Scheduling Social Golfers Locally

Iván Dotú[1] and Pascal Van Hentenryck[2]

[1] Departamento De Ingeniería Informática, Universidad Autónoma de Madrid
[2] Brown University, Box 1910, Providence, RI 02912

Abstract. The scheduling of social golfers has attracted significant attention in recent years because of its highly symmetrical and combinatorial nature. In particular, it has become one of the standard benchmarks for symmetry breaking in constraint programming. This paper presents a very effective, local search, algorithm for scheduling social golfers. The algorithm find the first known solutions to 11 instances and matches, or improves, state-of-the-art results from constraint programming on all but 3 instances. Moreover, most instances of the social golfers are solved within a couple of seconds. Interestingly, the algorithm does not incorporate any symmetry-breaking scheme and illustrates the nice complementarity between constraint programming and local search on this scheduling application.

1 Introduction

The social golfer problem has attracted significant interest since it was first posted on `sci.op-research` in May 1998. It consists of scheduling $n = g \times p$ golfers into g groups of p players every week for w weeks so that no two golfers play in the same group more than once. An instance of the social golfer is specified by a triple $g - p - w$, where g is the number of groups, p is the size of a group, and w is the number of weeks in the schedule.

The scheduling of social golfers is a highly combinatorial and symmetric problem and it is not surprising that it has generated significant attention from the constraint programming community (e.g., [5, 12, 6, 11, 10, 2, 9]). Indeed, it raises fundamentally interesting issues in modeling and symmetry breaking, and it has become one of the standard benchmarks for evaluating symmetry-breaking schemes. Recent developments (e.g., [2, 9]) approach the scheduling of social golfers using innovative, elegant, but also complex, symmetry-breaking schemes.

This paper approaches the problem from a very different angle. It proposes a local search algorithm for scheduling social golfers, whose local moves swap golfers within the same week and are guided by a tabu-search meta-heuristic. The local search algorithm matches, or improves upon, the best solutions found by constraint programming on all instances but 3. It also found the first solutions to 11 instances that were previously open for constraint programming.[1] Moreover, the local search algorithm solves

---

[1] For the statuses of the instances, see Warwick Harvey's web page at `http://www.icparc.ic.ac.uk/wh/golf`.

almost all instances easily in a few seconds and takes about 1 minute on the remaining (harder) instances. The algorithm also features a constructive heuristic which trivially solves many instances of the form $odd - odd - w$ and provides good starting points for others.

The main contributions of this paper are as follows.

1. It shows that local search is a very effective way to schedule social golfers. It found the first solutions to 11 instances and matches, or improves upon, all instances solved by constraint programming but 3. In addition, almost all instances are solved in a few seconds, the harder ones taking about 1 minute.
2. It demonstrates that the local search algorithm uses a natural modeling and does not involve complex symmetry-breaking schemes. In fact, it does not take symmetries into account at all, leading to an algorithm which is significant simpler than constraint programming solutions, both from a conceptual and implementation standpoint.
3. The experimental results indicate a nice complementarity between constraint programming and local search, as some of the hard instances for one technology are trivially solved by the other.

The rest of the paper is organized as follows. The paper starts by describing the basic local search algorithm, including its underlying modeling, its neighborhood, its metaheuristic, and its experimental results. It then presents the constructive heuristic and reports the new experimental results when the heuristic replaces the random configurations as starting points of the algorithm. Finally, the paper discusses related work and concludes by giving some preliminary results on generalizations of the problem.

2   The Modeling

There are many possible modelings for the social golfer problem, which is one of the reasons why it is so interesting. This paper uses a modeling that associates a decision variable $x[w, g, p]$ with every position p of every group g of every week w. Given a schedule σ, i.e., an assignment of values to the decision variables, the value $\sigma(x[w, g, p])$ denotes the golfer scheduled in position p of group g in week w. There are two kinds of constraints in the social golfer.

1. A golfer plays exactly once a week;
2. Two golfers can play together (i.e., in the same group of the same week) at most once.

The first type of constraints is implicit in the algorithms presented in this paper: It is satisfied by the initial assignments and is preserved by local moves. The second set of constraints is represented explicitly. The model contains a constraint $m[a, b]$ for every pair (a, b) of golfers: Constraint $m[a, b]$ holds for an assignment σ if golfers a and b are not assigned more than once to the same group. More precisely, if $\#_\sigma(a, b)$ denotes the number of times golfers a and b meet in schedule σ, i.e.,

$$\#\{(w, g) \mid \exists p, p' : \sigma(x[w, g, p]) = a \ \& \ \sigma(x[w, g, p']) = b\},$$

constraint $m[a, b]$ holds if

$$\#_\sigma(a, b) \leq 1.$$

To guide the algorithm, the model also specifies violations of the constraints. Informally speaking, the violations $v_\sigma(m[a, b])$ of a constraint $m[a, b]$ is the number of times golfers a and b are scheduled in the same group in schedule σ beyond their allowed meeting. In symbols,

$$v_\sigma(m[a, b]) = \max(0, \#_\sigma(a, b) - 1).$$

As a consequence, the social golfer problem can be modeled as the problem of finding a schedule σ minimizing the total number of violations $f(\sigma)$ where

$$f(\sigma) = \sum_{a,b \in \mathcal{G}} v_\sigma(m[a, b]).$$

and $\mathcal{G}$ is the set of $g \times p$ golfers. A schedule σ with $f(\sigma) = 0$ is a solution to the social golfer problem.

3   The Neighborhood

The neighborhood of the local search consists of swapping two golfers from different groups in the same week. The set of swaps is thus defined as

$$\mathcal{S} = \{(\langle w, g_1, p_1 \rangle, \langle w, g_2, p_2 \rangle) \mid g_1 \neq g_2\}.$$

It is more effective however to restrict attention to swaps involving at least one golfer in conflict with another golfer in the same group. This ensures that the algorithm focuses on swaps which may decrease the number of violations. More formally, a triple $\langle g, w, p \rangle$ is said to be in conflict in schedule σ, which is denoted by $v_\sigma(\langle g, w, p \rangle)$, if

$$\exists p' \in P : v_\sigma(m[\sigma(x[w, g, p]), \sigma(x[w, g, p'])]) > 1.$$

With this restriction in mind, the set of swaps $\mathcal{S}^-(\sigma)$ considered for a schedule σ becomes

$$\mathcal{S}^-(\sigma) = \{(\langle w_1, g_1, p_1 \rangle, \langle w_2, g_2, p_2 \rangle) \in \mathcal{S} \mid v_\sigma(\langle w_1, g_1, p_1 \rangle)\}.$$

4   The Tabu Component

The tabu component of the algorithm is based on three main ideas. First, the tabu list is distributed across the various weeks, which is natural since the swaps only consider golfers in the same week. The tabu component thus consists of an array $tabu$ where $tabu[w]$ represents the tabu list associated with week w. Second, for a given week w, the tabu list maintains triplet $\langle a, b, i \rangle$, where a and b are two golfers and i represents the first iteration where golfers a and b can be swapped again in week w. Observe that

the tabu lists store golfers, not positions $\langle w, g, p \rangle$. Third, the tabu tenure, i.e., the time a pair of golfers (a, b) stays in the list, is dynamic: It is randomly generated in the interval $[4, 100]$. In other words, each time a pair of golfers (a, b) is swapped, a random value ρ is drawn uniformly from the interval $[4, 100]$ and the pair (a, b) is tabu for the next ρ iterations. At iteration k, swapping two golfers a and b is tabu, which is denoted by

$$tabu[w](a, b, k)$$

if the Boolean expression

$$\langle a, b, i \rangle \in tabu[w] \ \& \ i \leq k$$

holds. As a consequence, for schedule σ and iteration k, the neighborhood consists of the set of moves $\mathcal{S}^t(\sigma, k)$ defined as

$$\mathcal{S}^t(\sigma, k) = \{(t_1, t_2) \in \mathcal{S}^-(\sigma) \mid \neg tabu[w](\sigma(x[t_1]), \sigma(x[t_2]), k)\}.$$

where we abuse notations and use $x[\langle w, g, p \rangle]$ to denote $x[w, g, p]$.

Aspiration In addition to the non-tabu moves, the neighborhood also considers moves that improve the best solution found so far, i.e., the set $\mathcal{S}^*(\sigma, \sigma^*)$ defined as

$$\mathcal{S}^*(\sigma, \sigma^*) = \{(t_1, t_2) \in \mathcal{S}^-(\sigma) \mid f(\sigma[x[t_1] \leftrightarrow x[t_2]]) < f(\sigma^*)\},$$

where $\sigma[x_1 \leftrightarrow x_2]$ denotes the schedule σ where the values of variables x_1 and x_2 have been swapped and σ^* denotes the best solution found so far.

5   The Tabu-Search Algorithm

We are now ready to present the basic local search algorithm SGLS. The algorithm, depicted in Figure 1, is a tabu search with a restarting component. Lines 2-7 perform the initializations. In particular, the tabu list is initialized in lines 2-3, the initial schedule is generated randomly in line 4, while lines 6 and 7 initialize the iteration counter k, and the stability counter s. The initial configuration σ randomly schedules all golfers in the various groups for every week, satisfying the constraint that each golfer plays exactly once a week. The best schedule found so far σ^* is initialized to σ.

The core of the algorithm is given in lines 8-23. They iterate local moves for a number of iterations or until a solution is found. The local move is selected in line 9. The key idea is to select the best swaps in the neighborhood

$$\mathcal{S}^t(\sigma, k) \ \cup \ \mathcal{S}^*(\sigma, \sigma^*),$$

i.e., the non-tabu swaps and those improving the best schedule. Observe that the expression

$$f(\sigma[x[t_1] \leftrightarrow x[t_2]])$$

represents the number of violations obtained after swapping t_1 and t_2. The tabu list is updated in line 11, where *week($\langle w,g,p \rangle$)* is defined as

$$week(\langle w, g, p \rangle) = w.$$

```
1.    SGLS(W, G, P)
2.        forall w ∈ W
3.            tabu[w] ← {};
4.        σ ← random configuration;
5.        σ* ← σ;
6.        k ← 0;
7.        s ← 0;
8.        while k ≤ maxIt  &  f(σ) > 0 do
9.            select (t₁, t₂) ∈ Sᵗ(σ, k) ∪ S*(σ, σ*) minimizing f(σ[x[t₁] ↔ x[t₂]]);
10.           τ ← RANDOM([4,100]);
11.           tabu[week(t₁)] ← tabu[week(t₁)] ∪ {⟨σ(x[t₁]), σ(x[t₂]), k + τ⟩};
12.           σ ← σ[x[t₁] ↔ x[t₂]];
13.           if f(σ) < f(σ*) then
14.               σ* ← σ;
15.               s ← 0;
16.           else if s > maxStable then
17.               σ ←random configuration;
18.               s ← 0;
19.               forall w ∈ W do
20.                   tabu[w] = {};
21.           else
22.               s++;
23.           k++;
```

Fig. 1. Algorithm SGLS for Scheduling Social Golfers

The new schedule is computed in line 12. Lines 13-15 update the best schedule, while lines 16-20 specify the restarting component.

The restarting component simply reinitializes the search from a random configuration whenever the best schedule found so far has not been improved upon for *maxStable* iterations. Note that the stability counter s is incremented in line 22 and reset to zero in line 15 (when a new best schedule is found) and in line 18 (when the search is restarted).

6   Experimental Results

This section reports the experimental results for the SGLS algorithm. The algorithm was implemented in C and the experiments were carried out on a 3.06GHz PC with 512MB of RAM. Algorithm SGLS was run 100 times on each instance and the results report average values for successful runs, as well as the percentage of unsuccessful runs (if any).

Tables 1 and 2 report the experimental results for SGLS. Given a number of groups g and a group size p, the tables only give the results for those instances $g - p - w$ maximizing w since they also provide solutions for $w' < w$. Table 1 reports the number of iterations (moves), while Table 2 reports the execution times. Bold entries indicate that SGLS matches the best known number of weeks for a given number of groups and a given group size. The percentage of unsuccessful runs is shown between parentheses in Table 2.

Table 1. Number of Iterations for SGLS with Maximal Number of Weeks. Bold Entries Indicate a Match with the Best Known Number of Weeks

	size 3		size 4		size 5		size 6		size 7		size 8		size 9		size 10	
g	w	I	w	I	w	I	w	I	w	I	w	I	w	I	w	I
6	**8**	282254.0	**6**	161530.3	**6**	16761.5	**3**	15.8	-	-	-	-	-	-	-	-
7	**9**	12507.6	**7**	274606.0	**5**	102.9	**4**	100.4	**3**	23.4	-	-	-	-	-	-
8	**10**	653.9	**8**	323141.5	**6**	423.7	**5**	1044.9	**4**	237.5	**4**	153301.6	-	-	-	-
9	**11**	128.3	**8**	84.4	**6**	52.7	**5**	55.5	**4**	44.8	**3**	27.7	**3**	43.9	-	-
10	**13**	45849.1	**9**	100.2	**7**	80.8	**6**	110.7	**5**	94.6	**4**	61.8	**3**	36.1	**3**	53.3

Table 2. CPU Time in Seconds for SGLS with Maximal Number of Weeks. Bold Entries Indicate a Match with the Best Known Number of Weeks

	size 3			size 4			size 5		size 6		size 7		size 8		size 9		size 10	
g	w	T	%F	w	T	%F	w	T	w	T	w	T	w	T	w	T	w	T
6	**8**	48.93	6	6	47.75		**6**	107.18	**3**	0.01	-	-	-	-	-	-	-	-
7	**9**	3.06		**7**	107.62	8	**5**	0.07	**4**	0.09	3	0.03	-	-	-	-	-	-
8	**10**	0.23		8	207.77	9	**6**	0.37	**5**	1.21	**4**	0.39	4	360.00	-	-	-	-
9	**11**	0.08		**8**	0.09		**6**	0.09	**5**	0.13	**4**	0.14	**3**	0.09	**3**	0.19	-	-
10	**13**	30.82		**9**	0.16		**7**	0.19	**6**	0.34	**5**	0.41	**4**	0.33	**3**	0.20	**3**	0.39

As can be seen from the tables, Algorithm SGLS finds solutions to all the instances solved by constraint programming except 4. Moreover, almost all entries are solved in less than a second. Only a few instances are hard for the algorithm and require around 1 minute of CPU time. Interestingly, algorithm SGLS also solves 7 new instances: $9-4-9$, $9-5-7$, $9-6-6$, $9-7-5$, $9-8-4$, $10-5-8$ and $10-9-4$.

It is interesting to observe that algorithm SGLS does not break symmetries and does not exploit specific properties of the solutions. This contrasts with constraint-programming solutions that are often quite sophisticated and involved. See, for instance, the recent papers [2, 9] which report the use of very interesting symmetry-breaking schemes to schedule social golfers.

7   A Constructive Heuristic

The quality of SGLS can be further improved by using a constructive heuristic to find a good starting, and restarting, configuration. The heuristic [3] trivially solves $p - p - (p + 1)$ instances when p is prime and provides good starting points (or solutions) for other instances as well. Examples of such initial configurations are given in Tables 3 and 4, which will be used to explain the intuition underlying the constructive heuristic. The heuristic simply aims at exploiting the fact that all golfers in a group for a given week must be assigned a different group in subsequent weeks. As a consequence, the heuristic attempts to distribute these golfers in different groups in subsequent weeks.

Table 4 is a simple illustration of the heuristic with 5 groups of size 5 (i.e., 25 golfers) and 6 weeks. The first week is simply the sequence 1..25. In the second week, group i consists of all golfers in position i in week 1. In particular, group 1 consists

Table 3. The initial configuration for the problem $4 - 3 - 3$

weeks	group 1	group 2	group 3	group 4
week 1	1 2 3	4 5 6	7 8 9	10 11 12
week 2	1 4 7	10 2 5	8 11 3	6 9 12
week 3	1 5 9	10 2 6	7 11 3	4 8 12

Table 4. The intial configuration for the problem $5 - 5 - 6$

weeks	group 1	group 2	group 3	group 4	group 5
week 1	1 2 3 4 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25
week 2	1 6 11 16 21	2 7 12 17 22	3 8 13 18 23	4 9 14 19 24	5 10 15 20 25
week 3	1 7 13 19 25	2 8 14 20 21	3 9 15 16 22	4 10 11 17 23	5 6 12 18 24
week 4	1 8 15 17 24	2 9 11 18 25	3 10 12 19 21	4 6 13 20 22	5 7 14 16 23
week 5	1 9 12 20 23	2 10 13 16 24	3 6 14 17 25	4 7 15 18 21	5 8 11 19 22
week 6	1 10 14 18 22	2 6 15 19 23	3 7 11 20 24	4 8 12 16 25	5 9 13 17 21

of golfers $1, 6, 11, 16, 21$, group 2 is composed of golfers $2, 7, 12, 17, 22$ and so on. In other words, the groups consist of golfers in the same group position in week 1. The third week is most interesting, since it gives the intuition behind the heuristic. The key idea is to try to select golfers whose positions are j,j+1,j+2,j+3,j+4 in the first week, the addition being modulo the group size. In particular, group 1 is obtained by selecting the golfers in position i from group i in week 1, i.e., golfers $1, 7, 13, 19, 25$. Subsequent weeks are obtained in similar fashion by simply incrementing the offset. In particular, the fourth week considers sequences of positions of the form j,j+2,j+4,j+6,j+8 and its first group is $1, 8, 15, 17, 24$. Table 3 illustrates the heuristic on the 4-3-3 instance. Note that the first group in week 2 has golfers in the first position in groups 1, 2, and 3 in week 1. However, the first golfer in week 4 must still be scheduled. Hence the second group must select golfer 10, as well as golfers 2 and 5.

Figure 2 depicts the code of the constructive heuristic. The code takes the convention that the weeks are numbered from 0 to $w-1$, the groups from 0 to $g-1$, and the positions from 0 to $p - 1$, since this simplifies the algorithm. The key intuition to understand the code is to recognize that a week can be seen as a permutation of the golfers on which the group structure is superimposed. Indeed, it suffices to assign the first p positions to the first group, the second set of p positions to the second group and so on. As a consequence, the constructive heuristic only focuses on the problem of generating w permutations $P_0, \ldots, P_{w-1}$.

The top-level function is HEURISTICSCHEDULE which specifies the first week and calls function SCHEDULEWEEEK for the remaining weeks. Scheduling a week is the core of the heuristic. All weeks start with golfer 1 (line 7) and initialize the position po to 0 (line 8), the group number gr to 1 (line 9), and the offset Δ to $we - 1$. The remaining golfers are scheduled in lines 11-15.

The key operation is line 12, which selects the first *unscheduled* golfer s from group gr of week 0 (specified by P_0) starting at position $(po + \Delta)\%p$ and proceeding by viewing the group as a circular list. The next three instructions update the position po to the position of s in group gr of week 0 (line 13), increment the group to select a

```
 1. HEURISTICSCHEDULE(w, g, p)
 2.     n ← g × p;
 3.     P₀ ← ⟨1, . . . , n⟩;
 4.     forall we ∈ 1..w − 1
 5.         P_we ← scheduleWeek(we, g, p, n);

 6. SCHEDULEWEEK(we, g, p, n)
 7.     P_we ← ⟨1⟩;
 8.     po ← 0;
 9.     gr ← 1;
10.     Δ ← we − 1;
11.     forall go ∈ 1..n − 1
12.         s ← SELECT(gr, (po + Δ)%p);
13.         po ← POSITION(s);
14.         gr ← (gr + 1)%g;
15.         P_we ← P_we :: ⟨s⟩;
16.     return P_we;
```

Fig. 2. The Constructive Heuristic for Scheduling Social Golfers

golfer from the next group, and extend the permutation by concatenating s to P_{we}. By specification of SELECT, which only selects unscheduled golfers and the fact that the heuristic selects the golfers from the groups in a round-robin fashion, the algorithm is guaranteed to generate a permutation.

8   Experimental Results Again

This section discusses the performance of algorithm SGLS-CH that enhances SGLS with the constructive heuristic to generate starting/restarting points. Although the starting point is deterministic, the algorithm still uses restarting, since the search itself is randomized, i.e., ties are broken randomly.

8.1   The $odd − odd − w$ Instances

It is known that the constructive heuristic finds solutions for $p − p − (p + 1)$ instances when p is prime. Moreover, it also provides solutions to many instances of the form $odd − odd − w$ as we now show experimentally. The results were performed up to $odd =$ 49. For all (odd) prime numbers p lower than 49, the heuristic solves the instances $p − p − w$, where w is the maximal number of weeks for p groups and periods. When odd is divisible by 3, the heuristic solves instances of the form $odd − odd − 4$, when odd is divisible by 5, it solves instances of the form $odd − odd − 6$, and when odd is divisible by 7, it solves instances of the form $odd − odd − 8$. For instance, the constructive heuristic solves instance 49-49-8.

It is interesting to relate these results to mutually orthogonal latin squares. Indeed, it is known that finding a solution for instances of the form $g − g − 4$ is equivalent to the problem of finding two orthogonal latin squares of size g. Moreover, instances of the form $g − g − n$ are equivalent to the problem of finding $n − 2$ mutually orthogonal latin

Table 5. Results on the $odd - odd - w$ Instances

instances	$CH : w$	Gol:LB
3-3-w	4	4
5-5-w	6	6
7-7-w	8	8
9-9-w	4	10
11-11-w	12	12
13-13-w	14	14
15-15-w	4	6
17-17-w	18	18
19-19-w	20	20
21-21-w	4	7
23-23-w	24	24
25-25-w	6	26
27-27-w	4	28
29-29-w	30	30
31-31-w	32	32
33-33-w	4	7
35-35-w	6	7
37-37-w	38	38
39-39-w	4	6
41-41-w	42	42
43-43-w	44	44
45-45-w	4	8
47-47-w	48	48
49-49-w	8	50

squares of size g [3, 10]. Instances of the form $g - g - 4$ can be solved in polynomial time when g is odd. This provides some insight into the structure of these instances and some rationale why the constructive heuristic is able to solve many of the $odd - odd - w$ instances. Table 5 summarizes the results on the $odd - odd - w$ instances. The columns respectively specify the instances, the largest w found by the constructive heuristic, and the number of weeks w for the social golfers that corresponds to the best lower bound on the latin square as given in [4]. Rows in bold faces indicate closed instances.

It is interesting to observe that the lower bounds on the mutually orthogonal latin squares vary significantly. Indeed, the lower bound for size 17 is 16, while it is 4 for size 15. These lower bounds give some additional insights on the inherent difficulty of these instances and on the behavior of the constructive heuristic.

8.2   Hard Instances

Table 6 compares the tabu-search algorithm with and without the constructive heuristic on the hard instances from Table 2. Note that $7 - 7 - 7$ and $7 - 7 - 8$ are now trivially solved, as well as $9 - 9 - 4$ which was also open. SGLS-CH does not strictly dominates SGLS, as there are instances where it is slightly slower. However, on some instances,

Table 6. Comparison between SGLS and SGLS-CH

instances	random			new		
	I	T	$\%F$	I	T	$\%F$
6-3-8	282254.07	48.93	6	250572	43.84	4
6-4-6	161530.35	47.75		168000	49.66	
7-4-7	274606.00	107.18		200087	124.15	
8-4-8	323141.52	107.62	8	316639	141.91	3
8-8-4	153301.61	360.00		8380.45	19.54	
8-8-5	–	–	100	108654.00	496.82	
10-3-13	45849.00	30.82		51015.00	34.28	

Table 7. Experimental Results of SGLS-CH on the New Solved Instances

instance	I	T	$\%$ solved
7-5-6	487025.0	370.50	10
9-4-9	469156.4	402.55	100
9-5-7	4615.0	5.39	100
9-6-6	118196.7	196.52	100
9-7-5	64283.9	155.16	100
9-8-4	1061.3	2.92	100
10-4-10	548071.6	635.20	100
10-5-8	45895.4	76.80	100
10-9-4	5497.9	24.42	100

it is clearly superior (including on $8 - 8 - 5$ which can now be solved). Algorithm SGLS-CH also closes two additional open problems: $7 - 5 - 6$ and $10 - 4 - 10$. Table 7 depicts the performance of algorithm SGLS-CH on the new solved instances.

8.3   Summary of the Results

Table 8 summarizes the results of this paper. It depicts the status of maximal instances for SGLS-CH and whether the instances are hard (more than 10 seconds) or easy (less than 10 seconds). The results indicate that SGLS-CH matches or improves the best results for all but 3 instances. In addition, it produces 11 new solutions with respect to earlier results. These results are quite remarkable given the simplicity of the approach. Indeed, constraint-programming approaches to the social golfer problem are typically very involved and use elegant, but complex, symmetry-breaking techniques. Algorithm SGLS-CH, in contrast, does not include any such symmetry breaking.

It is interesting to observe the highly constrained nature of the instances for which SGLS-CH does not match the best-known results. Hence it is not surprising that constraint programming outperforms local search on these instances. Note also that Brisset and Barnier [2] proposed a very simple constraint-programming model to solve $8-4-9$ in a few seconds. So, once again, there seems to be a nice complementarity between constraint programming and local search on the social golfer problem.

Table 8. Summary of the Results for SGLS-CH with Maximal Number of Weeks. Bold entries represent a match or an improvement over existing solutions. The status is *new* (for improvement), hard (> 10 seconds), and easy (≤ 10 seconds)

#groups	size 3 w status	size 4 w status	size 5 w status	size 6 w status	size 7 w status	size 8 w status	size 9 w status	size 10 w status
6	**8 Hard**	6 Hard	**6 Hard**	**3 Easy**	-	-	-	-
7	**9 Easy**	**7 Hard**	**6 New**	**4 Easy**	**8 New**	-	-	-
8	**10 Easy**	8 Hard	**6 Easy**	**5 Easy**	**4 Easy**	5 Hard	-	-
9	**11 Easy**	**9 New**	**7 New**	**6 New**	**5 New**	**4 New**	**4 New**	-
10	**13 Hard**	**10 New**	**8 New**	**6 Easy**	**5 Easy**	**4 Easy**	**4 New**	**3 Easy**

Table 9. Summary of the Results for Atmost Two Meetings. Easy < 10s. Medm > 20s & $< 5\%$ unsolved. Hard $< 50\%$ unsolved. Chal $> 50\%$ unsolved

#groups	size 3 w status	size 4 w status	size 5 w status	size 6 w status	size 7 w status	size 8 w status	size 9 w status	size 10 w status
3	8 Easy	6 Easy	6 Easy	6 Easy	4 Easy	4 Easy	4 Easy	2 -
4	11 Easy	10 Easy	8 Chal	6 Easy	6 Easy	7 Chal	6 Easy	6 Easy
5	14 Hard	11 Easy	9 Easy	8 Easy	7 Easy	7 Hard	6 Easy	6 Easy
6	16 Easy	13 Easy	11 Easy	10 Easy	9 Easy	8 Easy	7 Easy	7 Easy
7	19 Easy	15 Easy	13 Easy	12 Easy	11 Medm	10 Easy	9 Easy	8 Easy
8	22 Medm	18 Medm	15 Easy	14 Hard	12 Easy	11 Easy	10 Easy	10 Medm
9	25 Chal	20 Easy	17 Easy	15 Easy	14 Easy	13 Medm	12 Easy	11 Easy
10	27 Easy	22 Easy	19 Easy	17 Easy	15 Easy	14 Easy	13 Easy	12 Easy

9   Related Work

There is a considerable body of work on scheduling social golfers in the constraint programming community. References [2, 6, 9] describe state-of-the art results using constraint programming and are excellent starting points for more references.[2] See also [10] for interesting theoretical and experimental results on the social golfer problem, as well as the description of SBDD, a general scheme for symmetry breaking. Reference [1] describes a tabu-search algorithm for scheduling social golfers, where the neighborhood consists of swapping the value of a single variable and where all constraints are explicit. The results are very far in quality and performance from those reported here.

10   Conclusion

This paper reconsidered the scheduling of social golfers, a highly combinatorial and symmetric application which has raised significant interest in the constraint programming community. It presented an effective local search algorithm which found the first

---

[2] Reference [9] contains much more general results on symmetry breaking but the scheduling of social golfers is one of the main applications in evaluating the new techniques.

solutions to 11 new instances and matched, or improved upon, all instances solved by constraint programming solutions but 3. Moreover, the local search algorithm was shown to find almost all solutions in less than a couple of seconds, the harder instances taking about 1 minute. The algorithm also features a constructive heuristic which trivially solves many instances of the form $odd - odd - w$.

It is interesting to conclude with a number of interesting observations. First, the social golfer is a problem where the properties of the instances seem to determine which approach is best positioned to solve them. In particular, hard instances for constraint programming are easy for local search and vive-versa. There are of course other applications where this also holds. What is interesting here is the simplicity of local search compared to its constraint programming counterpart and the absence of symmetry-breaking schemes in local search. Whether this observation generalizes to other, highly symmetric, problems is an interesting issue for future work. See, for instance, [7, 8] for early results along these lines.

Second, there are many interesting variations around the social golfer problem that are generating increasing interest. These variations include the possibility of golfers to meet more than once, as well as the superimposition of a referee assignment minimizing the number of referees subject to "fairness" constraints. Our preliminary results with local search on these problems, which are motivated by real-life applications, are extremely encouraging. In particular, Table 9 reports some very preliminary results on the generalizations where golfers are allowed to meet more than once. The instances are classified into easy, medium, hard, and challenging. Hard instances mean that local search may occasionally fail to find a solution in 100,000 iterations (but in less than 50% of the time), while challenging instances fail in finding a solution more than 50% of the time within the iteration limit. Once again, given a number of groups g and a group size p, the tables only give the results for those instances $g - p - w$ maximizing w since they also provide solutions for $w' < w$.

Finally, there are many connections with Latin squares that could be exploited further. It is likely that new heuristic solutions based on this connection would close additional instances and provide good starting points on others.

Acknowledgements

References

1. M. Ågren.   Solving the Social Golfer Using Local Search. peg.it.uu.se/ saps02/ MagnusAgren/, 2003.
2. N. Barnier and P. Brisset. Solving kirkman's schoolgirl problem in a few seconds. *Constraints*, 2005. To appear in the Special Issue on CP'02.
3. C. Colbourn and Dinitz. *The CRC Handbook of Combinatorial Design*. CRC Press, Boca Raton, FL, 1996.

4. C.J. Colbourn and J.H. Dinitz. Mutually orthogonal latin squares: A brief survey of constructions. *"Journal of Statistical Planning and Inference"*, 95:9–48, 2001.
5. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In Toby Walsh, editor, *Principles and Practice of Constraint Programmingo*, volume 2293 of *LNCS*, pages 93–107. Springer Verlag, 2001.
6. S. Prestwich. Randomized backtracing for linear pseudo-boolean constraint problems. In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 7–19, Le Croisic, France, March 2002.
7. S.D. Prestwich. Supersymmetric Modeling for Local Search. In *Second International Workshop on Symmetry in Constraint Satisfaction Problems*, 2002.
8. S.D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research*, 118:137–150, 2003.
9. J.F. Puget. Symmetry breaking revisited. *Constraints*, 2005. To appear in the Special Issue on CP'02.
10. M. Sellmann. *Reduction Techniques in Constraint Programming and Combinatorial Optimization*. PhD thesis, University of Paderborn, Germany, 2003.
11. Meinolf Sellmann and Warwick Harvey. Heuristic constraint propagation – Using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem. In Narendra Jussien and François Laburthe, editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 191–204, Le Croisic, France, March 2002.
12. B. Smith. Reducing Symmetry in a Combinatorial Design Problem. In *CP-AI-OR'2001*, pages 351–359, Wye College (Imperial College), Ashford, Kent UK, April 2001.