishability, by verifying the inclusion of its class characteristics in a set (*Tried*) of already tried ones. This method corresponds to using nogoods [16].

The first week is thus automatically fixed and ordered with no possible backtracking, since values are indistinguishable throughout the week. Order of groups within weeks is assured by assigning the first possible value of the domain (which shortens along the week groups) with no possible backtracking, when reaching each first position of a group. This breaks symmetry of groups, since they could be permutated without affecting the solution (hence, no solution is lost here).

An assignment fails when the domain is empty. That is when backtracking takes place, until a solution is found. If the search space is exhausted with no solution found, the procedure fails, indicating that the problem is impossible.

This algorithm, as is, does not guarantee that weeks are sorted. That can be done optionally by comparisons on the first group, but the result is not necessarily better. To obtain a first solution, in general it is better not to force a sorting on weeks. On the other hand, to prove impossibility or in optimisation, where a complete search is needed, it is better to force this sorting since it prunes search space. In fact, our implementation also included this sorting, with a constraint $S_{i,1}(2) \#< S_{i+1,1}(2)$, for each week i. This can be done on the 2nd element of a first group since it is guaranteed that the first element will always take value 1.

We also extended this algorithm further with two particular look-ahead optimizations for this problem:

1. We considered the (fixed) last group of the first week, whose s elements must be spread through s different groups in subsequent weeks. Therefore, when assigning the last element of a group, we verify whether we have to place one of those (in case there are no more groups available for the yet unassigned elements in the current week). E.g., with 4 groups of 3 golfers, the first week is { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }. If we are to assign $S_{2,2}(3)$ in a 2nd week as { {1,4,7}, {2,5,?}, {?,?,?}, {?,?,?} }, it must be value 10 (rather than 8 or 9), because otherwise values {10,11,12} would not fit in this week.
2. In each group, the s (ordered) elements must come from s different groups of the first week. Since the first week is completely ordered (from 1 to the total number of golfers), on each assignment of a group position in subsequent weeks, the corresponding golfer has some maximum number possible (otherwise there would not be enough room in the group to receive members of different groups). E.g., with 5 groups of 4 golfers, the first week is { {1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}, {17,18,19,20} }. If we are to assign $S_{2,1}(2)$ in a 2nd week as { {1,?,?,?}, {?,?,?,?}, {?,?,?,?}, {?,?,?,?}, {?,?,?,?} }, it must be less than 13. Hence, on backtracking we do not need to consider values 13 to 20, because such values in that position could not lead to a complete group.

4.3 Results and Limitations

In Table 2 we present the results in seconds of our method (*EC*) to find a solution (or exhaust search space) to each of the instances of Table 1, with a C++ implementation on the same 2.4 GHz machine. We reproduce the results already shown for the other CP approaches, for comparison.

Table 2. Time results of other CP approaches for social golfers in comparison with our Equivalence Classes method

g-s-w	Cardinal	ic_sets	ROBDD	Symm	EC
4-3-5	165.63	—	44.4	0.27	0.01
4-3-6	94.67	—	29.6	0.30	0.00
5-3-6	—	—	2.0	367.00	0.13
5-3-7	—	—	28.4	—	1.12
5-4-2	0.83	5.3	0.1	0.07	0.00
5-4-3	1.89	9.3	0.5	0.12	0.00
5-4-4	3.13	10.5	1.3	0.20	0.01
5-4-5	28.65	267.3	4.4	0.84	0.01
5-5-7	—	—	0.4	19.30	0.00
6-3-6	1.20	2.7	2.5	—	0.01
6-4-2	1.75	35.5	0.2	0.20	0.00
6-4-3	4.62	59.2	2.3	0.34	0.00
6-5-4	—	—	171.5	2.01	0.01
7-4-2	2.82	70.3	0.6	7.70	0.01
7-4-3	6.37	113.6	3.5	0.50	0.01
7-4-4	12.46	135.8	21.8	—	0.01
7-4-5	17.18	—	54.7	—	0.02
8-3-5	1.01	4.1	6.6	—	0.04
8-5-2	—	—	3.1	24.70	0.00
9-4-4	42.45	22.7	338.4	—	0.01

First of all, the reason why *EC* is instantaneous in exhausting search space of instances *4-3-6* and *5-5-7* is that they are trivially impossible as checked by the simple test $w \leq (g*s\text{-}1)$ // $(s\text{-}1)$, which we added to our system (where // is the integer division). This test has a simple explanation: a golfer, x, has to play with $(s\text{-}1)$ different partners in each week, and there are only $(g*s\text{-}1)$ other golfers (as already noted also in CSPlib). Nonetheless, *4-3-5* is not trivially impossible, and *EC* exhausts search space with no solutions in just 1 hundredth of a second.

EC finds a solution for all other instances also almost instantaneously (only the *5-3-7* instance required 1.12 seconds). *EC* is always faster than *ROBDD*, generally outperforming it by 2 or more orders of magnitude.

In addition, since our method performs a sequential labelling with no definite number of variables *a priori*, it handles better the optimisation problem of finding the maximum number of weeks. In fact, weeks are constructed sequentially and each new week achieved is based on the search already performed for the previous weeks. Hence, our approach reports, for given numbers g and s, each new improved solution obtained, starting from the first week until a maximum is reached, also taking into account the pre-computed trivial upper bound. Thus, instead of trying in turn a specific number of weeks, the maximum is constructed incrementally. Using this method, we extended results of Table 2 to the more complete and organised Table 3 where, given g and s, an upper bound is computed and solutions keep coming (in a time limit of 5 minutes) until the maximum number of weeks is achieved (marked with √) either by reaching the upper bound or by exhausting the search space.

Table 3. Search for optimality with Equivalence Classes

Groups	Group Size	Weeks upper bound	Weeks achieved	Search Space Exhausted	Time
4	3	5	4	√	0.01
	4	5	5	√	0.00
5	3	7	7	√	1.10
	4	6	5	√	0.10
	5	6	6	√	0.03
6	3	8	7		1.36
	4	7	6		123.01
	5	7	5		6.56
7	3	10	6		0.08
	4	9	6		0.05
	5	8	5		0.11
8	3	11	8		0.08
	4	10	5		0.02
	5	9	6		15.63
9	3	13	9		4.29
	4	11	5		0.51

EC finds and exhausts search space in all instances with 4 and 5 groups (note that problems with $s>g$ are also trivially impossible for $w>1$, and are not considered in this table). For more than 5 groups, no search was complete, but more weeks than the maximum values presented by the other approaches were achieved. There were even 4 more weeks found in the instance with *g*-*s* values of 8-5, and 3 more in instances 6-4 and 8-3 (this one in just 8 hundredths of a second). In addition to presenting new results, all instances were improved.

Note that a particular instance can still be solved much faster with a specific labelling heuristic in a CP approach, but that heuristic will hardly be useful to a large set of instances. For example, using a heuristic of assigning each golfer in turn, to a group in every week, we obtained a 9-weeks solution for 8-4 (the original CSPlib problem) in 8 seconds with *Cardinal*. But then, many other instances remain unsolved: e.g. just by incrementing group size to 5, not even a 2 weeks solutions was then found to the 8-5 problem. *EC*, although seeming disappointing in this instance, is much less dependent on particular instances of this problem, showing a big consistency.

Unfortunately, the described method is incomplete. In fact, we later found a counter example to the possibility that conditions (**9**) and (**10**) could be sufficient for indistinguishability of golfers: in instance 6-5, after running for 6 minutes, search space was exhausted, finding no better than 5 weeks, when a 6 weeks solution is known. Thus, when we exhaust search space, no better solution can be found with the current algorithm; we may not conclude that the optimum was reached. Nevertheless, obtained results are correct and even the exhaustion of search space of Table 2 entries correspond to real optima. The described technique can be used to obtain fast results, or may serve as heuristics integrated in a complete tool. We obtain good results due to the restricted search space, and to our look-ahead optimisations. Notice that we did not include instances with $s=2$ or with $g=s$ in Table 2, cases where usually solutions

are easier to obtain (although not always). In such cases, *EC* is particularly fast; for example, for 7-7 the optimum 8-weeks solution is found and search exhausted in just 0.12 seconds. This result has just recently been recognised as the best solution on Warwick Harvey's page for the social golfers problem (formerly at http://www.icparc.ic.ac.uk/~wh/golf/, but currently unavailable), although a solution was not available there. Such page reports results from several different mathematical and CP sources, with specialised techniques based on mathematical constructions and local search, for instance. Currently we cannot improve such results, but we consider that promising results were obtained with this research work.

5 Conclusions and Further Research

In this paper we presented an approach using equivalence classes of values to break symmetries on highly symmetric problems such as those where the goal is to find, or somehow optimise, a set (or sets) of undifferentiated values for a set (or sets) of undifferentiated entities/variables. Current CP approaches model such problems with a fixed number of distinct variables, each with its own domain, thus lacking flexibility and losing information on the similarity of variables and values. This leads to allowing backtracking to an unnecessary re-exploration of search spaces that share the same characteristics. We showed that posting ordering constraints is clearly not sufficient to avoid this, still leaving much symmetry behind.

The Equivalence Classes scheme checks whether a new value in the domain is worth trying, by verifying whether a similar value has already been tried in that context. If so, then the value is simply discarded, thus breaking that symmetry and pruning search space. For this we introduced the notion of equivalent values on a dynamic execution. We applied this technique in the particular social golfers problem, where a sequential labelling allowed: *a)* an easy verification of equivalent values; *b)* immediate breaking of group symmetry (avoiding propagation); and *c)* the possibility for a solution to grow dynamically, thus facilitating solving the optimisation problem.

Experimental results in a C++ implementation showed improvements of orders of magnitude over other CP approaches with efficient propagators (such as an ROBDD-based one), namely over sets, and with global ordering constraints specially implemented to break symmetries [18]. Nevertheless, further comparison with other dynamic symmetry breaking techniques should yet be performed.

Although the indistinguishable conditions presented for this particular problem are incomplete, results are all valid and extended an already large set of instances to improved solutions in a matter of seconds. We consider that the notion of equivalent values and corresponding classes has been sufficiently illustrated with this example, showing its importance for future research. It also shows that it may be worth to partly relax symmetrical conditions in order to obtain larger equivalence classes, thus reducing search space, even if becoming incomplete. Such relaxation may also allow a faster dynamic computation of symmetry and should be further explored in general problems.

In future, for this particular problem, we intend to compare our method with others that search for all solutions, in order to better assess our search space reduction.

We believe this technique can be further improved since it can possibly benefit from other look-ahead techniques and be integrated with another CP approach using global constraints.

Future research should try to find indistinguishable conditions automatically from problem specification. For that, modelled problem variables and values that are really undifferentiated, should not be given names or numbers, *a priori* (as usually done in CP and other programming approaches). More efficiency may be achieved if allowing more declarative models, nearer natural language, to extract only the necessary information, correctly interpreting things like "a set of *n* golfers for a set of weeks" (capturing the implicit symmetries and also allowing a dynamic number of variables).

References

1. Anderson, I., Honkala, I.: A Short Course in Combinatorial Designs (1997), http://www.utu.fi/h onkala/cover.html
2. Azevedo, F.: Constraint Solving over Multi-valued Logics - Application to Digital Circuits. In: Frontiers of Artificial Intelligence and Applications, vol. 91, IOS Press, Amsterdam (2003)
3. Azevedo, F.: Cardinal: A Finite Sets Constraint Solver. Constraints journal, vol. 12(1), pp. 93–129, KAP (2007)
4. Azevedo, F., Barahona, P.: Modelling Digital Circuits Problems with Set Constraints. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 414–428. Springer, Heidelberg (2000)
5. Barnier, N., Brisset, P.: Solving the Kirkman's Schoolgirl Problem in a Few Seconds. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 477–491. Springer, Heidelberg (2002)
6. Beldiceanu, N.: An Example of Introduction of Global Constraints in CHIP: Application to Block Theory Problems. Technical Report TR-LP-49. ECRC, Munich, Germany (1990)
7. Choueiry, Noubir.: On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. In: Proc. AAAI'1998 (1998)
8. Colbourn, C.J., Dinitz, J.H. (eds.): Steiner Triple Systems. CRC Handbook of Combinatorial Designs. vol. 70, pp. 14–15, CRC Press, Boca Raton, FL (1996)
9. Fahle, T., Shamberger, S., Sellmann, M.: Symmetry Breaking. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 93–107. Springer, Heidelberg (2001)
10. Focacci, F., Milano, M.: Global Cut Framework for Removing Symmetries. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 75–92. Springer, Heidelberg (2001)
11. Gent, I.P., Harvey, W., Kelsey, T.: Groups and Constraints: Symmetry Breaking During Search. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 415–430. Springer, Heidelberg (2002)
12. Gent, I.P., Smith, B.M.: Symmetry Breaking During Search in Constraint Programming. In: Proc. ECAI'2000 (2000)
13. Gervet, C.: Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. In: Freuder, E.C. (ed.): Constraints journal, vol. 1(3), pp. 191–244, Kluwer Academic Publishers (1997)
14. Harvey, W.: Symmetry Breaking and the Social Golfer Problem. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, Springer, Heidelberg (2001)

15. Hawkins, P., Lagoon, V., Stuckey, P.J.: Set Bounds and (Split) Set Domain Propagation Using ROBDDs. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS (LNAI), vol. 3339, pp. 706–717. Springer, Heidelberg (2004)
16. Van Hentenryck, P.: A Logic Language for Combinatorial Optimization. In: Annals of Operations Research (1989)
17. Kirkman, T.P.: On a problem in combinatorics. Cambridge and Dublin Math. Journal 191–204 (1847)
18. Kiziltan, Z.: Symmetry Breaking Ordering Constraints. PhD thesis, Uppsala University (2004)
19. Lagoon, V., Stuckey, P.J.: Set Domain Propagation Using ROBDDs. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 347–361. Springer, Heidelberg (2004)
20. Lindner, C.C., Rosa, A.: Topics on Steiner Systems. Annals of Discrete Mathematics, North Holland, vol. 7 (1980)
21. Lueneburg, H.: Tools and Fundamental Constructions of Combinatorial Mathematics, Wissenschaftverlag (1989)
22. Meseguer, P., Torras, C.: Exploiting Symmetries Within Constraint Satisfaction Search. Art.Intell. 129, 133–163 (1999)
23. Puget, J.-F.: PECOS: A High Level Constraint programming Language. In: Proc. Spicis 92, Singapore (1992)
24. Puget, J.-F.: Symmetry Breaking Revisited. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 446–461. Springer, Heidelberg (2002)
25. Puget, J.-F.: Symmetry Breaking Using Stabilizers. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, Springer, Heidelberg (2003)
26. Roney-Dougal, C.M., Gent, I.P., Kelsey, T., Linton, S.A.: Tractable symmetry breaking using restricted search trees. In: Proceedings of ECAI-04 (2004)
27. Sellmann, M., Harvey, W.: Heuristic Constraint Propagation. In: Proceedings of CPAIOR'02 workshop, pp. 191–204 (2002)