

The Fully Social Golfer Problem

Warwick Harvey

IC-Parc, Imperial College London
Exhibition Road, London, SW7 2AZ, UK
wh@icparc.ic.ac.uk

Abstract. We present a variant of the Social Golfer Problem where every pair of players must play together at least once instead of at most once. We discuss its modelling and solution, and look at some of the features of the problem that are interesting from a symmetry point of view.

1 Introduction

In November 2002 I received an email from Dave Erb. He and 11 friends were going on a golfing trip, during which they would play 5 rounds of golf, playing in groups of 4 each round. (Yes, people really do go on golf trips, and they really do care about the scheduling.) He and his friends had been sitting round the table discussing how they should divide themselves up into groups when somebody suggested that everybody should play with everybody else at least once. It was agreed that this would be nice, but they were all sure it could not be done. However, Dave started to wonder, and eventually found my email address on the internet.

Ostensibly, this problem (which we will refer to as the Fully Social Golfer Problem) is very similar to the original Social Golfer Problem (problem 10 in [5]): the only difference is that instead of each pair meeting at most once, they must meet at least once. (The two problems are equivalent when the parameters are such that they are feasible instances of the resolvable balanced incomplete block design problem.) But unlike the Social Golfer Problem (and assuming players must meet at least two other players more than once) it is more obvious that all solutions need not be considered equally good, due to differences in the number and kind of repeated pairings (a given player might only meet one other player more than once, or a player's repeated pairings might be distributed among a number of other players). Potential (conflicting) optimisation criteria (assuming a fixed number of rounds) are:

- Minimise the number of occurrences of the most repeated pairing.
- Minimise the number of pairs that meet more than once.
- Minimise the largest number of players in common between any two groups (and then perhaps minimise the number of times this occurs).
- Minimise sequences of pairings repeated across consecutive rounds.

Of course one can also try to simply minimise the number of rounds required for all golfers to meet at least once.

When solving the problem in the first instance, we ignored the issue of optimisation, and just looked for all solutions; optimisation is discussed further in Section 3.

The rest of this paper presents our approach to solving the problem, the solutions found, and some discussion.

2 Solving the problem

2.1 Model

Assume the problem is to find a schedule for g groups of s players for w rounds, written $g-s-w$, and let $N = gs$ be the total number of players. Dave's problem is then $3-4-5$, with 12 players.

We used a set-based model, with each group represented by a set; denote the l th group in round k by $G_{k,l}$; the players are numbered from 1 to gs . The cardinality of each set was constrained to be s :

$$\#G_{k,l} = s \quad \forall k \in \{1 \dots w\}, l \in \{1 \dots g\}$$

Within each round the groups were constrained to be disjoint:

$$G_{k,l} \cap G_{k,m} = \emptyset \quad \forall k \in \{1 \dots w\}, l, m \in \{1 \dots g\}, l \neq m$$

Let $C_{i,j,k,l}$ be the number of players from the pair i and j that appear in group $G_{k,l}$, i.e.:

$$C_{i,j,k,l} = \#\{(i, j) \cap G_{k,l}\} \quad \forall i, j \in \{1 \dots N\}, i < j, k \in \{1 \dots w\}, l \in \{1 \dots g\}$$

We added a redundant constraint corresponding to the fact that i and j will each appear exactly once each round, for a total of two occurrences:

$$\sum_{l \in \{1 \dots g\}} C_{i,j,k,l} = 2 \quad \forall i, j \in \{1 \dots N\}, i < j, k \in \{1 \dots w\} \quad (\text{redundant}) \quad (1)$$

Let $P_{i,j,k}$ be the number of times players i and j play together in round k , i.e.:

$$P_{i,j,k} = \#\{G_{k,l} \mid C_{i,j,k,l} = 2, l \in \{1 \dots g\}\} \quad \forall i, j \in \{1 \dots N\}, i < j, k \in \{1 \dots w\}$$

Of course a given pair can play together at most once in a round:

$$0 \leq P_{i,j,k} \leq 1 \quad \forall i, j \in \{1 \dots N\}, i < j, k \in \{1 \dots w\} \quad (\text{redundant}) \quad (2)$$

Let $P_{i,j}$ be the number of times players i and j play together in total, i.e.:

$$P_{i,j} = \sum_{k \in \{1 \dots w\}} P_{i,j,k} \quad \forall i, j \in \{1 \dots N\}, i < j$$

In order to ensure that each pair does meet at least once:

$$P_{i,j} \geq 1 \quad \forall i, j \in \{1 \dots N\}, i < j$$

And one final redundant constraint, based on the number of pairings each player is involved in ($P_{i,j}$ is assumed to be a synonym for $P_{j,i}$ if $i > j$):

$$\sum_{j \in \{1 \dots N\}, j \neq i} P_{i,j} = w(s-1) \quad \forall i \in \{1 \dots N\} \quad (\text{redundant}) \quad (3)$$

Due to the symmetries of the problem the first round can be fixed arbitrarily:

$$G_{1,l} = \{(l-1)s + 1 \dots ls\} \quad \forall l \in \{1 \dots g\}$$

Player 1 can also be assigned to the first group in the remaining rounds:

$$1 \in G_{k,1} \quad \forall k \in \{2 \dots w\}$$

Note that while in the standard Social Golfer Problem player 2 can be assigned to the second group in the remaining rounds, we cannot automatically do this here. This is because the 1-2 pairing is allowed to be repeated, meaning player 2 may need to be assigned to the first group in some or all of the subsequent rounds. One could still exclude player 2 from groups three and above, exclude player 3 from groups four and above, etc., but we did not try this. Note also that, in general when using a dominance detection-based symmetry breaking approach [1, 2] (see Section 2.2), the impact of this kind of initial labelling typically has more to do with changing the labelling order of the search than it does with not having to consider the alternative choices. This is because if we were to backtrack to any of these choices and try the alternative, a dominance check would quite quickly show that the state is indeed dominated and can be pruned. Since the number of such dominance checks avoided is typically small in comparison to the number performed in during a complete search (and they would only be needed during a complete search), the overall impact can also be expected to be quite small.

2.2 Symmetry breaking

Due to the large amount of symmetry inherent in the problem, we employed a symmetry breaking technique. Specifically, we used the SBDD implementation described in [4], updated to work with set variables rather than integer variables, and augmented with the symmetry expression interface described in [6]. The extension to support sets was a straightforward one, exploiting the fact that the ECLⁱPS^e set library we used represents a set as a vector of boolean variables, each boolean indicating whether a particular element is in the set or not. This meant a wrapper for the SBDD implementation could be written which efficiently translates the user's set-based model and set-based search decisions into an integer-based model and integer-based search decisions, suitable for passing to the existing SBDD implementation.

The symmetries used were:

- consistent renaming of the players;
- reordering of the groups in a round; and
- reordering of the rounds.

Note that the symmetry among the players in a group is avoided through the choice of a set-based model.

2.3 Search heuristics

We experimented with two different search heuristics. The first was that used by Stefano Novello in his original program for solving the Social Golfer Problem [7]. This approach takes each player in turn and goes through each round in turn, assigning the player to one of the groups in the round, trying them from left to right. The second search heuristic was simply to label each round in turn, from left to right. A comparison of the results of these two heuristics appears in Section 4.1.

3 Solutions

The (unique) solutions to the problem are given in Table 1. The figures to the right of each solution show the repeated pairings of the solution; that is, there is one line between a pair of players for each time they meet beyond the required minimum of once. These figures illustrate (some of) the different structural properties of the solutions.

Solution 1 is equivalent to pairing up the players and finding a solution for 3 groups of 2 playing for 5 rounds. Such a configuration corresponds to a trivial resolvable balanced incomplete block design with a unique solution, the $(6, 2, 1)$ -design.

Solution 9 is of interest because it is the only solution which does not have at least one pair of players playing together in every round, and seems most likely to be the solution of choice for a schedule (unless, say, some pair of players consider themselves inseparable or something).

(Note that there are no solutions if we restrict ourselves to just four rounds. This is perhaps not immediately obvious: there are 66 pairings we need to schedule, and a four-round schedule yields 72 pairings; this means that any feasible schedule must have exactly 6 repeated pairings. Since there are four players to a group and only three groups to a round, by the pigeon-hole principle each group from, say, the first round must have at least two members appearing together in at least one group in each other round. Since there are three groups in the first round and three other rounds, this immediately means there must be at least 9 repeated pairings: too many to allow all the other pairings we need in the schedule.)

When setting up the symmetry breaking, we assumed that reordering the rounds of a schedule would yield an equivalent schedule. While it is true that reordering the rounds of any solution yields another solution, it may be that the

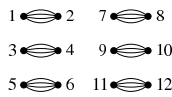
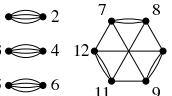
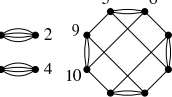
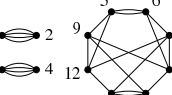
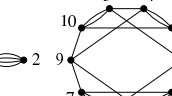
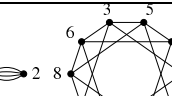
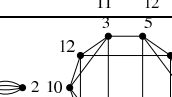
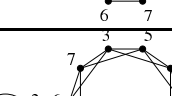
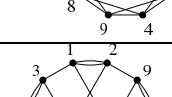
Solution 1	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 6\}, \{3, 4, 9, 10\}, \{7, 8, 11, 12\}\}$ $\{\{1, 2, 11, 12\}, \{3, 4, 5, 6\}, \{7, 8, 9, 10\}\}$ $\{\{1, 2, 7, 8\}, \{3, 4, 11, 12\}, \{5, 6, 9, 10\}\}$ $\{\{1, 2, 9, 10\}, \{3, 4, 7, 8\}, \{5, 6, 11, 12\}\}$	
Solution 2	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 6\}, \{3, 4, 9, 10\}, \{7, 8, 11, 12\}\}$ $\{\{1, 2, 11, 12\}, \{3, 4, 5, 6\}, \{7, 8, 9, 10\}\}$ $\{\{1, 2, 7, 9\}, \{3, 4, 8, 11\}, \{5, 6, 10, 12\}\}$ $\{\{1, 2, 8, 10\}, \{3, 4, 7, 12\}, \{5, 6, 9, 11\}\}$	
Solution 3	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 4, 6, 10\}, \{7, 8, 11, 12\}\}$ $\{\{1, 2, 6, 11\}, \{3, 4, 5, 12\}, \{7, 8, 9, 10\}\}$ $\{\{1, 2, 7, 10\}, \{3, 4, 8, 9\}, \{5, 6, 11, 12\}\}$ $\{\{1, 2, 8, 12\}, \{3, 4, 7, 11\}, \{5, 6, 9, 10\}\}$	
Solution 4	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 4, 6, 10\}, \{7, 8, 11, 12\}\}$ $\{\{1, 2, 6, 11\}, \{3, 4, 5, 12\}, \{7, 8, 9, 10\}\}$ $\{\{1, 2, 7, 12\}, \{3, 4, 8, 9\}, \{5, 6, 10, 11\}\}$ $\{\{1, 2, 8, 10\}, \{3, 4, 7, 11\}, \{5, 6, 9, 12\}\}$	
Solution 5	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 4, 6, 10\}, \{7, 8, 11, 12\}\}$ $\{\{1, 2, 6, 10\}, \{3, 5, 11, 12\}, \{4, 7, 8, 9\}\}$ $\{\{1, 2, 7, 11\}, \{3, 5, 8, 10\}, \{4, 6, 9, 12\}\}$ $\{\{1, 2, 8, 12\}, \{3, 7, 9, 10\}, \{4, 5, 6, 11\}\}$	
Solution 6	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 6, 7, 10\}, \{4, 8, 11, 12\}\}$ $\{\{1, 2, 6, 11\}, \{3, 5, 8, 12\}, \{4, 7, 9, 10\}\}$ $\{\{1, 2, 7, 12\}, \{3, 5, 10, 11\}, \{4, 6, 8, 9\}\}$ $\{\{1, 2, 8, 10\}, \{3, 6, 9, 12\}, \{4, 5, 7, 11\}\}$	
Solution 7	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 6, 7, 10\}, \{4, 8, 11, 12\}\}$ $\{\{1, 2, 6, 11\}, \{3, 5, 10, 12\}, \{4, 7, 8, 9\}\}$ $\{\{1, 2, 7, 12\}, \{3, 5, 8, 11\}, \{4, 6, 9, 10\}\}$ $\{\{1, 2, 8, 10\}, \{3, 6, 9, 12\}, \{4, 5, 7, 11\}\}$	
Solution 8	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 6, 7, 10\}, \{4, 8, 11, 12\}\}$ $\{\{1, 2, 6, 11\}, \{3, 5, 10, 12\}, \{4, 7, 8, 9\}\}$ $\{\{1, 2, 8, 10\}, \{3, 5, 7, 11\}, \{4, 6, 9, 12\}\}$ $\{\{1, 2, 7, 12\}, \{3, 6, 8, 9\}, \{4, 5, 10, 11\}\}$	
Solution 9	$\{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}\}$ $\{\{1, 2, 5, 9\}, \{3, 6, 10, 11\}, \{4, 7, 8, 12\}\}$ $\{\{1, 2, 6, 12\}, \{3, 5, 7, 10\}, \{4, 8, 9, 11\}\}$ $\{\{1, 3, 8, 10\}, \{2, 7, 9, 11\}, \{4, 5, 6, 12\}\}$ $\{\{1, 5, 7, 11\}, \{2, 4, 8, 10\}, \{3, 6, 9, 12\}\}$	

Table 1. All unique solutions to the 3-4-5 instance of the Fully Social Golfer Problem

result is not considered equivalent by the players. This is because the rounds form a sequence from the players' point of view, and changing the sequence may change the pattern of repeated pairings from a player's point of view. Taking Solution 9 as an example, consider the pairings that occur three times in the schedule. All of these pairings appear in the third round of the schedule as printed, distinguishing that round from the others (which each contain only half of these pairings). Also, it is impossible to avoid having at least one pair play together for three consecutive rounds, but the number of such pairs can vary from one (when round three is moved first or last) to three (when round three is in the middle; for the schedule in Table 1 the pairs playing three consecutive rounds together are 1 – 2, 3 – 10 and 6 – 12). On the other hand, for Solution 1 the order of the rounds appears to be irrelevant, since the repeated pairings occur in every round.

As can be seen from the above, optimisation criteria based on properties such as the sequence of the rounds may not respect what would otherwise be the symmetries of the problem. There seem to be several ways of handling this. One that works just fine when the number of solutions is small, as is the case here, and particularly when the optimisation criteria are a bit fuzzy or there are trade-offs to be made, is just to find all the solutions ignoring the optimisation criteria and then examine their properties to choose the one that looks best, possibly rearranging it to suit.

If there are too many solutions to examine by hand in this way, but the optimisation criteria are well-defined, one could add a second phase to solving the problem, which looks at each solution returned, and searches for the symmetric variant(s) of that solution that optimise the desired property or properties. This allows one to fully exploit the symmetry of the problem, but does not allow the objective function to prune the search.

Another alternative is to include the objective function in the model, and perform the search with a reduced number of symmetries. This allows the objective function to prune the search, but may result in redundant exploration of parts of the search space that would otherwise be considered symmetric.

Finally, one could try to get the best of both worlds by keeping all the symmetries but modifying the objective function so that it yields not the objective value of the current state but rather the objective value of the best possible of all the symmetric versions of the current state, with any final solutions found similarly transformed into their best possible symmetric version before being returned. This is somewhat related to the work presented in [3]. The main drawback with this approach is that it seems likely to be complicated to implement, and may be too slow and/or prune too weakly to be useful.

An exploration of these alternatives would be interesting, but is beyond the scope of this paper. (Note that the same issue comes up when there are, say, just a few constraints which break some or all of the symmetries of the problem; this can be handled using more or less the same techniques.)

Heuristic	Solution	Time			Choices	Dominance checks		
		GAP	ECL ⁱ PS ^e	Total		Success	Fail	Delete
player first	first	5.09	1.11	6.20	33	0	4	2
round first	first	9484.39	534.38	10018.77	23714	5482	20880	8016
player first	all	529.27	264.64	793.90	7468	1263	5676	4907
round first	all	24034.66	729.45	24764.12	31802	8441	25791	13302

Table 2. Comparison of search heuristics

4 Computational results

The implementation of the constraint program to solve the problem was done using ECLⁱPS^e. All experiments were run on a 933MHz Intel Pentium III machine with 512MB of RAM, running Red Hat Linux, using ECLⁱPS^e 5.6#36 and GAP 4r3fix5. In each case the times presented are the average of several runs, but the exact times given should of course be taken with a grain of salt.

4.1 Search heuristics

As mentioned in Section 2.3, we consider two search heuristics, what we will call “player first” and “round first”. The result of comparing these, for the case where all the redundant constraints from Section 2.1 are included but none of the possible initial arbitrary assignments of players to groups are done, is shown in Table 2, for finding the first solution and for finding all solutions. In each case, we list the time spent in GAP (doing dominance checks), the time spent in ECLⁱPS^e (solving the CSP and providing GAP with the information needed for the dominance checks) and the total time, all in seconds. We also list the number of times a choice¹ was made during search, the number of dominance checks that were successful, the number of dominance checks that failed without providing any useful information, and the number of dominance checks that failed but indicated that certain elements could be removed from particular domains (since they would lead to a dominated state).

Clearly, the player-first heuristic is superior on all counts. Interestingly, not only does the round-first heuristic have a substantially larger search tree for all solutions than player-first, but the average dominance check is also more than an order of magnitude slower, resulting in a huge blow-out in the total time taken.

It would appear from this and results from other problems we have looked at that the dominance checker from [4] takes substantially longer than expected on problems with this kind of wreath product symmetry when labelled in round order. This is worth investigating, in order to see whether it can be overcome

¹ We define a choice to be any decision that might be backtracked to later so that the alternative option can be tried. Note that this is not quite the same as the number of backtracks unless the entire search space is explored, since any intermediate point in the search will typically have a number of outstanding choices that have yet to have their alternatives tried.

Constraints	Solution	Time			Choices	Dominance checks		
		GAP	ECL ^{PS} ^c	Total		Success	Fail	Delete
none	first	3642.92	929.52	4572.44	32035	1253	34119	14011
1	first	2402.88	1011.62	3414.50	22538	1340	18883	12203
2	first	3642.65	900.06	4542.71	32035	1253	34119	14011
1,2	first	2401.40	981.58	3382.98	22538	1340	18883	12203
3	first	6.33	1.16	7.49	44	1	12	2
1,3	first	5.07	1.19	6.26	33	0	4	2
2,3	first	6.33	1.12	7.45	44	1	12	2
1,2,3	first	5.09	1.11	6.20	33	0	4	2
none	all	19592.66	5134.14	24726.81	189020	11259	201852	78219
1	all	12988.55	5552.33	18540.88	135116	12818	111328	68810
2	all	19598.22	4949.24	24547.46	189020	11259	201852	78219
1,2	all	12956.25	5362.08	18318.34	135116	12818	111328	68810
3	all	928.16	285.94	1214.10	11517	993	12906	6159
1,3	all	528.24	279.09	807.33	7468	1263	5676	4907
2,3	all	927.29	275.10	1202.39	11517	993	12906	6159
1,2,3	all	529.27	264.64	793.90	7468	1263	5676	4907

Table 3. Effect of redundant constraints for player-first labelling

by modifying the dominance check algorithm, possibly exploiting the structure of the symmetry group.

We also tried finding all solutions without employing any symmetry breaking. It had consumed about 9 weeks' worth of CPU time without finishing when the machine it was running on had a disk fail, effectively ending the experiment.

4.2 Redundant constraints

As noted in Section 2.1, several of the constraints added are redundant (namely, (1), (2) and (3)). Table 3 shows the effect of including or leaving out various combinations of the redundant constraints, for the player-first search heuristic with no initial arbitrary assignments of players to groups.

It is clear that (3) is crucial: it makes more than an order of magnitude difference across the board. This is because without this constraint, the solver often fails to detect when a player is already involved in too many repeated pairings, such that it is not possible for them to play with all the remaining players in the rest of the schedule, and this results in a lot of useless search.

(1) provides modest improvements, but (2) does not help at all.

The results for using the round-first search heuristic were similar, though of course much slower, and we didn't run to completion any experiment without (3) because it would have taken too long. Perhaps the only thing worth noting is that for this search heuristic, (2) did actually provide a very small reduction in the size of the search tree.

Initial labelling	Solution	Time			Choices	Dominance checks		
		GAP	ECL'PS ^e	Total		Success	Fail	Delete
none	first	5.09	1.11	6.20	33	0	4	2
p!	first	5.07	1.15	6.22	28	0	4	2
pr	first	5.11	1.15	6.26	33	0	4	1
pr!	first	5.09	1.15	6.23	21	0	4	1
r	first	5.03	1.17	6.20	33	0	4	1
rp!	first	5.09	1.13	6.21	21	0	4	1
none	all	529.27	264.64	793.90	7468	1263	5676	4907
p!	all	528.58	264.38	792.97	7461	1261	5676	4903
pr	all	4319.50	217.78	4537.28	5510	724	3547	3209
pr!	all	4318.55	216.54	4535.09	5496	721	3547	3199
r	all	4325.06	216.97	4542.03	5515	730	3547	3209
rp!	all	4316.52	216.16	4532.68	5496	721	3547	3199

Table 4. Effect of initial labelling for player-first labelling

4.3 Initial labelling

In Section 2.1, we argued that if making some initial assignments because they are guaranteed not to exclude any unique solutions had any significant effect, then it would be due to the perturbation of the search heuristic rather than any saving achieved from not having to look at the alternatives to the choices. In this section we back that argument with some computational results.

Table 4 shows the effect of labelling the first round (r) and/or player (p), before continuing with the player-first search heuristic, with this labelling done either as normal choices (to be backtracked later) or committed choices (never to be backtracked, indicated by a !). So “p!” commits to labelling the first player, which means that the search tree is exactly the same as no initial labelling, other than that these first choices are never reconsidered. “pr” labels the first player and then the first round, while “rp” labels the first round and then the first player (since we are labelling the rest in player order, “r” is the same as “rp” if no commit occurs).

(All redundant constraints were included.)

For the first solution, there is little to distinguish between the alternatives; the variance in the number of choices just reflects those choices that were committed to. For finding all solutions, the picture is much clearer. Committing to the initial choices has only a relatively minor effect, while changing the shape of the search tree by labelling the first round (either before or after the first player) has two significant effects: a marked reduction in the size of the search space, and an order of magnitude increase in the average time to perform a dominance check (presumably the result of having some round-wise labelling).

Table 5 shows the same thing for the round-first search heuristic: committing to the initial choices has only a minor impact, and the only significant change is when the first player is part of the initial labelling phase.

Initial labelling	Solution	Time			Choices	Dominance checks		
		GAP	ECL'PS ^e	Total		Success	Fail	Delete
none	first	9484.39	534.38	10018.77	23714	5482	20880	8016
r!	first	9483.72	533.37	10017.09	23706	5482	20880	8016
rp	first	6598.50	408.12	7006.62	14311	1942	10056	5255
rp!	first	6605.06	407.42	7012.48	14299	1942	10056	5255
p	first	6601.84	408.07	7009.92	14311	1942	10056	5255
pr!	first	6595.17	407.60	7002.77	14299	1942	10056	5255
none	all	24034.66	729.45	24764.12	31802	8441	25791	13302
r!	all	24041.10	727.72	24768.81	31787	8432	25791	13296
rp	all	17990.96	831.07	18822.02	28525	4544	19849	10665
rp!	all	18009.87	828.84	18838.72	28506	4535	19849	10655
p	all	17993.90	830.32	18824.22	28520	4538	19849	10665
pr!	all	17978.35	828.87	18807.22	28506	4535	19849	10655

Table 5. Effect of initial labelling for round-first labelling

5 Conclusions

We have presented an interesting variant of the Social Golfer Problem based on players being required to meet at least once instead of at most once. In particular, we have presented all the unique solutions of the 3-4-5 instance, which was of interest for a social golf tour. We have also examined a number of aspects of the modelling and solution of the problem, including the effect of redundant constraints, initial labelling (in particular its interaction with SBDD-based symmetry breaking) and search heuristics (particularly with regard to the effect of certain labelling strategies on the time taken to perform dominance checks using the method presented in [4]).

Acknowledgements

The author would like to thank his colleagues at the Universities of St Andrews and Huddersfield, and Meinolf Sellmann, for all their interesting discussions on symmetry and the Social Golfer Problem, and other support. The author would also like to thank the reviewers for their helpful comments.

This work was supported by EPSRC grant GR/S30658/01.

References

1. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *CP'01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 93–107, 2001.
2. F. Focacci and M. Milano. Global cut framework for removing symmetries. In T. Walsh, editor, *CP'01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.

3. F. Focacci and P. Shaw. Pruning sub-optimal search branches using local search. In *CPAIOR'02: Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 181–189, 2002.
4. I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In *CP'03: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, 2003. To appear.
5. I. P. Gent, T. Walsh, and B. Selman. CSPLib: a problem library for constraints. <http://csplib.org/>.
6. W. Harvey, T. Kelsey, and K. Petrie. Symmetry group expression for CSPs. In B. M. Smith and I. P. Gent, editors, *Proceedings of SymCon'03: The Third International Workshop on Symmetry in Constraint Satisfaction Problems*, 2003. To appear.
7. S. Novello. An ECLⁱPS^e program for the social golfer problem. <http://www.icparc.ic.ac.uk/eclipse/examples/golf.ecl.txt>.