Procedia Computer Science

Volume 51, 2015, Pages 336–345

ICCS 2015 International Conference On Computational Science

Expressively Modeling the Social Golfer Problem in SAT

Frédéric Lardeux[1] and Eric Monfroy[2]

[1] Université d'Angers, Angers, France. Frederic.Lardeux@univ-angers.fr
[2] LINA - UMR 6241, TASC - INRIA Université de Nantes, France. Eric.Monfroy@univ-nantes.fr

Abstract
Constraint Satisfaction Problems allow one to expressively model problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances. We thus present a technique to expressively model set constraint problems and to encode them automatically into SAT instances. Our technique is expressive and less error-prone. We apply it to the Social Golfer Problem and to symmetry breaking of the problem.

Keywords: Set Constraints, SAT Encoding, Social Golfer Problem, Constraint Programming, CSP

1 Introduction

Most combinatorial problems can be formulated as Constraint Satisfaction Problems (CSP) [12]. A CSP is defined by some variables and constraints between these variables. Solving a CSP consists in finding assignments of the variables that satisfy the constraints. One of the main strength of CSP is expressiveness: variables can be of various types (finite domains, intervals, sets, ...) and constraints as well (linear, non linear, arithmetic, set, Boolean, ...). On the other hand, the propositional satisfiability problem (SAT) is restricted (in terms of expressiveness) to Boolean variables and propositional formulae. However, SAT solvers can now handle huge SAT instances (millions of variables and clauses). It is thus attractive to encode CSPs into SAT (e.g., [3, 5]) in order to benefit from the expressiveness of CSP and the power of SAT.

In this paper we are concerned with the encoding of CSP set constraints into SAT instances. In the constraint programming community, various systems for set constraints have been designed (e.g., as special libraries [11] or integrated in a solver [1]). They have shown that numerous problems can easily be modeled with sets.

Coding set constraints directly into SAT is a tedious tasks (e.g., [13] or [10]). Moreover, when one wants to optimize its model in terms of variables and clauses this quickly leads to very complicated and unreadable models in which errors can easily appear. Thus, our approach is based on an automated encoding of set constraints into SAT instances. To this end, we define some encoding rules ($\Leftrightarrow_{enc}$) that encode set constraints (such as union, membership, cardinality, ...) into the corresponding SAT clauses and variables. The advantage is that the modeling language (i.e., standard set constraints) is declarative, expressive, simple, and readable. We have tried this technique on various problems, and the SAT instances which are automatically

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.252

generated have a complexity similar to the complexity of improved direct SAT formulations, and their solving with a SAT solver (in our case Minisat) is efficient. We illustrate our approach with the Social Golfer Problemn (SGP): q golfers play every weeks during w weeks split in g groups of p golfers ($q = p.g$); how to schedule the play of these golfers such that no golfer plays in the same group as any other golfer more than once?

Works such as [3] and [5] make a relation between CSP and SAT solving (in terms of consistency properties) for finite domain constraints. Here, we are concerned with a different type of constraints. [13] (a revision and improvement of [10]) is a direct modeling of the SGP into SAT, whereas we are concerned with a higher-level modeling language which is automatically transformed into SAT instances. [13] also proposes various symmetry breaking techniques to improve the model; some of these symmetries naturally disappear with set constraints (e.g., no permutation due to numbering of players in groups). The remaining symmetries can easily be broken in our model by adding constraints or by refining the initial model. Note that we use the *cardinality* global constraint of [4] to encode set cardinality.

2 Set Constraint Encoding

We present the encoding of usual (CSP) set constraints (e.g., $\in$, $\cup$, $\cap$, ...) into SAT clauses.

Universe and Supports. Informally, the universe is the set of all elements that are considered in a model of a given problem while the support $\mathcal{F}$ of a set F appearing in this model is a set of possible elements of F (i.e., $\mathcal{F}$ is a superset of F).

Definition 1. *Let P be a problem, and M be a model of P in $\mathcal{L}$, i.e., a description of P from the natural language to the language of constraints $\mathcal{L}$.*

- *The universe $\mathcal{U}$ of M is a finite set of constants.*
- *The support of the set F of the model M is a subset of the universe $\mathcal{U}$; we denote it by $\mathcal{F}$. $\mathcal{F}$ represents the elements of $\mathcal{U}$ that can possibly be elements of F:*

$$F \subseteq \mathcal{F} \subseteq \mathcal{U} \quad and \quad F \in \mathcal{P}(\mathcal{F})$$

where $\mathcal{P}(\mathcal{F}) = \{A | A \subseteq \mathcal{F}\}$ is the power set of $\mathcal{F}$. We say that F is over $\mathcal{F}$.

Note that each element of $\mathcal{U} \setminus \mathcal{F}$ cannot be an element of F. We denote sets by upper-case letters (e.g., F) and their supports by calligraphic upper-case letters (e.g., $\mathcal{F}$). When there is no confusion, we shorten "the set F of the model M" to "the set F". Consider a model M with a universe $\mathcal{U}$, and a set F over $\mathcal{F}$. For each element x of $\mathcal{F}$, we consider a Boolean variable $x_{\mathcal{F}}$ which is true if $x \in F$ and false otherwise. We call the set of such variables the support variables for F in $\mathcal{F}$. In the following, we write $x_{\mathcal{F}}$ for $x_{\mathcal{F}} = true$ and $\neg x_{\mathcal{F}}$ for $x_{\mathcal{F}} = false$.

The $\Leftrightarrow_{enc}$ Encoding Rule. In the following, we consider three sets F, G, and H respectively defined on the supports $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ of the universe $\mathcal{U}$, and for each $x \in \mathcal{U}$ the various Boolean variables $x_{\mathcal{F}}$, $x_{\mathcal{G}}$, and $x_{\mathcal{H}}$ as defined before. $|G|$ denotes the cardinality of the set G.

Our encoding rules also consider cases for which the supports are not minimal: for example, for the constraint $F = G$, the sets $\mathcal{F} \setminus \mathcal{G}$ and $\mathcal{G} \setminus \mathcal{F}$ can be non empty whereas $F \setminus G$ and $G \setminus F$ must be empty. Allowing the supports to be non minimal makes easier the modeling process: one does not have to compute the minimal support. This is practical when sets are built from numerous sets and constraints. Note also that using the minimal supports reduces the size of the generated SAT instances.

The encoding rule is noted $\Leftrightarrow_{enc}$. The clauses that are generated by this rule are of the form $\forall x \in \mathcal{F},\ \phi(x_{\mathcal{F}})$ which denotes the $|\mathcal{F}|$ formulae $\phi(x_{\mathcal{F}})$ built for each element x of the support $\mathcal{F}$ of F (x refers to the element of the universe/support, and $x_{\mathcal{F}}$ to the variable representing x for the set F). For the membership constraint, the rule is not quantified; for multi-intersection and multi-union, an additional universal quantifier over i is used to denote a set of encoding rules, each rule being related to one of the sets $\mathcal{F}_i$.

Table 1 presents our rules for set constraint: first the constraint, then its encoding into SAT, and finally, the number of clauses generated. Note that some constraints like $x \notin F$ or $F \neq G$ are defined similarly as constraints defined in Table 1. Moreover, constraints can be linked by $\vee$, $\wedge$, and $\rightarrow$. In our implementation, the result of a union must be stored in a set: thus, $H = \bigcup_{i=1}^{n} F_i$ is equivalent to $H = F_1 \cup H_1$, $H_1 = F_2 \cup H_2$, ... but the multi-union constraint significantly reduces the number of SAT variables.

3 SAT Encoding for Set Constraint Model

Among the various SAT models for the SGP, the most classical is the direct encoding (DE) [13] (which was already a revision of [10]). [13] also proposes a variant of the DE using intermediate variables (we call it TME). We propose a model for the SGP using set constraints in a solver independent way. These constraints are then encoded into SAT using our $\Leftrightarrow_{enc}$ rules.

Set constraints model An instance of the problem is given by a triple $g-p-w$: p players per group, g groups per week, and w weeks. The universe is the set of players $\mathcal{P} = \{p_1, \ldots, p_q\}$ with $q = g.p$ being the total number of players. We need $w.g$ set variables to model the groups $G_{1,1}$, ..., $G_{w,g}$. The set $G_{i,j}$ is the group j of week i and is over the support $\mathcal{G}_{i,j} = \mathcal{P}$. Each $G_{i,j}$ will contain p players from $\mathcal{P}$. The supports are "minimal": they cannot be reduced without loosing (symmetric) solutions. We now give the constraints of the Social Golfer Problem.

- p players per group every weeks
$$\forall i \in [1..w], \forall j \in [1..g],\quad |G_{i,j}| = p \tag{1}$$
- Every golfer plays every weeks
$$\forall i \in [1..w] \quad \bigcup_{j=1..g} G_{i,j} = \mathcal{P} \tag{2}$$
- No golfer plays in two groups the same week
$$\forall i \in [1..w] \quad \bigcap_{j=1..g} G_{i,j} = \emptyset \tag{3}$$
Constraints (3) are not required since they are implied by Constraints (1) and (2).
- Two players cannot play twice together in the same group
$$\begin{aligned}\forall w_1, w_2 \in [1..w], p_i, p_j \in \mathcal{P}, g_1, g_2 \in [1..g], w_1 > w_2 \ \wedge\ i > j \ \wedge \\ p_i \in G_{w_1,g_1} \ \wedge\ p_j \in G_{w_1,g_1} \ \wedge\ p_i \in G_{w_2,g_2} \ \rightarrow\ p_j \notin G_{w_2,g_2}\end{aligned} \tag{4}$$
Another formulation of these constraints can be given using the cardinality constraints:
$$\forall w_1, w_2 \in [1..w], g_1, g_2 \in [1..g], w_1 > w_2 \ \wedge\ g_1 \geq g_2 \ \wedge |G_{w_1,g_1} \cap G_{w_2,g_2}| \leq 1 \tag{5}$$

SCE: Set Constraint Encoding From the above set model, our $\Leftrightarrow_{enc}$ encoding rules automatically generate SAT instances (see Section 2). The number of generated clauses is:

- Constraints (1) generates $w.g.w.(g.p + \sum_{i=1}^{g.p}[2u_i^{g.p}(\lfloor \frac{u_i^{g.p}}{2} \rfloor + 1)(\lceil \frac{u_i^{g.p}}{2} \rceil + 1) - (\frac{u_i^{g.p}}{2} + 1)])$ clauses with $u_{g.p}^{g.p} = 1$,$u_1^{g.p} = g.p$ and $u_i^{g.p} = u_{2i-1}^{g.p} + 2u_{2i}^{g.p} + u_{2i+1}^{g.p}$. The complexity of the formula generated by Constraints (1) is $\mathcal{O}(w^2.g^3.p^2)$.

Table 1: Set constraints encoding into SAT

$x \in F$	$\Leftrightarrow_{enc}$	$x \in \mathcal{F},\ x_{\mathcal{F}}$	1 unit clause
		$x \notin \mathcal{F},\ false$	1 empty clause
$F = G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{G}}$	$2.\lvert\mathcal{F} \cap \mathcal{G}\rvert$ binary clauses
		$\forall x \in \mathcal{F} \setminus \mathcal{G},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{G}\rvert$ unit clauses
		$\forall x \in \mathcal{G} \setminus \mathcal{F},\ \neg x_{\mathcal{G}}$	$\lvert\mathcal{G} \setminus \mathcal{F}\rvert$ unit clauses
$F \cap G = H$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ ternary clauses $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ binary clauses
		$\forall x \in (\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H},\ \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}}$	$\lvert(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}\rvert$ binary clauses
		$\forall x \in \mathcal{H} \setminus (\mathcal{F} \cap \mathcal{G}),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\mathcal{F} \cap \mathcal{G})\rvert$ unit clauses
$F \cup G = H$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \vee x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ ternary clauses $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ binary clauses
		$\forall x \in (\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}\rvert$ binary clauses
		$\forall x \in (\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F},\ x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F}\rvert$ binary clauses
		$\forall x \in \mathcal{H} \setminus (\mathcal{F} \cup \mathcal{G}),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\mathcal{F} \cup \mathcal{G})\rvert$ unit clauses
		$\forall x \in \mathcal{F} \setminus \mathcal{H},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{H}\rvert$ unit clauses
		$\forall x \in \mathcal{G} \setminus \mathcal{H},\ \neg x_{\mathcal{G}}$	$\lvert\mathcal{G} \setminus \mathcal{H}\rvert$ unit clauses
$F \subseteq G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}}$	$\lvert\mathcal{F} \cap \mathcal{G}\rvert$ binary clauses
		$\forall x \in \mathcal{F} \setminus \mathcal{G},\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus \mathcal{G}\rvert$ unit clauses
$H = F \setminus G$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge \neg x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}}$	$\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ ternary clauses $+ 2.\lvert\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}\rvert$ binary clauses
		$\forall x \in \mathcal{F} \setminus (\mathcal{G} \cup \mathcal{H}),\ \neg x_{\mathcal{F}}$	$\lvert\mathcal{F} \setminus (\mathcal{G} \cup \mathcal{H})\rvert$ ternary clauses
		$\forall x \in \mathcal{H} \setminus \mathcal{F},\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus \mathcal{F}\rvert$ unit clauses
		$\forall x \in (\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}\rvert$ binary clauses
		$\forall x \in (\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}}$	$\lvert(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}\rvert$ binary clauses
$H = \bigcup_{i=1}^{n} F_i$	$\Leftrightarrow_{enc}$	$\forall I, J \in \mathcal{P}(N), I \neq \emptyset, I \cup J = N,$ $\forall x \in \mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j),$ $\bigvee_{i \in I} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}}$	$\sum_{\substack{I,J \in \mathcal{P}(N),\\ I \neq \emptyset,\\ I \cup J = N}} \left(\lvert\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j)\rvert.(\lvert I\rvert + 1)\right)$ binary clauses and $\sum_{\substack{I,J \in \mathcal{P}(N),\\ I \neq \emptyset,\\ I \cup J = N}} \left(\lvert\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i) \setminus (\bigcup_{j \in J} \mathcal{F}_j)\rvert\right)$ $(\lvert I\rvert + 1)$-ary clauses
		$\forall x \in \mathcal{H} \setminus (\bigcup_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\bigcup_{i=1}^{n} \mathcal{F}_i)\rvert$ unit clauses
		$\forall i \in [1..n],\ \forall x \in \mathcal{F}_i \setminus \mathcal{H},\ \neg x_{\mathcal{F}_i}$	$\sum_{i=1}^{n} \lvert(\mathcal{F}_i \setminus \mathcal{H}\rvert$ unit clauses
$H = \bigcap_{i=1}^{n} F_i$	$\Leftrightarrow_{enc}$	$\forall x \in \mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i),\ \bigwedge_{i=1}^{n} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}}$	$2.\lvert\mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i)\rvert$ $(n+1)$-ary cl.
		$\forall x \in \bigcap_{i=1}^{n} \mathcal{F}_i \setminus \mathcal{H},\ \bigvee_{i=1}^{n} (\neg x_{\mathcal{F}_i})$	$\lvert\bigcap_{i=1}^{n} \mathcal{F}_i \setminus \mathcal{H}\rvert$ n-ary clauses
		$\forall x \in \mathcal{H} \setminus (\bigcap_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}}$	$\lvert\mathcal{H} \setminus (\bigcap_{i=1}^{n} \mathcal{F}_i)\rvert$ unit clauses
$\lvert G\rvert = k$	$\Leftrightarrow_{enc}$	[4]	$n + \sum_{i=1}^{n} 2u_i^n(\lfloor \frac{u_i^n}{2} \rfloor + 1)(\lceil \frac{u_i^n}{2} \rceil + 1) - (\frac{u_i^n}{2} + 1)$ *clauses and* $\sum_{i=1}^{n} u_i^n$ *variables with* $u_n^n = 1, u_1^n = n$ *and* $u_i^n = u_{2i-1}^n + 2u_{2i}^n + u_{2i+1}^n$.

- Constraints (2) generates $w.g.p$ clauses.
- Two formulations are possible: Constraints (4) generates $w.(w-1).g.(g+1).q.(q-1)/2)$ clauses ($\mathcal{O}(w^2.g^4.p^2)$) and Constraints (5) generates $w.((w-1)/2).g.((g+1)/2).3.q.(q+\sum_{i=1}^{q}[2u_i^q(\lfloor\frac{u_i^q}{2}\rfloor+1)(\lceil\frac{u_i^q}{2}\rceil+1)-(\frac{u_i^q}{2}+1)])$ clauses ($\mathcal{O}(w^2.g^5.p^3)$).

Complexity of the SAT instances Since the complexity of Constraints (4) is $\mathcal{O}(w^2.g^4.p^2)$ whereas the complexity of Constraints (5) is $\mathcal{O}(w^2.g^5.p^3)$, we will only focus on the implication formulation (Constraints (4)). The complexity of the SAT instances generated by the Set Constraint Encoding (SCE) model made from Constraints (1), (2), and (4) is $\mathcal{O}(w^2.g^4.p^2)$.

Post-treatment by Unit Propagation Unit propagation is a simple process to eliminate unit clauses (clauses with only false literals and one free literal) by giving the *true* value to the free literals. This valuation can produce new unit clauses; the process is applied until there is no longer any unit clause. Algorithms for unit propagation may significantly reduce: 1) instance sizes, 2) number of variables and 3) solving time. Note that the cardinality constraint encoding that we have chosen generates numerous unit clauses that vanish using unit propagation.

4 Symmetry Breaking for the SGP

Removing symmetric solutions eases the work of a (SAT) solver. The Social Golfer Problem is highly symmetric: the position of a player in a group is not relevant; the groups in a week can be renumbered; the weeks can be swapped. Symmetry breaking thus consists in eliminating symmetries by adding new constraints or modifying the model. [10] proposes some clauses to remove symmetries among players, to order groups within a week to order lexicographically the weeks, ... However, these clauses become more and more complicated and mistakes can easily be introduced. Indeed, [13] revised the clauses for symmetry breaking of [10] in order to correct the ranges of the various $\bigvee$ and $\bigwedge$. We call this encoding TME$^{\text{SB}}$. More symmetries can be broken, such as in [9]. All symmetries can be broken [6], but this is often at the cost of a super exponential number of constraints. Thus, this cannot be considered in practice.

With our set language, we have two possibilities to break symmetries: by adding extra constraints to the initial model or by refining the model itself by modifying the supports of sets and the constraints. Since our model is different from the one of [10] and [13], we do not obtain the same symmetries. However, we try to break similar symmetries.The first group of symmetry breaking ($SB1$) consists in filling the first week as follows: the first p players are sent to the first group of the first week; the next p players, on the second group of the first week; and so on. We consider a second group $SB2$ of symmetry breaking to complete $SB1$: the first p players are spread in different groups each week. When p is greater than g we can rather obviously see that the problem has no solution. In the following, we thus consider $g \geq p$.

Symmetry breaking for the set constraint model by adding constraints. For $SB1$, we only have to add the following simple constraints to the model of the SCE.

$$\forall i \in [1..p.g], p_i \in G_{1,i\ div\ (p+1)} \tag{6}$$

For the second group $SB2$ of symmetry breaking, the required constraints are also simple:

$$\forall i \in [2..w], \forall j \in [1..p], p_j \in G_{i,j} \tag{7}$$

These constraints add clauses to the SAT instances: however, these extra constraints are unit clauses that will produce unit propagation and thus will vanish. The SAT encoding of the

set model with symmetry breaking by adding constraints to the model is named SCE$^{\text{SBC}}$ and consists in Constraints (1), (2), (4), (6), and (7).

Symmetry breaking for the set constraint model by modifying the model. Although modifying the model is more tedious, this reduces the supports of sets and cardinality constraints, and consequently the generated SAT instances. The only modification for $SB1$ consists in both modifying the supports of the groups of the first week and to fix these groups:

$$\forall i \in [1..g], \mathcal{G}_{1,i} = \{p_{1+(i-1).g}, \dots p_{p+(i-1).g}\} \text{ and } \forall i \in [1..g], G_{1,i} = \mathcal{G}_{1,i} \tag{8}$$

The other sets, variables, and constraints remain unchanged. To introduce $SB2$, we change the group variables. Instead of the $G_{i,j}$, we now consider the sets $G'_{1,1}, \dots, G'_{w,g}$ such that:

- for the first week $G_{i,j} = G'_{i,j}$;
- for the following weeks $G_{i,j} = G'_{i,j} \cup \{p_j\}$ if $j \leq p$, $G_{i,j} = G'_{i,j}$ otherwise.

The support of the $G'_{1,i}$ (i.e., the groups of the first week) are defined as with $SB1$. Since the p first players are spread on the p first groups of each week, the supports of the other groups can be reduced to $\mathcal{P}' = \{p_{p+1}, \dots, p_q\}$, and $\forall i \in [2..w], \forall j \in [1..g], \mathcal{G}_{i,j} = \mathcal{P}'$

P players per group every weeks. Constraints (1) must be replaced by Constraints (9)–(11).

$$\forall i \in [1..g], \quad |G'_{1,i}| = p \tag{9}$$

$$\forall j \in [2..w], \forall i \in [1..p], \quad |G'_{j,i}| = p - 1 \tag{10}$$

$$\forall j \in [2..w], \forall i \in [p+1..g], \quad |G'_{j,i}| = p \tag{11}$$

Every golfer plays every week. Constraints (12) replace Constraints (2).

$$\forall j \in [2..w] \bigcup_{i=1..g} G_{j,i} = \mathcal{P}' \tag{12}$$

Two players cannot play twice together in the same group. Constraints (4) are replaced by Constraints (13)–(16). Since 2 groups $G_{i,j}$ with $j \leq p$ and $i > 1$ have player p_j in common, the corresponding groups $G'_{i,j}$ (which supports do not contain the p_l, $l \leq p$) cannot have any other player p_k in common:

$$\forall w_1, w_2 \in [2..w], \ p_i \in \mathcal{P}, \ g_1 \in [1..p], \ w_1 > w_2, p_i \in G'_{w_1,g_1} \ \rightarrow \ p_i \notin G'_{w_2,g_1} \tag{13}$$

The relation between other two groups is not changed.
Between a group of the first week (except the first group) and groups of other weeks:

$$\begin{aligned} &\forall w_1 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [2..g], g_2 \in [1..g], \ i > j, \\ &p_i \in G'_{1,g_1} \ \wedge \ p_j \in G'_{1,g_1} \ \wedge \ p_i \in G'_{w_1,g_2} \ \rightarrow \ p_j \notin G'_{w_1,g_2} \end{aligned} \tag{14}$$

Without considering $p \leq g$, then g_1 must be considered in $[2..g]$ to generate the proper constraints (that will generate a failure during the resolution of the SAT instance).
Between two groups (except of the first week) equally numbered with an index greater than p:

$$\begin{aligned} &\forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [p+1..g], \ w_1 > w_2, \ i > j, \\ &p_i \in G'_{w_1,g_1} \wedge p_j \in G'_{w_1,g_1} \wedge p_i \in G'_{w_2,g_1} \ \rightarrow \ p_j \notin G'_{w_2,g_1} \end{aligned} \tag{15}$$

Between two groups (except of the first week) not equally numbered:

$$\begin{aligned} &\forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1, g_2 \in [1..g], \ w_1 > w_2, \ g_1 \neq g_2, \ i > j, \\ &p_i \in G'_{w_1,g_1} \ \wedge \ p_j \in G'_{w_1,g_1} \ \wedge \ p_i \in G'_{w_2,g_2} \ \rightarrow \ p_j \notin G'_{w_2,g_2} \end{aligned} \tag{16}$$

The SAT encoding of the set model with symmetry breaking by modifying the model is named SCE$^{\text{SBM}}$ and consists in Constraints (8)–(16).

Table 2: List of the encoding names

Name	Description	Constraints
DE	Direct Encoding	[13]
TME	Triska-Musliu encoding	[13]
TME^{SB}	TME with symmetry breaking	[13]
SCE	SAT encoding of the set constraint model	(1), (2), (4)
SCE^{SBC}	SCE with symmetry breaking by adding constraints	(1), (2), (4),(6), (7)
SCE^{SBM}	SCE with symmetry breaking by modifying the mode	(8)–(16)
$NAME_{UP}$	encoding after unit propagation treatment	

5 Comparisons of Models

Table 2 summarizes the encodings (described in the previous sections) that we compare in the following sections. $NAME_{UP}$ denotes the encoding NAME after unit propagation.

Expressiveness. The variables of our set model are much simpler: we need 2 indices instead of 4, making them more readable. The second difference is the simplicity and readability of constraints. Indeed, set constraints are more expressive than pure SAT clauses. Then, the encoding into SAT is performed by the $\Leftrightarrow_{enc}$ encoding rules. The advantage is double: 1) constraints are readable, expressive, easy to modify, resulting in a much understandable model; 2) less mistakes are introduced since the modeling process is much simpler. Moreover, the set encoding is solver independent. Indeed, the same model (changing the syntax) can be used in a set constraint CSP solver or in a SAT solver after the $\Leftrightarrow_{enc}$ encoding proposed above.

Adding symmetry breaking in the set model can be done by adding constraints/clauses or by modifying the model itself. The process is a bit more complicated than just adding constraints, but the result is worth: instances are smaller and solving time is faster.

In terms of expressiveness, readability, error introduction, and solver dependence, our set model is superior to direct encodings such as DE or TME. Breaking symmetries is also easier.

Model Structure We generated a set of SGP instances with: the direct encoding DE, the Triska-Musliu encoding [13] (TME), and our set constraint encoding with unit propagation post-treatment (SCE_{UP}) and without (SCE). In Table 3, each instance is defined by the triple (groups, players per group, weeks) and for each encoding the numbers of variables and clauses are given. For each instance, encodings generating the smallest number of clauses and variables are in bold. Instance size is not the only comparison criterion. Nevertheless, big instances are intractable due to the limited computer memory. It is thus necessary to generate as small as possible instances. Direct encoding (DE) is clearly unsuitable when the number of players or groups increases; With the introduction of auxiliary variables the number of clauses is less important for TME but the number of variables increases. SCE produces more variables but less clauses. As might be expected, SCE_{UP} provides the most interesting encoding in terms of number of clauses and number of variables: indeed, SCE generates a lot of unit clauses and binary clauses (Section 3) that vanish using unit propagation.

Impact of the symmetry breaking We applied the two symmetry breaking processes of Section 4 and results are presented in Table 3. For TME, introducing symmetry breaking only increases the number of clauses (around 10% more clauses), the number of variables does not change. Note also that unit propagation is not worth for TME instances nor for TME^{SB} instances: there is no unit clause. For SCE, symmetry breaking by adding constraints adds a negligible amount of constraints (see SCE^{SBC}). More interestingly, adding symmetry breaking by modifying the model (SCE^{SBM}) significantly reduces the size of the generated SAT instances: from 20 up to 60% less variables and from 40 to 60% less clauses. This significant reduction is

Table 3: Size of instances generated using different encodings.

Prob.	DE var	DE cl. $\times 10^6$	TME var	TME cl. $\times 10^3$	TME_*^{SB} var	TME_*^{SB} cl. $\times 10^3$	SCE var	SCE cl. $\times 10^3$	SCE^{SBM} var	SCE^{SBM} cl. $\times 10^3$	SCE^{SBC} var	SCE^{SBC} cl. $\times 10^3$	SCE_{UP} var	SCE_{UP} cl. $\times 10^3$	SCE_{UP}^{SBM} var	SCE_{UP}^{SBM} cl. $\times 10^3$	SCE_{UP}^{SBC} var	SCE_{UP}^{SBC} cl. $\times 10^3$
5-3-6	1 350	3	1 800	60	1 800	71	8 625	50	5 702	21	8 625	50	1 410	44	**860**	**18**	980	23
5-3-7	1 575	4	2 100	79	2 100	92	11 110	68	7 734	30	11 110	68	1 645	60	**1 032**	**26**	1 176	34
8-4-4	4 096	48	5 120	323	5 120	390	24 224	235	14 192	96	24 224	235	3 840	205	**2 376**	**78**	2 580	92
8-4-5	5 120	81	6 400	483	6 400	567	34 752	373	22 476	173	34 752	373	4 800	335	**3 168**	**149**	3 440	176
8-4-6	6 144	121	7 680	675	7 680	776	47 072	543	32 552	273	47 072	543	5 760	498	**3 960**	**243**	4 300	288
8-4-7	7 168	170	8 960	898	8 960	1 016	61 184	744	44 420	396	61 184	744	6 720	692	**4 752**	**361**	5 160	427
8-4-8	8 192	227	10 240	1 153	10 240	1 288	77 088	978	58 080	542	77 088	978	7 680	918	**5 544**	**500**	6 020	593
8-4-9	9 216	292	11 520	1 441	11 520	1 592	94 784	1 243	73 532	711	94 784	1 243	8 640	1 175	**6 336**	**663**	6 880	786
8-4-10	10 240	365	12 800	1 759	12 800	1 928	114 272	1 540	90 776	902	114 272	1 540	9 600	1 465	**7 128**	**848**	7 740	1 006
9-4-6	7 776	196	9 720	1 048	9 720	1 191	117 324	858	46 344	448	117 324	858	7 344	793	**5 620**	**472**	5 620	472
9-4-7	9 072	274	11 340	1 401	11 340	1 568	157 284	1 180	63 368	652	157 284	1 180	8 568	1 104	**6 008**	**562**	6 744	701
9-4-8	10 368	366	12 960	1 805	12 960	1 996	203 076	1 553	82 984	895	203 076	1 553	9 792	1 465	**7 024**	**783**	7 868	975
9-4-9	11 664	470	14 580	2 261	14 580	2 261	254 700	1 976	105 192	1 176	254 700	1 977	11 016	1 878	**8 040**	**1 040**	8 992	1 294
9-4-10	12 960	588	16 200	2 767	16 200	3 006	312 156	2 451	129 992	1 496	312 156	2 451	12 240	2 342	**9 056**	**1 334**	10 116	1 658

due to the reduction of supports and to the cardinality constraints.

Without unit propagation, the instances of SCE^{SBM} are always the smallest ones w.r.t. the number of clauses. Unit propagation has no impact at all on TME. However, its impact is significant on SCE, SCE^{SBM}, and SCE^{SBC}. For SCE, unit propagation divides the number of variables by 6 to 25: this is mainly due to the variables of the cardinality constraints. The number of clauses is reduced of around 10%. For SCE^{SBC}, unit propagation reduces even more the number of variables (up to 30 times less variables). The number of clauses is reduced from 30 to 60%. For SCE^{SBM}, unit propagation is less spectacular: indeed, the initial model itself is reduced. However, the number of variables is divided by 5 up to 15. The number of clauses is reduced of about 10%.

To summarize, unit propagation is more beneficial to SCE^{SBC}; however, SCE_{UP}^{SBM} always gives the best instances in terms of number of clauses and number of variables.

6 Experimental Analysis

We now compare the efficiency of the encodings in terms of solving time, using the well known solver Minisat [8]. SatELite [7] is a pre-treatment in Minisat to drastically reduce the number of clauses and variables. This pre-treatment has a cost but it generally improves the global running time. It can also be deactivated. Table 4 represents respectively the running time of Minisat with and without SatElite as pre-treatment.

Experimentations are realized on a 2.60GHz Intel Core i5-2540M CPU and 4 GB RAM. For each experiment, the time-out is 300 seconds ("-" if time-out is reached). The direct encoding DE is not presented since, as supposed, no results are obtained in a reasonable time. Table 4 shows that the use of SatElite is difficult to predict: depending on the instance, it significantly improves or degrades the results. On average, it does not improve the results and the best running times are obtained without pre-treatment. Symmetry breaking modifying the model (SCE^{SBM}) provides the best (or very close to the best) results, with or without pre-treatment. The use of unit propagation has a weak impact to the resolution time of SCE^{SBM}. Adding constraints to break symmetries (SCE^{SBC}) does not produce any improvement except when unit propagation is applied (SCE_{UP}^{SBC}). Indeed, SCE_{UP}^{SBC} obtains results as good as SCE^{SBM}. Breaking symmetries in TME is rather fluctuating: depending on the instances and depending on the use of SatELite, it significantly improves or degrades the results.

To summarize, the best results are obtained with our set constraint model, with SCE_{UP}^{SBC} when the pre-treatment is applied, or predominantly with SCE_{UP}^{SBM} when the pre-treatment is

Table 4: Minisat Solving Time

Prob.	TME	TME^{SB}	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}	SCE^{SBM}_{UP}	SCE^{SBC}_{UP}	TME	TME^{SB}	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}	SCE^{SBM}_{UP}	SCE^{SBC}_{UP}
	with SatElite								**without** SatElite							
5-3-6	8.92	0.69	0.18	0.06	0.12	0.12	0.07	**0.04**	9.37	0.30	1.05	**0.01**	**0.01**	0.26	**0.01**	**0.01**
5-3-7	98.28	13.37	1.42	0.13	1.21	5.09	0.09	**0.08**	97.47	24.86	9.19	**0.06**	0.13	5.67	1.79	0.28
8-4-4	1.04	1.33	0.97	0.32	1.19	0.90	0.29	**0.27**	0.05	0.23	0.09	**0.03**	0.07	0.07	**0.03**	**0.03**
8-4-5	2.26	2.64	1.93	0.86	2.51	1.89	0.84	**0.78**	0.08	0.58	0.13	0.06	0.11	0.06	**0.05**	0.07
8-4-6	4.44	5.16	3.65	1.87	4.74	3.65	1.82	**1.71**	0.25	3.58	0.27	0.14	0.18	0.19	**0.08**	0.09
8-4-7	34.25	94.68	8.66	3.59	8.52	7.52	3.64	**3.46**	27.05	25.88	3.53	**0.48**	1.71	1.94	0.56	0.98
8-4-8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8-4-9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8-4-10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9-4-6	8.45	10.52	11.24	3.15	10.34	11.10	**2.71**	4.58	0.23	3.72	0.37	0.13	0.29	0.25	**0.11**	0.13
9-4-7	13.69	27.16	18.95	5.80	17.8	19.04	**5.12**	8.76	0.31	6.61	0.58	0.22	0.51	0.36	**0.14**	0.24
9-4-8	-	-	31.87	**11.10**	29.60	31.48	12.72	14.90	247.83	-	14.66	5.03	1.10	20.93	2.62	**0.68**
9-4-9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9-4-10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

not applied. Finally, the best results are obtained without pre-treatment.

7 Discussion and Conclusion

Modeling. Modeling a problem with set constraints and then automatically generating the corresponding SAT instances is much simpler than writing direct encodings. Breaking symmetries is rather tedious in direct encodings, very easy by adding constraints in the set model, and rather easy by modifying the set model. Using a higher level language is thus beneficial to the modeling phase: it simplifies the task, and avoid making errors (mainly in the numerous indices required by a direct encoding). The SAT encoding is then automatically done.

SAT Instances The SAT instances that are automatically produced by our encoding are of good quality: 1) they always produce significantly less clauses (with or without symmetry breaking, and with or without unit propagation); 2) with unit propagation, they also generate less variables; and 3) they are solved faster with Minisat, without "tuning parameters", with or without pre-treatment with SatElite.

Symmetry breaking Breaking symmetries by adding constraint to the set model is very simple. Moreover, the generated SAT instances after unit-propagation are much smaller, and the solving time is also improved. Symmetry breaking by modifying the model is even more beneficial, however, the effort is more important. This extra work is very beneficial for the size of the generated SAT instances, but not so much worth for the solving time (it depends on instances, and pre-treatment). Thus, one has to make the trade-off between solving time and modeling time. Instance size can be the deciding factor: larger problems can be modeled and generated introducing symmetry breaking into the model as in SCE^{SBM}.

Set constraints In terms of sets, the expressiveness of systems such as [1]) is more or less the same as the one we propose: that was our goal. The advantage of [11] or [1] is that sets can be mixed with other constraints. In the future, we also plan to encode other types of constraints. In systems such as [11] or [1], a special solver has to be designed (e.g., a domain reduction phase interleaved with search). [2] compares set constraint solvers for the SGP: most of the results are obtained with special (dynamic) search heuristics orl solving mechanisms. Our approach is very different: we do not want to design a special solver, nor to tune an existing one for efficiently solving our SAT instances; we want to transform a high level set model into a "good" quality SAT instance that is efficiently solved by an existing multi-purpose SAT solver. In terms of

computing time, the 5-3-6 instance of the same set model of the SGP is not solved before the time-out (300s.) with the standard solver of [1] whereas it takes less that 1s. in Minisat.

Conclusion We have presented a technique for encoding set constraints into SAT: the modeling process is done with some expressive set constraints which are then automatically encoded into SAT instances using our $\Leftrightarrow_{enc}$ rules. This technique has been applied successfully to the Social Golfer Problem, and to study some symmetry breaking on this problem. The advantages of our technique are the following: 1) the modeling process is expressive, declarative, and readable. Moreover, it is solver independent and independent from CSP or SAT; 2) the technique is less error-prone than direct SAT encodings; 3) symmetries can be broken by just adding new constraints or by refining the model; 4) the SAT instances which are automatically generated are smaller than the ones of [13]; with unit propagation, our instances also contain less variables than the ones of [13]; 5) finally, with respect to solving time, our automatically generated instances of the Social Golfer Problem are solved faster with or without unit propagation, with or without constraint breaking, with or without the SatElite pre-treatment. We have also modeled and solved other problems (such as car-sequencing). We obtained very readable and simple set models. The generated SAT instances also appeared to be well-suited for Minisat.

We plan to refine the notion of supports and integrate a pre-process to reduce them. This does not have any impact on the SGP, but for many problems (in which supports are not clear at the principle), it is important to reduce supports before generating SAT instances.

References

[1] Minizinc. http://www.minizinc.org/.

[2] F. Azevedo. An attempt to dynamically break symmetries in the social golfers problem. In *CSCLP*, pages 33–47, 2006.

[3] F. Bacchus. Gac via unit propagation. In *Proc. of CP 2007*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.

[4] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Proc. of CP 2003*, volume 2833, pages 108–122. Springer, 2003.

[5] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in sat. In *Selected Revised Papers of SAT 2003.*, volume 2919 of *LNCS*, pages 299–314. Springer, 2004.

[6] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR'96*, pages 148–159. Morgan Kaufmann, 1996.

[7] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT 2005*, volume 3569, pages 61–75, 2005.

[8] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT 2003*, volume 2919, pages 502–518, 2003.

[9] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP 2002*, volume 2470, pages 462–476. Springer, 2002.

[10] I. Gent and I. Lynce. A sat encoding for the social golfer problem. In *IJCAI05 workshop on modelling and solving problems with constraints*, 2005.

[11] C. Gervet. Conjunto: Constraint propagation over set constraints with finite set domain variables. In *Proc. of ICLP'94*, page 733. MIT Press, 1994.

[12] F. Rossi, T. P. van Beek, and Walsh, editors. *Handbook of Constraint Programming.* Elsevier, 2006.

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