

Scheduling Social Tournaments Locally

Iván Dotú¹ and Pascal Van Hentenryck²

¹ Departamento De Ingeniería Informática, Universidad Autónoma de Madrid

² Brown University, Box 1910, Providence, RI 02912

Abstract. Tournament scheduling, such as the social golfer problem, has attracted significant attention in recent years because of their highly symmetrical and combinatorial nature. This paper presents an effective local search algorithm for a variety of tournament scheduling problems, including social golfer problems, debating tournaments, judge assignments, and very social golfers. The algorithm finds high-quality solutions on all problems, including new solutions to open problems. Interestingly, the algorithm does not incorporate any symmetry-breaking schemes and is conceptually simple when compared to advanced constraint-programming solutions.

1 Introduction

Tournament scheduling problems arise in many practical applications and their highly symmetric and combinatorial nature makes them particularly challenging for search algorithms. It is thus not surprising that tournament scheduling problems have generated significant attention from the constraint programming community (e.g., [7, 16, 10, 15, 14, 3, 13]). The social golfer in particular has become one of the standard benchmarks for evaluating symmetry-breaking schemes. Recent developments (e.g., [3, 13]) approach this problem using innovative, elegant, yet complex, symmetry-breaking schemes.

This paper describes the local search approach to tournament scheduling, initially proposed in [6]. It demonstrates that this approach can be successfully applied to the social golfer problem but also to other challenging, real-life, social tournaments, including a debating tournament, a judge assignment problem, and the very social golfer problem. The underlying algorithm is conceptually simple and uses “natural” modelings of these applications, relatively simple neighborhoods, and a traditional tabu-search meta-heuristic. Moreover, it does not incorporate any symmetry-breaking scheme.

The main contribution of the paper is thus to show the versatility and effectiveness of local search for a wide variety of social tournaments. In particular, it shows that

1. the algorithm finds solutions to all instances solved by constraint programming (except 2) and solves many new instances.
2. the algorithm schedules a challenging debating tournament which was open before and exhibits excellent performance across a wide range of such instances.
3. the algorithm finds the best-known solution to a real-life, judge-scheduling problem which superimposes a referee assignment over a debating tournament.
4. the approach finds high-quality solutions to the very social golfer problem where the *atmost* constraints are replaced by *atleast* constraints.

It is important to emphasize that these problems are not academic: social tournaments are increasingly pervasive and arise in very diverse settings. The applications described in this paper are also prototypical of many sport competitions. Moreover, they are extremely challenging computationally and have attracted significant research in recent years. In particular, these applications have been tackled by a variety of algorithms and could not be solved so far. In addition, earlier local search algorithms (e.g., [1]) are not competitive with the approach presented here. The results also complement earlier work on sport scheduling in [2, 17], where simulated annealing was shown to outperform other approaches.

The rest of the paper is organized as follows. The first section describes the generalized social golfer problem and the basic algorithm. The next sections present the rest of the problems and show how to adapt the algorithm to these new applications. The paper concludes with related work and some open issues.

2 The Social Golfer

The first application is the well-known social golfer problem has attracted significant interest since its posting on `sci.op-research` in May 1998. The social golfer 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.

2.1 The Modeling

There are many possible modelings for the social golfer problem, which is one of the reasons it is so interesting. This paper uses a modeling which 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:

1. A golfer plays exactly once a week;
2. Two golfers can play together 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

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

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, and generalizing

$$v_\sigma(m[a, b]) = \max(0, \#_\sigma(a, b) - 1). \quad (2)$$

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 solution golfer problem.

2.2 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

$$\exists p' \in P : v_\sigma(m[\sigma(x[w, g, p]), \sigma(x[w, g, p'])]) > 1. \quad (3)$$

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

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

2.3 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 stores 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 $[4, 100]$. 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 result, for schedule σ and iteration k , the neighborhood consists of the set of moves $\mathcal{S}^t(\sigma, k)$ defined as

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

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

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

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.

2.4 The Tabu-Search Algorithm

We are now ready to present the basic local search algorithm SGLS. The algorithm, depicted in Figure 1, 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 are 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 which improve 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

$$\text{week}(\langle w, g, p \rangle) = w,$$

and 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).

2.5 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 [4] 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 1 and 2, which will be used to explain the intuition underlying the constructive heuristic.

1. $SGLS(W, G, P)$
2. **forall** $w \in W$
3. $tabu[w] \leftarrow \{\}$;
4. $\sigma \leftarrow$ random configuration;
5. $\sigma^* \leftarrow \sigma$;
6. $k \leftarrow 0$;
7. $s \leftarrow 0$;
8. **while** $k \leq maxIt$ & $f(\sigma) > 0$ **do**
9. **select** $(t_1, t_2) \in \mathcal{S}^t(\sigma, k) \cup \mathcal{S}^*(\sigma, \sigma^*)$
 minimizing $f(\sigma[x[t_1] \leftrightarrow x[t_2]])$;
10. $\tau \leftarrow \text{RANDOM}([4, 100])$;
11. $tabu[week(t_1)] \leftarrow$
 $tabu[week(t_1)] \cup \{(\sigma(x[t_1]), \sigma(x[t_2]), k + \tau)\}$;
12. $\sigma \leftarrow \sigma[x[t_1] \leftrightarrow x[t_2]]$;
13. **if** $f(\sigma) < f(\sigma^*)$ **then**
14. $\sigma^* \leftarrow \sigma$;
15. $s \leftarrow 0$;
16. **else if** $s > maxStable$ **then**
17. $\sigma \leftarrow$ random configuration;
18. $s \leftarrow 0$;
19. **forall** $w \in W$ **do**
20. $tabu[w] = \{\}$;
21. **else**
22. $s++$;
23. $k++$;

Fig. 1. Algorithm SGLS for Scheduling Social Golfers

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 2 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 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 1 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.

| 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 1. The initial configuration for the problem 4 – 3 – 3

| 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 |

Table 2. The initial configuration for the problem 5 – 5 – 6

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, \dots, P_{w-1} .

The top-level function is HEURISTICSCHEDULE which specifies the first week and calls function SCHEDULEWEEK 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 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.

In [9], Warwick Harvey observed that the above heuristic is a special case of a construction to find Mutually Orthogonal Latin Squares (MOLS) when the order of the Latin Squares is prime or prime power. The above heuristic only finds solutions when the order is prime but Harvey showed how to generalize it to find solutions when the order is a prime power as well. The generalized heuristic is described in detail in [9]

1. HEURISTICSCHEDULE(w, g, p)
2. $n \leftarrow g \times p$;
3. $P_0 \leftarrow \langle 1, \dots, n \rangle$;
4. **forall** $we \in 1..w - 1$
5. $P_{we} \leftarrow \text{scheduleWeek}(we, g, p, n)$;

6. SCHEDULEWEEK(we, g, p, n)
7. $P_{we} \leftarrow \langle 1 \rangle$;
8. $po \leftarrow 0$;
9. $gr \leftarrow 1$;
10. $\Delta \leftarrow we - 1$;
11. **forall** $go \in 1..n - 1$
12. $s \leftarrow \text{SELECT}(gr, (po + \Delta)\%p)$;
13. $po \leftarrow \text{POSITION}(s)$;
14. $gr \leftarrow (gr + 1)\%g$;
15. $P_{we} \leftarrow P_{we} :: \langle s \rangle$;
16. **return** P_{we} ;

Fig. 2. The Constructive Heuristic for Scheduling Social Golfers

and is based on the algebraic concept of a Galois field, i.e., finite fields closed under addition and multiplication.

2.6 Experimental Results

This section reports the experimental results for the social golfer problem.

The Basic Local Search We first report the experimental results for algorithm SGLS without using the constructive heuristic as a starting point. 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, depicted in Tables 3 and 4, report average values for successful runs and the percentage of unsuccessful runs (if any). 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 3 reports the number of iterations (moves), while Table 4 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 as available at the start of this research.³

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 solved 7 instances, i.e., $9 - 4 - 9$, $9 - 5 - 7$, $9 - 6 - 6$, $9 - 7 - 5$, $9 - 8 - 4$, $10 - 5 - 8$ and $10 - 9 - 4$, for the

³For the current statuses of the instances, see Warwick Harvey's web page at www.icparc.ic.ac.uk/wh/golf.

| g | size 3 | | size 4 | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | | size 10 | |
|-----------|-----------|-----------------|----------|-----------------|----------|----------------|----------|---------------|----------|--------------|----------|-------------|----------|-------------|----------|-------------|
| | 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 3. Number of Iterations for SGLS with Maximal Number of Weeks.
Bold Entries Indicate a Match with the Best Known Number of Weeks.

| g | size 3 | | | size 4 | | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | | size 10 | |
|-----------|-----------|--------------|----------|---------------|----------|-------------|---------------|-------------|-------------|-------------|-------------|-------------|----------|-------------|----------|-------------|---------|---|
| | 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 | - | - |

Table 4. CPU Time in Seconds for SGLS with Maximal Number of Weeks.
Bold Entries Indicate a Match with the Best Known Number of Weeks.

first time [6]. For simplicity, the computational results for these entries are not shown in the tables but will be given for the local search with the constructive heuristic.

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 [3, 13] which report the use of very interesting symmetry-breaking schemes to schedule social golfers. There is also a complementary between constraint programming and local search: instances that are hard for local search seem easy for constraint programming and vice-versa.

The Local Search with the Constructive Heuristic Table 5 depicts the set of instances solved by SGLS-CH. Once again, the table only gives the results for those instances $g - p - w$ maximizing w (for a given q and p) since they also provide solutions for $w' < w$. The table also reports in column $\Delta(w)$ by how much they improve/degrade the best number of weeks (as known at the beginning of this research). Table 6 reports the CPU Times for the instances solved by SGLS-CH.

The results show that the combination of local search with a good starting point is very effective on the social golfer problem. The algorithm finds all constraint programming solutions but 2 and obtains 12 new solutions. The computation times remain reasonable, although they increase on most of the (previously) open instances. They remain below 10 minutes however.

| | size 3 | | size 4 | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | | size 10 | |
|-----------|-----------|-------------|-----------|-------------|----------|-------------|----------|-------------|----------|-------------|----------|-------------|-----------|-------------|---------|-------------|
| g | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ | w | $\Delta(w)$ |
| 6 | 8 | 0 | 6 | -1 | 6 | 0 | 3 | 0 | - | - | - | - | - | - | - | - |
| 7 | 9 | 0 | 7 | 0 | 7 | 2 | 5 | 1 | 8 | 0 | - | - | - | - | - | - |
| 8 | 10 | 0 | 8 | -1 | 6 | 0 | 8 | 3 | 4 | 0 | 9 | 5 | - | - | - | - |
| 9 | 12 | 1 | 9 | 1 | 7 | 1 | 9 | 4 | 5 | 1 | 4 | 1 | 10 | 7 | - | - |
| 10 | 13 | 0 | 10 | 1 | 8 | 1 | 6 | 0 | 5 | 0 | 4 | 0 | 4 | 1 | 3 | 0 |

Table 5. Maximal Number of Weeks for SGLS-CH.
Bold Entries Indicate Improvement over Best Known Solutions.

| | size 3 | | | size 4 | | | size 5 | | | size 6 | | | size 7 | | | size 8 | | | size 9 | | | size 10 | | |
|-----------|-----------|---------------|-----------|-----------|---------------|----------|---------------|----------|---------------|----------|---------------|----------|-------------|-----------|-------------|--------|------|---|--------|---|-----|---------|-----|--|
| g | w | T | %F | w | T | w | T | w | T | w | T | w | T | w | T | w | T | w | T | w | T | w | T | |
| 6 | 8 | 51.93 | 5 | 6 | 49.98 | 6 | 143.72 | 3 | 0.01 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | |
| 7 | 9 | 2.58 | | 7 | 99.26 | 7 | 45.56 | 5 | 479.88 | 8 | 0.01 | - | - | - | - | - | - | - | - | - | - | - | - | |
| 8 | 10 | 0.23 | | 8 | 188.17 | 6 | 0.27 | 8 | 1.55 | 4 | 0.23 | 9 | 0.01 | - | - | - | - | - | - | - | - | - | - | |
| 9 | 12 | 584.76 | 60 | 9 | 471.39 | 7 | 5.78 | 9 | 269.62 | 5 | 199.93 | 4 | 2.46 | 10 | 0.01 | - | - | - | - | - | - | - | - | |
| 10 | 13 | 23.45 | | 10 | 431.31 | 8 | 159.67 | 6 | 0.26 | 5 | 0.34 | 4 | 0.22 | 4 | 28.78 | 3 | 0.01 | - | - | - | - | - | - | |

Table 6. CPU Time in Seconds for SGLS-CH with Maximal Number of Weeks
Bold Entries Indicate Improvement over Best Known Solutions.

3 The Debating Tournament Problem

We now turn to the debating tournament problem (DTT), a problem that can be seen as a generalization of the social golfer. The debating tournament problem is a real-life application given to us by W. Harvey [8]:

A debating tournament involves 15 students. The students are grouped into 3 groups of 5 in each round, for 9 rounds. Due to the nature of the scoring system, it is desired to have as balanced a schedule as possible (i.e. each pair meets about the same number of times as every other pair). Having each pair meet 2 or 3 times is as balanced as you're going to get. I don't know whether this is possible.

In the best known solution, obtained by constraint programming, every two students meet at most four times. The local search algorithm presented here closes this DDT instance and finds a solution where every two students meet at most three times. The experimental results also consider a variety of the DTTs which are generalizations of the social golfer problem, where golfers are allowed to meet up to k times.

3.1 The Modeling

The DTP can be modeled like the SGP by relaxing the cardinality constraint, i.e., replacing the value "1" by k in equations 1, 2, and 3, that define the constraints, the violations, and the neighborhood.

| g | size 3 | | size 4 | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | | size 10 | |
|-----------|--------|------------|--------|----------|--------|-------------|--------|------------|--------|------|--------|------------|--------|-----|---------|------|
| | w | T | w | T | w | T | w | T | w | T | w | T | w | T | w | T |
| 3 | 8 | 0 | 6 | 0 | 6 | 0.0 | 6 | 0.0 | 4 | 0 | 4 | 0.0 | 4 | 0.0 | 2 | - |
| 4 | 11 | 0.3 | 10 | 0.5 | 8 | 34.7 (90) | 6 | 0.0 | 6 | 6.7 | 7 | 107.3 (60) | 6 | 2.2 | 6 | 2.5 |
| 5 | 14 | 9.7 (19) | 11 | 0.5 | 9 | 0.1 | 8 | 0.1 | 7 | 0.1 | 7 | 30.4 (4) | 6 | 0.1 | 6 | 1.5 |
| 6 | 16 | 0.1 | 13 | 0.1 | 12 | 102.54 (99) | 10 | 0.3 | 9 | 0.4 | 8 | 0.3 | 7 | 0.2 | 7 | 0.5 |
| 7 | 19 | 1 | 15 | 0.1 | 13 | 0.2 | 12 | 4.0 | 11 | 47.3 | 10 | 9.5 | 9 | 1.4 | 8 | 0.8 |
| 8 | 22 | 12.7 (2) | 18 | 12.5 (1) | 15 | 0.4 | 14 | 152.5 (23) | 12 | 1.1 | 11 | 1.4 | 10 | 1.5 | 10 | 23.3 |
| 9 | 25 | 103.3 (86) | 20 | 1.2 | 17 | 0.8 | 15 | 1.2 | 14 | 4.7 | 13 | 12.3 | 12 | 9.3 | 11 | 5.4 |
| 10 | 27 | 0.3 | 22 | 0.7 | 19 | 1.4 | 17 | 2.5 | 15 | 2.8 | 14 | 4.5 | 13 | 6.0 | 12 | 6.5 |

Table 7. CPU Times in Seconds for DDT(2) with the Maximal Number of Weeks.

| g | size 3 | | size 4 | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | | size 10 | |
|-----------|--------|---------|--------|------------|--------|------|--------|------|--------|-----------|--------|-----------|--------|-----------|---------|----------|
| | w | T | w | T | w | T | w | T | w | T | w | T | w | T | w | T |
| 3 | 12 | 0.0 | 11 | 2.1 | 9 | 0.1 | 9 | 0.1 | 9 | 0.5 | 9 | 0.4 | 9 | 0.4 | 7 | 30.3 (7) |
| 4 | 16 | 0.1 | 14 | 6.0 (3) | 12 | 0.1 | 11 | 0.2 | 10 | 0.1 | 9 | 0.1 | 9 | 0.5 | 8 | 0.1 |
| 5 | 21 | 5.7 (3) | 17 | 0.1 | 15 | 0.1 | 14 | 0.6 | 13 | 1.3 | 12 | 0.6 | 11 | 0.4 | 11 | 8.5 |
| 6 | 25 | 5.0 | 21 | 2.0 | 18 | 0.2 | 17 | 2.1 | 16 | 61.4 (2) | 15 | 107.6 (7) | 14 | 7.8 | 13 | 1.9 |
| 7 | 29 | 0.4 | 25 | 144.2 (88) | 22 | 12.3 | 20 | 7.7 | 18 | 1.3 | 17 | 2.3 | 16 | 2.8 | 15 | 3.0 |
| 8 | 33 | 0.2 | 28 | 0.8 | 25 | 2.8 | 23 | 19.4 | 21 | 4.0 | 20 | 27.1 | 19 | 101.9 | 18 | 36.5 |
| 9 | 38 | 82(19) | 32 | 19.8 | 28 | 2.2 | 26 | 35.6 | 24 | 15.3 | 22 | 7.3 | 21 | 11.7 | 20 | 16.7 |
| 10 | 42 | 2.2 | 35 | 1.5 | 31 | 2.9 | 29 | 64.0 | 27 | 334.5 (3) | 25 | 22.8 | 24 | 372.5 (1) | 22 | 21.2 |

Table 8. CPU Times in Seconds for the DTP(3) with the Maximal Number of Weeks.

3.2 Experimental Results

Tables 7 and 8 report the experimental results for SGLS for $k = 2, 3$ when the maximum number of moves was set to 100.000. Once again, the tables only give the results for those instances $g-p-w$ maximizing w since they also provide solutions for $w' < w$.

First observe that SGLS solves the real-life instance 3-5-9 for $k = 3$ in 0.1 second. This instance is in fact particularly challenging for systematic search, since constraint programming solutions were only able to find a solution with $k = 4$, demonstrating the effectiveness of SGLS [8].

The remaining results are not easy to comment since no other results are available at this point. It is useful however to make a few observations. A solution to the debating tournament problem for a given k could be obtained by solving the corresponding social golfer instance and repeating its schedule k times. Of course, such a solution is not really desirable in practice, since the exact same players will be meeting k times, but it provides a lower bound $k \times w$ on the maximum number of weeks (where w is the maximal number of weeks for the social golfer problem).

Consider now the results for the social golfer problem without the constructive heuristics. The experimental results depicted in Tables 7 and 8 indicate that SGLS

| Players | group 1 | group 2 | group 3 | Judges | group 1 | group 2 | group 3 |
|---------|-------------|--------------|---------------|--------|----------|----------|---------|
| week 1 | 3 9 0 8 5 | 1 10 2 7 4 | 6 12 11 14 13 | week 1 | 7 6 12 | 15 9 11 | 4 16 2 |
| week 2 | 11 8 9 0 7 | 14 10 1 6 3 | 2 4 5 12 13 | week 2 | 2 10 4 | 7 8 12 | 1 17 14 |
| week 3 | 0 4 9 14 10 | 3 2 11 6 5 | 1 8 7 12 13 | week 3 | 1 16 13 | 8 10 0 | 6 11 3 |
| week 4 | 1 2 3 8 11 | 0 6 7 4 12 | 14 9 5 12 10 | week 4 | 14 16 13 | 10 9 2 | 11 6 3 |
| week 5 | 9 13 4 3 1 | 11 12 7 10 5 | 14 8 0 6 2 | week 5 | 0 10 2 | 1 8 15 | 17 9 3 |
| week 6 | 8 12 9 10 6 | 2 0 5 13 1 | 11 14 3 4 7 | week 6 | 5 4 14 | 12 7 16 | 6 17 13 |
| week 7 | 14 3 2 12 0 | 9 1 6 5 7 | 13 11 4 8 10 | week 7 | 5 11 15 | 14 17 13 | 0 7 12 |
| week 8 | 9 11 2 4 6 | 14 8 1 12 5 | 0 7 13 10 3 | week 8 | 3 12 7 | 10 9 0 | 14 4 5 |
| week 9 | 6 8 3 5 4 | 9 2 14 13 7 | 0 10 1 12 11 | week 9 | 11 15 16 | 8 1 5 | 6 3 17 |

Table 9. A Solution to the 3 – 5 – 9 Judge Assignment Problem with 3 Judges per Group.

improves this (previous) lower bound in almost every instance and matches it in the remaining ones. Moreover the improvements become more significant as the number of groups and the group sizes grow. It is also interesting to emphasize that most instances are solved really quickly (i.e., in less than a second in many cases), confirming the results of the social golfer.

Finally, observe that instance 6-6-3 is optimal for the social golfer [14]. However, its special structure does not carry over to larger k , since SGLS finds schedules with up to 10 weeks ($k = 2$) and 17 weeks ($k = 3$).

4 The Judge Assignment Problem

This section considers the judge assignment problem, another real-life problem (also given to us by W. Harvey [8]). The problem can be viewed as superimposing a judge assignment on top of a debating tournament instance. It was stated as follows:

This problem involves the assignment of judges to the player schematic. Every player is required to furnish the tournament director with one judge. Each judge has to judge whichever sections in whichever rounds that the tournament director assigns. Every section is judged by exactly three judges. A judge can not judge two or more sections in the same round (because all three sections in a round are played simultaneously), and a judge can not judge any section in which his own player is playing. The tournament director wants to make sure that no judge judges any particular player more than twice over the nine rounds. Ideally, each judge should judge each player the same number of times (or something close to that). Given a particular player schematic, is it possible to get away with using only the fifteen judges provided by the players, and still meet the constraints? If not, the goal is to get close; the tournament director has extra judges available for spot duty, but it is better to use them as little as possible, for all kinds of good and valid reasons. The judge schematic is generated after the player schematic.

The algorithm described in this section only considers the judge schematic: the player schematic is obtained first and is a debating tournament problem. The original problem

superimposes the judge assignment on the real-life instance of the debating tournament. The experimental results consider a variety of instances and the goal is to minimize the number of additional judges while satisfying the feasibility constraints. Figure 9 depicts a solution to the judge assignment problem for the (real-life) 3-5-9 debating tournament problem and 3 judges per group. This solution uses three additional judges only.

The Modeling The modeling is also similar to the SGP. It receives, as input, a solution to the debating tournament problem. In particular, $x[w, g, p]$ will denote the player scheduled in position p of group g in week w . The decision variables $y[w, g, p]$ associates a decision variable with every week, group, and position, where the set P_r of positions for the judges is included in the set of positions of the debating tournament (i.e., $P_r \subset P$). The goal is to find a schedule σ , where the value $\sigma(y[w, g, p])$ denotes the judge scheduled in position $\langle w, g, p \rangle$. In the following, the judge provided by player p is denoted j_p , the set of judges by \mathcal{J} , and the set of players by \mathcal{P} . There are four kinds of constraints in the referee assignment.

1. A judge judges exactly once a week;
2. Each group is composed of three judges;
3. Judges can grade the same player at most twice.
4. No judge can grade his own player.

The first and second types of constraints are implicit in the algorithms presented in this paper: they are satisfied by the initial assignments and preserved by local moves. The third and fourth sets of constraints are represented explicitly. The model contains a constraint $m[a, b]$ for every pair (a, b) of judge-player: Constraint $m[a, b]$ holds for an assignment σ if judge a does not grade player b more than twice (if judge a is not the judge furnished by player b). More precisely, if $\#_\sigma(a, b)$ denotes the number of times judge a judges player b in σ , i.e.,

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

constraint $m[a, b]$ holds if

$$\#_\sigma(a, b) \leq \begin{cases} 2 & \text{if } b \neq j_a; \\ 0 & \text{otherwise.} \end{cases}$$

The violations $v_\sigma(m[a, b])$ of a constraint $m[a, b]$ is the number of times judge a judges player b in schedule σ beyond their allowed meeting, i.e.,

$$v_\sigma(m[a, b]) = \max(0, \#_\sigma(a, b) - (\text{if } b \neq j_a \text{ then } 2 \text{ else } 0)).$$

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

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

| instances | 3-5-7 | 3-5-8 | 3-5-9 | 4-5-10 | 4-5-11 | 4-5-12 | 5-5-13 | 5-5-14 | 5-5-15 |
|-----------------|-----------|---------|---------|--------|---------|--------|--------|-----------|--------|
| Nb. Add. judges | 0 (50) | 2 (20) | 3 (80) | 0 (10) | 1 (90) | 3 | 0 | 1 (50) | 3 |
| Failures (%) | 50 | 20 | 80 | 10 | 90 | 0 | 0 | 50 | 0 |
| Iterations | 2506261.5 | 16559.5 | 5000000 | 76517 | 5000000 | 75173 | 59219 | 4674736.5 | 297947 |
| Med. time | 300.48 | 1.52 | >1200 | 15.08 | >1200 | 19.84 | 20.97 | 3458.23 | 128.65 |
| Avg. time | 0.48 | 2.39 | 35.86 | 26.88 | 189.48 | 61.98 | 52.68 | 543.39 | 145.8 |
| Med. time (+1) | 0.03 | - | 0.86 | - | 4.59 | - | - | 5.3 | - |

Table 10. Experimental Results on the Referee Assignment Problem.

The Neighborhood The neighborhood in the judge assignment problem also consists of swapping judges in each week. However, since there are more judges than necessary each week (since $P_r \subset P$), it is important to introduce insertion moves that swap a scheduled and an uncheduled judge for a given week. The set of moves is thus the union $\mathcal{S} \cup \mathcal{I}$ of swap moves

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

and insertion moves

$$\mathcal{I} = \{(\langle w, g, p \rangle, j) \mid j \notin J_\sigma(w)\}$$

where $J_\sigma(w)$ denotes all judges scheduled in week w by σ . Once again, the neighborhood will be restricted to consider only moves whose variables are involved in a violation, i.e., $y[w, g_1, p_1]$ for swaps and $y[w, g, p]$ for insertions.

The Tabu Component The tabu component of the algorithm exploits the same principles as in earlier problems. For a given week w , the tabu list maintains triplet $\langle j_a, j_b, i \rangle$, where j_a and j_b are two judges and i represents the first iteration where judges j_a and j_b can be swapped again in week w . Observe that this tabu list captures both swap and insertion moves. In addition to the non-tabu moves, the neighborhood also includes the traditional aspiration criterion and considers moves that improve the current best solution. It is not difficult to upgrade SGLS to include these changes.

Experimental Results Table 10 depicts the experimental results on a variety of instances related to the real-life problem. It reports the minimal number of additional judges found by SGLS for these instances, the percentage of failures of the algorithm with 5,000,000 iterations, the median number of iterations, the average time for the successful runs, and, for those instances where the failure is above 50%, the median time when an additional judge is included. First observe that SGLS solves the real-life problem with 3 additional judges in about 35 seconds in 20% of the case. For the other instances, SGLS finds solutions with no additional judge, one additional judge, or at most 3 additional judges. These instances are typically more challenging for SGLS than the earlier tournament problems, but the running times are completely acceptable. Note that the running times decrease significantly when an additional judge is included.

| | size 3 | | size 4 | | size 5 | | size 6 | | size 7 | | size 8 | | size 9 | |
|---|--------|------|--------|------------|--------|-------------|--------|-------------|--------|-------|--------|--------|--------|---|
| g | w | T | w | T | w | T | w | T | w | T | w | T | w | T |
| 6 | 9 | 4.68 | 9 | 26.52(20%) | 10 | 0.18 | 10 | 2.03 | - | - | - | - | - | - |
| 7 | 11 | 0.12 | 11 | 0.64 | 10 | 101.4(90%) | 9 | 30.68 | 8 | 0 | - | - | - | - |
| 8 | 13 | 0.07 | 13 | 0.19 | 13 | 1.48 | 11 | 54.44 | 11 | 24.73 | 9 | 0 | - | - |
| 9 | 14 | 6.68 | 14 | 7.75 | 14 | 239.79(80%) | 14 | 572.96(90%) | 13 | 49.15 | 12 | 141.02 | 10 | 0 |

Table 11. Smallest Number of Weeks and Time in Seconds for the Very Social Golfer Problem. The Percentage of Failures (if any) is Shown Between Parentheses.

5 The Very Social Golfer Problem

We now consider the very social golfer problem, i.e., the social golfer where the *atmost* constraints on the number of meetings is replaced by an *atleast* constraints. The goal here is thus to find a schedule where two golfers play together (i.e., in the same group of the same week) and where the number of weeks is minimized. In the very social golfer problem, the constraints are of a fundamentally different nature and it is thus interesting to determine whether the local search algorithm is also appropriate in this setting.

5.1 The Modeling

This generalization can be modeled like the SGP by changing the cardinality constraint, i.e., changing the direction of the inequality symbol in equations 1, 2, and 3, that define the constraints, the violations, and the neighborhood. For instance, constraint $m[a, b]$ now holds if

$$\#_{\sigma}(a, b) \geq 1.$$

5.2 Experimental Results

Table 11 depicts the results for the very social golfer when the algorithm was limited to 20000 moves. The table reports the smallest number of weeks found by the algorithm, as well as the CPU Time in seconds. Once again, it is difficult to compare the results since there is no prior work. However we can make two observations.

First, for instances of the form $\langle g, p, w \rangle$ where $g = p$ and g is *prime or prime power*, there is a complete solution where all players meet exactly once in the same group. Thus, for those instances, the number of weeks will be equal to that of the corresponding instance of the social golfer problem. The constructive heuristic described earlier achieves the optimal number of weeks for the mentioned instances.

For the remaining instances, a simple counting argument enable us to show that the obtained solutions are of high-quality. Consider $g = 9$ and $p = 3$ for simplicity. There are 27 players and hence the solution must contain $27 \times 26/2$ meetings. Each week can only schedule 9×3 meetings, which gives a lower bound of 13 weeks. The results show that the algorithm produces a schedule with one additional week when compared to the lower bound.

6 Related Work

There is a considerable body of work on scheduling social golfers in the constraint programming community. References [3, 13] describe state-of-the art results using constraint programming and are excellent starting points for more references. See also [14] 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. The neighborhood used in this paper, which implicitly maintain the group and week structures, and the randomized tabu-list strategy are fundamental in scheduling hard instances. The algorithm for the social golfer was first presented in [6], where a constructive heuristic (for primes only) was proposed and detailed experimental results are given. The improved constructive heuristics was proposed in [9]. The debating tournament, judge assignment, and generalizations of the social golfers were given to us by W. Harvey [8]. On the debating tournament, only a solution with $k = 4$ was known and the approach presented here closed the problem by providing a balanced solution with $k = 3$. No solutions were known on the judge assignment.

7 Conclusion

This paper proposed the local search algorithm SGLS for tournament scheduling and applied it to the social golfer, the debating tournament, the judge assignment problem, and generalizations of the social golfers. A posteriori, the algorithm is conceptually simple: it uses “natural” modelings of social tournaments and it ignores their highly symmetric nature. Yet the algorithm is shown to be particularly effective and robust. In particular, it was able to schedule real-life applications that were open so far despite the considerable research devoted to social tournaments in recent years. In particular, on the social golfer problem, it finds solutions to all instances solved by constraint programming (except 2) and solves many novel instances. On debating tournaments, SGLS solves an open, real-life, instance and exhibits excellent performance over a wide variety of instances. On judge assignments, SGLS finds a high-quality solution to a real-life instance with few additional judges and is very effective on other instances as well. On the very social golfers, the algorithm also yields high-quality solutions for many instances.

There are some intriguing open issues raised by this research. On the social golfer problem, the experimental results show a nice synergy between constraint programming and local search, since instances that are easy for one technology seem to be hard for the other. It would be interesting to determine if this holds for the debating tournament and the judge assignment problems. It would also be interesting to find out whether more sophisticated neighborhoods may improve the quality of local search on the more constrained instances.

Moreover, we are interested by the effect of the seeding heuristic. It not only constructs optimal solutions for several instances, but represents an effective starting point

for the algorithm. However, we believe that a deeper study on its effects should be performed in order to adapt the heuristic to certain instances and to develop new intensification and diversification mechanisms.

8 Acknowledgements

Special thanks to Warwick Harvey for many interesting discussions. This work was partially supported by NSF ITR Awards ACI-0121497.

References

1. M. Ågren. Solving the Social Golfer Using Local Search. peg.it.uu.se/saps02/MagnusAgren/, 2003.
2. A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A Simulated Annealing Approach to the Traveling Tournament Problem. In *CP-AI-OR'03*. 2003.
3. N. Barnier and P. Brisset. Solving Kirkman's Schoolgirl Problem in a Few Seconds. *Constraints*, 10(1), 7–21, 2005.
4. C. Colbourn and Dinitz. *The CRC Handbook of Combinatorial Design*. CRC Press, Boca Raton, FL, 1996.
5. 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.
6. Dotú, I., and Van Hentenryck, P. Scheduling Social Golfers Locally. In *Proceedings of the Second International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, Prague, May 2005.
7. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In Toby Walsh, editor, *Principles and Practice of Constraint Programming*, volume 2293 of *LNCS*, pages 93–107. Springer Verlag, 2001.
8. W. Harvey. Personal communication. 2004.
9. W. Harvey and T. Winterer Solving the MOLR and Social Golfers Problems. In *Proceedings of the 11th International Conference on Constraint Programming (CP-2005)* Stiges, Spain, September 2005.
10. 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.
11. S.D. Prestwich. Supersymmetric Modeling for Local Search. In *Second International Workshop on Symmetry in Constraint Satisfaction Problems*, 2002.
12. S.D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research*, 118:137–150, 2003.
13. J.F. Puget. Symmetry breaking revisited. *Constraints*, 10(1), 23–46, 2005.
14. M. Sellmann. *Reduction Techniques in Constraint Programming and Combinatorial Optimization*. PhD thesis, University of Paderborn, Germany, 2003.
15. 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.

16. 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.
17. P. Van Hentenryck and Y. Vergados. Minimizing Breaks in Sport Scheduling with Local Search. In *ICAPS'05*. 2005.