Symmetry Breaking and the Social Golfer Problem

Warwick Harvey

IC-Parc, Imperial College, Exhibition Road, London SW7 2AZ, United Kingdom

Abstract. Dealing with highly symmetric problems in a constraint programming context is an area of growing interest. The Social Golfer Problem is a highly symmetric problem on which many researchers are trying out their algorithms. In this paper we take a closer look at the social golfer problem and some of the techniques which can be used to solve it, focussing in particular on Fahle et. al's Symmetry Breaking via Dominance Detection.

1 Introduction

Within the constraint programming community there is growing interest in finding more effective ways of tackling problems which contain symmetry. Of course, the more symmetry a problem contains, the greater the benefit which can be derived from dealing effectively with that symmetry. The big challenge facing those studying symmetry-breaking techniques is handling problems which are inherently highly symmetric. For these problems ignoring the symmetry and attempting to tackle them directly using current constraint-based techniques is impractical.

One such problem, which is a current favourite among symmetry researchers, is the Social Golfer Problem (see problem 10 in [4]). A standard generalisation of this problem can be stated as follows.

> A group of N golfers wish to play golf each week, arranged into g groups of s golfers ($N = gs$). Find a playing schedule which covers w weeks such that no two golfers play together more than once.

This problem is of interest mainly because of its high degree of symmetry. A very naïve formulation has $(s!)^{gw}(g!)^w w!(gs)!$ symmetries (which obviously grows very rapidly as the size of the problem increases!). Even the best published formulation in terms of symmetry reduction of which I am aware still has $(gs)!$ symmetries[1] [7].

This is a real problem of practical relevance. A member of a golf club in Melbourne, Australia, when organising a 5-day golf tour along the Murray River, wished to arrange the daily foursomes such that everyone played with different people each day. The solution provided to them (courtesy of Keith Edwards) was highly valued, and has been re-used for the golf tour every year since [2].

In this paper we will give an overview of a number of different techniques for solving the social golfer problem. We will look at several of these in more detail, including the Symmetry Breaking via Dominance Detection (SBDD) method of Fahle et al. [1], and its relationship to the Symmetry Breaking During Search (SBDS) of Gent and Smith [3]. We will propose a refinement to SBDD, and present a detailed description of how SBDD can be implemented for the social golfer problem. Finally, we will present some experimental results evaluating SBDD and our modification, and make some suggestions as to how the technique could be improved further.

2 Background and Related Work

Three main ways have been proposed for tackling symmetry in constraint problems. These are:

1. re-model the problem;
2. add constraints; and
3. modify the search procedure.

It is usually not necessary to remove all of the symmetry in the problem; often one is just interested in finding the solution(s) of interest as quickly as possible, rather than guaranteeing that only distinct solutions are returned (particularly when the first solution found will do). Thus there is often a trade-off between the cost of eliminating symmetries and the savings derived

[1] Though this can be reduced to $w(s!)^g g!$ by making an initial assignment to one week.

from having done so. Also, in general, the sooner a symmetric part of the search space can be removed from consideration, the better; waiting until a solution is found and then discarding it because it is symmetric to one already discovered cannot result in finding solutions faster.

Definition 1. Two solutions to an instance of the social golfer problem are considered equivalent if one can be obtained from the other by applying any sequence of operations from the following classes:

1. Rename the players.
2. Rearrange the players within a group.
3. Rearrange the groups within a week.
4. Rearrange the weeks.

One could argue that there are actually relatively few symmetries inherent to the social golfer problem: specifically, that most of the "symmetries" listed above are artifacts of models of the problem, not of the problem itself. Indeed, most published approaches avoid symmetry class 2, either through modelling or by imposing symmetry-breaking constraints. Moreover, by a suitable choice of model, it seems that all but one of the above classes of symmetry can be eliminated.

Joachim P. Walser proposed a pseudo-boolean formulation (see problem 10 in [4]), with a boolean variable representing whether player i is a member of group j in week k for all i, j, k, and an auxiliary boolean variable representing whether players i and j play together in week k for all i, j, k, $i < j$. This model avoids symmetry class 2, but not the others. He then used the local search heuristic Wsat(PB) to solve this formulation, finding an 8-week solution for 8 groups of 4 players.

Stefano Novello used a model where each group was a set (see [6]). Again, this avoids symmetry class 2, but not the others. He then used a simple labelling strategy which chose groups for each player in turn, finding a 9-week solution for 8 groups of 4 players.

Barbara Smith has studied a number of different models with a view to reducing symmetry [7]. In particular, by considering pairs of players, and then considering which pairs of players play in the same week as other pairs of players, she has been able to formulate a model that eliminates symmetry classes 2, 3 and 4 simultaneously. She then used the SBDS technique developed with Ian Gent [3] to eliminate some of the remaining symmetry during the search phase.

Torsten Fahle, Stefan Schamberger and Meinolf Sellmann's SBDD technique [1] also eliminates symmetry during the search phase, and has been applied to the social golfer problem. The two techniques are compared further in Section 3.

Another approach to handling symmetry which operates during the search phase is that of Pedro Meseguer and Carme Torras [5]. They present variable and value selection heuristics which try to increase the likelihood of finding a solution quickly by exploiting knowledge of symmetries of the current state of the search. To the best of our knowledge these techniques have not been applied to the social golfer problem, but they have been applied to the closely related problem of finding Balanced Incomplete Block Designs (BIBDs) with pleasing results: there was a much higher likelihood of finding a solution early when a symmetry-exploiting heuristic was used. It would be interesting to try to combine these heuristics with other symmetry-breaking techniques.

As mentioned above, it is also possible to add constraints a priori in order to remove symmetry from a problem. For example, an ordering could be applied to labellings of the problem variables, and lexicographic constraints imposed in order to ensure that only the smallest solution from each symmetrically equivalent set is considered feasible (or perhaps just to ensure that only quite small solutions from each set are feasible). We are not aware of much work in this regard being applied to the social golfer problem. However, it is quite common to remove some initial symmetry by assigning values to some of the variables at the start. Usually this involves assigning values for one full week, and then taking the players from one group from that week and assigning them to distinct groups in all the other weeks. Note that if one is simply looking for a first solution (rather than all solutions) then performing initial labelling does not help because any solution will have a symmetric solution which is compatible with the initial labelling. In such cases the labelling only helps when there are no solutions.

3 Breaking Symmetry While Searching

To the best of our knowledge, there are two general techniques for breaking symmetry during the search phase which have been applied to the social golfer problem. These are Symmetry Breaking During Search (SBDS) [3] and Symmetry Breaking via Dominance Detection (SBDD) [1]. The fundamental principle behind these two approaches is the same: don't bother exploring any part of the search space which is symmetric to somewhere which has already been considered. The implementations apply this principle in different ways (assuming depth-first search):

- SBDS, when it has finished exploring a subtree, imposes constraints to exclude all symmetric subtrees from consideration.
- SBDD, when exploring a new node, performs a check to see whether it is dominated by a symmetric equivalent of a subtree already explored.

In both cases the symmetries of a subtree are only considered for exclusion during the exploration of the siblings of that subtree. This is because as soon as all the siblings are explored, the symmetries of the subtree rooted at the parent are considered, and these subsume those of its children.

Current implementations of SBDS require an explicit list of all the symmetries which are to be broken. Obviously, for highly symmetric problems such as the social golfer problem, listing all the symmetries is not practical. However, using a partial set of symmetries is possible, and can be quite effective at reducing both the size of the search tree and the time required to find a solution [7]. It is also possible that SBDS can be adapted so that an explicit list of the symmetries is not required, for example by exploiting the group-theoretic nature of the symmetries. This should hopefully make it possible to use SBDS to eliminate all symmetries from the social golfer problem and return only unique solutions, without resorting to other techniques.

SBDD is capable of eliminating all symmetries from the social golfer problem. Currently, the main drawback of using SBDD for the social golfer problem is that the dominance check can be quite expensive. The solution proposed in [1] is to do full dominance checks only occasionally; at the other nodes, a much cheaper check is done which eliminates just some symmetries.

3.1 Relationship between SBDS and SBDD

SBDD checks at each node (or at whatever desired frequency) whether the current state is dominated by any state visited previously. That is, it repeatedly checks to see whether the following condition holds:

$$\neg\exists\phi \in \Phi, S' \in \mathcal{S} : S \preceq \phi(S')$$

where Φ is the set of symmetries of the problem, S is the current state and $\mathcal{S}$ is the set of previous states to be compared against (the roots of the completed subtrees).

Given that these active dominance checks are quite slow for highly symmetric problems, one might consider whether they could be converted into constraints on the problem instead, so that they can be performed incrementally as the computation progresses. For this to work, the negation in the above expression should be removed, to achieve monotonicity. Pushing the negation through the existential quantifier results in:

$$\forall\phi \in \Phi, S' \in \mathcal{S} : S \not\preceq \phi(S')$$

That is, for each symmetry and each completed subtree, we wish to impose a constraint that the state resulting from applying that symmetry to the state from that subtree does not dominate the current state. But this is exactly what SBDS does (with the optimisation that constraints are not imposed if they correspond to symmetries which cannot hold during the lifetime of the constraint[2]). Thus SBDS appears to be the passive, incremental equivalent of SBDD.

It would be interesting to investigate a hybrid SBDS/SBDD approach, where SBDS was used to exclude a manageable subset of the symmetries, and full SBDD used at appropriate points to exclude the remaining symmetries. In particular, if one is doing full SBDD checks only occasionally and weaker checks at the other nodes, it would be interesting to see what the effect of replacing the weaker checks with their SBDS equivalent would be.

[2] Note that one would normally expect an SBDD dominance check to quickly discount such symmetries from consideration, so applying a similar mechanism to SBDD seems unlikely to be useful.

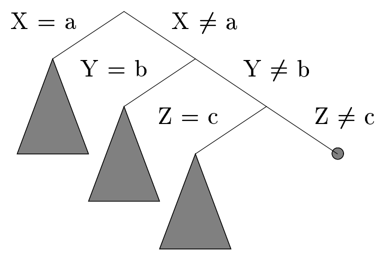

Fig. 1. A partial search tree

4 Our SBDD Implementation

We have implemented SBDD in ECL^{i}PS^e, a Constraint Logic Programming (CLP) system [8]. The implementation allows essentially any data structure to represent the state of the search space, containing almost any kind of variable (finite domain, set, etc. — or even user-defined).[3] The current implementation also allows any kind of depth-first search, being independent of variable and value ordering, and has been designed so that it can be used as a wrapper around variable labelling routines without modifying the internals of such routines. In future it is expected to support other kinds of search by assuming a pre-defined ordering on the solutions, as outlined in [1].

As noted earlier, for highly symmetric problems such as the social golfer problem the cost of performing a dominance check can be very high. Here we present a simple method which can reduce the cost of dominance checks without sacrificing pruning power. The basic idea is that it should be possible to compare the state of the current node against the states of all completed subtrees simultaneously, for less than the cost of performing them individually (and hopefully for not much more than the cost of checking one). Obviously this gains us nothing anywhere in the search tree where there is only one prior state to compare against, but if the search tree is non-trivial, it would be expected that having several prior states to compare against would not be uncommon.

Consider the search tree in Figure 1. The current state is characterised by the choices $\{X \neq a, Y \neq b, Z \neq c\}$. It is to be compared for dominance against states characterised by the choices $\{X = a\}$, $\{X \neq a, Y = b\}$ and $\{X \neq a, Y \neq b, Z = c\}$. Essentially, the current state is dominated by at least one of these three subtrees if it is dominated by any state characterised by $X = a \vee (X \neq a \wedge Y = b) \vee (X \neq a \wedge Y \neq b \wedge Z = c)$ (or simply $X = a \vee Y = b \vee Z = c$ in this case). If a reasonably general technique is being employed to check dominance, then it should be possible to use constructive disjunction or some similar method to allow the dominance check to consider all the states simultaneously. We describe a concrete example of this for the social golfer problem in Section 4.2 and evaluate it in Section 5.

4.1 Checking Dominance for the Social Golfer Problem

With any implementation of a dominance check for a highly symmetric problem, the key is not to consider all symmetries in order to find out whether one of them results in a dominance relation, but rather to *search for a symmetry which makes the dominance relation true.* The obvious way to do this is to set up a constraint subproblem where the search space corresponds to the symmetry space of the main problem. Constraints are then imposed which restrict the symmetry space based on the states being compared, and a search for a symmetry is performed. If a solution is found, then the dominance relation holds, and the current node in the master problem should be pruned.

To begin with, we consider comparing the current state with just one other; we will explain how to adapt this approach for multiple states in the next section.

---

[3] The only requirement is that the data structure and its contents be copyable using **copy_term/2**. Generally this means circular data structures are not permitted, but just about anything else is.

To compare one state A with another B to see whether A is dominated by B, we need to determine whether there is a permutation of the player names, the groups and the weeks which makes A an instance of B; that is, every possible further instantiation of A is also a possible further instantiation of B. In order to set this up as a constraint problem, we need to impose constraints which ensure that some permutation of A is an instance of B. To assist in this, we introduce three maps: one mapping player names to player names (P), one mapping groups to groups (G), and one mapping weeks to weeks (W). (Note that W is implicit in G, but we use W to help ensure that groups from the same week get mapped to groups from the same week.) The idea is that if P and G are applied to A, then the result should be constrained to be an instance of B. For this we introduce an intermediate state C which is the result of applying P to A, and the result of applying G to C is constrained to be an instance of B.

To be more concrete, each of the maps is implemented as a boolean matrix where each row and column is constrained to have exactly one '1' entry, and the group map is further constrained to keep groups from the same week together, based on the week map. (Essentially, the group map is partitioned into submatrices by week, and each submatrix is zero iff the corresponding entry in the week map is zero.) In the dominance subproblem, the states A, B and C are each represented by a pair of vectors of boolean vectors, one indicating which players are definitely members of each group (the lower bound), and one indicating which players might be members of each group (the upper bound). We write the lower bound of A as $\underline{A}$ and the upper bound as $\overline{A}$; similarly for B and C.

For the player map P, if it maps player i to player j (i.e. $P[i,j]=1$), then for each group k, if i appears in group k in A, then j must appear in group k in C, and vice-versa; i.e.

$$P[i,j] \Rightarrow (\underline{A}[k,i] = \underline{C}[k,j] \wedge \overline{A}[k,i] = \overline{C}[k,j]) \quad \forall k = 1 \ldots wg$$

In practice we use the following equivalent set of constraints, since they seem to propagate faster:

$$P[i,j] \leq C[k,j] \quad \forall k = 1 \ldots wg, i \in I_k \qquad \sum_{i \in I_k} P[i,j] \geq C[k,j] \quad \forall k = 1 \ldots wg$$

$$\sum_{j=1}^{gs} C[k,j] = \#I_k \quad \forall k = 1 \ldots wg$$

where $I_k = \{i : A[k,i] = 1\}$ (A is fixed for any given subproblem).

For the group map G, if it maps group i to group j ($G[i,j]=1$), then group i of C must be an instance of group j of B; i.e. each player k in the $\underline{B}$ group must be in the $\underline{C}$ group, and there must be no player k in the $\overline{C}$ group which is not also in the $\overline{B}$ group:

$$G[i,j] \Rightarrow (\underline{C}[i,k] \geq \underline{B}[j,k] \wedge \overline{C}[i,k] \leq \overline{B}[j,k]) \quad \forall k = 1 \ldots gs$$

Once these constraints are set up, all that is left to do is to try to find an assignment to the variables such that the constraints hold. Note that it is sufficient to label the player and group maps: if they have fixed values then the constraints ensure that all other variables have fixed values as well. We do not use any particular variable ordering heuristic; variables are simply assigned values in the order they appear in the matrices. The only exception to this is that we assign values to the group map before the player map, because it seems to result in shorter running times.

4.2 Checking Multiple Dominance Relations Simultaneously

As noted earlier, it is possible to check to see whether one state is dominated by any of several other states at the same time. To do this, we use essentially the same approach as in the previous section. The difference is that in this case B does not correspond to a single potentially dominating state, but is instead constrained to be equal to one of several potentially dominating states. If we denote these candidate states by B_1, B_2, etc., then we want to impose the constraint $B = B_1 \vee B = B_2 \vee \ldots$. We do this by introducing a variable b indicating which of these states B should be equal to, and then imposing a set of **element** constraints:

$$\mathbf{element}(b, [\underline{B_1}[i,j], \underline{B_2}[i,j], \ldots], \underline{B}[i,j]) \quad \forall i = 1 \ldots wg, j = 1 \ldots gs$$

$$\mathbf{element}(b, [\overline{B_1}[i,j], \overline{B_2}[i,j], \ldots], \overline{B}[i,j]) \quad \forall i = 1 \ldots wg, j = 1 \ldots gs$$

As discussed in the next section, this appears to be quite an efficient way of performing dominance checks when a state has to be compared to several others.

5 Experimental Results

We have conducted a number of experiments to compare the effect of using the above-described SBDD approaches on the social golfer problem against not using any symmetry breaking technique. The experiments were performed on 450 MHz Pentium II machines running Linux. The software was all written in ECL^{i}PS^e.

We used a set-based model, essentially that of Stefano Novello [6], modified to allow different instances. We added redundant constraints which other experiments have shown to be helpful in pruning the search space: constraining the sum of the cardinalities of the intersections of a group with the groups of another week to be equal to the group size. I.e.

$$\sum_{k=1}^{g} \#(G_{i,j} \wedge G_{k,l}) = s \quad \forall i = 1 \ldots g, j = 1 \ldots w, l = 1 \ldots w, l \neq j$$

where $G_{i,j}$ denotes the ith group in week j.

We used a "week-first" labelling strategy. This is simply to go through the weeks in order, labelling the groups in order. Of all the search strategies we have studied so far for this model of the social golfer problem, this is by far the most stable (less prone to do unexpectedly well or unexpectedly poorly on any given instance), and for larger numbers of weeks, usually also the fastest. For the purposes of counting nodes and for applying SBDD, a choice was considered to be the labelling of one group which had not yet been fully determined (though perhaps something more fine-grained would have been more appropriate for larger group sizes). When applying SBDD, a full dominance check was performed at every node (after the first backtrack).

For each instance, we report the time taken and number of nodes explored when finding the first solution and when finding all solutions, subject to a 2 hour time limit. When SBDD was used, as well as the number of nodes explored, we report the number of dominance checks performed and the number of times this lead to the node being pruned (nodes/checks/prunes in the table). Finally, we also report the number of solutions found within the time limit. The results are shown in Table 1. We have included all non-trivial instances of up to 6 groups, in each case for a number of weeks up to one more than the best upper bound of which we are aware (to see whether infeasibility can be proved).

Consider first the problem of finding all solutions. As noted earlier, each dominance check can be quite expensive, so it is little surprise that for small problems, it is usually faster if SBDD is not used. However, as the problems get larger, the average number of nodes pruned for each successful dominance check increases, so the overhead of doing the checks pays off, offering substantial savings in execution time in addition to the search space reduction. This also leads to being able to completely solve 9 cases using SBDD within the time limit which could not be solved without it.

Note also that using merged dominance checks is clearly superior. When the entire search space was explored, it was almost never slower, and often about twice as fast (depending upon the scope for merging dominance checks). In the other cases (when the time limit was reached) it usually found more solutions, and usually explored many more nodes (the search trees are identical). The exceptions can probably be attributed to a lack of robustness in the search strategy used to perform the dominance checks. For example, the results for SBDD with merged dominance checks for 6 groups of 2 for 8 or more weeks strongly suggest that the search got stuck on a particular dominance check which took an inordinately long time compared with other checks for the same or similar instances.

It is perhaps a little disappointing that in no case did SBDD help find a first solution faster — and it was often substantially slower. This can be ascribed to several factors. First, many of the first solutions found were found with little or no backtracking; in such cases, there is little scope for SBDD to prune the search space. Second, early in the search, the fully-explored

	No symmetry					SBDD					SBDD with merged checks				
	First solution		#	All solutions		First solution		#	All solutions		First solution		#	All solutions	
$g{\times}s{\times}w$	Time	Nodes	sols	Time	Nodes	Time	Stats	sols	Time	Stats	Time	Stats	sols	Time	Stats
3×2×2	0.0	2	2	0.0	3	0.0	2/0/0	1	0.0	3/1/1	0.0	2/0/0	1	0.0	3/1/1
3×2×3	0.0	3	8	0.1	15	0.0	3/0/0	2	0.4	9/8/3	0.0	3/0/0	2	0.4	9/6/3
3×2×4	0.0	4	12	0.1	27	0.1	4/0/0	1	0.7	13/11/5	0.1	4/0/0	1	0.7	13/9/5
3×2×5	0.1	4	12	0.3	27	0.1	4/0/0	1	1.1	13/11/5	0.1	4/0/0	1	1.2	13/9/5
3×2×6	-	-	0	0.3	15	-	-/-/-	0	1.4	9/8/2	-	-/-/-	0	1.6	9/6/2
3×3×2	0.0	2	4	0.0	5	0.0	2/0/0	1	0.1	5/3/3	0.0	2/0/0	1	0.1	5/3/3
3×3×3	0.0	3	8	0.1	13	0.0	3/0/0	1	0.2	7/4/4	0.0	3/0/0	1	0.2	7/4/4
3×3×4	0.1	3	8	0.2	13	0.1	3/0/0	1	0.3	7/4/4	0.1	3/0/0	1	0.4	7/4/4
3×3×5	-	-	0	0.2	5	-	-/-/-	0	0.4	5/3/3	-	-/-/-	0	0.4	5/3/3
4×2×2	0.0	3	20	0.1	26	0.0	3/0/0	2	0.5	14/11/9	0.0	3/0/0	2	0.5	14/11/9
4×2×3	0.1	6	1256	5.4	1802	0.1	6/0/0	8	22.3	96/161/63	0.1	6/0/0	8	16.9	96/90/63
4×2×4	0.1	9	36608	264.3	60554	0.1	9/0/0	16	290.3	436/1008/288	0.1	9/0/0	16	151.4	436/427/288
4×2×5	0.2	12	419328	5168.7	876938	0.2	12/0/0	19	1320.4	1027/2861/637	0.2	12/0/0	19	752.4	1027/1015/637
4×2×6	0.3	14	355662	-	945166	0.3	14/0/0	13	2300.0	1511/4436/865	0.3	14/0/0	13	1669.2	1511/1497/865
4×2×7	0.4	15	325456	-	732577	0.4	15/0/0	6	2810.7	1457/4174/840	0.3	15/0/0	6	3400.8	1457/1442/840
4×2×8	-	-	0	-	342016	-	-/-/-	0	1555.5	1167/3561/605	-	-/-/-	0	1358.1	1167/1155/605
4×3×2	0.0	3	48	0.2	61	0.0	3/0/0	1	0.7	17/14/14	0.0	3/0/0	1	0.7	17/14/14
4×3×3	0.1	6	2688	21.1	4909	0.1	6/0/0	4	28.2	66/107/43	0.1	6/0/0	4	14.6	66/60/43
4×3×4	0.2	12	3936	128.3	10861	9.9	12/10/1	3	96.9	95/192/55	7.6	12/6/1	3	49.1	95/89/55
4×3×5	-	-	0	143.4	5581	-	-/-/-	0	47.5	76/156/40	-	-/-/-	0	29.4	76/70/40
4×4×2	0.0	3	216	0.5	244	0.0	3/0/0	1	2.2	36/33/33	0.0	3/0/0	1	2.2	36/33/33
4×4×3	0.1	6	5184	38.2	8020	0.1	6/0/0	2	8.5	47/41/40	0.1	6/0/0	2	8.5	47/41/40
4×4×4	0.2	7	2592	121.6	10612	0.2	7/0/0	1	8.5	49/42/41	0.2	7/0/0	1	8.5	49/42/41
4×4×5	0.3	7	2592	185.6	10612	0.3	7/0/0	1	12.4	49/42/41	0.3	7/0/0	1	12.4	49/42/41
4×4×6	-	-	0	188.6	8020	-	-/-/-	0	21.1	47/41/40	-	-/-/-	0	21.1	47/41/40
5×2×2	0.0	4	408	1.1	506	0.0	4/0/0	2	5.5	51/53/42	0.0	4/0/0	2	4.7	51/47/42
5×2×3	0.1	8	716544	3369.9	937322	0.1	8/0/0	23	1517.9	998/3093/803	0.1	8/0/0	23	693.0	998/990/803
5×2×4	0.2	12	944149	-	1329467	0.2	12/0/0	51	-	1576/6984/1146	0.2	12/0/0	185	-	5251/5239/3777
5×2×5	0.3	16	592746	-	921398	0.3	16/0/0	66	-	729/3071/399	0.3	16/0/0	106	-	2500/2484/1578
5×2×6	0.5	22	386640	-	697476	27.2	22/5/0	30	-	391/1359/175	16.5	22/3/0	74	-	966/947/463
5×2×7	0.8	34	239264	-	537160	504.4	34/41/2	11	-	184/600/68	216.1	34/15/2	26	-	521/502/218
5×2×8	1.3	43	94152	-	382588	2842.4	43/95/2	2	-	81/256/19	1140.8	43/24/2	5	-	184/165/63
5×2×9	1.4	39	193713	-	412702	5468.5	39/66/2	1	-	44/81/5	2410.9	39/20/2	2	-	133/114/51
5×2×10	-	-	0	-	181368	-	-/-/-	0	-	1085/4577/465	-	-/-/-	0	-	2747/2728/1189
5×3×2	0.1	4	4896	15.2	5546	0.1	4/0/0	2	18.3	110/119/101	0.1	4/0/0	2	16.7	110/106/101
5×3×3	0.2	8	1027470	-	1381712	0.2	8/0/0	109	-	691/4572/404	0.1	8/0/0	220	-	4958/4950/3702
5×3×4	0.4	18	347268	-	762070	29.9	18/13/0	101	-	431/2597/101	17.8	18/7/0	356	-	1513/1502/370
5×3×5	0.7	29	61356	-	484260	347.8	29/76/0	14	-	245/1329/21	109.3	29/18/0	47	-	837/826/78
5×3×6	5129.4	248134	288	-	350906	-	-/-/-	0	-	399/2952/36	-	-/-/-	0	-	1636/1625/136
5×3×7	-	-	0	-	266950	-	-/-/-	0	-	696/6805/61	-	-/-/-	0	-	4026/4015/877
5×3×8	-	-	0	-	240935	-	-/-/-	0	-	548/4815/51	-	-/-/-	0	-	2827/2816/444
5×4×2	0.1	4	31104	114.9	35101	0.1	4/0/0	1	17.0	153/149/149	0.1	4/0/0	1	16.9	153/149/149
5×4×3	0.2	7	643357	-	1098949	0.2	7/0/0	33	-	306/1748/167	0.2	7/0/0	40	2502.6	1502/1495/1143
5×4×4	1.1	34	31526	-	248830	118.9	26/38/6	11	-	265/1471/45	110.6	26/18/6	14	-	767/759/137
5×4×5	4.1	107	7104	-	186689	1374.6	94/431/10	2	-	533/3632/53	653.0	94/86/10	7	-	1493/1485/269
5×4×6	-	-	0	-	167168	-	-/-/-	0	-	402/2505/43	-	-/-/-	0	-	1753/1745/322
5×5×2	0.1	4	331776	967.2	352769	0.1	4/0/0	1	52.4	354/350/350	0.1	4/0/0	1	52.2	354/350/350
5×5×3	0.2	7	637258	-	888118	0.2	7/0/0	2	479.7	405/410/393	0.2	7/0/0	2	249.4	405/398/393
5×5×4	1.1	36	36585	-	292770	2277.2	23/26/9	1	2635.0	410/425/394	999.9	23/16/9	1	1304.9	410/403/394
5×5×5	1.4	37	48528	-	242706	-	-/-/-	0	-	24/29/9	3639.3	24/17/9	1	4027.8	412/405/395
5×5×6	1.8	37	33372	-	166907	-	-/-/-	0	-	24/28/9	-	-/-/-	0	-	24/17/9
5×5×7	-	-	0	-	147334	-	-/-/-	0	1448.5	410/425/394	-	-/-/-	0	1038.6	410/403/394
6×2×2	0.1	5	14496	41.3	17730	0.0	5/0/0	4	79.1	149/171/130	0.1	5/0/0	4	41.8	149/144/130
6×2×3	0.2	10	1391216	-	1769513	0.2	10/0/0	55	-	1444/4459/1155	0.1	10/0/0	89	-	2565/2555/2046
6×2×4	0.6	37	871776	-	1165830	38.4	37/28/16	29	-	502/1652/348	29.1	37/23/16	57	-	1554/1540/1160
6×2×5	1.2	60	600837	-	858771	288.2	46/47/18	12	-	201/630/115	160.2	46/32/18	20	-	490/476/322
6×2×6	1.9	83	434205	-	667750	6494.6	55/63/22	1	-	68/86/32	3607.1	55/41/22	1	-	73/59/35
6×2×7	2.6	102	320280	-	535589	-	-/-/-	0	-	60/71/24	-	-/-/-	0	-	63/49/26
6×2×8	3.4	121	251520	-	444991	-	-/-/-	0	-	67/84/26	-	-/-/-	0	-	64/50/26
6×2×9	4.3	136	198612	-	373264	-	-/-/-	0	-	74/96/30	-	-/-/-	0	-	64/50/26
6×2×10	4.9	140	100224	-	265006	-	-/-/-	0	-	84/114/34	-	-/-/-	0	-	64/50/26
6×2×11	5.6	143	168888	-	295012	-	-/-/-	0	-	86/118/34	-	-/-/-	0	-	64/50/26
6×2×12	-	-	0	-	108496	-	-/-/-	0	-	451/1518/258	-	-/-/-	0	-	64/50/26
6×3×2	0.1	5	1283904	4458.3	1422360	0.1	5/0/0	6	683.2	588/695/557	0.1	5/0/0	6	329.8	588/583/557
6×3×3	0.2	10	873706	-	1072996	0.2	10/0/0	27	-	429/1813/324	0.2	10/0/0	180	-	4052/4042/3266
6×3×4	0.4	15	393569	-	618686	0.4	15/0/0	21	-	234/1235/97	0.4	15/0/0	178	-	1684/1669/891
6×3×5	1.8	57	126674	-	348370	1683.6	57/181/0	5	-	138/729/25	374.1	57/38/0	47	-	817/798/231
6×3×6	103.4	2695	24108	-	220336	-	-/-/-	0	-	75/334/0	-	-/-/-	0	-	155/136/0
6×3×7	-	-	0	-	179683	-	-/-/-	0	-	348/3147/6	-	-/-/-	0	-	1715/1696/123
6×3×8	-	-	0	-	163815	-	-/-/-	0	-	279/2402/5	-	-/-/-	0	-	1517/1498/107
6×3×9	-	-	0	-	140030	-	-/-/-	0	-	247/2052/5	-	-/-/-	0	-	1216/1197/80
6×4×2	0.1	5	1744240	-	1855537	0.2	5/0/0	4	2007.9	1117/1427/1100	0.1	5/0/0	4	605.2	1117/1112/1100
6×4×3	0.3	10	711615	-	915943	0.4	10/0/0	48	-	204/581/111	0.4	10/0/0	187	-	575/565/256
6×4×4	0.6	15	176996	-	444825	0.6	15/0/0	27	-	117/327/36	0.6	15/0/0	74	-	357/342/91
6×4×5	2.0	36	18508	-	219288	1461.8	35/42/2	5	-	78/204/17	573.2	35/17/2	15	-	181/163/48
6×4×6	-	-	0	-	152811	-	-/-/-	0	-	226/1120/50	-	-/-/-	0	-	685/668/117
6×4×7	-	-	0	-	121842	-	-/-/-	0	-	223/1405/24	-	-/-/-	0	-	970/953/144
6×4×8	-	-	0	-	110828	-	-/-/-	0	-	194/1088/22	-	-/-/-	0	-	729/712/102
6×5×2	0.2	5	1454008	-	1545729	0.2	5/0/0	1	557.9	1837/1832/1832	0.2	5/0/0	1	541.7	1837/1832/1832
6×5×3	0.4	10	434236	-	723722	0.4	10/0/0	14	-	39/97/4	0.4	10/0/0	46	-	93/83/5
6×5×4	2.1	31	7458	-	117915	1929.3	31/64/0	3	-	68/313/0	602.7	31/19/0	8	-	118/106/0
6×5×5	97.5	1122	26	-	88173	-	-/-/-	0	-	87/468/0	-	-/-/-	0	-	157/145/0
6×5×6	-	-	0	-	77408	-	-/-/-	0	-	75/350/0	-	-/-/-	0	-	261/249/5
6×5×7	-	-	0	-	66229	-	-/-/-	0	-	62/249/0	-	-/-/-	0	-	163/151/0
6×5×8	-	-	0	-	59614	-	-/-/-	0	-	48/182/0	-	-/-/-	0	-	132/120/0
6×6×2	0.1	5	1939744	-	2000614	0.1	5/0/0	1	1967.7	4425/4420/4420	0.1	5/0/0	1	1924.9	4425/4420/4420
6×6×3	0.4	10	506711	-	702671	0.4	10/0/0	7	-	33/31/12	0.3	10/0/0	10	-	50/40/22
6×6×4	-	-	0	-	133476	-	-/-/-	0	-	59/97/19	-	-/-/-	0	-	59/49/19

Table 1. Experimental results

subtrees are quite small, and thus it will often be faster to simply explore a symmetric version of one of these subtrees rather than pay the cost of the dominance check which would show it need not be explored.[4] Third, early in the search, the chance of a dominance check succeeding can be significantly lower than later in the search. This is because early in the search the subtrees already explored only dominate a small fraction of the search space (if they dominated a large fraction of it, the search would likely be nearly complete) and hence the chance of entering a dominated part of the search tree would, in general, be small.

Combined, these last two points mean that early in the search, not only is the expensive dominance check unlikely to succeed, even when it does it is likely to have cost more than it saved. This suggests that, from a cost/benefit point of view, it is not worth doing full dominance checks early in the search. Instead, such checks should become more frequent as the search progresses, as both their chance of success and the expected saving from a success increase.

6 Conclusions and Further Work

We have studied in detail the application of SBDD to the social golfer problem, showing its utility for finding all solutions for medium-sized instances despite the significant cost of applying the dominance checks at every node. We have proposed a refinement to SBDD to reduce the number of dominance checks without sacrificing pruning power, and showed that it was a beneficial approach in practice. We have also provided a detailed description of how the SBDD dominance checks can be implemented for the social golfer problem. Finally, we have looked at the relationship between SBDD and SBDS.

It is clear that more work needs to be done to reduce the cost of using SBDD. One approach which we have proposed is to adapt the frequency of dominance checks such that checks are rarely done early in the search, but increase in frequency as the search progresses and the expected payoff increases.

It would also be nice to extend the scope of the experimental results presented here, by trying different models and exploring more search strategies. In particular, we would like to investigate Meseguer and Torras's symmetry-based search heuristics [5], since we think combining them with some of the approaches presented here could yield synergistic improvements.

7 Acknowledgments

The author would like to thank the many people with whom he has had interesting and stimulating discussions regarding dealing with symmetry and solving the social golfer problem, but in particular Barbara Smith, Mark Wallace, Ian Gent, Torsten Fahle and Meinolf Sellmann.

References

1. Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In *CP'01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, 2001. To appear.
2. Graham Farr, Keith Edwards, and Nigel Hooke. Personal communication.
3. Ian P. Gent and Barbara M. Smith. Symmetry breaking during search in constraint programming. In Werner Horn, editor, *ECAI'2000: Proceedings of the 14th European Conference on Artificial Intelligence*, pages 599–603, 2000.
4. Ian P. Gent, Toby Walsh, and Bart Selman. CSPLib: a problem library for constraints. http://www-users.cs.york.ac.uk/~tw/csplib/.
5. Pedro Meseguer and Carme Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129:133–163, 2001.
6. Stefano Novello. An ECL^{i}PS^e program for the social golfer problem. http://www.icparc.ic.ac.uk/eclipse/examples/golf.ecl.txt.
7. Barbara M. Smith. Reducing symmetry in a combinatorial design problem. In *CPAIOR'01: Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 351–359, April 2001.
8. Mark G. Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe : A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, May 1997.

[4] Though if we wish to ensure only unique solutions are returned, any solutions found after the first still need to be checked.