Simulation Tools and Techniques Project

Social Golfer Problem

Vladimír Marcin (xmarci10)[1]

FIT, Brno University of Technology, Brno, Czech Republic
xmarci10@stud.fit.vutbr.cz

1 Introduction

In this project, a selected *combinatorial optimization problem* is described according to the certain published research paper. This paper, as well as other resources, are stated in Section 1.1. A given problem is defined in Section 1.2. Approaches of a solution to the problem from the paper are re-implemented and experimental results are finally compared with the results in the paper.

The rest of the project documentation is organised as follows. An approach of our solution is presented in Chapter 2. Chapter 6 presents some observations and findings based on which we were able to improve the implemented method. Re-implementation of the methods from [4] is then described in Chapter 4. Experimental evaluation is discussed in Chapter 5 and finally, Chapter 6 concludes the documentation and project results.

1.1 Resources

The project is based on the paper [4], which improves the methods presented in [1]. Almost the same paper with slight differences was previously published in [5]. A more detailed description of a used constraint solver can be found in [7] and catalogue of used constraints can be found in [2].

1.2 Social Golfer Problem

The SGP, i.e., 010 problem in CSPLib [3], is a typical combinatorial optimization problem that has attracted significant attention from the constraints community because of its highly symmetrical and combinatorial nature. According to the constraints community's convention on this problem, an *instance* of the SGP is denoted by a triple g-s-w, where g is the number of groups, s is the number of players within a group, and w is the number of weeks in the schedule.

Definition 1. [4]**: Social Golfer Problem**

In a golf club, $\boldsymbol{n} = \boldsymbol{g} * \boldsymbol{s}$ golfers want to play in $\boldsymbol{g}$ groups of $\boldsymbol{s}$ golfers for $\boldsymbol{w}$ weeks. Does there exist a schedule for each golfer to play no more than once with any other golfer ?

The SGP can also be regarded as a discrete optimization problem that maximizes the number of weeks w^* for a given g and s, where $w^* \leq \frac{g*s-1}{s-1}$. Clearly a solution for an instance g-s-w^* indicates itself as the solution for all instances g-s-w with $0 < w < w*$. In practice, the computational difficulty of solving the g-s-w^* and g-s-w instance is often no in the same order of magnitude. In the light of this, paper [4] concentrates on solving the g-s-w^* instances with the maximal number of weeks, which are called the *full instances*.

2 Constraint satisfaction approach for solving the SGP

The constraint programming (CP) is a powerful technique to tackle combinatorial problems, generally $\mathcal{NP}$-complete or $\mathcal{NP}$-hard. The idea behind the CP is that a user states a problem by using constraints and a general purpose constraint solver is used to solve the problem. The classic definition of a constraint satisfaction problem (CSP) is as follows. A CSP P is a triple $\langle X, D, C\rangle$, where $X = \{x_0, \ldots, x_n\}$ is a set of decision variables, $D = \{D_{(x_o)}, \ldots, D_{(x_n)}\}$ contains associated finite domains for each variable in X, and $C = \{c_0, \ldots, c_t\}$ is a collection of constraints. Each constraint $c_i \in C$ is a relation defined over a subset of X, and restricts the values that can be simultaneously assigned to these variables. A solution of a CSP is a complete instantiation satisfying all constraints of the CSP P. The present paper involves the use of several types of constraints, including the **GCC, AllDifferent, Count, Table,** and **Arithmetic** constraints [2].

2.1 The Basic Model

There are various ways of modeling the SGP as a CSP proposed in the literature, which is one of the reasons why the problem is so compelling. In the [4], they used a model presented in [1] and added more constraints into the model to tackle larger instances piece by piece.

The decision variables of the model is a $w \times n$ matrix G in which each element $G_{i,j}$ of the matrix G represents that player j is assigned to group $G_{i,j}$ in week i (cf. Table 1). Hence, the domain of decision variable $G_{i,j}$ is a set of integers $\{1, \ldots, g\}$, where $0 \leq i < w,\ 0 \leq j < n$. The major advantage of the decision variables defined in this model is that the range of the variables is reduced from $\{1, \ldots, n\}$ to $\{1, \ldots, g\}$ while keeping an unchanged number of variables, compared to the naive model. Another benefit of this representation is that, easily observable "patterns" can be used to define instance-specific constraints.

The first constraint of the basic model freezes the elements of the first row to, at least partially, mitigate the symmetries caused by renumbering golfers. Formally:

$$\forall j \in J(G_{0,j} = j/s + 1), J = \{j \in \mathbb{Z} \mid 0 \leq j \leq n\} \quad (1)$$

where the operator "/" denotes the integer division. Equation (1) produces a sequence of integers from 1 to g in non-descending order, and each integer repeats exactly s times (cf. Table 1). Moreover, the basic model can also freeze the first s

columns by assigning the first s players to the first s groups after the first week, given by:

$$\begin{gathered} \forall i \in I \forall j \in J(G_{i,j} = j + 1) \\ I = \{i \in \mathbb{Z} \mid 0 < i < w\}, J = \{j \in \mathbb{Z} \mid 0 \leq j < s\} \end{gathered} \tag{2}$$

Application of Eq. (2) to the model, significantly reduces the search space, as well as a part of symmetries caused by interchangeable groups inside of weeks (cf. Table 1).

Table 1: A solution of 5-5-6 expressed by groups. It can be converted to the solution expressed by the number of golfers easily. The submatrices GS_1 and GS_2 are distinguished by color. The text in bold indicates that the values have been initialized before search.

Player \ Week	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
1	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**5**	**5**	**5**	**5**	**5**
2	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
3	**1**	**2**	**3**	**4**	**5**	2	5	1	3	4	3	1	4	5	2	4	3	5	2	1	5	4	2	1	3
4	**1**	**2**	**3**	**4**	**5**	3	1	4	5	2	2	5	1	3	4	5	4	2	1	3	4	3	5	2	1
5	**1**	**2**	**3**	**4**	**5**	4	3	5	2	1	5	4	2	1	3	3	1	4	5	2	2	5	1	3	4
6	**1**	**2**	**3**	**4**	**5**	5	4	2	1	3	4	3	5	2	1	2	5	1	3	4	3	1	4	5	2

By the definition of the SGP, n different players are divided into g groups, which implies that each group includes exactly s players. Hence, the constraint required by this property, which is imposed on the rows of the matrix G, can be stated as:

$$\begin{aligned} \forall i \in I(GCC(G_{i,*}, V, O)) \\ I = \{i \in \mathbb{Z} \mid 0 < i < w\} \\ V = \{v \in \mathbb{Z} \mid 1 \leq v \leq g\} \\ O = [s \dots s] \end{aligned} \tag{3}$$

where the length of O is g. The constraints (3) ensure that every value in the set of integers $\{1, \dots, g\}$ must occur exactly s times in all the rows of the matrix G (cf. Table 1).

The restriction, according to which no player meets any other player more than once, can be interpreted as saying that no two columns of the matrix G have the same value at the same row more than once, given by:

$$\begin{gathered} \sum_{0 \leq i < w} |\, G_{i,j_1} - G_{i,j_2} = 0 \,| \leq 1 \\ j_1 \in \mathbb{Z}, j_2 \in \mathbb{Z}, 0 \leq j_1 < j_2 < n \end{gathered} \tag{4}$$

Constraint (4) was implemented in a different way to avoid using the *reified* constraints which often slow the resolution speed down [4]. Specifically, the need for the *reified* constraints can be bypassed by introducing a $w \times m$ matrix C, where $m = \binom{n}{2}$. Then every column is subtracted from all other columns in the matrix G and the differences between two columns of the matrix G are assigned to a column of another matrix C, given by:

$$\begin{aligned}\forall i \in I \forall j_1 \in J_1 \forall j_2 \in J_2 (G_{i,j_1} - G_{i,j_2} = C_{i,j_3})\\ (j_1 < j_2) \wedge (0 \leq j_3 < m) \wedge (j_3 \in \mathbb{Z})\\ I = \{i \in \mathbb{Z} \mid 0 \leq i < w\}\\ J_1 = \{j_1 \in \mathbb{Z} \mid 0 \leq j_1 < n\}\\ J_2 = \{j_2 \in \mathbb{Z} \mid 0 \leq j_2 < n\}\end{aligned} \tag{5}$$

Next, the **Count** constraint is imposed on every column of the matrix C so that the number of occurrences of value 0 on each column is no more than once:

$$\begin{aligned}\forall j \in J (count(0, C_{*,j}, occ))\\ J = \{j \in \mathbb{Z} \mid 0 \leq j < m\}\\ occ \in \{0, 1\}\end{aligned} \tag{6}$$

So far, all the constraints as mentioned earlier have fully satisfied all the restrictions defined by the definition of the SGP and can be used to solve some small instances (e.g., 3-3-4, 5-3-7). However, we can further shrink the search space by placing *implied constraints*, which do not change the set of solutions, and hence are logically redundant.

Equation (1) has already fixed the first row of the matrix G, which implies that those players who have met in the first week cannot play in the same group in the subsequent weeks. Therefore **AllDifferent** constraint can be used to enforce that the groups of these players are pairwise distinct after the first week, and we express these **AllDifferent** constraints by:

$$\begin{aligned}\forall i \forall j \neq j' \wedge j/s = j'/s \, (G_{i,j} \neq G_{i,j'})\\ i \in \{i \in \mathbb{Z} \mid 0 < i < w\}\\ j, j' \in \{x \in \mathbb{Z} \mid 0 \leq x < n\}\end{aligned} \tag{7}$$

2.2 Instance Specific Constraints

For a given number of groups g and a group size s, the goal is to compute a first solution for a *full instance g-s-w**. In this section, we consider a particular type of instance *s-s-(s+1)*, which means that the number of groups in each week is the same as the number of players in the groups within each week, and the number of weeks is equal to the number of groups within each week plus one. The specific properties of the instances of the form *s-s-(s+1)* enable one to discover the *instance-specific* constraints. Furthermore, the observed pattern is utilized from the relatively small instances to deduce more *instance-specific* constraints for the instances of the form *prime-prime-(prime+1) (p-p-(p+1))* and the form *even-even-(even+1) (e-e-(e+1))*.

Before introducing the constraints, we first define the submatrix GS of the decision variables matrix G. In the present paper, a submatrix GS of G is a $(w-1) \times s$ matrix formed by removing the first row of G and selecting columns $[j...(j+s-1)]$, where j must be divisible by s, i.e. $j\%s = 0$. Thus, G has exactly s such submatrices, each of which has $w-1$ rows and s columns. The i-th submatrix of G is denoted by GS_i, where $0 \leq i \leq s-1$ (Table 1).

2.3 *s-s-(s+1)* instances

For the instances of the form *s-s-(s+1)*, every player must play with every other exactly once since $s * s - 1$ is divisible by $s - 1$. Since the first s players, in turn, are assigned to the first s groups after the first week and there are only s groups within each week, all the rest of $n - s$ players have to be assigned to these s groups to avoid meeting the first s players more than once. Based on this analysis, the following constraints on the columns of the matrix G are placed:

$$\begin{aligned} \forall i \in I \wedge i' \in I \wedge i \neq i' \forall j \in J (G_{i,j} \neq G_{i',j}) \\ I = \{x \in \mathbb{Z} \mid 0 < x < w\} \\ J = \{x \in \mathbb{Z} \mid s \leq x < n\} \end{aligned} \tag{8}$$

Constraint (8) states that starting with submatrix GS_1 of the matrix G, every column in the matrix GS_i $(i \geq 1)$ must be pairwise distinct (cf. Table 1). Therefore, all the possible values of columns (column space) of the matrix GS_i are reduced from s^s to $s!$.

We can also fix the second row of the instances of the form *s-s-(s+1)* for the following reason. In Constraint (7), we have explained that s players assigned in the same group in the first week cannot meet again in the subsequent weeks. Besides, for the form *s-s-(s+1)*, there are only s different groups, which implies that the possible groups assigned to these s players must by a permutation of the set of integers $\{1 \ldots s\}$. Thus, every row of the submatrix GS_i $(i \geq 1)$ is a permutation of the set of integers $\{1 \ldots s\}$. Moreover, arbitrary swapping two columns in the submatrix GS_i $(i \geq 1)$ leads to an isomorphism even when the first row of the matrix G is fixed. Therefore, for the instances of the form *s-s-(s+1)*, the first row of the submatrix GS_i $(i \geq 1)$ is fixed with the array $[1 \ldots s]$ (cf. the second row of Table 1), which can be expressed as:

$$\begin{aligned} \forall j \in J (G_{1,j} = j\%s + 1) \\ J = \{x \in \mathbb{Z} \mid s \leq x < n\} \end{aligned} \tag{9}$$

Thus, the symmetries caused by renumbering players in the second row can be eliminated by imposing Constraint (9).

In [4], they observed that the solutions of the 4-4-5, 5-5-6, and 7-7-8 instances; and discovered that GS_1 can always be a symmetric matrix, namely $GS_1 = GS_1^T$. Hence, they conjecture that bigger instances can also have a symmetric submatrix and then impose the following constraints on the decision variables

G:

$$\begin{aligned} \forall i \in I \forall j \in J (G_{1,j} = G_{(j-s),(i+s)}) \\ I = \{x \in \mathbb{Z} \mid 0 \leq x < w\} \\ J = \{x \in \mathbb{Z} \mid s \leq x < 2 * s\} \end{aligned} \tag{10}$$

Constraint (**??**) states that the entries of GS_1 are symmetric with respect to the main diagonal. Besides, the main diagonal of the submatrix GS_1 is pairwise distinct for 5-5-6 and 7-7-8, given by:

$$\begin{aligned} \forall i \in I \forall j \in J (G_{1,j} \neq G_{(i+1),(j+1)}) \\ I = \{x \in \mathbb{Z} \mid s \leq x < w\} \\ J = \{x \in \mathbb{Z} \mid s \leq x < 2 * s\} \end{aligned} \tag{11}$$

Also, because the second row has already been fixed by Constraint (9), we can impose the **AllDifferent** constraints on the subsequent rows for those players who have played together in the second week since any two columns of G can only have identical values in exactly one row (e.g. **AllDifferent**$(G_{3,5}, G_{3,10}, G_{3,15}, G_{3,20}$ in Table 1). These constraints are implied constraints and can be expressed as:

$$\begin{aligned} \forall i \in I \forall j \in J \wedge j' \in J \wedge j\%s = j'\%s \wedge j \neq j' (G_{1,j} \neq G_{i,j'}) \\ I = \{x \in \mathbb{Z} \mid 1 \leq x < w\} \\ J = \{x \in \mathbb{Z} \mid s \leq x < n\} \end{aligned} \tag{12}$$

The addition of these constraints into the basic model presents the way to tackle the larger *s-s-(s+1)* instances.

2.4 *p-p-(p+1)* instances

In [4], they've also discovered that the submatrix GS_i, $1 < i < s$, consists of s s-tuples that are derived from the second submatrix GS_1 on the 5-5-6 and 7-7-8 (cf. Table 1). Simply put, the rest of submatrices can be obtained by interchanging rows of GS_1 on these instances. Thus, we can solve larger instances of the form *p-p-(p+1)* by restricting row space of the submatrix GS_i $(1 < i < s)$ to the rows of the submatrix GS_1. Formally:

$$\begin{aligned} PT = \{(G_{i,j}, G_{i,j+1}, \ldots, G_{i,j+s-1}) \mid \\ s \leq j < 2 * s \wedge 2 \leq i < w \wedge i, j \in \mathbb{Z}\} \end{aligned} \tag{13}$$

$$\begin{aligned} (G_{i,j}, G_{i,j+1}, \ldots, G_{i,j+s-1}) \in PT, \\ 2 \leq i < w, 2 * s \leq j < n, j\%s = 0,\ i, j \in \mathbb{Z}\} \end{aligned} \tag{14}$$

where Constraint (13) defines the potential combination of values of columns of GS_1 as PT. Then we can limit the row space of the submatrices except GS_0 and GS_1 to PT by Constraint 14, which can be implemented by the **Table** constraint. So the question then is, how to find the submatrix GS_1 that can lead to a solution of the instance.

To find the correct GS_1, a separate model is used. This model is defined on a $s \times s$ matrix (s must be a prime number), which comprises Constraints (10) and (11), and the **AllDifferent** constraint imposing on each row and each column of the matrix. In addition the first and second row were fixed with $[1 \dots s]$ and $[2, s, 1, 3, 4, ..., s-1]$ respectively (fix based on 5-5-6 and 7-7-8 instances). Incidentally, GS_1 is a *Latin square*[1]. Moreover, for the last row ($i = s-1$) of GS_1, starting with the third element ($j = 2$) to the last element, the values are fixed with the array $[2, 1, 3, 4, 5, ..., s-2]$. Along with decrementing the row ($i--$), the element at the tail of the array is removed and the starting position of the first element of the array incrementing ($j++$) until the array is reduced to containing exactly one element $\{2\}$, as illustrated in Table 2.

Table 2: The second matrix GS_1 for the instance 13-13-14. The text in bold indicates that the values have been initialized before search.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
2	**13**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
3	1	4	5	6	7	8	9	10	11	12	13	**2**
4	3	5	6	7	8	9	10	11	12	13	**2**	**1**
5	4	6	7	8	9	10	11	12	13	**2**	**1**	**3**
6	5	7	8	9	10	11	12	13	**2**	**1**	**3**	**4**
7	6	8	9	10	11	12	13	**2**	**1**	**3**	**4**	**5**
8	7	9	10	11	12	13	**2**	**1**	**3**	**4**	**5**	**6**
9	8	10	11	12	13	**2**	**1**	**3**	**4**	**5**	**6**	**7**
10	9	11	12	13	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**
11	10	12	13	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
12	11	13	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
13	12	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

Having this observed pattern and aforementioned separated model, we can obtain exactly one GS_1 for the instance of the form *p-p-(p+1)*, and then utilize it as an input for Constraint (14). Note that since GS_1 has already been initialized before solving process, we do not need Constraints (10), (11), and (12) on the model.

2.5 *e-e-(e+1)* instances

We now consider the form of instances *even-even-(even+1)*. From instance 4-4-5 were derived the following conjectures:

$$\forall i \in I(G_{i,(i+s-1)} = 1) \\ I = \{x \in \mathbb{Z} \mid 0 < x < w\} \tag{15}$$

[1] Latin square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.

$$\begin{aligned} \forall j \neq j' \wedge i \neq i' \wedge j/s = j'/s \wedge j\%s + 1 = i' \\ (G_{i,j} \neq G_{i',j'}) \\ i, i' \in \{x \in \mathbb{Z} \mid 0 < x < w\} \\ j, j' \in \{x \in \mathbb{Z} \mid 2 * s < x < n\} \end{aligned} \tag{16}$$

Constraint (15) states that the main diagonal of the matrix GS_1 consists of the fixed values $[1, 1, ..., 1]$; and the rest of submatrices have the main diagonal whose values must be pairwise distinct (Constraint (16)).

Table 3: The second matrix GS_1 for a solution 8-8-9.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
2	*1*	*4*	**3**	**6**	*5*	*8*	**7**
3	*4*	*1*	**2**	**7**	*8*	*5*	**6**
4	**3**	**2**	**1**	**8**	**7**	**6**	**5**
5	**6**	**7**	**8**	**1**	**2**	**3**	**4**
6	*5*	*8*	**7**	**2**	*1*	*4*	**3**
7	*8*	*5*	**6**	**3**	*4*	*1*	**2**
8	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Table 3 depicts the submatrix GS_1 of the solution of 8-8-9. It is interesting to observe that the GS_1 matrices of 4-4-5 and 8-8-9 are composed of four symmetric matrices. Moreover, in [4], they discover that their solutions also satisfy the Constraints (13) and (14), and it is still unclear whether or not the 16-16-17 instance shares these common features with 4-4-5 and 8-8-9.

Table 4: Part of the matrix G for instance *7-7-8*. The text in bold indicates that the values have been initialized before search. The submatrices GS_1, GS_2, and first column of GS_3 are distinguished by color.

Player \ *Week*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	18	19	20	21	22	23	...
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**...**
2	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**1**	**...**
3	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**2**	**7**	**1**	**3**	**4**	**5**	**6**	*3*	**1**	*4*	*5*	*6*	*7*	*2*	**4**	...
4	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**3**	**1**	**4**	**5**	**6**	**7**	**2**	*2*	*7*	**1**	*3*	*4*	*5*	*6*	**7**	...
5	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**4**	**3**	**5**	**6**	**7**	**2**	**1**	*7*	*6*	*2*	**1**	*3*	*4*	*5*	**5**	...
6	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**5**	**4**	**6**	**7**	**2**	**1**	**3**	*6*	*5*	*7*	*2*	**1**	*3*	*4*	**3**	...
7	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**6**	**5**	**7**	**2**	**1**	**3**	**4**	*5*	*4*	*6*	*7*	*2*	**1**	*3*	**2**	...
8	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**7**	**6**	**2**	**1**	**3**	**4**	**5**	*4*	*3*	*5*	*6*	*7*	*2*	**1**	**6**	...

3 Observations and findings

After implementing and obtaining results for *p-p-(p+1)* instances, patterns not mentioned in [4] were discovered in their matrices. Specifically, there was always a solution, in all instances of this type, in which the main diagonal of the GS_2 consists of the fixed values $[1, 1, \ldots, 1]$ (see Tables 1, 4, and 5 and notice blue coloured matrices). Formally:

$$\begin{aligned} \forall i \in I(G_{i,(i+(2*s)-1)} = 1) \\ I = \{x \in \mathbb{Z} \,|\, 0 < x < w\} \end{aligned} \tag{17}$$

Besides, Constraint (14) implies that by fixing the main diagonal of the GS_2 matrix, we get the whole GS_2 matrix before searching, which significantly reduces the solution time for instances 11-11-12 and 13-13-14 (see Table 6).

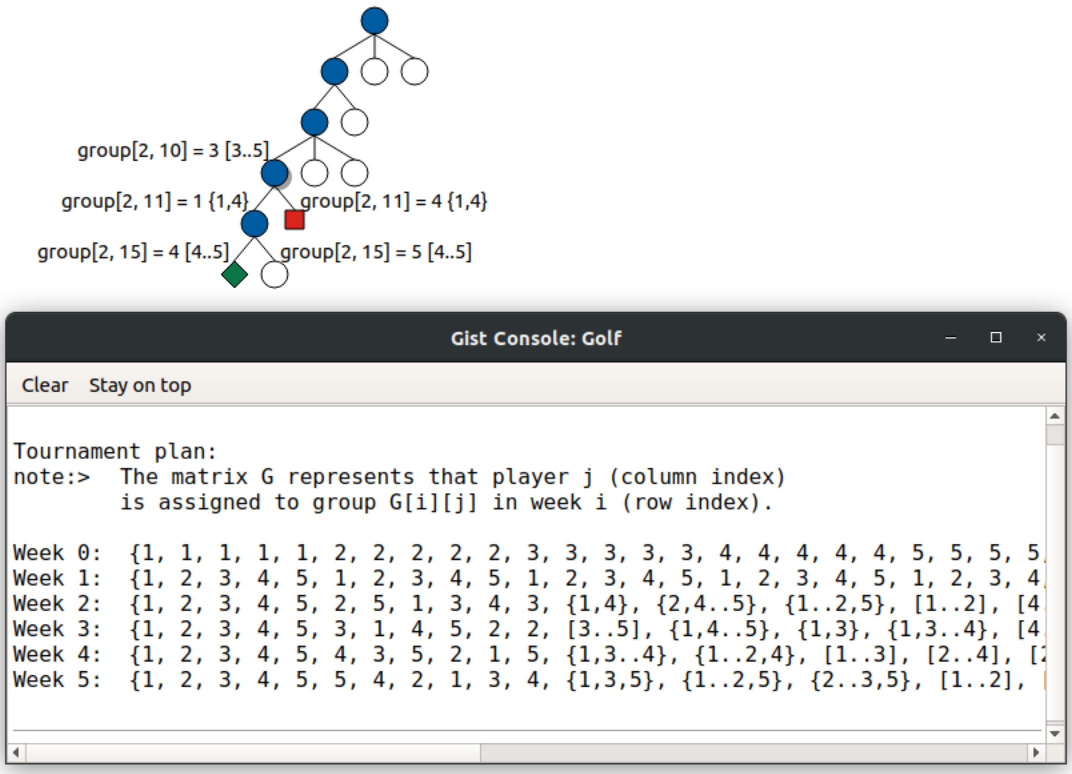
We can also notice a certain trend in the first column of the GS_3 matrix (see Tables 4 and 5, and notice green coloured values). The first three rows of the of the GS_3 matrix's first column always have the values $4, 1, 4$ which are followed by the value s. Each subsequent row then has a value two less than the previous one until we reach value 2. Then the value $(s-1)$ follows, and the trend of the countdown of 2 is repeated. Based on these observations, it was possible to resolve even instance 17-17-18 by fixing the mentioned column with values $[4, 1, 4, 17, 15, \ldots, 2, 16, 14, \ldots, 6]$ in less time than was needed to resolve instance 13-13-14 in [4], where they do not even mention instance 17-17-18. As in the previous case, Constraint (14) implies that by fixing the first column of the GS_3 matrix, we actually obtained the entire GS_3 matrix before searching.

Table 5: Part of the matrix G for instance *11-11-12*. The text in bold indicates that the values have been initialized before search. The submatrices GS_1, GS_2, and first column of GS_3 are distinguished by color.

$Player \backslash Week$	...	12	13	14	15	16	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	...
1	**...**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**...**
2	**...**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**1**	**...**
3	**...**	**2**	**11**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**3**	**1**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**2**	**4**	...
4	**...**	**3**	**1**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**2**	2	11	**1**	3	4	5	6	7	8	9	10	**11**	...
5	**...**	**4**	**3**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**2**	**1**	11	10	2	**1**	3	4	5	6	7	8	9	**9**	...
6	**...**	**5**	**4**	**6**	**7**	**8**	**9**	**10**	**11**	**2**	**1**	**3**	10	9	11	2	**1**	3	4	5	6	7	8	**7**	...
7	**...**	**6**	**5**	**7**	**8**	**9**	**10**	**11**	**2**	**1**	**3**	**4**	9	8	10	11	2	**1**	3	4	5	6	7	**5**	...
8	**...**	**7**	**6**	**8**	**9**	**10**	**11**	**2**	**1**	**3**	**4**	**5**	8	7	9	10	11	2	**1**	3	4	5	6	**3**	...
9	**...**	**8**	**7**	**9**	**10**	**11**	**2**	**1**	**3**	**4**	**5**	**6**	7	6	8	9	10	11	2	**1**	3	4	5	**2**	...
10	**...**	**9**	**8**	**10**	**11**	**2**	**1**	**3**	**4**	**5**	**6**	**7**	6	5	7	8	9	10	11	2	**1**	3	4	**10**	...
11	**...**	**10**	**9**	**11**	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	5	4	6	7	8	9	10	11	2	**1**	3	**8**	...
12	**...**	**11**	**10**	**2**	**1**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	4	3	5	6	7	8	9	10	11	2	**1**	**6**	...

4 Implementation

The presented model was implemented using a Gecode [7] constraint solver (implemented in `C++`), opposed to publication [4] where they used a Choco solver [6]. Gecode was chosen for its excellent performance with respect to runtime, memory usage, and scalability. For example, Gecode won all gold medals in the MiniZinc Challenge in 2012, 2011, 2010, 2009, and 2008 [7]. Another reason was the graphical interactive search engine Gist (an example of which is shown in Figure 1), which was very helpful during development.



```
Tournament plan:
note:>  The matrix G represents that player j (column index)
        is assigned to group G[i][j] in week i (row index).

Week 0:  {1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5
Week 1:  {1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4
Week 2:  {1, 2, 3, 4, 5, 2, 5, 1, 3, 4, 3, {1,4}, {2,4..5}, {1..2,5}, [1..2], [4
Week 3:  {1, 2, 3, 4, 5, 3, 1, 4, 5, 2, 2, [3..5], {1,4..5}, {1,3}, {1,3..4}, [4
Week 4:  {1, 2, 3, 4, 5, 4, 3, 5, 2, 1, 5, {1,3..4}, {1..2,4}, [1..3], [2..4], [
Week 5:  {1, 2, 3, 4, 5, 5, 4, 2, 1, 3, 4, {1,3,5}, {1..2,5}, {2..3,5}, [1..2],
```

Fig. 1: Gist, solving the SGP problem (5-5-6). The inner nodes are choices, the red square leaf node is failure, and the green diamond leaf node is a solution. As we can see in the example, after implementing the `print()` function, assigning a label to each node is possible. Besides, after selecting the `Inspect node` option, the current state of the model in that node will be displayed.

4.1 Gecode Model

Models in Gecode are implemented using *spaces*. A space is *home* to the *variables, propagators* (implementations of constraints), *branchers* (implementation of branchings, describing the search tree's shape), and possibly - an *order* determining a best solution during search.

Not surprisingly in an object oriented language such as `C++`, an elegant approach to programming a model is by inheritance: a model inherits from the class `Space` and the subclass constructor implements the model. In addition to the constructor, a model must implement a copy constructor and a copy function such that search for the model works.

4.2 Implemented Models

The implementation consists of two models:

– **Golf model** (`sqp.cpp`): The model closely follows the above problem description. The `Golf` class defines an array of integer *variables* `groups` of size $g * s * w$, where each variable can contain the values $1...g$. The `Golf` class also defines a matrix `G` with $g * s$ columns and w rows on top of the variable array, such that `G(i,j)` is the group for player `i` in week `j`. Subsequently, the methods of the Golf class define the constraints (*propagators*) described in sections 2 and 6. During the evaluation, as many corresponding constraints as possible are used for each input instance according to its type in order to get the solution as fast as possible.
  For the searching strategy was chosen the `Gecode::INT_VAR_SIZE_MIN()`, which assigns the non-instantiated variable of the smallest size to its lower bound.
– **Latin square model** (`latin.cpp`): This model is used to generate a GS_1 matrix for p-p-(p+1) instances. The matrix is then used as input for constraints (13) and (14) from Section 2.4. Like the previous model, it defines a matrix of integer *variables* and *constraints* from section 2.4. It also uses the same searching strategies as the `Golf` model.

5 Experimental Evaluation

In this section, we report the experimental results on the dataset presented in [4]. In addition, we added an instance 17-17-18, which we were able to resolve thanks to our newly added constraints.

Our experiments were run on a CORE i7-7700HQ processor at 2.80 GHz, and 16 GB DDR4 memory running Ubuntu 18.04 with 64-bit binaries. The Choco experiments were run on a laptop with an Intel i7-3720QM CPU, 2.60 GHz, and 8 GB DDR3 memory running Linux Mint 18.3.

Table 6 summarizes the experimental results, including the total CPU time, the number of visited nodes, and fails. We can see, that our approach is better than [4] on all the solved instances. We were able to prove the nonexistence of the solution of 6-6-7 and solve all the other instances. The results can be found in `output.txt` file or you can reproduce experiments by typing `make benchmark` on a command line.

Table 6: Results on the s-s-(s+1) Instances. **Choco** refers to solver presented in [4] and **Gecode** refers to our implementation. A superscript "c" means that the instance was open for constraint satisfaction approach before presented in [4].

	Choco			**Gecode**		
Instance	**Time (s)**	**Nodes**	**Fails**	**Time (s)**	**Nodes**	**Fails**
5-3-7	0.095	111	94	0.026	518	245
5-5-6	0.069	7	0	0.001	6	0
6-6-7^c	25	1.38e5	1.38e5	7.782	49,503	49,503
7-7-8^c	111	3.62e5	3.62e5	6.842	21,851	10,914
8-8-9^c	12	15,370	15,350	0.038	21	6
11-11-12^c	62	3,150	3,144	0.588	57	24
13-13-14^c	2563	5.80e4	5.79e4	142.126	6,969	3,481
17-17-18	-	-	-	552.958	6,666	3,327

6 Conclusion

In this project, we have successfully implemented the method set out in [4] to solve the SGP problem. By experimental evaluation, we proved that the method was not only understood but also implemented correctly, as we were able to achieve similar respectively better results as presented in the article. Better results are due to the application of as many constraints as possible to a given instance, what the authors of the article did not do. They chose the approach of gradually adding constraints with an increasing instance size. The Gecode solver and its searching strategies also contributed to the better results. During the implementation, the most advantageous of all tested searching strategies was chosen. The last thing that signed on the better results was the newly introduced constraints, which accelerated instances 11-11-12 and 13-13-14 and at the same time allowed us to solve a much larger instance 17-17-18, which was not even presented in the original article.

References

1. Barnier, N., Brisset, P.: Solving the kirkman's schoolgirl problem in a few seconds. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming. p. 477–491. CP '02, Springer-Verlag, Berlin, Heidelberg (2002)
2. Beldiceanu, N., Carlsson, M., Rampon, j.x.: Global constraint catalog (01 2005)
3. Harvey, W.: CSPLib problem 010: Social golfers problem. http://www.csplib.org/Problems/prob010
4. Liu, K., Löffler, S., Hofstedt, P.: Social golfer problem revisited. In: van den Herik, J., Rocha, A.P., Steels, L. (eds.) Agents and Artificial Intelligence. pp. 72–99. Springer International Publishing, Cham (2019)
5. Liu, K., Löffler, S., Hofstedt, P.: Solving the social golfers problems by constraint programming in sequential. In: Rocha, A.P., Steels, L., van den Herik, H.J. (eds.) Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019, Volume 2, Prague, Czech Republic, February 19-21, 2019. pp. 29–39. SciTePress (2019). https://doi.org/10.5220/0007252300290039, https://doi.org/10.5220/0007252300290039
6. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016), http://www.choco-solver.org
7. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling. In: Schulte, C., Tack, G., Lagerkvist, M.Z. (eds.) Modeling and Programming with Gecode (2019), corresponds to Gecode 6.2.0