**Noname manuscript No.**
(will be inserted by the editor)

# An Effective Greedy Heuristic for the Social Golfer Problem

**Markus Triska · Nysret Musliu**

Received: date / Accepted: date

**Abstract** The Social Golfer Problem (SGP) is a combinatorial optimization problem that exhibits a lot of symmetry and has recently attracted significant attention. In this paper, we present a new greedy heuristic for the SGP, based on the intuitive concept of *freedom* among players. We use this heuristic in a complete backtracking search, and match the best current results of constraint solvers for several SGP instances with a much simpler method. We then use the main idea of the heuristic to construct initial configurations for a metaheuristic approach, and show that this significantly improves results obtained by local search alone. In particular, our method is the first metaheuristic technique that can solve the original problem instance optimally. We show that our approach is also highly competitive with other metaheuristic and constraint-based methods on many other benchmark instances from the literature.

**Keywords** sports scheduling · combinatorial optimization · design theory · finite geometry

## 1 Introduction

The *Social Golfer Problem* (SGP) is a combinatorial optimization problem derived from a question that was posted to `sci.op-research` in May 1998:

The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

Markus Triska (✉)
Database and Artificial Intelligence Group
Vienna University of Technology
triska@dbai.tuwien.ac.at

Nysret Musliu
Database and Artificial Intelligence Group
Vienna University of Technology
musliu@dbai.tuwien.ac.at

> 32 golfers play golf once a week, and always in groups of 4. For how many weeks can they play such that no two players play together more than once in the same group?

The problem is readily generalized to the following decision problem: Is it possible to schedule $n = g \times p$ golfers in $g$ groups of $p$ players for $w$ weeks such that no two golfers play in the same group more than once? An *instance* of the SGP is then denoted by a triple $g{-}p{-}w$ of natural numbers. In practice, one is interested in the corresponding optimization problem that asks for the maximum number $w^*$ of weeks that admits a solution for given $g$ and $p$, since such a solution also implies solutions for all instances $g{-}p{-}w$ with $w \leq w^*$.

Clearly, not all combinations of parameters admit a solution, and upper bounds are easy to determine. For example, in the original problem, $8{-}4{-}w$, $w$ can be no more than 10: Suppose $w \geq 11$, and observe the schedule of an arbitrary but fixed player $\alpha$: Each week, $\alpha$ plays in a group with 3 (distinct) other players. To play 11 weeks, $\alpha$ would have to partner $3 \cdot 11 > 32$ players.

The SGP exhibits many symmetries: Weeks, groups within weeks, and players within groups, can all be ordered arbitrarily. In addition, players can be assigned arbitrary names. Due to its highly constrained and symmetric nature, the SGP has attracted much attention from the constraint programming community and has led to the development of powerful but complex dynamic symmetry breaking schemes (Barnier and Brisset 2005; Petrie et al. 2004). The SGP is included as problem number 10 in CSPLib, a benchmark library for constraints (Gent and Walsh 1999). No constraint solver or metaheuristic approach was so far able to solve the $8{-}4{-}10$ instance, although a solution is known to exist. In addition to being a hard and interesting benchmark for constraint solvers, the SGP and closely related problems arise in many practical applications such as encoding, encryption and covering problems (Douglas R. Stinson 1994; Gordon et al. 1995; Hsiao et al. 1970).

## 2 Related Work

Research on a problem that is very closely related to the SGP actually dates back to Euler, who considered an instance of the SGP in a different context: Euler asked whether two orthogonal Latin squares of order 6 exist, which has become known as "Euler's Officer Problem" (Colbourn and Dinitz 1996). In terms of the SGP, this corresponds to solving the $6{-}6{-}4$ instance, which is now known to be impossible. As another special case of the SGP, the $5{-}3{-}7$ instance also has a long history and is known as Kirkman's schoolgirl problem (Barnier and Brisset 2005).

In general, the task of finding $w$ mutually orthogonal Latin squares (MOLS) of order $q$ is equivalent to solving the SGP instance $q{-}q{-}(w+2)$. MOLS play an important rôle in the design of statistical experiments, coding theory and cryptography, and several (in)existence results and construction methods for specific instances are known from *design theory*, a branch of discrete mathematics. Harvey and Winterer have compared and successfully applied several of these interesting techniques to the SGP (Harvey and Winterer 2005).

The computational complexity of the SGP is currently unknown. Some instances are easily solved using construction methods from design theory, but such methods are typically restricted to certain families of instances. However, from a result derived by

Colbourn (Colbourn 1984), one can show that the *completion problem* of the SGP, i.e., deciding whether a partially filled schedule can be completed to conflict-free one, is NP-complete.

Metaheuristic approaches towards the SGP include local search with tabu-lists (Dotú and Hentenryck 2005) and an evolutionary approach (Cotta et al. 2006). Dotú and Hentenryck also use a constructive heuristic, but instead of a greedy heuristic, they use a deterministic construction method from design theory that is known to trivially solve instances of the form $p{-}p{-}(p+1)$ when $p$ is prime. However, the construction method is not easy to randomize meaningfully.

Other approaches include a SAT encoding (Gent and Lynce 2005, Triska and Musliu 2010), which so far is not competitive with other methods, and various constraint-based formulations including sophisticated dynamic symmetry breaking techniques (Barnier and Brisset 2005; Fahle et al. 2001).

For the original instance, state-of-the-art constraint solvers can currently generate solutions for no more than 9 weeks, using either set-based formulations or by posting more constraints than strictly necessary to break symmetries. Whether there is a solution for 10 weeks was an open question that was answered – six years after the problem was posed – in the affirmative by Aguado (Aguado 2004) who constructed an explicit solution based on a result by Colbourn (Colbourn 1999), derived using a combination of design-theoretic techniques, backtracking search and instance-specific considerations. Unfortunately, his approach does not generalize to other instances.

## 3 Modeling the SGP

Assuming the correct group sizes, there are essentially only two constraints in the SGP:

1. Each player plays exactly once each week.
2. Each pair of golfers can play in the same group at most once.

These constraints can be enforced using many different formulations of the problem, which is one of the reasons it is so interesting. On the most abstract level, all constraints can be expressed using a *set-based* formulation, in which the sets $G_{ij}$ denote groups of golfers that play in week $i$. The constraints are then as follows, written as first-order formulae in the language of set theory, with the players being represented by the numbers $1, \ldots, n$:

$$G_{ij} \subseteq \{1, \ldots, n\} \wedge |G_{ij}| = p \qquad \text{for all } 1 \leq i \leq w,\ 1 \leq j \leq g \tag{1}$$

$$\bigcup_{1 \leq j \leq g} G_{ij} = \{1, \ldots, n\} \qquad \text{for all } 1 \leq i \leq w \tag{2}$$

$$|G_{ij} \cap G_{i'j'}| \leq 1 \qquad \text{for all } 1 \leq i < i' \leq w,\ 1 \leq j, j' \leq g \tag{3}$$

In most actual implementations, the abstract concept of sets is simulated by the use of variables $G_{ijk}$, $1 \leq G_{ijk} \leq n$, that denote which player plays in week $i$, group $j$ and position $k$. The constraints are then either expressed using built-in primitives provided by solvers or encoded manually. This is also the model we chose. We call the set of variables $G_{ijk}$ and their respective values $v(G_{ijk})$ a *configuration*. There are several interesting observations to be made about this model:

- The model admits many identical solutions due to its inherent symmetries. For example, the order of groups or of players within groups does not distinguish actual solutions, but leads to distinct solutions in this model.
- Most of the symmetries can in fact be *removed* by imposing additional requirements on solutions. For example, players within groups could be required to occur in ascending order (Barnier and Brisset 2005).
- There is one kind of symmetry that cannot be removed statically in this formulation: Players can be renumbered arbitrarily.
- While removing symmetries can dramatically reduce computation time in constraint-based approaches, it can make metaheuristic approaches significantly slower, since it actually removes solutions. Therefore, we do not break any symmetries in our approach, which also has a local search component.

We remark that this is not the only sensible way to encode the problem, and in fact there is a model that allows to statically break the symmetry arising from player permutations: Let $M_{(w \cdot g),n}$ denote a Boolean $(w \cdot g) \times n$ matrix. Each column corresponds to a player, and each row corresponds to one group. The Boolean value at position $(i, j)$ denotes whether player $j$ plays in group $i$. The necessary constraints on $M$ are quite obvious. The symmetry arising from player permutations can now be broken by imposing a lexicographic "less than" constraint on columns. Similarly, the symmetry among groups can be broken by imposing a lexicographic "less than" constraint on the rows corresponding to the groups of each week. However, a drawback of this model is that it uses $g$ times more variables than the model we use, which we also consider more natural.

## 4 Freedom of Sets of Players

We now introduce the concept of *freedom* of sets of players. Let $C$ be a partial configuration. For an arbitrary player $x$, we denote with $P_C(x)$ the *potential partner-set* of $x$ with respect to $C$, i.e., the set of players that $x$ can still *partner* in any group, assuming $C$ as given. In other words, $P_C(x)$ is the set of all players, minus the players that $x$ has already partnered in any group of $C$. For any set $S$ of players, we denote with $\varphi_C(S)$ the *freedom* of $S$ with respect to $C$, and define it as the cardinality of the intersection of the potential partner-sets of all players in $S$, i.e.:

$$\varphi_C(S) = \left| \bigcap_{x \in S} P_C(x) \right| \tag{4}$$

Informally, the freedom of a set of players denotes how many players they can still "partner together". As an example, consider the configuration $C$ of Fig. 1, where players 0 and 31 are highlighted in each week. These two players partner complementary subsets of remaining players, and the freedom of the player set $\{0, 31\}$ is:

$$\varphi_C(\{0, 31\}) = |\{16, \ldots, 31\} \cap \{0, \ldots, 15\}| = |\emptyset| = 0$$

As a consequence, if we wanted to extend the schedule by another week, we know that player 0 and 31 cannot occur together in the same group, because there is no other player that can still play with both of them. In fact, the freedom of *all* pairs of players that have not yet played together in any group in $C$ is 0, and this configuration

| | Week 1 | | | | Week 2 | | | | Week 3 | | | | Week 4 | | | | Week 5 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group 1 | 0 | 1 | 2 | 3 | 0 | 4 | 8 | 12 | 0 | 5 | 10 | 15 | 0 | 6 | 11 | 13 | 0 | 7 | 9 | 14 |
| Group 2 | 4 | 5 | 6 | 7 | 1 | 5 | 9 | 13 | 1 | 4 | 11 | 14 | 1 | 7 | 10 | 12 | 1 | 6 | 8 | 15 |
| Group 3 | 8 | 9 | 10 | 11 | 2 | 6 | 10 | 14 | 2 | 7 | 8 | 13 | 2 | 4 | 9 | 15 | 2 | 5 | 11 | 12 |
| Group 4 | 12 | 13 | 14 | 15 | 3 | 7 | 11 | 15 | 3 | 6 | 9 | 12 | 3 | 5 | 8 | 14 | 3 | 4 | 10 | 13 |
| Group 5 | 16 | 17 | 18 | 19 | 16 | 20 | 24 | 28 | 16 | 21 | 26 | 31 | 16 | 22 | 27 | 29 | 16 | 23 | 25 | 30 |
| Group 6 | 20 | 21 | 22 | 23 | 17 | 21 | 25 | 29 | 17 | 20 | 27 | 30 | 17 | 23 | 26 | 28 | 17 | 22 | 24 | 31 |
| Group 7 | 24 | 25 | 26 | 27 | 18 | 22 | 26 | 30 | 18 | 23 | 24 | 29 | 18 | 20 | 25 | 31 | 18 | 21 | 27 | 28 |
| Group 8 | 28 | 29 | 30 | 31 | 19 | 23 | 27 | 31 | 19 | 22 | 25 | 28 | 19 | 21 | 24 | 30 | 19 | 20 | 26 | 29 |

**Fig. 1** A solution for the 8−4−5 instance, with players 0 and 31 highlighted in each week

can therefore not even be extended by a single further *group* if the shown weeks are considered fixed.

In the following, we describe how to use this measure to improve two different methods: Complete backtracking and local search.

## 5 A Greedy Heuristic for Complete Backtracking

We now describe a greedy heuristic that can be used to guide a complete backtracking search for the SGP.

Let us first suppose that $p$ is even. Then the task of scheduling the players into groups in each week can also be seen as scheduling *pairs* of players into groups. Thus, we visit the weeks one after another, and in each week, we traverse the groups in their natural order. For each pair of adjacent positions in a group, we need to select a pair of players still remaining to be scheduled in the current week. Here, we use the degree of *freedom* to guide the selection, and select a pair having *minimal* freedom with respect to the current partial configuration. The intention behind this choice is that if a pair of players is close to running out of potential partners, then they should be scheduled in the same group while that is still possible at all. This reasoning is analogous to the well-known labeling strategy "first-fail" in constraint satisfaction problems.

If a group is encountered that cannot be completed, or a conflict is found, *backtracking* occurs: We undo the most recent choice of players, and select a pair with next larger degree of freedom instead.

If $p$ is odd, there are several options. A simple solution is to schedule pairs of players for each group as far as possible, and then to fill the remaining position with any player that is compatible with all other players scheduled in this group.

Another approach is to generalise the heuristic to triples and larger sets of players. Here, a trade-off must be reached between accurate assessment of a scheduling choice and computational tractability.

We have implemented the described backtracking method in such a way that a *pattern* can be specified to denote in which way each individual group is considered. For example, the pattern 3-2-2-1 can be applied to $g-8-w$ instances and means that for each group, we proceed as follows: First a *triple* of players with minimal freedom is scheduled; on the remaining places, two *pairs* are placed next (each chosen according to minimal freedom), and any player that can still play with all others in this group is put in the final position.

Table 1 shows results for selected instances and patterns with this method, tested on an Apple MacBook with a 2.16 GHz Intel Core 2 Duo CPU and 1GB RAM, using a

**Table 1** Complete backtracking with greedy heuristic on selected instances

| instance | pattern | time | instance | pattern | time |
|---|---|---|---|---|---|
| 5−3−7 | 3 | 0.51s | 6−6−3 | 3-3 | 0.29s |
| 8−4−9 | 2-2 | 0.08s | 6−6−3 | 4-2 | 1.54s |
| 8−4−9 | 4 | 1.58s | 9−9−3 | 3-3-3 | 14.4s |

Prolog implementation of the algorithm. We benefit from arbitrary precision arithmetic to represent sets of players as bit vectors. This lets us efficiently intersect sets by using fast bitwise operations. Determining a set's cardinality is thus also very efficient. For each instance, we match the best number of weeks of current constraint solvers (Harvey 2002). In particular, we solve Kirkman's schoolgirl problem optimally, and we solve the original SGP for 9 weeks with pattern 2-2 (i.e., two pairs) and 4 (i.e., by considering quadruples).

During the search, the freedom of remaining sets of players could be used to prune the search earlier. We have implemented this idea and found that it did not in itself lead to new solutions to open instances. However, as we show in the next section, it can be useful to obtain greedily constructed configurations that still contain conflicts, and for them early backtracking is not even necessary.

## 6 Randomized Greedy Initial Configurations for Local Search

We now use the main idea of our greedy heuristic to generate good initial configurations for a local-search approach. For reasons that will become clear below, the *dual* of the previous heuristic is beneficial in this case. In addition, we add a randomization component that lets us add slight perturbations to initial configurations. First, we describe the modified heuristic.

To produce an initial configuration, the heuristic visits the weeks one after another. A single week is produced as follows: The week's groups are traversed one after another. As before, for each pair of adjacent positions in a group, the heuristic needs to select a pair of players still remaining to be scheduled in the current week. Now, it selects the pair having *maximal* freedom with respect to the current partial configuration. In addition, there is a parameter $\gamma$, with $0 \leq \gamma \leq 1$, that can be used to randomize the heuristic: In the case of ties, a random choice is made among the pairs of players having maximal freedom with probability $\gamma$. With probability $1-\gamma$, pairs are regarded as ordered, with the numerically smaller player first, and the lexicographically smallest pair is selected. After a pair of players was selected and is placed into a group, a large number is subtracted as a penalty from that pair's freedom in further weeks, to discourage that pair from being selected again in a different group. Other than that, the heuristic pays no attention to potential conflicts in a group, and never undoes a choice of pairs.

The remaining case is when $p$ is odd. Here, the heuristic can still work with pairs of players, except for the last player in each group. With probability $\gamma$, that player is randomly selected from all players that are still remaining to be scheduled in that week. With probability $1-\gamma$, the numerically smallest remaining player is selected.

As before, the heuristic is readily generalized from pairs to larger sets of players, and again there is a clear trade-off between maximizing freedom of groups and efficiency.

The intuition behind maximizing the freedom among players of a group is to "make room" for good local moves, which we discuss in the next section.

## 7 The Local Search Component

We aimed to keep the local search component as simple as possible. We chose the approach underlying the memetic algorithm (Cotta et al. 2006) and tabu-search (Dotú and Hentenryck 2005) as our basis, and then simplified it further. In particular, we eliminated the restart component, which we found not to be necessary in experiments: Even after omitting the restart component, our local search matched or exceeded the local search approaches reported in the literature. Also, based on experiments with different lengths of tabu lists, we fixed the length of tabu lists to 10 instead of imposing random limits as in (Dotú and Hentenryck (2005)).

Let $C$ denote a configuration. The triple $(i, j, k)$ is a *conflict position* iff there is a $k' \neq k$ such that there is a week $i' \neq i$ with $v(G_{ijk}) = v(G_{i'lm})$ and $v(G_{ijk'}) = v(G_{i'ln})$ for any $l$, $m$ and $n$, i.e., wherever a player is in the same group with another player more than once. Let $f(C)$ denote the number of conflict positions in $C$. Clearly, $f(C)$ is in the range $0 \ldots g \times p \times w$. A *swap* affecting position $(i, j, k)$ means the exchange of the values of $G_{ijk}$ with that of any other variable $G_{ij'k'}$, $j \neq j'$. Note that swaps preserve constraints (1) and (2).

A local search iteration starting from a configuration $C$ that satisfies constraints (1) and (2) proceeds as follows: First, determine $f = f(C)$. If $f = 0$, we have found a solution and are done. Otherwise, of all swaps affecting conflict positions, make the swap that leads to a configuration $C'$ with minimal $f(C')$ among all considered configurations (breaking ties lexicographically).

In addition, we associate a *tabu list* with each week that stores all pairs of *players* that were exchanged within the last 10 iterations in that week. Swaps that affect a pair of players in the corresponding week's tabu list are only considered if they result in an improvement over the best solution found so far. Also, if there was no improvement for 4 iterations, two random swaps are made.

We determined these parameters in test runs over all instances (see Section 8), and this combination worked best when averaged over the running times of all tested instances. By varying these parameters, we sometimes solved specific instances much faster, but others then turned out to be unsolvable within the time limit. In that respect, the local search is sensitive to these parameters. We tested all combinations of parameters up to a tabu list length of 20, up to 5 random swaps, and swapping after up to 10 iterations of non-improvement. To illustrate the algorithm's sensitivity to these parameters, Table 2 gives average running times (over 10 runs) for selected parameter combinations for three instances (10−6−7, 6−5−6 and 9−3−12) that we identified as hard instances for many parameter combinations. Each instance was tested with an initial configuration generated with $\gamma = 0$ as explained in the next section; "–" means that no solution was found within 20 minutes. With the parameter configuration that we eventually used, a solution was only found in two out of ten runs for instance 10−6−7 (first after 11 minutes, then after 9).

**Table 2** Timing results for selected parameter configurations on hard instances

| instance | # iter. of non-improvement | # random swaps | tabu list length | time |
|---|---|---|---|---|
| 6−5−6 | 2 | 1 | 5 | – |
| 6−5−6 | 2 | 3 | 5 | 2min |
| 6−5−6 | 4 | 2 | 10 | 4min |
| 6−5−6 | 4 | 2 | 20 | 2min |
| 6−5−6 | 4 | 5 | 10 | – |
| 9−3−12 | 2 | 1 | 5 | – |
| 9−3−12 | 2 | 3 | 5 | – |
| 9−3−12 | 4 | 2 | 10 | 15min |
| 9−3−12 | 4 | 2 | 20 | – |
| 9−3−12 | 4 | 5 | 10 | 20s |
| 10−6−7 | 2 | 1 | 5 | – |
| 10−6−7 | 2 | 3 | 5 | – |
| 10−6−7 | 4 | 2 | 10 | 10min |
| 10−6−7 | 4 | 2 | 20 | – |
| 10−6−7 | 4 | 5 | 10 | – |

**Table 3** Running times for selected instances using the parameters described in Section 7 ($\gamma = 0$)

| instance | time | instance | time |
|---|---|---|---|
| 5−3−7 | 0.2s | 8−8−5 | 37s |
| 8−3−10 | 0.4s | 9−3−11 | 0.2s |
| 8−4−9 | 1.2s | 9−4−9 | 5.6s |
| 8−4−10 | 11min | 10−3−13 | 7.8s |

## 8 Experimental Results

We executed 10 runs for each instance $g-p-w$ presented here, using an Apple MacBook with a 2.16 GHz Intel Core 2 Duo CPU and 1GB RAM. First, the greedy heuristic was used to generate an initial configuration with $\gamma = 0$. The time it takes to generate an initial configuration for the benchmark instances is negligible, i.e., at most half a second. Then the local search component is run with this starting configuration.

We chose 20 minutes as the maximum running time of the algorithm, since this is also the limit used in the benchmarks of the memetic algorithm (Cotta et al. 2006) on a similar (even faster) machine configuration, with which we wanted to allow a fair comparison. We considered an instance solved if a solution was found in at least one of the 10 runs we performed. With a single exception, all tested instances were consistently solved well within the 20 minutes time limit. The only exception was the instance 10−6−7, which was solved only in two runs out of ten within the time limit.

Our results are shown in Fig. 2. For various values of $g$ and $p$, we show groups of three bars, which denote the maximum $w$ such that the instance could be solved with (from left to right): Our scheme, the best memetic algorithm (Cotta et al. 2006), and local search alone (Dotú and Hentenryck 2005). The latter values are similar to those we obtain if we run the local search component in isolation, starting from the trivial configuration where each week is the same. In particular, we cannot even solve the 8−4−9 instance, let alone 8−4−10, without our greedy heuristic. For each instance, the thin horizontal lines show the (optimistic) upper bound and the best solution obtained with a mix of constraint-based formulations and basic design-theoretic techniques (Harvey 2002), respectively. Table 3 shows average running times for selected instances with $\gamma = 0$.

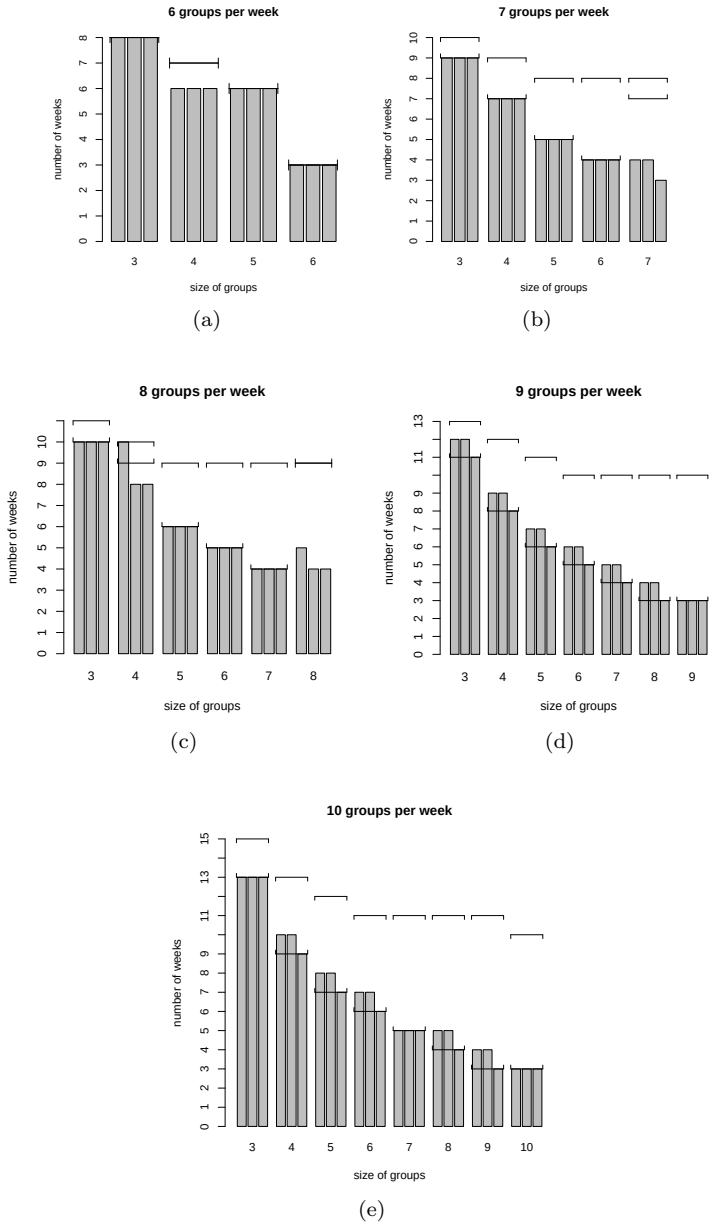


**Fig. 2** Solved number of weeks for $g$ equal to (a) 6, (b) 7, (c) 8, (d) 9 and (e) 10, with various values of $p$. Each group of three bars represents the maximum $w$ obtained by (from left to right): Our scheme, the best memetic approach proposed in (Cotta et al. 2006), and local search alone (Dotú and Hentenryck 2005). The thin horizontal lines show the best $w$ found with constraint solvers and design theory (Harvey 2002) and optimistic upper bounds, respectively.

It is clear from these figures that our approach is highly competitive on other instances besides the original problem as well: On all tested instances, it finds solutions for as many weeks as the best variant of the memetic algorithm (surpassing it on 8−4−10 and 8−8−5), and surpasses plain local search and constraint-based solutions in many cases.

## 9 New Solutions for the 8−4−10 Instance

The 8−4−10 instance of the SGP is of particular interest due to two reasons: First, it is the optimal solution for "the" Social Golfer Problem in the original sense, which is problem number 10 in CSPLib, a benchmark library for constraints (Gent and Walsh 1999). Second, being on the verge of solvability, the instance was previously thought to be amenable only to constraint solvers due to its highly constrained nature (previous metaheuristics could only solve for eight weeks). However, even the most sophisticated constraint solvers are currently unable to solve the instance. In contrast, by starting from our greedy initial configuration, solutions for this instance are readily generated. Two such solutions are depicted in Fig. 3. They were found by varying the randomization factor $\gamma$ of the greedy heuristic. Computation time was 11 and 4 minutes, respectively. We used McKay's `dreadnaut` program (McKay 1990) on the Levi graphs (Colbourn and Dinitz 1996) of the two solutions to verify that they are not isomorphic. The fact that we could obtain structurally different solutions is an indication that the greedy heuristic is meaningful and does not work only by accident.

It is left to explain *why* the greedy heuristic works so well on the 8−4−10 instance. Here, we believe that observing the solution process can give an important indication: To see an effect of the greedy heuristic, consider first Fig. 4, which shows what happens when the search is started from the trivial initial configuration of simply lining up the players in order for each week. For comparison, Fig. 5 shows different states of the local search component starting from a *greedy* initial configuration with $\gamma = 0$, with conflict positions highlighted. Initially, *every* position is a conflict position in both cases.

When contrasting the distribution of conflicts in the two figures, one effect of the greedy initial heuristic is apparent: Conflicts become more concentrated, and some weeks become conflict-free very early. In contrast, with a bad initial heuristic (Fig. 4), remaining conflicts are dispersed throughout all weeks. We believe that Fig. 5 gives a valuable suggestion on how the SGP could be successfully approached with a completely different local search method, which explicitly encodes a behaviour that is similar to the one found in this case. For example, one could build conflict-free groups incrementally, while exchanging players in existing weeks or dropping already built groups on occasion.

## 10 Conclusions and Future Work

We introduced a new greedy heuristic for the SGP, based on the intuitive concept of *freedom* among players. The heuristic is readily randomized and generalized, and was shown to improve results obtained by local search alone. In particular, we have obtained new solutions for the 8−4−10 instance. This makes our approach the first metaheuristic method that solves the original problem optimally, and also surpasses current constraint solvers on this instance. In addition, our approach is among the simplest and was shown to be highly competitive with other metaheuristic and constraint-

| | Week 1 | | | | Week 2 | | | | Week 3 | | | | Week 4 | | | | Week 5 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group 1 | 3 | 7 | 26 | 1 | 3 | 24 | 0 | 27 | 15 | 24 | 29 | 4 | 22 | 7 | 30 | 15 | 27 | 13 | 6 | 15 |
| Group 2 | 15 | 11 | 9 | 18 | 15 | 8 | 20 | 19 | 14 | 8 | 11 | 13 | 23 | 31 | 14 | 25 | 4 | 25 | 18 | 16 |
| Group 3 | 20 | 13 | 16 | 22 | 9 | 21 | 17 | 13 | 9 | 19 | 30 | 10 | 18 | 26 | 5 | 13 | 14 | 24 | 26 | 12 |
| Group 4 | 23 | 21 | 19 | 0 | 11 | 16 | 23 | 12 | 5 | 3 | 6 | 31 | 4 | 19 | 27 | 12 | 0 | 11 | 29 | 22 |
| Group 5 | 28 | 24 | 5 | 30 | 6 | 29 | 30 | 26 | 17 | 20 | 23 | 18 | 1 | 24 | 16 | 9 | 17 | 5 | 7 | 19 |
| Group 6 | 27 | 25 | 29 | 17 | 18 | 22 | 10 | 14 | 12 | 1 | 22 | 21 | 8 | 0 | 17 | 6 | 8 | 30 | 3 | 21 |
| Group 7 | 14 | 6 | 2 | 4 | 5 | 25 | 1 | 2 | 26 | 28 | 25 | 0 | 2 | 10 | 29 | 21 | 2 | 31 | 9 | 20 |
| Group 8 | 31 | 12 | 10 | 8 | 7 | 28 | 4 | 31 | 7 | 16 | 27 | 2 | 28 | 3 | 11 | 20 | 28 | 10 | 1 | 23 |

| | Week 6 | | | | Week 7 | | | | Week 8 | | | | Week 9 | | | | Week 10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group 1 | 19 | 6 | 1 | 11 | 3 | 16 | 19 | 14 | 22 | 9 | 25 | 6 | 18 | 8 | 1 | 27 | 4 | 20 | 1 | 0 |
| Group 2 | 24 | 21 | 31 | 18 | 1 | 13 | 29 | 31 | 28 | 29 | 19 | 18 | 15 | 3 | 25 | 10 | 8 | 2 | 28 | 22 |
| Group 3 | 3 | 22 | 4 | 17 | 11 | 7 | 21 | 25 | 21 | 26 | 27 | 20 | 17 | 24 | 2 | 11 | 18 | 6 | 12 | 7 |
| Group 4 | 0 | 10 | 7 | 13 | 20 | 24 | 6 | 10 | 0 | 16 | 31 | 15 | 20 | 14 | 29 | 7 | 24 | 19 | 13 | 25 |
| Group 5 | 5 | 8 | 29 | 16 | 17 | 15 | 28 | 12 | 1 | 17 | 14 | 30 | 0 | 5 | 9 | 12 | 3 | 23 | 9 | 29 |
| Group 6 | 25 | 12 | 20 | 30 | 0 | 30 | 2 | 18 | 4 | 11 | 10 | 5 | 22 | 31 | 19 | 26 | 16 | 17 | 26 | 10 |
| Group 7 | 9 | 28 | 14 | 27 | 23 | 22 | 27 | 5 | 13 | 12 | 2 | 3 | 4 | 30 | 23 | 13 | 31 | 27 | 30 | 11 |
| Group 8 | 2 | 26 | 15 | 23 | 9 | 4 | 26 | 8 | 23 | 7 | 24 | 8 | 16 | 21 | 28 | 6 | 14 | 5 | 15 | 21 |

| | Week 1 | | | | Week 2 | | | | Week 3 | | | | Week 4 | | | | Week 5 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group 1 | 19 | 9 | 3 | 8 | 24 | 2 | 30 | 4 | 9 | 15 | 0 | 27 | 16 | 0 | 25 | 24 | 15 | 17 | 18 | 12 |
| Group 2 | 28 | 25 | 18 | 5 | 18 | 13 | 10 | 8 | 7 | 28 | 4 | 31 | 17 | 23 | 31 | 8 | 19 | 24 | 5 | 14 |
| Group 3 | 4 | 1 | 15 | 6 | 20 | 15 | 19 | 21 | 25 | 26 | 2 | 1 | 5 | 26 | 13 | 3 | 7 | 11 | 26 | 8 |
| Group 4 | 14 | 11 | 20 | 0 | 1 | 29 | 5 | 7 | 10 | 5 | 12 | 30 | 30 | 6 | 7 | 19 | 25 | 22 | 4 | 21 |
| Group 5 | 13 | 23 | 30 | 21 | 23 | 12 | 11 | 16 | 20 | 6 | 3 | 17 | 18 | 22 | 20 | 1 | 3 | 29 | 0 | 30 |
| Group 6 | 16 | 10 | 29 | 22 | 26 | 6 | 28 | 0 | 24 | 29 | 23 | 18 | 11 | 2 | 15 | 10 | 13 | 6 | 16 | 27 |
| Group 7 | 7 | 2 | 27 | 17 | 3 | 31 | 27 | 25 | 14 | 8 | 16 | 21 | 21 | 12 | 9 | 28 | 28 | 10 | 23 | 1 |
| Group 8 | 31 | 26 | 12 | 24 | 14 | 17 | 9 | 22 | 11 | 22 | 19 | 13 | 29 | 4 | 14 | 27 | 2 | 9 | 20 | 31 |

| | Week 6 | | | | Week 7 | | | | Week 8 | | | | Week 9 | | | | Week 10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group 1 | 0 | 5 | 2 | 8 | 20 | 10 | 27 | 24 | 14 | 18 | 31 | 30 | 17 | 11 | 24 | 28 | 1 | 30 | 11 | 9 |
| Group 2 | 11 | 29 | 6 | 31 | 7 | 22 | 12 | 0 | 20 | 23 | 7 | 25 | 9 | 25 | 13 | 29 | 6 | 12 | 14 | 25 |
| Group 3 | 28 | 27 | 22 | 30 | 8 | 15 | 30 | 25 | 9 | 4 | 10 | 26 | 30 | 16 | 20 | 26 | 24 | 13 | 7 | 15 |
| Group 4 | 15 | 26 | 23 | 14 | 23 | 6 | 9 | 5 | 22 | 6 | 8 | 24 | 27 | 12 | 1 | 8 | 21 | 31 | 0 | 10 |
| Group 5 | 19 | 10 | 17 | 25 | 11 | 18 | 3 | 4 | 19 | 12 | 2 | 29 | 31 | 22 | 5 | 15 | 28 | 20 | 29 | 8 |
| Group 6 | 12 | 13 | 4 | 20 | 26 | 29 | 21 | 17 | 28 | 15 | 3 | 16 | 2 | 18 | 21 | 6 | 4 | 17 | 16 | 5 |
| Group 7 | 18 | 9 | 7 | 16 | 14 | 13 | 28 | 2 | 27 | 11 | 5 | 21 | 19 | 23 | 0 | 4 | 26 | 27 | 19 | 18 |
| Group 8 | 1 | 24 | 21 | 3 | 16 | 19 | 31 | 1 | 13 | 0 | 1 | 17 | 14 | 10 | 7 | 3 | 2 | 23 | 22 | 3 |

**Fig. 3** Two new non-isomorphic solutions for the 8−4−10 instance

based techniques on other instances as well. The dual of this heuristic was used to guide a complete backtracking search and solved Kirkman's schoolgirl problem with a very simple method.

Several interesting questions and opportunities for future research arise from our results: First, what other good greedy heuristics are there for the SGP and related problems? Second, it seems interesting to use the schedules produced by our greedy heuristic as initial configurations in combination with a proposed SAT formulation

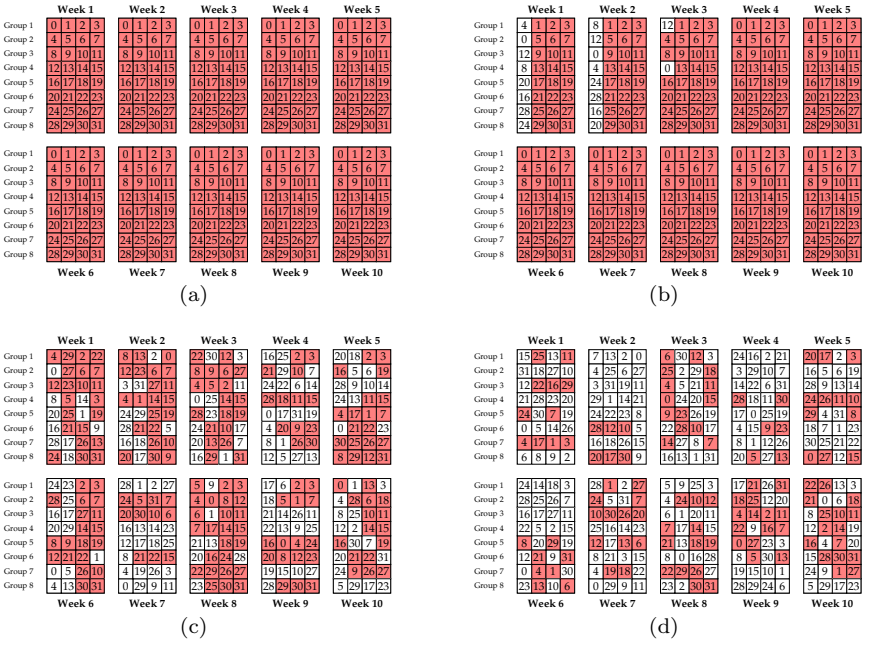


**Fig. 4** Instance 8−4−10, local search using a trivial initial configuration. Conflicts after (a) 0, (b) 10, (c) 100, (d) 500 iterations.

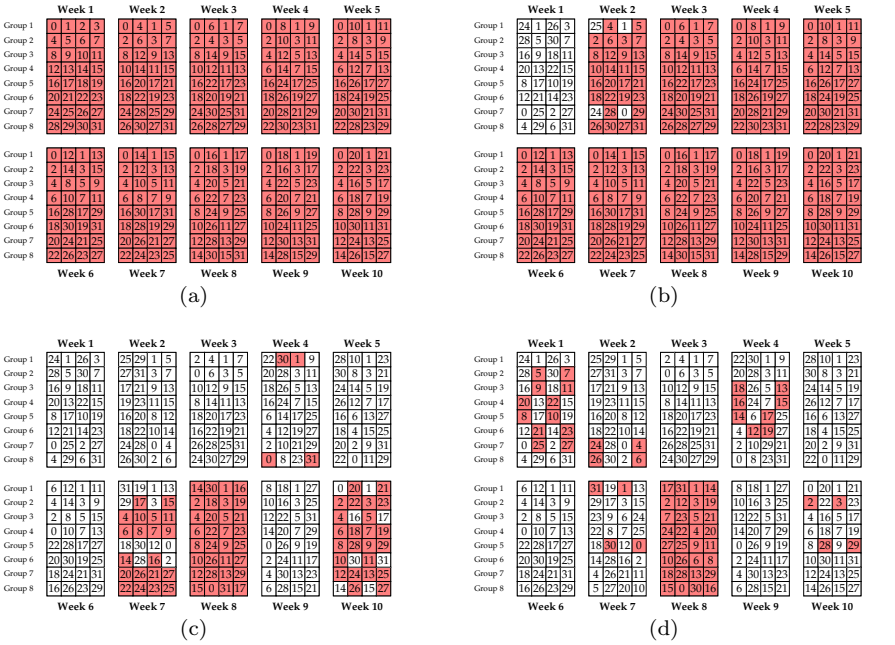


**Fig. 5** Instance 8−4−10, local search with our greedy initial configuration ($\gamma = 0$). Conflicts after (a) 0, (b) 10, (c) 100, (d) 500 iterations.

from the literature (Gent and Lynce 2005), and to use existing SAT solvers as the local search component. This could further reduce the effort for solving the problem. Third, we considered only one among several options for neighbourhoods and evaluation functions in the local search component, and it would be interesting to compare this with other variants. For example, it seems promising to take the freedom of players also into account in the local search component. Finally, observing the solution process of heuristics that are known to work well can give valuable hints on what else to try.

## 11 Acknowledgements

We thank the anonymous reviewers for their helpful comments.

## References

Aguado, A. (2004). A 10 days solution to the social golfer problem. *Manuscript*.

Barnier, N. and Brisset, P. (2005). Solving Kirkman's schoolgirl problem in a few seconds. *Constraints*, 10(1):7–21.

Colbourn, C. H. and Dinitz, J. H. (1996). *The CRC Handbook of Combinatorial Designs*. CRC Press.

Colbourn, C. J. (1984). The complexity of completing partial latin squares. *Discrete Appl. Math.*, 8:25–30.

Colbourn, C. J. (1999). A Steiner 2-design with an automorphism fixing exactly $r + 2$ points. *Journal of Combinatorial Designs*, 7:375–380.

Cotta, C., Dotú, I., Fernández, A. J., and Hentenryck, P. V. (2006). Scheduling social golfers with memetic evolutionary programming. In *Hybrid Metaheuristics*, volume 4030 of *LNCS*, pages 150–161.

Dotú, I. and Hentenryck, P. V. (2005). Scheduling social golfers locally. In *CPAIOR*, volume 3524 of *LNCS*, pages 155–167.

Douglas R. Stinson (1994). Universal hashing and authentication codes. *Designs, Codes and Cryptography*, 4:369–380.

Fahle, T., Schamberger, S., and Sellmann, M. (2001). Symmetry breaking. In *CP'01, the 7th Int. Conf. on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 93–107.

Gent, I. and Lynce, I. (2005). A SAT encoding for the social golfer problem. *IJCAI'05 Workshop on Modelling and Solving Problems with Constraints*.

Gent, I. P. and Walsh, T. (1999). CSPLib: A benchmark library for constraints. In *CP'99*, volume 1713 of *LNCS*.

Gordon, D., Kuperberg, G., and Patashnik, O. (1995). New constructions for covering designs. *Journal of Combinatorial Designs*, 4:269–284.

Harvey, W. (2002). Warwick's results page for the social golfer problem. *http://www.icparc.ic.ac.uk/∼wh/golf/*.

Harvey, W. and Winterer, T. (2005). Solving the MOLR and social golfers problems. In *CP'05*, volume 3709 of *LNCS*, pages 286–300.

Hsiao, M. Y., Bossen, D. C., and Chien, R. T. (1970). Orthogonal Latin square codes. *IBM Journal of Research and Development*, 14(4):390–394.

McKay, B. (1990). Nauty user's guide (version 1.5). Technical report, Dept. Comp. Sci., Australian National University.

Petrie, K. E., Smith, B., and Yorke-Smith, N. (2004). Dynamic symmetry breaking in constraint programming and linear programming hybrids. In *In European Starting AI Researcher Symp*.

Triska, M. and Musliu, N. (2010). An improved sat formulation for the social golfer problem. *Annals of Operations Research*, pages 1–12. 10.1007/s10479-010-0702-5.