Solving the Social Golfers Problems by Constraint Programming in Sequential and Parallel

Ke Liu[a], Sven Löffler and Petra Hofstedt

Brandenburg University of Technology Cottbus-Senftenberg, Germany
Department of Mathematics and Computer Science, MINT,
Konrad-Wachsmann-Allee 5, 03044 Cottbus, Germany

Keywords: Constraint Programming, Constraint Satisfaction, Parallel Constraint Solving, Sports Scheduling, Social Golfer Problem.

Abstract: The social golfer problem (SGP) has received plenty of attention in constraint satisfaction problem (CSP) research as a standard benchmark for symmetry breaking. However, the constraint satisfaction approach has stagnated for solving larger SGP instances over the last decade. We improve the existing model of the SGP by introducing more constraints that effectively reduce the search space, particularly for instances of special form. Furthermore, we present a search space splitting method to solve the SGP in parallel through data-level parallelism. Our implementation of the presented techniques allows us to attain solutions for eight instances with maximized weeks, in which six of them were open instances for the constraint satisfaction approach, and two of them are computed for the first time. Besides, super-linear speedups are observed for all the instances solved in parallel.

1 INTRODUCTION

The social golfer problem (SGP), i.e., 010 problem in CSPLib (Harvey, 2002), is a typical combinatorial optimization problem that has attracted significant attention from the constraint programming community due to its highly symmetrical and combinatorial nature. The computational complexity of the SGP is yet unknown. However, determining whether a partial assignment can be extended to a solution is NP-complete (Colbourn, 1984). The problem consists of scheduling $n{=}g*s$ golfers into g groups of s golfers for w weeks so that any two golfers are assigned to the same group at most once in w weeks. By the convention, 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 golfers within a group, and w is the number of weeks in the schedule.

The SGP can be viewed as an optimization problem which seeks a schedule (solution) for a given g and s with the maximum number w^* weeks ($w^* \leq \frac{n-1}{s-1}$). Obviously, a solution for an instance g-s-w^* implies the solutions for all instances g-s-w with $w < w^*$. In light of this, this paper considers only solving the g-s-w^* instances with maximized weeks, which we call the full instance. For example, Table 1 depicts one solution for the full instance 7-3-10 social golfer problem.

The SGP is a simple-sounding question, and it can be modeled by several common 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. 8-4-10, 9-3-13), which are mainly caused by the following two reasons (difficulties):

- The inherent highly symmetrical nature of the SGP cannot be entirely known before solving process. Despite the symmetries, which are caused by interchangeable players inside groups ($s!$), interchangeable groups inside weeks ($g!$), and arbitrarily ordered weeks ($w!$), can be removed through model reformulation and static symmetry breaking constraints, it is difficult to eliminate all symmetries among golfers caused by renumbering n golfers ($n!$).[1] Consequently, the unnecessary symmetrical search space is explored redundantly (Barnier and Brisset, 2002).

[a] https://orcid.org/0000-0002-5256-9253

[1] For example, if golfers $[16,17,18,19,20,21]$ 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 $[1,2,3,...,19,20,21]$.

Liu, K., Löffler, S. and Hofstedt, P.
Solving the Social Golfers Problems by Constraint Programming in Sequential and Parallel.
DOI: 10.5220/0007252300290039
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 29-39
ISBN: 978-989-758-350-6

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 the search.

Group / Week	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

- A large proportion of partial assignments that cannot lead to a solution is likely to cause the backtrack search to trap in a sizeable fruitless subspace and therefore waste valuable computation time. More importantly, it is often hard to determine the usefulness of a partial assignment until almost all variables are instantiated.

The research on the SGP is meaningful not only in itself but also for other constraint satisfaction problems (CSPs), such as balanced incomplete block design (BIBD) (Prestwich, 2001) and steel mill slab design (Miguel, 2012), etc. The reason is that we are likely to face the same difficulties as the SGP when solving other CSPs through the constraint satisfaction approach.

In this paper, we first classify the researches on the SGP by how they resolve the two difficulties mentioned above and also survey the studies related to the SGP outside the context of constraint programming (Section 2). Next, Section 3 present a modeling approach improved on the model proposed by Barnier and Brisset (2002). Then, some instance-specific constraints are introduced in Section 4. After that, we elaborate on how to employ Embarrassingly Parallel Search (EPS) (Régin et al., 2013) to solve the SGP in Section 5. Finally, we conclude in Section 6. The contributions of this paper are twofold. First, the performance of the CP approach on the SGP is improved by the introduction of additional problem-specific constraints. Second, it shows 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.

2 RELATED WORK

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

Smith (2001) presented the integer set model with extra auxiliary variables that automatically eliminates the symmetries inside of groups. Moreover, Symmetry Breaking During Search (SBDS) with symmetry breaking constraints is employed to break renaming symmetry but not entirely, where SBDS is essentially a search space reduction technique by adding constraints to remove symmetrical search space during the search. Law and Lee (2004) introduced the **Precedence** constraint to break the symmetries of groups inside of weeks for the integer model and the symmetries caused by renaming golfers for the set model. Symmetry Breaking via Dominance Detection (SBDD), another dynamic symmetry breaking technique, was developed separately by Focacci and Milano (2001) and by Fahle et al. (2001); Rossi et al. (2006). SBDD utilizes no-good learning to avoid exploring search space that is symmetrical of previously explored nodes recorded on the no-goods. As with SBDD, Fahle & Milano discovered 7 non-symmetric solutions for the 5-3-7 instance in less than 2 hours on a computer with an UltraSparc-II 400 MHz processor.

Barnier and Brisset (2002) 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 takes around 8 seconds to compute all 7 non-symmetric solutions for 5-3-7, which is a significant improvement compared with (Fahle et al., 2001). However, they also point out that SBDD+ has to tackle the explosion of node store and the time overhead due to nodes dominance checking if one wants to apply SBDD+ to a larger instance and we also believe this is the common problem for other dynamic symmetry breaking techniques, including SBDS. Puget (2005) combined SBDD with Symmetry Breaking Using Stabilizers (STAB) to obtain a solution of the instance 5-5-6 in 38 seconds 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 the specified partial assignment. To tackle the second difficulty, Sellmann and Harvey (2002) 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 seconds on a computer with Pentium III 933 MHz processor by imposing vertical constraints and horizontal constraints on the decision variables. But the constraints are developed for the original naive model, and no efficient algorithm for finding the golfers who have conflicting residual graphs is given.

Despite its elegant and sophisticated search space reduction techniques such as SBDS, SBDD, etc., the CP approach, a systematic search method, cannot compete with metaheuristic approaches on the SGP if the goal is to obtain one solution instead of all non-symmetric solutions. Please note that the problem grows much faster even from 5-3-7 to 6-3-8 than the performance boost out of the processors. Dotú and Van Hentenryck (2005) 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). They also found a solution 9-9-10 and 6-3-8 by using tabu search with a good starting point in 0.01 second and 51.93 seconds on a Pentium IV 3.06 GHz processor (Dotú and Van Hentenryck, 2007). Besides, Cotta et al. (2006) used an evolutionary approach to solve 6-3-8 on a computer with a Pentium IV 3.06 GHz processor (no CPU time is given). Triska and Musliu (2012a) are the only computer scientists who successfully solved the 8-4-10 instance published so far. They employed a greedy heuristic for tabu search with the well-designed greedy initial configuration to solve the instances 8-4-10 in 11 minutes on a computer with an Intel Core 2 Duo 2.16 processor. Besides, they also explored a SAT encoding for the SGP (Triska and Musliu, 2012b). Unfortunately, their SAT encoding is not competitive with other approaches.

In addition to these approaches mentioned above which address the SGP head-on, Harvey and Winterer (2005) used the Mutually Orthogonal Latin Rectangle (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 relies heavily on the construction of MOLR.

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 (de Resmini, 1981; Rees and Wallis, 2003).

3 THE BASIC MODEL

There are various ways to model the SGP, which is one of the reasons why the problem is so compelling. We use a model improved on the model presented in (Barnier and Brisset, 2002), and more constraints are added into the model to tackle larger instances piece by piece. The present paper involves the use of several types of constraints, including the **GCC**, **AllDifferent**, **Count**, **Table**, and **Arithmetic** constraints (Beldiceanu et al., 2012).

The decision variables in our model is a $w \times n$ matrix G, where the first row (week) and first s columns (golfers) of the matrix are fixed (cf. Table 2). An element $G_{i,j}$ of the matrix G stands for golfer j is assigned to group $G_{i,j}$ in week i. The domain of the decision variable $G_{i,j}$ is a set of integers consisting of $\{1..g\}$, where $0 \leq i < w,\ 0 \leq j < n$. (In the present paper, the index of each subscript of a matrix starts from 0.) The main advantage of the decision variables used in this model is the range of the variables reduced from $\{1..n\}$ to $\{1..g\}$ and the number of decision variables is unchanged, compared with the naive model that derived from the problem definition. In addition to this, the elements of the first row and first s columns without the first element are initialized to:

$$0 \leq j < n,\ G_{0,j} = j/s + 1 \tag{1}$$

$$0 < i < w,\ 0 \leq j < s,\ G_{i,j} = j + 1 \tag{2}$$

Equation 1 produces a sequence of integers from 1 to g in non-descending order, and each integer repeats exactly s times. Freezing the first row by this sequence of integers in our model can partially mitigate the symmetries caused by renumbering the golfers. Applying Equation 2 amounts to assigning the first s golfers to the first s groups after the first week, which can significantly reduce the search space, as well as a part of symmetries caused by interchangeable groups inside of weeks (cf. Table 1 and 2).

Since each group is composed of s number of golfers in every week, the constraint imposed on each row of the matrix G can be stated as:

$$\begin{aligned} 0 < i < w,\ V = \{1..g\},\ O = [s, .., s], \\ GCC(G_{i,*}, V, O) \end{aligned} \tag{3}$$

where GCC is an acronym for the **Global Cardinality Constraint** and the length of the list of variables O is g. The constraints ensure that each value in the set $\{1..g\}$ must occur exactly s times in each row of the matrix G. In other words, there are the n distinct golfers divided into g groups in each week via these constraints. The restriction, which says that no golfer meets the same golfer more than once, can be interpreted as no two columns of the matrix G have the

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 the search.

Week \ Golfers	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

same value at the same row more than once, which can be expressed as:

$$0 \leq j_1 < j_2 < n, \sum_{0 \leq i < w} \mid G_{i,j_1} - G_{i,j_2} = 0 \mid \leq 1 \quad (4)$$

Constraints 3 and 4 are the only two constraints that are the same as the model presented in (Barnier and Brisset, 2002). In particular, unlike Barnier and Brisset (2002), we realize the Constraint 4 in a different way to avoid using the **Reified** constraints because these constraints slow down the resolution speed. Specifically, the need for the **Reified** constraints can be replaced by introducing a $w \times m$ matrix C, where $m = \binom{n}{2}$. We subtract each 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 the matrix C. Simply put, the two matrices are linked by the **Arithmetic** constraints, which is given by:

$$0 \leq i < w,\ 0 \leq j_1 < j_2 < n,\ 0 \leq j_3 < \binom{n}{2}, \quad (5)$$
$$\forall j_1 \forall j_2,\ G_{i,j_1} - G_{i,j_2} = C_{i,j_3}$$

Then, we realize the restriction defined by Constraint 4 by imposing the **Count** constraint on each column of the matrix C, which can be stated as:

$$0 \leq j < m,\ occ = \{0,1\},\ count(C_{*,j},\ occ,\ 0) \quad (6)$$

where occ is an integer variable whose domain is $\{0,1\}$. Constraint 6 restricts the number of occurrences of value 0 on each column is no more than once.

Until now, the constraints presented have already stated all the restrictions defined by 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 (Smith, 2006). As we mentioned earlier, the first row of the matrix G is fixed with a sequence of integers by Equation 1, which implies that those golfers who have met in the first week cannot be assigned to the same group in the subsequent weeks. Thus, the **AllDifferent** constraint can be used to enforce the group numbers of these golfers are pairwise distinct after the first week, given by:

$$\forall 0 < i < w, \forall j \neq j' \wedge j/s = j'/s, G_{i,j} \neq G_{i,j'} \quad (7)$$

In summary, the basic model is composed of Constraints 1, 2, 3, 5, 6, and 7. Nevertheless, the performance of this model can be greatly improved by the introduction of additional constraints, such as symmetry breaking constraints, and the constraints derived by instance-specific information. On the basis of this model, in the subsequent sections, we will present the additional constraints dedicated to different instances and how to solve these instances in parallel if the instance cannot be solved sequentially.

4 INSTANCES SOLVED SEQUENTIALLY

For a given number of groups g and a group size s, our goal is to compute the first solution for a full instance g-s-w^*, where w^* maximizes the 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 equal to the number of golfers in the groups within each week, and the number of weeks is maximized because $\frac{s^2-1}{s-1} = s+1$. The exclusive 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)*) and *prime-prime-(prime+1)* (*p-p-(p+1)*). We also give our experimental results of the models executed sequentially.

4.1 7-7-8

Since $s * s$-1 is divisible by s-1, every golfer must play with every other exactly once. Thus, golfers whose number is greater than s must meet every golfer whose number is less than or equal to s exactly once in every week except the first week because the first s golfers have played together in the first week since Equation 1 freezes the first week.

Based on the above analysis, we can place the following constraints on the columns of the matrix G:

$$0 < i < i',\ s \leq j,\ \forall i \forall i' \forall j,\ G_{i,j} \neq G_{i',j} \tag{8}$$

Constraint 8 states that starting with s-th column, every column in the matrix G except its first element must be pairwise different, which can be implemented by the **AllDifferent** constraint. For the sake of illustration, we solved the instance 5-5-6 as an example. Begin with the sixth golfer, each column of Table 3 is a permutation of the 5-element array $[1,2,3,4,5]$. Therefore, apart from the first s columns, all the possible values of columns (column space) of the matrix G is reduced from s^s to $s!$ by introducing the Constraint 8, which is a significant search space reduction.

In Section 3 we have presented Constraint 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. The s number of golfers assigned in the same group cannot meet again in the subsequent weeks, and the domain of these golfers is the set of integers $\{1..s\}$, which implies that the possible values assigned to these s golfers must be a permutation of s-element array $[1,2,..,s]$. Moreover, arbitrary swapping two columns that have the same first elements leads to an isomorphism when the first row is fixed by the Constraint 1; hence, to avoid exploring the isomorphisms, we can choose a fixed column ordering. Therefore, for the instances of the form $s-s-(s+1)$, we can initialize the variables representing the golfers who have met in the first week with the array $[1,2,..,s]$ in the second week (cf. the second row of Table 3), which can be expressed as:[2]

$$s \leq j < n,\ G_{1,j} = j\%s+1 \tag{9}$$

The symmetries caused by renumbering the golfers in the second row can be eliminated by imposing Constraint 9. Summary: The constraints of the basic model and the Constraints 8 and 9 forms the model used to tackle *5-5-6*, 6-6-7, and 7-7-8.

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.

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

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 deleting the first row of G and selecting columns $[j,..,(j+s-1)]$, where j must be divisible by s. Hence, G has exactly s number of such submatrices with $w-1$ rows and s columns, and the i-th submatrix of G is denoted by GS_i, where $1 \leq i \leq s$ (Table 3).

We observe the solutions of the 4-4-5, 5-5-6, and 7-7-8 instances; and discover that GS_2 is always a symmetric matrix, namely $GS_2 = GS_2^T$. Hence, we conjecture that 9-9-10 also has a symmetric submatrix and then impose the following constraints on the decision variables G:

$$0 < i < w,\ s \leq j < 2*s,\ G_{i,j} = G_{(j-s),(i+s)} \tag{10}$$

Constraint 10 states that the entries of GS_2 are symmetric with respect to the main diagonal. Besides, the main diagonal of the sub-matrix GS_2 is pairwise distinct for 5-5-6 and 7-7-8, given by:

$$0 < i < w,\ s \leq j < 2s,\ \forall i \forall j,\ G_{i,j} \neq G_{(i+1),(j+1)} \tag{11}$$

Apart from the fixed pattern of GS_2, there is also a fixed pattern among the sub-matrices 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 golfers 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:

$$\begin{aligned} 1 < i < w,\ s \leq j < j' < n,\ j\%s = j'\%s, \\ \forall i \forall j \forall j',\ G_{i,j} \neq G_{i,j'} \end{aligned} \tag{12}$$

We notice that for 5-5-6 and 7-7-8, there is always a type solution in which the second row of GS_2 is fixed by the array $[2,s,1,3,4,..,s-1]$ (cf. the third row of Table 3). We therefore assume that 9-9-10 also exists such solution, and use it to solve 9-9-10. 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_2.

4.3 13-13-14 etc.

We have discovered some common features of the instances of the form *s-s-(s+1)*, particularly the instances of the form *o-o-(o+1)*, for the solution expressed by the number of groups; and these common features are mostly focused on the second submatrix GS_2 of

Table 3: A solution of 5-5-6 expressed by the number of groups. It can be converted to the solution expressed by the number of golfers easily; we don't provide it due to lack of space. The submatrices GS_2 and GS_3 are surrounded in the dotted line.

Week \ Golfers	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

G. It is also interesting to observe that the submatrix GS_i, $2 < i \leq s$, consists of s number of s-tuples that are derived from the second submatrix GS_2 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_2 on these instances. Note that we do not consider GS_1 since it is fixed by Constraint 2. Thus, we can solve larger instances of the form *p-p-(p+1)* by restricting row space of the submatrix GS_i, $2 < i \leq s$, to the rows of the submatrix GS_2. Formally:

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

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

where Constraint 13 defines the potential combination of values of columns of GS_2 as PT. Then we can limit the row space of the submatrices except GS_1 and GS_2 to PT by Constraint 14, which can be realized through the **Table** constraint. Hence, the problem is transferred to finding the GS_2 that can lead to a solution of the instance.

To find the correct GS_2, we create a separate model defined on a $s \times s$ matrix (s must be a prime number), which consists of 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 $[1, 2, ..., s]$ and $[2, s, 1, 3, 4, .., s-1]$ respectively, as we did for the 9-9-10 instance. Moreover, the last row ($i = s-1$) starting with the third element ($j = 2$) to the last element is fixed with the array $[2, 1, 3, 4, 5, ..., s-2]$, and the element at the tail of the array is removed with the row number decrementing (i−−) 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.

Having this observed pattern and aforementioned separated model, we can obtain exactly one GS_2 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_2 has already initialized before the solving process, we do not use the model of 9-9-10 because it is unnecessary to impose Constraints 10, 11, and 12 on the model.

Table 4: The second matrix GS_2 for the instance 13-13-14.

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**

All the instances we have introduced so far conform to the form of *o-o-(o+1)*; now we consider the form of instances *even-even-(even+1)*, we solved 8-8-9 by the following conjectures derived from 4-4-5 with the model for 7-7-8:

$$0 < i < w,\ G_{i,(i+s-1)} = 1 \tag{15}$$

$$0 < i < i' < w,\ 2*s \leq j < j' < n, \\ \forall j/s = j'/s \land j\%s+1 = i \land j'\%s+1 = i' \\ G_{i,j} \neq G_{i',j'} \tag{16}$$

Constraint 15 states that the main diagonal of the matrix GS_2 contains the fixed values [1,1,...,1]; and the rest of submatrices have the main diagonal whose variables must be pairwise distinct (Constraint 16).

Table 5: The second matrix GS_2 for a solution the instance 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 5 depicts the submatrix GS_2 of the solution of 8-8-9 we solved. It is interesting to observe that the GS_2 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. Unfortunately, we could not take advantage of these features to solve 10-10-11 and 12-12-13, 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.

4.4 Experimental Results

To confirm our theoretical discussion and the conjecture for the instances of 9-9-10 and 11-11-12 etc., we implemented the model as described in this section via the Choco Solver 4.0.6 (Prud'homme et al., 2017) with JDK version 10.0.1. All experiments were performed on a laptop with an Intel i7-3720QM CPU, 2.60GHz with 4 physical and 8 logical cores, and 8 GB DDR3 memory running Linux Mint 18.3. Table 6 shows the experimental results of above mentioned instances.

Table 6: 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.

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

Using the models presented in this section, we were able to prove 6-6-7 has no solution and solved 6 open instances for constraint satisfaction approach (but not for metaheuristic approach (Dotú and Van Hentenryck, 2005)). It is interesting to relate no solution for 6-6-7 to no Mutually Orthogonal Latin Squares (MOLS) of order 6 Benadé et al. (2013). The results 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.

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 which must deal with through parallel processing to obtain one solution. The term difficult refers 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 Hennessy and Patterson (2011). Moreover, since no communication is required during the solving process, to some extent, our parallel approach can also be seen as Embarrassingly Parallel Search (EPS) (Régin et al., 2013). Régin *et al.* defines that the EPS assigns the task to worker dynamically (Palmieri et al., 2016). 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. The generic procedure can be summarized as follows:

1. A subset of the decision variables of the model is selected.
2. All the partial assignments over selected variables in the subset are generated by a separate model 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 own 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. We adhere to the following principles when selecting the subset of the decision variables: 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 section, 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. As shown in Section 4, the 5-3-7 instance can be solved in less than one second using constraint satisfaction approach, but 6-3-8 was open for sequential constraint solving. For our modeling approach, the number of decision variables grows from $5*3*7 = 105$ to $6*3*8 = 144$, and the domain size of each variable is increased by one, which implies the overall underlying search space greatly increased when the target instance switches from 5-3-7 to 6-3-8 (5^{105} to 6^{144}).

Thus, we freeze a part of the decision variables so that the size of the sub-problem is shrunk to solvable, hence the original problem is likely to be solved. For 6-3-8, since the first row is always fixed in our modeling approach, we select the second row for the search space splitting and use a separate model to generate them, which is composed of the following constraints:

$$0 \le j < s,\ F_j = j+1 \tag{17}$$

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

$$\forall j \ne j' \wedge j/s = j'/s,\ F_j \ne F_{j'} \tag{19}$$

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

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

where F is an array of decision variables, and the domain of each variable is also $\{1..g\}$. Constraints 17, 18, and 19 are identical to Constraints 2, 3, and 7 stated in the basic model respectively. Constraints 20 and 21, which are not included in the basic model, are static symmetry breaking constraints. Constraint 20 breaks the symmetries caused by interchangeable submatrices GS_i, $1 \le i \le s$. We remove these symmetries by arranging the elements in the first column of the first row of all submatrices of G in non-decreasing order. 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.

Apart from the constraints of the separate model we also place the constraints to break the symmetries caused by interchangeable weeks partially.

$$0 \le i < w-1,\ G_{i,s} \le G_{(i+1),s} \tag{22}$$

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 2, interchanging the 5th week with 9th 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. Nonetheless, we amend the separate model for 6-3-8 to produce less number of solutions for the second row of 6-4-7. The separate model for 6-4-7 is formed by adding the following constraints to the separate model of 6-3-8:

$$\begin{aligned} &0 \le j < n,\ j\%s = 0,\\ &j = (s-1)*s \Rightarrow j+s \ne s*s,\\ &\forall j(F_{j+1} \le F_{j+s+1}) \end{aligned} \tag{23}$$

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

$$\begin{aligned} &\forall j(F_{j+1} = F_{j+s+1} \wedge F_{j+2} = F_{j+s+2}\\ &\Rightarrow F_{j+3} \le F_{j+s+3}) \end{aligned} \tag{25}$$

In short, Constraints (23-25) ensure that the elements occupying the same position in the first row of the first s submatrices (GS_1, GS_2, GS_3, GS_2) and the latter two submatrices (GS_5, GS_6) are in non-decreasing order respectively (Table 7). 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] would be generated. In that case, we would require more workers to work on these fruitless search trees due to symmetries.

As with the 6-3-8 instance, we map the solutions of the separate model to different workers before the solving process. Moreover, we add the following constraints to the basic model:

$$\begin{aligned} &s \le j < 3s,\ 0 < * < w,\\ &V = \{1..s\},\ O = \{1..1\},\\ &GCC(G_{*,j},V,O) \end{aligned} \tag{26}$$

where $G_{*,j}$ denotes the columns from the s^{th} column to the *(3s-1)*th column of the matrix G, each of which deletes its first element. For the 6-4-7 instance, this constraint guarantees that every column without the first element indexed between s to $3*s-1$ contains every value in the set {1,2,3,4} exactly once (Table 7). We impose Constraint 26 on only $2*s$ number of columns because $(24-1)\%(4-1) = 2$, which implies that each golfer only plays with other 21 golfers; thus not every column contains the set {1,2,3,4}. Though the columns imposed Constraint 26 do not include all columns containing the values {1,2,3,4}, it reduces much search space; experiments show 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. Just as we imposed many constraints on

Table 7: A solution of 6-4-7. The numbers with the same superscript are in non-decreasing order in the second row.

Week \ Golfers	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	$\boldsymbol{1^a}$	$\boldsymbol{2^b}$	$\boldsymbol{3^c}$	$\boldsymbol{4^d}$	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

GS_2 for 13-13-14 etc., we impose the following constraints on the model for 7-3-10:

$$0 < i \leq s \ , \ G_{i,s} = i+1 \tag{27}$$

$$s+1 < i < w \ , \ G_{i,s} = s+1 \tag{28}$$

$$V = \{1,2,3,6,7\}, \ O = [1,1,1,0,0], \quad 0 < * < w, \ GCC(G_{*,(s+1)}, V, O) \tag{29}$$

$$V = \{1,2,3,6\}, \ O = [1,1,1,0], \quad 0 < * < w, \ GCC(G_{*,(s+2)}, V, O) \tag{30}$$

$$s+s \leq j < n, \ V = \{1..s\}, \ O = [1..1], \quad 0 < * < w, \ GCC(G_{*,j}, V, O) \tag{31}$$

We strictly limit the positions of golfer 4 so that he/she will never be assigned to group 5 and 6. The reason is that golfer 4 is always the smallest golfer in a group after week 4 (Table 1) and golfer 4 also must be less than the smallest golfer in other groups except the first s groups. Therefore, Constraint 27 and 28 remove the symmetries caused by swapping the group containing golfer 4 with other groups after week 4. Since golfer 4 cannot appear in group 5, 6 and 7, golfer 5 is impossible to be assigned to group 6 and 7, because then there would be no golfer assigned in group 5. Similarly, golfer 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 values 6 and 7. Furthermore, because $(21-1)\%(3-1) = 0$, each golfer must play with other golfers exactly once. Hence, we guarantee every golfer other than the first s golfers meet the first s golfers once, which are ensured by Constaints 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 concrete 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. Moreover, we combine the search space splitting with the portfolio, where a partial assignment is distributed to several different workers, and these workers use the same model with different search strategies, including *fail-first* (Rossi et al., 2006) and *dom/wdeg* (Boussemart et al., 2004). Since we employ two different parallel constraint approaches including search space splitting and portfolio, our parallel approach for 7-3-10 can be viewed as a hybrid parallel constraint solving.

5.4 Experimental Results

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 2.13GHz processors running on Linux with kernel release 2.6.32-431.17.1.el6.x86_64, where each processor has 8 physical cores. The versions of Choco Solver and the JDK are unchanged.

Table 8: 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 time span which equals to the number of workers multiplied by the execution time in parallel.

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

Table 8 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 superliner speedup was only observed when using 8 workers, because the partial assignment that can extend to a solution does not happen to be evaluated first otherwise.

Then, for 6-4-7, we use 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 experimental results show that parallel constraint solving through search space splitting is a very effective means to prevent backtrack search getting stuck into the 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 unable to be solved when solving sequentially. In other words, super-linear speedups in parallel implementations are not in contradiction with Amdahl's law because workers can evaluate the partial solutions that can lead to a solution in early resolution process.

6 CONCLUSION AND FUTURE WORK

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 which are in the same form as the smaller ones. 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 serially. 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. As a reference point, the individual core performance of i7-3720QM and E7-4830 is around three and two times faster than Pentium IV 3.06, respectively.[3]

Unlike the earlier researches on the SGP which mainly focus on dynamic symmetry breaking, we attribute the effectiveness of the instance-specific constraints and parallelism to mitigating the two problems mentioned in the Introduction. Specifically, the instance-specific constraints imposed on the second sub-matrix of the matrix of the decision variables 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. Besides, we also remove the symmetries of the second row 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 (Puget, 2005). Not only that, but search space splitting can result in the partial assignments that can extend to a solution to be proceeded much earlier than serial search. Therefore, with the trend of integrating more and more cores on a single chip, we regard the parallel constraint solving via search space splitting as an excellent alternative to fast restart policies (Perron, 2003) and discrepancy search (Shaw, 1998) for constraint solving. Moreover, as we emphasized before, research on the SGP is not restricted to the problem itself, we believe the parallel approach can be generalized to deal with other combinatorial optimization problems.

Indeed, there is still a lot of potential to improve the performance of our approach. In particular, we have not removed the symmetries among weeks after week s, though we employ Constraints 22 when solving 6-3-8, 6-4-7, and 7-3-10. In fact, we have resolved it by enforcing the indices of the first "1" of all the weeks in ascending order, which means that the second golfers assigned in the first group are in ascending order. But the performance is not satisfactory. Future work should figure out whether the performance degradation is due to the use of the **IfThen** constraints or removal of symmetries which also simultaneously removes solutions. Besides, despite better than using the **Reified** constraints, Constraints 5 and 6 introduce too many auxiliary variables; thus, we have also implemented our dedicated constraint to replace them. However, our constraint increases the difficulty of variable-selection since the constraint requires an additional variable to shift the equality relationship from row to row of the matrix C. We would also like to resolve this problem in the future work. To solve larger instances, in addition to using more processors and discovering more instance-specific constraints, we would like to investigate combining the dynamic symmetry breaking and parallel constraint solving for the SGP.

And finally, we must regretfully admit that even if we have made some progress, some interesting instances are are still open (e.g. 7-4-9, 8-3-11, 9-3-13, 10-10-11, and 12-12-13); notably, the original SGP 8-4-10 (Harvey, 2002) is still unsolved for constraint satisfaction approach, despite many efforts from the constraint programming community. Constraint technology should be amenable to solve these instances to demonstrate itself as the first choice for solving combinatorial problems.

[3]http://cpu.userbenchmark.com/, Accessed: August 2018

REFERENCES

Barnier, N. and Brisset, P. (2002). Solving the kirkmans schoolgirl problem in a few seconds. In *International Conference on Principles and Practice of Constraint Programming*, pages 477–491. Springer.

Beldiceanu, N., Carlsson, M., and Rampon, J.-X. (2012). Global constraint catalog, (revision a).

Benadé, J., Burger, A., and van Vuuren, J. (2013). 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*, pages 40–49.

Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004). Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 146–150.

Colbourn, C. J. (1984). The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30.

Cotta, C., Dotú, I., Fernández, A. J., and Van Hentenryck, P. (2006). Scheduling social golfers with memetic evolutionary programming. In *International Workshop on Hybrid Metaheuristics*, pages 150–161. Springer.

de Resmini, M. J. (1981). There exist at least three non-isomorphic s (2, 4, 28)'s. *Journal of Geometry*, 16(1):148–151.

Dotú, I. and Van Hentenryck, P. (2005). Scheduling social golfers locally. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 155–167. Springer.

Dotú, I. and Van Hentenryck, P. (2007). Scheduling social tournaments locally. *AI Communications*, 20(3):151–162.

Fahle, T., Schamberger, S., and Sellmann, M. (2001). Symmetry breaking. In *International Conference on Principles and Practice of Constraint Programming*, pages 93–107. Springer.

Focacci, F. and Milano, M. (2001). Global cut framework for removing symmetries. In *International Conference on Principles and Practice of Constraint Programming*, pages 77–92. Springer.

Harvey, W. (2002). CSPLib problem 010: Social golfers problem. http://www.csplib.org/Problems/prob010.

Harvey, W. and Winterer, T. (2005). Solving the molr and social golfers problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 286–300. Springer.

Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Morgan Kaufmann.

Law, Y. C. and Lee, J. H. (2004). Global constraints for integer and set value precedence. In *International Conference on Principles and Practice of Constraint Programming*, pages 362–376. Springer.

Miguel, I. (2012). CSPLib problem 038: Steel mill slab design. http://www.csplib.org/Problems/prob038.

Palmieri, A., Régin, J.-C., and Schaus, P. (2016). Parallel strategies selection. In *International Conference on Principles and Practice of Constraint Programming*, pages 388–404. Springer.

Perron, L. (2003). Fast restart policies and large neighborhood search. *Principles and Practice of Constraint Programming at CP*, 2833.

Prestwich, S. (2001). CSPLib problem 028: Balanced incomplete block designs. http://www.csplib.org/Problems/prob028.

Prud'homme, C., Fages, J.-G., and Lorca, X. (2017). *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S.

Puget, J.-F. (2005). Symmetry breaking revisited. *Constraints*, 10(1):23–46.

Rees, R. S. and Wallis, W. (2003). Kirkman triple systems and their generalizations: A survey. In *Designs 2002*, pages 317–368. Springer.

Régin, J.-C., Rezgui, M., and Malapert, A. (2013). Embarrassingly parallel search. In *International Conference on Principles and Practice of Constraint Programming. Lecture Notes in Computer Science*, volume 8124, pages 596–610. Springer, Berlin, Heidelberg.

Rossi, F., van Beek, P., and Walsh, T., editors (2006). *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier.

Sellmann, M. and Harvey, W. (2002). Heuristic constraint propagation–using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem. In *CPAIOR'02*. Citeseer.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

Smith, B. M. (2006). Modelling. In *Foundations of Artificial Intelligence*, volume 2, pages 377–406. Elsevier.

Smith, B. M. (April 2001). Reducing symmetry in a combinatorial design problem. In *CPAIOR'01*, pages 351–359. http://www.icparc.ic.ac.uk/cpAIOR01.

Triska, M. and Musliu, N. (2012a). An effective greedy heuristic for the social golfer problem. *Annals of Operations Research*, 194(1):413–425.

Triska, M. and Musliu, N. (2012b). An improved sat formulation for the social golfer problem. *Annals of Operations Research*, 194(1):427–438.