Social Golfer Problem Revisited

Ke Liu(✉), Sven Löffler, and Petra Hofstedt

Department of Mathematics and Computer Science (MINT),
Brandenburg University of Technology Cottbus-Senftenberg,
Konrad-Wachsmann-Allee 5, 03044 Cottbus, Germany
{liuke,sven.loeffler,hofstedt}@b-tu.de

Abstract. In a golf club, $n = g * s$ golfers want to play in g groups of s golfers for w weeks. Does there exist a schedule for each golfer to play no more than once with any other golfer? This simple but overwhelmingly challenging problem, which is called social golfer problem (SGP), has received considerable attention in constraint satisfaction problem (CSP) research as a standard benchmark for symmetry breaking. However, constraint satisfaction approach for solving the SGP has stagnated in terms of larger instance over the last decade. In this article, we improve the existing model of the SGP by introducing more constraints that effectively reduce the search space, particularly for the instances of the specific form. And on this basis, we also provide a search space splitting method to solve the SGP in parallel via data-level parallelism. Our implementation of the presented techniques allows us to attain the solutions for eight instances with maximal number of weeks, in which six of them were open instances for constraint satisfaction approach, and two of them are computed for the first time, and super-linear speedups are observed for all the instances solved in parallel. Besides, we survey the extensive literature on solving the SGP, including the best results they have achieved, and analyse the cause of difficulties in solving the SGP.

Keywords: Constraint programming · Parallel constraint solving · Resolvable steiner systems · Combinatorial optimization · Design theory · Mutually Orthogonal Latin Squares · Affine plane

1 Introduction

The *social golfer problem* (SGP), i.e., 010 problem in CSPLib [13], is a typical combinatorial optimization problem that has attracted significant attention from the constraints community because of its highly symmetrical and combinatorial nature. The original SGP, which was posted to `sci.op-research` in May 1998 [13,38], can be stated as follows: In a golf club, 32 golfers wish to play in foursomes for 10 weeks. Is it possible to find a schedule for maximum socialization; that is, each golfer can only meet any other no more than once? In fact, the SGP dates back to Thomas Penyngton Kirkman's 1850 query [16] in which the number of golfers and the size of a group are 15 and 3 respectively.

J. van den Herik et al. (Eds.): ICAART 2019, LNAI 11978, pp. 72–99, 2019.
https://doi.org/10.1007/978-3-030-37494-5_5

Hence, we can readily generalize the SGP to the following: The SGP consists of scheduling $n = g * s$ players into g groups of s players for w weeks so that any two players are assigned to the same group at most once in w weeks. 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. In addition, we can also regard the SGP 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 not in the same order of magnitude due to the huge difference in the solution density of two instances. For example, the 8-4-9 instance can be solved in a second on a state-of-the-art constraint solver. The 8-4-10 instance, by contrast, is still unsolvable for constraint approach at the time of writing, although at least three non-isomorphic solutions are known to exist. In light of this, this research concentrates on solving the g-s-w^* instances with maximal number of weeks, which we call *the full instance*. For instance, Table 1 depicts one solution for the full instance 7-3-10.

Table 1. A solution for 7-3-10 (transformed from the solution depicted in Table 2) The text in bold indicates that the values have been initialized before search. (Table reproduced from [20]).

Week \ Group	1			2			3			4			5			6			7		
1	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***
2	***1***	4	7	***2***	10	13	***3***	16	19	5	8	17	6	11	14	9	12	20	15	18	21
3	***1***	10	14	***2***	4	17	***3***	9	11	5	7	21	6	12	16	8	15	19	13	18	20
4	***1***	17	21	***2***	6	19	***3***	4	12	5	10	20	7	15	16	8	11	13	9	14	18
5	***1***	8	12	***2***	5	16	***3***	7	13	4	14	21	6	9	15	11	17	20	10	18	19
6	***1***	9	16	***2***	11	15	***3***	10	21	4	8	20	5	14	19	6	13	17	7	12	18
7	***1***	13	19	***2***	9	21	***3***	6	20	4	11	18	5	12	15	8	10	16	7	14	17
8	***1***	5	11	***2***	8	18	***3***	15	17	4	9	19	6	7	10	14	16	20	12	13	21
9	***1***	6	18	***2***	7	20	***3***	8	14	4	10	15	5	9	13	11	16	21	12	17	19
10	***1***	15	20	***2***	12	14	***3***	5	18	4	13	16	6	8	21	9	10	17	7	11	19

The research on the SGP is not only meaningful to itself, but also for other Constraint Satisfaction Problems (CSPs) that exhibit symmetrical and combinatorial nature. For example, balanced incomplete block design (BIBD), problem 28 in CSPLib [26], is a standard combinatorial problem from design theory and also a test bed for symmetry breaking methods. Moreover, steel mill slab design [23], which is a real industry problem, can also benefit from the SGP. The reason is that we are likely to face the same difficulties as the SGP when solving other CSPs through the constraint satisfaction approach.

This article is an extended and revised version of our previous work [20]. In this version, the formulations of the constraints in the models have been improved. Besides, we have added a new section which explains the difficulties of the SGP and presents the background constraint programming. We also present additional experimental results for the solved instances in Appendix A. We make the following contributions: (1) The improvement of existing constraint model on the SGP enables constraint approach to solve larger instances. (2) We discover some patterns in the solutions of particular forms of the instances. (3) We show that the two-stage models with static partitioning are well-suited for solving the SGP in parallel since the instances which are unsolvable for a single model can be solved in parallel.

The remainder of this article is organized as follows. Section 2 analyses the causes of the difficulties of solving the SGP in the context of the CSP and introduces the constraints required to encode the model. A modeling approach improved on the model proposed by [3] are described in Sect. 3, and some instance-specific constraints are presented in Sect. 4. In addition, we elaborate on how to employ Embarrassingly Parallel Search (EPS) [30] to solve the SGP in Sect. 5. We then present the experimental results in Sect. 6. In Sect. 7, we classify the researches on the SGP and also survey the studies relevant to the SGP outside the context of the CSP. We finally conclude in Sect. 8.

2 Background

In this section, we first explain why it is challenging to solve the SGP in the context of the CSP. Then we review the definition of the CSP and the constraints relevant to our model of the SGP.

2.1 The Difficulties of Solving the SGP

At first sight, the SGP is a simple-sounding question. And indeed, one can model the problem by using several frequently-used constraints derived from the problem definition. The constraint satisfaction approach, however, still has enormous difficulties in obtaining the solution even for some small instances (e.g. 7-4-9, 8-4-10, etc.). We believe that the following two reasons result in the difficulties of the SGP:

The First Difficulty. The inherent highly symmetrical nature of the SGP cannot be entirely known before solving process. There exist four types of symmetries: (1) We can permute the w weeks, that is, arbitrarily ordered weeks ($w!$ symmetries). (2) Within each week, we can (separately) permute the g groups, that is, interchangeable groups inside weeks ($g!$ symmetries). (3) Within each group, we can permute the s players, that is, interchangeable players inside groups ($s!$ symmetries). (4) Finally, we can also permute the n players ($n!$ symmetries), which can also be viewed by renumbering n golfers. The first three types of symmetries can be relatively easy to remove through model reformulation or static symmetry breaking constraints. Nevertheless, it is difficult to eliminate all the symmetries among players caused by the fourth type of symmetry.

For example, if players $[16, 17, 18, 19, 20, 21]$ in Table 1 replace with $[19, 20, 21, 16, 17, 18]$ in turn, an isomorphism of the solution depicted in Table 1 will be generated even if the first row of the solution is fixed with $[1, \ldots, 21]$. Apparently, we are unable to foresee this symmetry before search. Consequently, the unnecessary symmetrical search space is explored redundantly.

The Second Difficulty. It is common to observe that some unfortunate choices of variables early on are to blame for a long-running search process [12]. The SGP has also been experienced such phenomena. More precisely, invalid partial assignments lead to the backtrack search to trap in a barren part of the search space since no consistent assignment can be found. More importantly, it is often hard to determine the usefulness of a partial assignment until almost all variables are instantiated; and these invalid partial assignments predominate in the overall search space.

2.2 The Global Constraints

The constraint programming (CP) is a powerful technique to tackle combinatorial problems, generally NP-complete or NP-hard. The idea behind the CP is that the user states the 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, $D = \{D_{(x_0)}, \ldots, D_{(x_n)}\}$ 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 P is a complete instantiation satisfying all constraints of the CSP P.

The *allDifferent*[1] constraint is the most influential global constraint in constraint programming and widely implemented in almost every constraint solver, such as Choco solver [27] and Gecode [34]. Formally, let X_a denote a subset of variables of X, the *alldifferent*(X_a) constraint can be defined as:

$$\forall x_i \in X_a \forall x_j \in X_a (x_i \neq x_j)$$

The *global cardinality constraint* $GCC(X_g, V, O)$ is defined using two lists of variables X_g and O, and an array of integer values V, where $X_g = \{x_l, \ldots, x_m\} \subseteq X$ and O is a list of variables not defined in X and predefines the range of the number of occurrences for each value in V. The GCC constraint restricts that each value V_i appearing exactly O_{i_j} times in X_g, where O_{i_j} is in the domain of O_i. More formally:

$$\{(d_l, \ldots, d_m) |\ d_l \in D_{(x_l)}, \ldots, d_m \in D_{(x_m)} \wedge$$
$$\forall i \forall O_{i_j} \in O_i (occur(V_i, (d_l, \ldots, d_m)) = O_{i_j})\}$$

where *occur* counts the number of occurrences of V_i in $(d_l, \ldots, d_m)$.

[1] This article follows the naming convention and order of the arguments of constraints in Choco solver.

The *count* constraint is similar to the GCC, but with the restriction for only one value. More precisely, the $count(v, X_c, occ)$ constraint only restricts the number of occurrences of value v for the list of variables $X_c = \{x_l, \ldots, x_m\}$, given by:

$$\{(d_l, \ldots, d_m) |\ d_l \in D_{(x_l)}, \ldots, d_m \in D_{(x_m)} \wedge \\ \forall O_j \in occ(occur(v, (d_l, \ldots, d_m) = O_j)\}$$

The *table* constraint is another one of the most frequently-used constraints in practice. For an ordered subset of variables $X_o = \{x_i, \ldots, x_j\} \subseteq X$, a positive (negative) *table* constraint defines that any solution of the CSP P must (not) be explicitly assigned to a tuple in the tuples that consists of the allowed (disallowed) combinations of values for X_o. For a given list of tuples T, we can state the positive *table* constraint as:

$$\left\{(d_i, \ldots, d_j) \mid d_i \in D_{(x_i)}, \ldots, d_j \in D_{(x_j)}\right\} \subseteq T$$

Finally, the *arithm* constraint is used to enforce relations between integer variables or between integer variables and integer values. For example, an integer value can be assigned to an integer variable by using the *arithm* constraint. We refer to [6,19,33] for more comprehensive and profound introduction to the CP.

3 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 this article, we use a model improved on the model presented in [3] due to its untapped potential. Specifically, we can add more constraints into the model to tackle larger instances piece by piece.

The decision variables of our 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. Hence, the domain of decision variable $G_{i,j}$ is a set of integers $\{1 \ldots g\}$, where $0 \leq i < w,\ 0 \leq j < n$.[2] The major advantage of the decision variables defined in this model is that the range of the variables are reduced from $\{1 \ldots n\}$ to $\{1 \ldots g\}$ while keeping an unchanged number of variables, compared with the naive model.

We mentioned, in Sect. 2.1, that symmetries among players are difficult to handle and only dynamic checks can remove them completely. However, we can partially eliminate the symmetries among players by fixing the first week (cf. Tables 1 and 2), i.e. the first row of the matrix G, which can be expressed as:

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

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

[2] In this article, we follow the Zero-based index.

s columns by assigning the first s players to the first s groups after the first week (cf. Table 2), given by:

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

By applying Eq. (2) to the model, we can significantly reduce the search space. Note that, in the implementation, we can realize Eqs. (1) and (2) by either restricting the domain of the variables or using the *arithm* constraint. Therefore, we use the term Equation instead of Constraint.

Table 2. A solution is obtained by our model for 7-3-10 instance, and it is equivalent to the solution depicted in Table 1. The text in bold indicates that the values have been initialized before search. (Table reproduced from [20]).

Week \ Player	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	***1***	***1***	***1***	***2***	***2***	***2***	***3***	***3***	***3***	***4***	***4***	***4***	***5***	***5***	***5***	***6***	***6***	***6***	***7***	***7***	***7***
2	***1***	***2***	***3***	1	4	5	1	4	6	2	5	6	2	5	7	3	4	7	3	6	7
3	***1***	***2***	***3***	2	4	5	4	6	3	1	3	5	7	1	6	5	2	7	6	7	4
4	***1***	***2***	***3***	3	4	2	5	6	7	4	6	3	6	7	5	5	1	7	2	4	1
5	***1***	***2***	***3***	4	2	5	3	1	5	7	6	1	3	4	5	2	6	7	7	6	4
6	***1***	***2***	***3***	4	5	6	7	4	1	3	2	7	6	5	2	1	6	7	5	4	3
7	***1***	***2***	***3***	4	5	3	7	6	2	6	4	5	1	7	5	6	7	4	1	3	2
8	***1***	***2***	***3***	4	1	5	5	2	4	5	1	7	7	6	3	6	3	2	4	6	7
9	***1***	***2***	***3***	4	5	1	2	3	5	4	6	7	5	3	4	6	7	1	7	2	6
10	***1***	***2***	***3***	4	3	5	7	5	6	6	7	2	4	2	1	4	6	3	7	1	5

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 are imposed on the rows of the matrix G, can be stated as:

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

where the length of O is g. The constraints (3) ensure that every value in the set of integers $\{1 \ldots g\}$ must occur exactly s times in all the rows of the matrix G (cf. Table 2).

The restriction, 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:

$$\begin{aligned} \sum_{0 \le i < w} & \mid G_{i,j_1} - G_{i,j_2} = 0 \mid \le 1 \\ j_1 \in \mathbb{Z},\ & j_2 \in \mathbb{Z}\ , 0 \le j_1 < j_2 < n \end{aligned} \tag{4}$$

Constraints (3) and (4) are the only two constraints presented in [3]. In particular, unlike [3], we implement Constraint (4) in a different way to avoid using the *reified* constraints because these constraints often slow the resolution speed down in solvers like Choco [21,27]. Specifically, the need for the *reified* constraints can be bypassed by introducing a $w \times m$ matrix C, where $m = \binom{n}{2}$. We then subtract every column 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. Simply put, the two matrices G and C are linked by the equations expressed by the *arithm* constraints, given by:

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

Next, we impose the *count* constraint on every column of the matrix C so that the number of occurrences of value 0 on each column is no more than once. So the constraint are defined by:

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

where *occ* is an integer variable whose domain is $\{0, 1\}$. Thus, the conjunction of Constraints (5) and (6) can logically realize the restriction required from Constraint (4).

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 [36].

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, the *allDifferent* constraint can be used to enforce the groups of these players are pairwise distinct after the first week, and we express these *allDifferent* constraints by:

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

In summary, the basic model comprises Eqs. (1), (2), and Constraints (3), (5), (6), and (7). Nevertheless, the problem-solving ability of this model can be greatly improved by the introduction of additional constraints, such as static symmetry breaking constraints, and the constraints derived by instance-specific pattern. In the subsequent sections, we will present the additional constraints dedicated to different types of instances based on this model and discuss how to solve the instances, which cannot be solved sequentially, in parallel.

4 Instances Solved Sequentially

For a given number of groups g and a group size s, our goal is to compute a first solution for a full instance g-s-w^*, where w^* represents the maximum number of weeks. 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. Moreover, the number of weeks is maximized because $\frac{s*s-1}{s-1} = s+1$. The specific properties of the instances of the form s-s-$(s+1)$ enable us to discover the instance-specific constraints. Furthermore, we utilize the observed pattern from the relatively small instances to deduce more instance-specific constraints for the instances of the form *odd*-*odd*-$(odd+1)$ (o-o-$(o+1)$), especially for 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 $[j \ldots (j+s-1)]$, where j must be divisible by s, i.e. $j\%s = 0$.[3] 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 3).

Table 3. 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 surrounded in the dotted line. (Table reproduced from [20]).

Week \ Player	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

[3] The % (modulo) operator yields the remainder from the division of the first operand by the second.

4.1 7-7-8

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 $s - 1$. Thus, players whose number is greater than s must meet every player whose number is less than or equal to s exactly once in every week except the first week since the first row and the first s columns of the matrix G are frozen by Eqs. (1) and (2). To put it another way, 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, we place the following constraints on the columns of the matrix G:

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

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, which can be implemented by the *allDifferent* constraint (cf. Table 3). Therefore, all the possible values of columns (column space) of the matrix GS_i $(i \geq 1)$ is reduced from s^s to $s!$ by introducing the Constraint 8, which is a significant search space reduction.

In Sect. 3, we have presented Eq. (1) to fix the first row of the matrix G. 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 be 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 by Eq. (1). Therefore, for the instances of the form *s-s-(s + 1)*, we fix all the first rows of the submatrix GS_i $(i \geq 1)$ with the array $[1 \ldots s]$ (cf. the second row of Table 3), which can be expressed as:

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

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

In summary, the model used to tackle 5-5-6, 6-6-7, and 7-7-8 consists of the constraints of the basic model and the additional constraints including Constraints (8) and (9).

4.2 9-9-10

The additional constraints for 7-7-8 are insufficient to solve 9-9-10 in an appropriate time since the size of the problem grows significantly. One possible way to

tackle the larger instance is to shrink the overall search space by imposing more instance-specific constraints.

We observe the solutions of the 4-4-5, 5-5-6, and 7-7-8 instances; and discover that GS_1 can always be a symmetric matrix, namely $GS_1 = GS_1^T$. Hence, we conjecture that 9-9-10 can also have a symmetric submatrix and then impose the following constraints on the decision variables G:

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

Constraint (10) 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:

$$\begin{aligned} \forall i \in I \forall j \in J(G_{i,j} \neq G_{(i+1),(j+1)}) \\ I = \{x \in \mathbb{Z} |\ 0 \leq x < w\} \\ J = \{x \in \mathbb{Z} |\ s \leq x < 2 * s\} \end{aligned} \tag{11}$$

Apart from the fixed pattern of GS_1, there is also a fixed pattern among the submatrices of G. 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 3). These constraints are implied constraints and can be expressed as:

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

We also notice that for 5-5-6 and 7-7-8, there is always a type solution in which the second row of GS_1 is fixed by the array $[2, s, 1, 3, 4, \ldots, s-1]$ (cf. the second row of GS_1 Table 3). We therefore assume that 9-9-10 also exists such solution, and solve 9-9-10 by fixing the second row of GS_1 with $[2, 9, 1, 3, 4, 5, 6, 7, 8]$.

In conclusion, we solve 9-9-10 by adding Constraints (10), (11), and (12) to the model of 7-7-8, as well as the fixed values for the second row of GS_1.

4.3 13-13-14 etc.

We have discovered some common features of the instances of the form *s-s-(s + 1)*, particularly for the instances of the form *o-o-(o + 1)* when expressing a solution by groups; and these common features are mostly focused on the second submatrix GS_1 of G. It is also interesting to observe 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 7-7-9 but 9-9-10 (cf. Table 3). 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:

$$PT = \{(G_{i,j}, G_{i,j+1}, ..., G_{i,j+s-1}) | \\ s \leq j < 2 * s \ \wedge \ 2 \leq i < w \ \wedge \ i, j \in \mathbb{Z}\} \tag{13}$$

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

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, we create a separate model 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. We also fix the first row and the second row with $[1 \ldots s]$ and $[2, s, 1, 3, 4, \ldots, s-1]$ respectively, as we did for the 9-9-10 instance. Incidentally, GS_1 is a *Latin square* since it is a $s \times s$ matrix filled with s distinct numbers and every row and column of the matrix is all different. Moreover, for the last row $(i = s - 1)$ of GS_1, starting with the third element $(j = 2)$ to the last element is fixed with the array $[2, 1, 3, 4, 5, \ldots, s-2]$. 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 4.

Table 4. The second matrix GS_1 for the instance 13-13-14 (Table reproduced from [20]).

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) with the model of 7-7-8 to solve 11-11-12 and 13-13-14. Note that since GS_1 has already initialized before solving process, we do not use the model of 9-9-10 because it is redundant to impose Constraints (10), (11), and (12) on the model.

4.4 8-8-9

So far, all the instances we have discussed conform to the form of o-o-$(o+1)$. We now consider the form of instances *even*-*even*-(*even* + 1). The 8-8-9 is solved by the following conjectures derived from 4-4-5 with the model for 7-7-8:

$$\begin{aligned} &\forall i \in I(G_{i,(i+s-1)} = 1) \\ &I = \{x \in \mathbb{Z} |\ 0 < x < w\} \qquad (15) \\ \forall j \neq j' \wedge i \neq i' \wedge j/s = j'/s \wedge j\%s + 1 = i \wedge j'\%s + 1 = i' \\ &(G_{i,j} \neq G_{i',j'}) \\ &i, i' \in \{x \in \mathbb{Z} |\ 0 < x < w\} \\ &j, j' \in \{x \in \mathbb{Z} |\ 2 * s < x < n\} \qquad (16) \end{aligned}$$

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 5. The second matrix GS_1 for a solution 8-8-9 (Table reproduced from [20]).

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 5 depicts the submatrix GS_1 of the solution of 8-8-9 we solved. It is interesting to observe that the GS_1 matrix of 4-4-5 and 8-8-9 are composed of four symmetric matrices. Moreover, we 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.

5 Instances Solved in Parallel

In the previous section, we have presented the instances that can be solved sequentially via our modeling approach. We now turn to more difficult instances that must deal with by way of parallel processing to obtain one solution. The difficult instances refer to no fixed pattern discovered so far, which implies no instance-specific constraints to shrink search space for these instances and hence there are large search spaces even for relatively small size.

Our idea is to partition the search tree of the SGP into independent subtrees; then each worker that is associated with a thread works on distinct subtrees using the same CP model. Thus, this approach can be classified as *data-level parallelism* based on the taxonomy for parallelism in applications from [15]. Furthermore, since no communication is required during the solving process, to some extent, our parallel approach can also be seen as Embarrassingly Parallel Search (EPS) [31]. The EPS is defined as decomposing the problem in many sub-problems and assigning the sub-problems to workers dynamically [24]. By contrast, our parallel approach differs from the EPS due to the use of a separate model that is used to generate the sub-problems instead of Depth-bounded Depth First Search [31]. The generic procedure can be summarized as follows:

1. A subset of the decision variables of the model is selected.
2. A separate model generates all the partial assignments over selected variables in the subset before the search process.
3. The partial assignments are mapped to the workers so that each worker can work on its own independent search space by using its constraint solver.
4. Once a solution is found, the worker that finds the solution notifies other workers to stop.

Step 1 is crucial to the search space splitting because it determines the subtrees explored by each worker. The selection of the subset of the decision variables adhere to the following rules: First, they should be easy to generate by a separate model. Second, each worker should not be assigned too many partial assignments because one partial assignment might take a long time to evaluate for a large instance. Because of the usage of the separate model, the partial assignments are consistent with the propagation (i.e., running the propagation mechanism on them does not detect any inconsistency). Besides, the number of solutions of the separate model can help us decide the workload of each worker and workload distribution. In the following sections, we will gradually describe CP models for generating partial assignments for search-space splitting and the constraints imposed on the basic model for the 6-3-8, 6-4-7, and 7-3-10 instances in detail.

5.1 6-3-8

The 6-3-8 instance is a representative example to illustrate the effectiveness of our parallel approach for the SGP since the instances smaller than it can be solved quickly and the instances bigger than it are difficult to be solved sequentially by

constraint solving. When switching the target instance from 5-3-7 to 6-3-8, the number of decision variables grows from $5*3*7 = 105$ to $6*3*8 = 144$, and the domain size of each variable is incremented by one for our modeling approach, which indicates the overall underlying search space significantly increased from 5^{105} to 6^{144} if we do not take account of the search space pruned by constraint propagation.

The idea behind the parallel approach is to freeze a part of the decision variables so that the size of the sub-problem is shrunk to solvable, thereby solving the original problem. For the 6-3-8 instance, we select the second row of the matrix G for the search space splitting since the first row of the matrix G is fixed by Constraint (1). A separate model is used to generate the solutions for the second row of the matrix G as the partial assignments for the search space splitting, which is composed of the following constraints:

$$J = \{x \in \mathbb{Z} |\ 0 \leq x < s\}$$

$$\forall j \in J(F_j = j + 1) \tag{17}$$

$$GCC(F, V, O), V = \{1 \ldots g\},\ O = [s \ldots s] \tag{18}$$

$$\forall j \neq j' \wedge j/s = j'/s(F_j \neq F_{j'})$$

$$j' \in J \tag{19}$$

$$\forall j\%s = 0(F_j \leq F_{(j+s)}) \tag{20}$$

$$\forall j/s = (j+1)/s(F_j < F_{(j+1)}) \tag{21}$$

where F is an array of decision variables for the separate model, and the domain of each vairbales is also $\{1 \ldots g\}$. Constraints (17), (18), and (19) are identical to Constraints (1), (3), and (7) stated in the basic model (see Sect. 3) respectively. Constraints (20) and (21), which are not included in the basic model, are static symmetry breaking constraints. Constraint (20) removes the symmetries caused by interchangeable submatrices GS_i, $0 < i < s$. We eliminate these symmetries by arranging the values assigned to the first column of the first row of all the submatrices GS_i in non-decreasing order. (Please refer to the numbers with

Table 6. A solution of 6-3-8 expressed by groups.

Week \ Player	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	***1***	***1***	***1***	***2***	***2***	***2***	***3***	***3***	***3***	***4***	***4***	***4***	***5***	***5***	***5***	***6***	***6***	***6***
2	***1***[a]	***2***	***3***	1^a	4	5	1^a	3	6	2^a	5	6	2^a	4	5	3^a	4	6
3	***1***	***2***	***3***	2^a	1	3	3	4	6	5	6	4	6	2	5	1	4	5
4	***1***	***2***	***3***	3^a	2	1	5	6	3	1	6	4	5	6	4	4	5	2
5	***1***	***2***	***3***	4^a	3	5	6	5	2	6	3	1	1	6	4	2	4	5
6	***1***	***2***	***3***	4^a	5	2	6	4	5	3	6	2	4	1	5	6	3	1
7	***1***	***2***	***3***	4^a	5	6	2	5	1	5	4	3	6	3	2	6	1	4
8	***1***	***2***	***3***	4^a	5	6	5	1	6	4	2	5	3	6	1	4	2	3

superscript a in the second row of Table 6.) Additionally, interchanging any two columns of a submatrix GS_i generates a solution symmetrical with the original one, which entails Constraint (21) to remove these symmetries. Because of Constraint (21), the players played together in the first week must be in ascending order of groups in the second week (cf. the second row of Table 6).

In addition to the constraints of the separate model, we also place the constraints to break the symmetries caused by interchangeable weeks partially. The idea is to restrict the groups of the 4^{th} player in non-decreasing order from week two, given by:

$$\begin{aligned} \forall i \in I(G_{i,s} \leq G_{(i+1),s}) \\ I = \{x \in \mathbb{Z} |\ 0 < x < w - 1\} \end{aligned} \tag{22}$$

Please note that Constraint (22) cannot fully remove the symmetries among weeks because there are still symmetries whenever $G_{i,s} = G_{(i+1),s}$. For example in Table 6, interchanging the 7^{th} week with 8^{th} week results in a symmetrical solution.

Finally, the results of the above model are equally distributed to each worker that runs the basic model.

5.2 6-4-7

The separate model for 6-3-8 produces 424 solutions for the second row, while it produces 351 for the second row of 6-4-7. However, because of the increasing difficulty, we add the following constraints based on the separate model for 6-3-8 to produce less number of solutions for the second row of 6-4-7:

$$J = \left\{j \in \mathbb{Z} |\ 0 \leq j < n - s \wedge j\%s = 0 \wedge j = (s-1) * s \Rightarrow j + s \neq s^2\right\}$$

$$\forall j \in J(F_{j+1} \leq F_{j+s+1}) \tag{23}$$

$$\forall j \in J(F_{j+1} = F_{j+s+1} \Rightarrow F_{j+2} \leq F_{j+s+2}) \tag{24}$$

$$\forall j \in J(F_{j+1} = F_{j+s+1} \wedge F_{j+2} = F_{j+s+2} \Rightarrow F_{j+3} \leq F_{j+s+3}) \tag{25}$$

Table 7. A solution of 6-4-7. The numbers with the same superscript are in non-decreasing order in the second row. (Table reproduced from [20]).

Week \ Player	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	***1***	***1***	***1***	***1***	***2***	***2***	***2***	***2***	***3***	***3***	***3***	***3***	***4***	***4***	***4***	***4***	***5***	***5***	***5***	***5***	***6***	***6***	***6***	***6***
2	$\mathbf{1}^a$	$\mathbf{2}^b$	$\mathbf{3}^c$	$\mathbf{4}^d$	1^a	2^b	3^c	4^d	1^a	4^b	5^c	6^d	1^a	4^b	5^c	6^d	2^e	3^f	5^g	6^h	2^e	3^f	5^g	6^h
3	***1***	***2***	***3***	***4***	2	4	6	5	5	3	6	1	6	2	4	5	3	4	1	2	5	1	3	6
4	***1***	***2***	***3***	***4***	3	6	4	5	4	2	1	3	6	1	5	2	5	1	2	6	3	6	4	5
5	***1***	***2***	***3***	***4***	4	6	1	2	3	5	2	6	5	6	3	1	1	5	4	3	5	2	6	4
6	***1***	***2***	***3***	***4***	6	3	5	1	2	6	3	4	4	5	6	3	4	2	5	1	5	6	1	2
7	***1***	***2***	***3***	***4***	6	1	2	3	5	1	4	2	3	5	2	6	5	6	3	4	4	5	6	1

In short, Constraints (23)–(25) ensure that the values occupying the same positions in the first row of the first s submatrices (GS_0, GS_1, GS_2, GS_3) and the last two submatrices (GS_4, GS_5) are in non-decreasing order respectively (see Table 7). The reason why the submatrices are divided into two groups is that numeral 1 always takes up the first row of the first column in the first s submatrices due to the restrictions from Constraint (20) and (21). Thus, if a constraint enforces $G_{1,13} \leq G_{1,17}$, the solution shown in Table 7 will not be obtained. These additional constraints also reduce search space by removing symmetries. For example, if we do not impose Constraint (23) on the separate model, a second row such like [1 2 3 4 1 4 5 6 1 2 3 4 1 4 5 6 2 3 5 6 2 3 5 6] will be generated. In that case, we will require more workers to work on these symmetrical search spaces.

As with the 6-3-8 instance, we map the solutions of the separate model to different workers before the solving process. Then, to solve the 6-4-7 instance, we further reduce the search space by adding the following constraints onto the basic model:

$$\begin{aligned} \forall j \in J(GCC(G_{*,j}, V, O)) \\ J = \{j \in \mathbb{Z} |\ s \leq j < 3s\} \\ V = \{1 \ldots s\},\ O = \{1 \ldots 1\},\ 0 < * < w \end{aligned} \tag{26}$$

where $G_{*,j}$ denotes the columns from the s^{th} column to the $(3s-1)^{\text{th}}$ column of the matrix G with removed first element. More particularly, every value in the set $\{1, 2, 3, 4\}$ can appear only once in all the columns of the submatrices GS_1 and GS_2. We impose Constraint (26) on only the columns of GS_1 and GS_2 because each player only plays with other 21 players since $(24-1)\%(4-1) = 2$; thus not every column contains the set $\{1, 2, 3, 4\}$. Though Constraint (26) does not enforce all columns containing the values {1, 2, 3, 4}, it reduces much search space; our experiments show that we cannot solve 6-4-7 without these constraints.

5.3 7-3-10

The problem size of 7-3-10 is much larger than 6-4-7 and 6-3-8, we must harness more instance-specific constraints, which are given by:

$$\forall i \in I(G_{i,s} = i + 1), I = \{i \in \mathbb{Z} |\ 0 < i \leq s\} \tag{27}$$

$$\forall i \in I'(G_{i,s} = s + 1), I' = \{i \in \mathbb{Z} |\ s + 1 < i < 2\} \tag{28}$$

$$GCC(G_{*,(s+1)}, V, O),\ V = \{1, 2, 3, 6, 7\}$$

$$0 < * < w,\ O = [1, 1, 1, 0, 0] \tag{29}$$

$$GCC(G_{*,(s+2)}, V', O'),\ V' = \{1, 2, 3, 6\}$$

$$O' = [1, 1, 1, 0] \tag{30}$$

$$\forall s + s \leq j < n(GCC(G_{*,j}, V'', O''))$$

$$V'' = \{1 \ldots s\},\ O'' = [1 \ldots 1] \tag{31}$$

We strictly limit the positions of player 4 so that he/she will never be assigned to groups 5, 6, and 7. The reason is that player 4 must meet players 1, 2, and 3, and the groups of the first three (s) players are frozen by Eq. 2 after the first week, which implies that player 4 must stay in the first three groups from week 2 to week 4. Hence, player 4 can only play in the groups that are greater than or equal to 4 (s + 1) from week 5. Moreover, player 4 is always the smallest player starting from the 9^{th} column of a solution (cf. Table 1). Therefore, player 4 cannot appear in groups 5, 6, and 7, and only stay in group 4 from week 5. Consequently, the 4^{th} column of Table 2 is the result by imposing Constraint (27) and (28). These two constraints not only shrink the search space but also remove the symmetries caused by swapping the group containing player 4 with other groups after week 4.

Since player 4 can only play in group 4 after week 4, player 5 is impossible to stay in groups 6 and 7, because then there will be no player assigned in group 5. Similarly, player 6 cannot appear in group 7 and can only appear in group 6 once. Thus, we use Constraints (29) and (30) to limit the number of occurrences of the values 6 and 7.

Furthermore, because $(21 - 1)\%(3 - 1) = 0$, each player must play with other players exactly once. Hence, we guarantee the first s players must meet the rest of players once, which are ensured by Constraints (29), (30), and (31). Incidentally, Constraint (31) can be applied to any full instance that satisfies $(n - 1)\%(s - 1) = 0$ in our modeling approach (e.g. 7-4-9).

In the implementation of parallelism for 7-3-10, we also use the same separate model as the model for 6-4-7 to generate solutions of the second row and distribute them to the workers.

6 Experiments

In this section, we report the experimental results on instances discussed in Sects. 4 and 5 separately since different hardware and methods were used.

6.1 Experimental Results on Instance Solved Sequentially

To confirm our theoretical discussion and the conjecture for the instances discussed in Sect. 4, we implemented the basic model as described in Sect. 3 and the instance-specific constraints in Sect. 4 via the Choco Solver 4.0.6 [27] with JDK version 10.0.1. All experiments were performed on a laptop with an Intel i7-3720QM CPU, 2.60 GHz with 4 physical and 8 logical cores, and 8 GB DDR3 memory running Linux Mint 18.3.

Table 8 summarizes the experimental results on the instances solved sequentially, including the total CPU time, the number of visited nodes, backtracks, and fails. It also provides search strategies we used. By using our approach, we were able to prove the nonexistence of the solution of 6-6-7 and solved six open instances for constraint satisfaction approach but not for metaheuristic approach [7].

Table 8. Results on the s-s-$(s + 1)$ Instances. A superscript "c" means that the instance was open for constraint satisfaction approach; "dom" and "min" denote the predefined search strategies domOverWDegSearch and minDomLBSearch in Choco Solver, respectively. (Table reproduced from [20]).

Instance	Time(s)	Nodes	Backtracks	Fails	Strategy
5-3-7	0.095	111	179	94	dom
5-5-6	0.069	7	1	0	min
6-6-7^{c}	25	1.38e5	2.77e5	1.38e5	min
7-7-8^{c}	111	3.62e5	723e5	3.62e5	min
8-8-9^{c}	12	15,370	30,680	15,350	min
9-9-10^{c}	2559	2.08e6	4.16e6	2.08e6	min
11-11-12^{c}	62	3,150	6,279	3,144	min
13-13-14^{c}	2563	5.80e4	1.16e5	5.79e4	min

6.2 Experimental Results on Instance Solved in Parallel

To validate our parallel approach for the SGP, we switch to a computer with 250 GB DDR3 1066 memory and 4 Intel Xeon CPU E7-4830 2.13 GHz processors running on Linux CentOS 6.5, where each processor has 8 physical cores. The versions of Choco Solver and the JDK are unchanged. Table 9 reports the experimental results for comparing parallel and sequential execution when using the same model to solve the same instance. For parallel execution, the number of workers we used varies from instance to instance. For 6-3-8, we specified 8, 16 and 32 workers to execute in parallel, but super-liner speedup was only observed when using 8 workers, because the partial assignment that can lead to a solution does not happen to be evaluated first.

Table 9. Results on the Instances solved in parallel. A superscript "f" means that the instance is solved by computer for the first time. A "-" sign means the program was still running after a period which is equal to the number of workers multiplied by the execution time in parallel. (Table reproduced from [20]).

Instance	Workers	Time(s)	Nodes	Backtracks	Fails	Strategy
6-3-8^{c}	1	2.95e4	2.91e8	5.83e8	2.91e8	min
	8	50.2	2.09e5	4.18e5	2.09e5	min
	16	2.62e4	2.50e8	5.13e8	2.31e8	min
6-4-7^{f}	1	-	-	-	-	min
	48	8.59e3	1.66e7	3.32e7	1.66e7	min
7-3-10^{f}	1	-	-	-	-	dom
	32	7.61e4	1.86e8	3.73e8	1.86e8	dom

Then, for 6-4-7, we used 48 workers because there are only 48 solutions generated by the separate model. Finally, the result of 7-3-8 is given by selecting the first 8 solutions of the separate model, and every solution is allocated to 4 different workers, each of which employs their respective search strategies that are predefined in Choco Solver, including *minDomUBSearch*, *minDomLBSearch*, *defaultSearch* and *domOverWDegSearch*. Besides, we also performed three more experiments in which the separate model was specified with above mentioned search strategies. As a consequence, the first 8 solutions are different from the first experiment, and we obtained three more non-isomorphic solutions for the 7-3-10 instance. The solutions of the instances in Table 9, which are not given in the main body of this article, are provided in Appendix A.

6.3 Discussion

It is interesting to observe the results for the instances of the form *s-s-(s + 1)* ($s = \{5, 7, 8, 11, 13\}$) consisting of $s-1$ mutually orthogonal $s \times s$ latin squares[4] (cf. GS_1, GS_2, GS_3, and GS_4 of Table 3). The results of these instances are consistent with the basic correspondence of *affine planes* and Latin squares, which proves that there exist $n-1$ Mutually Orthogonal Latin Squares (MOLS) of order n iff there exists an affine plane of order n [2,25], i.e., there are affine planes of order 5, 7, 8, 11, and 13. It is also not difficult to relate no solution for 6-6-7 to no MOLS of order 6 [4]. And we argue that the solution of 10-10-11 is nonexistent because there is no set of 7 MOLS of order 10 [22] and thereby no affine plane of order 10 [17]. More generally, we speculate that the solutions for the form *np-np-(np+1)* (e.g., $np = 14, 21, 22, 30, 33$) do not exist because of the nonexistence of *projective planes*[5] for them according to the *Bruck-Ryser-Chowla theorem* [2], where $np \equiv 1$ *or* $2 \pmod 4$ and the square-free part of np contain at least one prime $p \equiv 3 \pmod 4$.[6] Moreover, the 12-12-13 instance is hard for the CP approach, which corresponds to searching an affine plane of order 12—an unsettled case.

In addition to the results of the instances, we also show that more instance-specific constraints can shorten the execution time even if the size of instances increases. For example, 11-11-12 took much less time than 9-9-10 since more constraints are posted. The experimental results also show that parallel constraint solving through search space splitting is a very effective means to prevent backtrack search from getting stuck into a fruitless search area. Without surprise, the super-linear speedup was observed since only one invalid partial assignment is enough to cause instances such as 6-4-7 to be unsolvable for sequential solving and one valid partial solution can easily lead to backtrack search into a search area with a solution. Note that observed super-linear speedups are not in

[4] Two Latin squares are mutually orthogonal if, they have the same order n and when superimposed, each of the possible n^2 ordered pairs occur exactly once.

[5] An affine plane of order n exists iff a projective plane of order n exists.

[6] For instance, $14 = 2 * 7 \equiv 2 \pmod 4$, and the primes in the square-free part are 2 and 7.

contradiction with Amdahl's law since our goal is to obtain a first solution instead of all solutions.

7 Related Work

There is a substantial body of work available on symmetry breaking for the SGP from the constraints community, including model reformulation, static symmetry breaking constraints, and dynamic symmetry breaking.

7.1 Methods from the CSP Literature

Smith [37] presented the integer set model with extra auxiliary variables that automatically eliminates the symmetries inside of groups, which is probably one of the first works that break the symmetry of the SGP via model reformulation. Besides, Symmetry Breaking During Search (SBDS) with symmetry breaking constraints is employed to break renumbering symmetry but not entirely, where SBDS is essentially a search space reduction technique that adds constraints to remove symmetrical search space during search. Law and Lee [18] developed the *Precedence* constraint to break the symmetries of groups inside of weeks for the integer model and the symmetries caused by renumbering players for the set model. Symmetry Breaking via Dominance Detection (SBDD), another dynamic symmetry breaking technique, was developed separately by Focaci and Milano [10] and by Fahle *et al.* [9,11]. The main idea of SBDD is to utilize no-good learning to avoid exploring search space that is symmetrical of previously explored nodes recorded on the no-goods. By using SBDD, Fahle and Milano discovered seven non-symmetric solutions for the 5-3-7 instance in less than two hours on a computer with an UltraSparc-II 400 MHz processor.

Barnier and Brisset [3] proposed SBDD+ for the SGP, which computes isomorphism not only for leaves of the search tree but also on current non-leaves node. The experimental results showed that SBDD+ only took around eight seconds to compute all the seven non-symmetric solutions for 5-3-7, which is a significant improvement compared with [9]. However, they also pointed out that SBDD+ has to tackle the explosion of node store and the time overhead due to nodes dominance checking for a larger instance. Puget [28] combined SBDD with Symmetry Breaking Using Stabilizers (STAB) to obtain a solution of 5-5-6 in 38 s on a laptop with a Pentium M 1.4 GHz processor, where STAB is a variant of SBDS that adds symmetry breaking constraints without changing specified partial assignment.

All of the above mentioned works aim at eliminating the symmetries of the SGP, which is the first difficulty mentioned in Sect. 2.1. To tackle the second difficulty, Sellmann and Harvey [35] developed the vertical constraints and horizontal constraints for propagation, which can check whether a given partial assignment is extensible to a solution. They obtained all unique solutions of the 5-3-7 instance in 393.96 s on a computer with Pentium III 933 MHz processor by using the dedicated constraints. However, the dedicated constraints are

developed for the original naive model, and no efficient algorithm for finding the players who have conflicting residual graphs is given.

7.2 Methods from the Metaheuristic Literature

Despite having elegant and sophisticated search space reduction techniques such as SBDS, SBDD, etc., the constraint satisfaction approach, a systematic search method, cannot compete with the metaheuristic approaches on the SGP when the goal is to obtain one solution instead of all non-symmetric solutions. Dotú and Van Hentenryck [7] employed tabu search with a constructive seeding heuristic and good starting points to achieve significant results on the instances of the form *prime-prime-(prime + 1)* (e.g. 43-43-44, 47-47-48). Dotú and Van Hentenryck also solved 9-9-10 and 6-3-8 by using tabu search with a good starting point in 0.01 s and 51.93 s on a computer with Pentium IV 3.06 GHz processor [8]. Besides, the 6-3-8 instance was also solved by the evolutionary approach on a Pentium IV 3.06 GHz processor [5]. Unfortunately, the total CPU time is not reported in [8].

Triska and Musliu [38] are the first to solve the 8-4-10 instance reported in the literature, although one solution of 8-4-10 had already been published before [1] but without any explanation. The idea behind their metaheuristic approach is to employ a greedy heuristic for tabu search with the well-designed greedy initial configuration. The first solution of 8-4-10 instance was obtained in 11 min on a computer with an Intel Core 2 Duo 2.16 processor. Moreover, after varying the randomization factor of the greedy heuristic, they obtained two new non-isomorphic solutions for 8-4-10. In addition to the metaheuristic approach, they also explored a SAT encoding for the SGP [39]. Unfortunately, their SAT encoding is not competitive with other approaches.

Generally, solving the *q-q-(w + 2)* instance of the SGP amounts to finding w Mutually Orthogonal Latin Squares (MOLS). Thus, in addition to these approaches mentioned above which address the SGP head-on, Harvey and Winterer [14] exploited MOLS (in practice, MOLR) solutions found to construct solutions to the SGP. The most notable instance they solved is 20-16-6, which indicates that this is probably the most efficient method so far. However, no full instance g-s-w^* was resolved since this method heavily relies on the construction of MOLR.

7.3 Summary

Most of the research from the constraints community focus on search space reduction techniques, mainly dynamic symmetry breaking. The metaheuristic approach, by contrast, aims at finding a first solution as quickly as possible. For example, the 6-3-8 instance could be solved within reasonable time via the metaheuristic approach but not the constraint satisfaction approach. Note that the problem grows much faster even from 5-3-7 to 6-3-8 than the performance boost out of the processors. Table 10 summarizes the main accomplishments in

the SPG from the computer-science community, including both the constraints and metaheuristics communities.

Table 10. The summary of the most significant results on the SGP from the computer-science community.

Instance	Year	Authors	Method	Description
4-3-4	2001	Smith [37]	SBDS	42 solutions found
5-3-7	2001	Fahle and Milnano [9]	SBDD	7 unique solutions in 2 h
5-3-7	2001	Barnier and Brisset [3]	SBDD+	7 unique solutions in 8 s
5-3-7	2002	Sellmann and Harvey [35]	Specific Constraints	7 unique solutions in 394 s
5-5-6	2005	Puget [28]	SBDD and STAB	A solution in 38 s
20-16-6	2005	Harvey and Winterer [14]	MOLR	Tabu search for MOLR
47-47-48	2005	Dotú *et al.* [7]	Tabu-search	Efficient for p-p-$(p+1)$
6-3-8	2007	Dotú *et al.* [8]	Tabu-search	A solution in 52 s
8-4-10	2011	Triska and Musliu [38]	Tabu-search	2 new unique solutions found

Finally, some instances which have not been solved by computer at present have already been constructed by combinatorics (e.g., 7-4-9, 9-3-13). For a detailed introduction, please refer to [29,32].

8 Conclusion

In this paper, we have presented a combination of techniques which allows us to find solutions for eight open instances, where six of these instances are solved sequentially, and three of these instances are solved in parallel. In particular, we have shown the constraints derived from the relatively small instances can be used to solve larger instances that are in the same form as the smaller ones. In other words, we explore the properties of the instances of the form s-s-$(s+1)$ from the perspective of constraint programming. Besides, we have also shown that it is not uncommon for solving the SGP in parallel via search space splitting or with portfolio to gain super-linear speedups and parallel solving the SGP can be an effective method to address the instances that cannot be solved sequentially. The results show that our method is much more successful, even if we consider that the computers used for the other methods are up to 10 times slower than ours.

Unlike the earlier researches on the SGP which mainly focus on dynamic symmetry breaking, we attribute the success of our approach to the effectiveness of the instance-specific constraints and parallelism due to mitigating the two problems of solving the SGP mentioned in Sect. 2.1. Specifically, the instance-specific constraints imposed on the second submatrix of the decision variables matrix prune a large number of the sub-search trees near the root, including some symmetries. And since many partial assignments are extended simultaneously, fruitless partial assignments have no impact on overall execution time.

Not only that, but search space splitting can result in the partial assignments that can lead to a solution to be proceeded much earlier than the sequential search, which is the reason for super-linear speedup. Furthermore, we can conclude that early diversity brought by search space splitting before search can effectively alleviate the strong commitment due to the early decisions made by search strategy. Besides, we also remove the symmetries of the second row in the decision variables matrix when generating the partial assignments, which is helpful because nodes near the root contain much more symmetries than the nodes near the leaves of the search tree [28]. Therefore, with mainstream computers turning into parallel architectures, we believe that parallel constraint solving through search space splitting is a promising approach to solving more significant instances of the SGP.

Indeed, there is still a lot of potential to improve the performance of our approach. In particular, Constraints (22) is unable to eliminate the symmetries among weeks after week s when solving 6-3-8, 6-4-7, and 7-3-10. In fact, we have resolved it by enforcing the indices of the second "1" of all the weeks in ascending order, which means that the second golfers assigned in the first group are in ascending order. Unfortunately the performance is not satisfactory. As future work, we want to know whether the performance degradation is due to the use of the *IfThen* constraints or removal of symmetries that also simultaneously removes solutions. Besides, despite better than the *Reified* constraints, Constraints (5) and (6) introduce too many auxiliary variables that inevitably slow down the resolution process; thus, we have also implemented a specialized constraint to replace them. However, our constraint increases the difficulty of variable-selection since the constraint requires an additional variable to record the equality relationship among rows of the matrix G. To solve larger instances, in addition to using more processors and discovering more instance-specific constraints, we would like to consider combining the dynamic symmetry breaking and parallel constraint solving for the SGP.

In the end, we must regretfully admit that even if we have made some progress, some interesting instances are still open (e.g. 7-4-9, 8-3-11, and 9-3-13); notably, the original SGP 8-4-10 [13] is still unsolved for the CP approach, despite many efforts from the constraint programming community. Constraint technology should solve these instances to demonstrate itself as the first choice for solving combinatorial problems.

A Appendix

The Solutions

See Tables 11, 12, 13, 14, 15, 16 and 17.

Table 11. The solution for 6-3-8 transformed from the solution shown in Table 6.

Week \ Group	1			2			3			4			5			6		
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1	4	7	2	10	13	3	8	16	5	14	17	6	11	15	9	12	18
3	1	5	16	2	4	14	3	6	7	8	12	17	9	11	13	10	15	18
4	1	6	10	2	5	18	3	4	9	7	13	17	8	11	14	12	15	16
5	1	8	15	2	11	17	3	13	18	4	10	16	5	7	12	6	9	14
6	1	9	17	2	7	15	3	12	14	4	11	18	5	8	10	6	13	16
7	1	12	13	2	9	16	3	5	11	4	15	17	6	8	18	7	10	14
8	1	14	18	2	6	12	3	10	17	4	8	13	5	9	15	7	11	16

Table 12. A new non-isomorphic solution for the 6-3-8 instance.

Week \ Group	1			2			3			4			5			6		
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	1	4	7	2	5	8	3	6	10	9	13	16	11	14	17	12	15	18
3	1	5	16	2	4	15	3	7	12	6	8	14	9	10	17	11	13	18
4	1	6	9	2	10	18	3	4	11	5	12	13	7	14	16	8	15	17
5	1	8	18	2	12	16	3	9	15	4	13	17	5	10	14	6	7	11
6	1	10	13	2	7	17	3	14	18	4	8	12	5	9	11	6	15	16
7	1	11	15	2	9	14	3	8	13	4	10	16	5	7	18	6	12	17
8	1	12	14	2	6	13	3	5	17	4	9	18	7	10	15	8	11	16

Table 13. The solution for 6-4-7 transformed from the solution shown in Table 7.

Week \ Group	1				2				3				4				5				6			
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
2	1	5	9	13	2	6	17	21	3	7	18	22	4	8	10	14	11	15	19	23	12	16	20	24
3	1	6	10	24	2	7	12	15	3	8	13	19	4	11	20	21	5	16	18	23	9	14	17	22
4	1	7	16	17	2	8	11	22	3	9	15	20	4	5	19	24	6	12	14	23	10	13	18	21
5	1	8	20	23	2	9	18	24	3	6	11	16	4	12	13	17	5	10	15	22	7	14	19	21
6	1	11	14	18	2	10	16	19	3	5	12	21	4	7	9	23	6	13	20	22	8	15	17	24
7	1	12	19	22	2	5	14	20	3	10	17	23	4	6	15	18	7	11	13	24	8	9	16	21

Table 14. A new non-isomorphic solution for the 7-3-10 instance.

Week \ Group	1			2			3			4			5			6			7		
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	1	4	7	2	5	10	3	8	11	6	13	16	9	14	19	12	17	20	15	18	21
3	1	5	18	2	8	15	3	17	19	4	14	20	6	9	11	7	12	13	10	16	21
4	1	6	19	2	7	16	3	5	12	4	13	17	8	14	21	9	10	18	11	15	20
5	1	8	17	2	4	11	3	9	16	5	13	21	6	14	18	7	10	20	12	15	19
6	1	9	15	2	13	19	3	4	21	5	8	20	6	10	17	7	11	18	12	14	16
7	1	10	14	2	12	18	3	6	15	4	8	16	5	7	19	9	13	20	11	17	21
8	1	11	13	2	14	17	3	18	20	4	9	12	5	15	16	6	7	21	8	10	19
9	1	12	21	2	6	20	3	7	14	4	10	15	5	9	17	8	13	18	11	16	19
10	1	16	20	2	9	21	3	10	13	4	18	19	5	11	14	6	8	12	7	15	17

Table 15. A new non-isomorphic solution for the 7-3-10 instance.

Week \ Group	1			2			3			4			5			6			7		
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	1	4	7	2	10	13	3	16	19	5	8	20	6	11	14	9	12	17	15	18	21
3	1	5	11	2	8	21	3	15	20	4	9	16	6	7	10	12	13	18	14	17	19
4	1	6	21	2	7	17	3	8	14	4	10	15	5	9	13	11	18	19	12	16	20
5	1	8	12	2	5	19	3	7	13	4	14	18	6	9	15	10	16	21	11	17	20
6	1	9	19	2	11	15	3	10	18	4	8	17	5	14	16	6	13	20	7	12	21
7	1	10	14	2	4	20	3	9	11	5	7	18	6	12	19	8	15	16	13	17	21
8	1	13	16	2	9	18	3	6	17	4	11	21	5	12	15	7	14	20	8	10	19
9	1	15	17	2	12	14	3	5	21	4	13	19	6	8	18	7	11	16	9	10	20
10	1	18	20	2	6	16	3	4	12	5	10	17	7	15	19	8	11	13	9	14	21

Table 16. A new non-isomorphic solution for the 7-3-10 instance.

Week \ Group	1			2			3			4			5			6			7		
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	1	4	7	2	5	10	3	6	11	8	12	13	9	16	19	14	17	20	15	18	21
3	1	5	19	2	13	16	3	10	15	4	18	20	6	8	14	7	11	17	9	12	21
4	1	6	18	2	9	11	3	8	21	4	10	14	5	13	20	7	15	16	12	17	19
5	1	8	15	2	6	17	3	9	18	4	13	19	5	11	21	7	10	20	12	14	16
6	1	9	17	2	15	20	3	4	12	5	8	16	6	10	19	7	14	21	11	13	18
7	1	10	16	2	4	21	3	13	17	5	9	14	6	12	15	7	18	19	8	11	20
8	1	11	14	2	8	19	3	16	20	4	9	15	5	12	18	6	7	13	10	17	21
9	1	12	20	2	14	18	3	5	7	4	8	17	6	16	21	9	10	13	11	15	19
10	1	13	21	2	7	12	3	14	19	4	11	16	5	15	17	6	9	20	8	10	18

Table 17. A solution of 8-8-9 expressed by groups.

Week \ Player	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3
2	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
3	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	1	2	3	4	5	6	7	8	3	4	1	2	7	8	5	6	6	5	8	7	2	1	4	3
5	1	2	3	4	5	6	7	8	4	3	2	1	8	7	6	5	8	7	6	5	4	3	2	1
6	1	2	3	4	5	6	7	8	5	6	7	8	1	2	3	4	2	1	4	3	6	5	8	7
7	1	2	3	4	5	6	7	8	6	5	8	7	2	1	4	3	4	3	2	1	8	7	6	5
8	1	2	3	4	5	6	7	8	7	8	5	6	3	4	1	2	5	6	7	8	1	2	3	4
9	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1	7	8	5	6	3	4	1	2

Week \ Player	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
1	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6
2	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
3	4	3	2	1	8	7	6	5	5	6	7	8	1	2	3	4	6	5	8	7	2	1	4	3
4	8	7	6	5	4	3	2	1	2	1	4	3	6	5	8	7	4	3	2	1	8	7	6	5
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
6	6	5	8	7	2	1	4	3	7	8	5	6	3	4	1	2	3	4	1	2	7	8	5	6
7	7	8	5	6	3	4	1	2	3	4	1	2	7	8	5	6	8	7	6	5	4	3	2	1
8	3	4	1	2	7	8	5	6	8	7	6	5	4	3	2	1	2	1	4	3	6	5	8	7
9	2	1	4	3	6	5	8	7	4	3	2	1	8	7	6	5	5	6	7	8	1	2	3	4

Week \ Player	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
1	7	7	7	7	7	7	7	7	8	8	8	8	8	8	8	8
2	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
3	7	8	5	6	3	4	1	2	8	7	6	5	4	3	2	1
4	5	6	7	8	1	2	3	4	7	8	5	6	3	4	1	2
5	3	4	1	2	7	8	5	6	2	1	4	3	6	5	8	7
6	8	7	6	5	4	3	2	1	4	3	2	1	8	7	6	5
7	2	1	4	3	6	5	8	7	5	6	7	8	1	2	3	4
8	4	3	2	1	8	7	6	5	6	5	8	7	2	1	4	3
9	6	5	8	7	2	1	4	3	3	4	1	2	7	8	5	6

References

1. Aguado, A.: A 10 days solution to the social golfer problem. Math games: Social Golfer problem. MAA Online (2004)
2. Ball, S.: Finite Geometry and Combinatorial Applications, vol. 82. Cambridge University Press, Cambridge (2015)
3. Barnier, N., Brisset, P.: Solving the Kirkman's schoolgirl problem in a few seconds. In: Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Proceedings, Ithaca, NY, USA, 9–13 September 2002, pp. 477–491 (2002). https://doi.org/10.1007/3-540-46135-3_32
4. Benadé, J., Burger, A., van Vuuren, J.: The enumeration of k-sets of mutually orthogonal latin squares. In: Proceedings of the 42th Conference of the Operations Research Society of South Africa, Stellenbosch, pp. 40–49 (2013)
5. Cotta, C., Dotú, I., Fernández, A.J., Hentenryck, P.V.: Scheduling social golfers with memetic evolutionary programming. In: Hybrid Metaheuristics, Third International Workshop, HM 2006, Proceedings, Gran Canaria, Spain, 13–15 October 2006, pp. 150–161 (2006). https://doi.org/10.1007/11890584_12
6. Dechter, R.: Constraint Processing. Elsevier Morgan Kaufmann (2003). http://www.elsevier.com/wps/find/bookdescription.agents/678024/description

7. Dotú, I., Hentenryck, P.V.: Scheduling social golfers locally. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Proceedings, Prague, Czech Republic, 30 May–1 June, 2005, pp. 155–167 (2005). https://doi.org/10.1007/11493853_13
8. Dotú, I., Hentenryck, P.V.: Scheduling social tournaments locally. AI Commun. **20**(3), 151–162 (2007). http://content.iospress.com/articles/ai-communications/aic402
9. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In: Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Proceedings, Paphos, Cyprus, 26 November–1 December 2001, pp. 93–107 (2001). https://doi.org/10.1007/3-540-45578-7_7
10. Focacci, F., Milano, M.: Global cut framework for removing symmetries. In: Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Proceedings, Paphos, Cyprus, 26 November–1 December, 2001, pp. 77–92 (2001). https://doi.org/10.1007/3-540-45578-7_6
11. Gent, I.P., Petrie, K.E., Puget, J.: Symmetry in constraint programming. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 10, pp. 329–376. Elsevier (2006). https://doi.org/10.1016/S1574-6526(06)80014-3
12. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena insatisfiability and constraint satisfaction problems. J. Autom. Reasoning **24**(1/2), 67–100 (2000). https://doi.org/10.1023/A:1006314320276
13. Harvey, W.: CSPLib problem 010: Social golfers problem (2002). http://www.csplib.org/Problems/prob010. Accessed 28 Apr 2019
14. Harvey, W., Winterer, T.J.: Solving the MOLR and social golfers problems. In: Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Proceedings, Sitges, Spain, 1–5 October 2005, pp. 286–300 (2005). https://doi.org/10.1007/11564751_23
15. Hennessy, J.L., Patterson, D.A.: Computer Architecture - A Quantitative Approach, 5th edn. Morgan Kaufmann, Burlington (2012)
16. Kirkman, T.P.: Note on an unanswered prize question. Cambridge Dublin Math. J. **5**, 255–262 (1850)
17. Lam, C.W., Thiel, L., Swiercz, S.: The non-existence of finite projective planes of order 10. Can. J. Math. **41**(6), 1117–1123 (1989)
18. Law, Y.C., Lee, J.H.: Global constraints for integer and set value precedence. In: Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Proceedings, Toronto, Canada, 27 September–1 October 2004, pp. 362–376 (2004). https://doi.org/10.1007/978-3-540-30201-8_28
19. Lecoutre, C.: Constraint Networks: Techniques and Algorithms. Wiley, Hoboken (2009)
20. Liu., K., Löffler., S., Hofstedt., P.: Solving the social golfers problems by constraint programming in sequential and parallel. In: Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, pp. 29–39. INSTICC, SciTePress (2019). https://doi.org/10.5220/0007252300290039
21. Liu, K., Löffler, S., Hofstedt, P.: Solving the traveling tournament problem with predefined venues by parallel constraint programming. In: Mining Intelligence and Knowledge Exploration - 6th International Conference, MIKE 2018, Proceedings, Cluj-Napoca, Romania, 20–22 December 2018, pp. 64–79 (2018). https://doi.org/10.1007/978-3-030-05918-7_7
22. McKay, B.D., Meynert, A., Myrvold, W.: Small latin squares, quasigroups, and loops. J. Comb. Des. **15**(2), 98–119 (2007)

23. Miguel, I.: CSPLib problem 038: steel mill slab design (2012). http://www.csplib.org/Problems/prob010. Accessed 28 Apr 2019
24. Palmieri, A., Régin, J., Schaus, P.: Parallel strategies selection. In: Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Proceedings, Toulouse, France, 5–9 September 2016, pp. 388–404 (2016). https://doi.org/10.1007/978-3-319-44953-1_25
25. Parker, E.T.: Construction of some sets of mutually orthogonal latin squares. Proc. Am. Math. Soc. **10**(6), 946–949 (1959)
26. Prestwich, S.: CSPLib problem 028: balanced incomplete block designs (2001). http://www.csplib.org/Problems/prob010. Accessed 28 Apr 2019
27. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC - LS2N CNRS UMR 6241, COSLING S.A.S. (2017). http://www.choco-solver.org
28. Puget, J.: Symmetry breaking revisited. Constraints **10**(1), 23–46 (2005). https://doi.org/10.1007/s10601-004-5306-8
29. Rees, R.S., Wallis, W.D.: Kirkman triple systems and their generalizations: a survey. In: Wallis, W.D. (ed.) Designs 2002. MIA, vol. 563, pp. 317–368. Springer, Boston, MA (2003). https://doi.org/10.1007/978-1-4613-0245-2_13
30. Régin, J.-C., Rezgui, M., Malapert, A.: Embarrassingly parallel search. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 596–610. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40627-0_45
31. Régin, J., Rezgui, M., Malapert, A.: Embarrassingly parallel search. In: Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Proceedings, Uppsala, Sweden, 16–20 September 2013, pp. 596–610 (2013). https://doi.org/10.1007/978-3-642-40627-0_45
32. de Resmini, M.J.: There exist at least three non-isomorphic s (2, 4, 28)'s. J. Geom. **16**(1), 148–151 (1981)
33. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006). http://www.sciencedirect.com/science/bookseries/15746526/2
34. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with gecode. Gecode Team (2017). https://www.gecode.org/
35. Sellmann, M., Harvey, W.: Heuristic constraint propagation-using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem. In: CPAIOR 2002, Citeseer (2002)
36. Smith, B.M.: Modelling. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 11, pp. 377–406. Elsevier (2006). https://doi.org/10.1016/S1574-6526(06)80015-5
37. Smith, B.M.: Reducing symmetry in a combinatorial design problem. In: CPAIOR 2001, pp. 351–359, April 2001. http://www.icparc.ic.ac.uk/cpAIOR01
38. Triska, M., Musliu, N.: An effective greedy heuristic for the social golfer problem. Ann. Oper. Res. **194**(1), 413–425 (2012). https://doi.org/10.1007/s10479-011-0866-7
39. Triska, M., Musliu, N.: An improved SAT formulation for the social golferproblem. Ann. Oper. Res. **194**(1), 427–438 (2012). https://doi.org/10.1007/s10479-010-0702-5