Ann Oper Res (2015) 235:423–452
DOI 10.1007/s10479-015-1914-5

Set constraint model and automated encoding into SAT: application to the social golfer problem

Frédéric Lardeux[1] · **Eric Monfroy**[2] · **Broderick Crawford**[3,4,5] · **Ricardo Soto**[3,6,7]

Published online: 9 June 2015

Abstract On the one hand, 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. We apply our technique to the social golfer problem and we also use it to break symmetries of the problem. Our technique is simpler, more expressive, and less error-prone than direct modeling. The SAT instances that we automatically generate contain less clauses than improved direct instances such as in Triska and Musliu (Ann Oper Res 194(1):427–438, 2012), and with unit propagation they also contain less variables. Moreover, they are well-suited for SAT solvers and they are solved faster as shown when solving difficult instances of the social golfer problem.

Keywords Constraint programming · CSP · Set constraints · SAT encoding · Social golfer problem

✉ Frédéric Lardeux
Frederic.Lardeux@univ-angers.fr

Eric Monfroy
Eric.Monfroy@univ-nantes.fr

Broderick Crawford
broderick.crawford@ucv.cl

Ricardo Soto
ricardo.soto@ucv.cl

1 LERIA, Université d'Angers, Angers, France

2 LINA, UMR CNRS 6241, TASC INRIA, Université de Nantes, Nantes, France

3 Pontificia Universidad Católica de Valparaiso, 2362807 Valparaiso, Chile

4 Universidad San Sebastián, Valdivia, Chile

5 Universidad Central de Chile, Santiago, Chile

6 Universidad Autónoma de Chile, Temuco, Chile

7 Universidad Cientifica del Sur, Lima, Perú

1 Introduction

Most of combinatorial problems can be formulated as constraint satisfaction problems (CSP) Rossi et al. (2006). A CSP is defined by some variables (generally over finite domains) 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, floating point numbers, intervals, sets, …) and constraints as well (linear arithmetic constraints, set constraints, non linear constraints, Boolean constraints, symbolic constraints, …). Moreover, the so-called global constraints not only improve solving efficiency but also expressiveness: they propose new constructs and relations such as *alldifferent* (to enforce that all the variables of a list have different values), *cumulative* (to schedule tasks sharing resources), …

On the other hand, the propositional satisfiability problem (SAT) Garey and Johnson (1979) is restricted (in terms of expressiveness) to Boolean variables and propositional formulae. However, SAT solvers can now handle huge SAT instances (millions of variables). It is thus attractive to (1) encode CSPs into SAT (e.g., Bacchus 2007; Bessière et al. 2004) in order to benefit from the expressiveness of CSP and the power of SAT, or (2) introduce more expressiveness into SAT, for example with global constraints (e.g., alldifferent Lardeux et al. 2009; cardinality Bailleux and Boufkhad 2003).

In this paper we are concerned with the transformation of set constraints into SAT instances: we often refer to this transformation as "encoding". Various systems of set constraints (either specialized systems Legeard and Legros 1991, libraries for constraint programming systems such as Gervet 1994, the set constraint library of CHOCO http://www.emn.fr/z-info/choco-solver/, or constraint systems and modelers such as MiniZinc (http://www.minizinc.org/) have been designed for solving problems such as prototyping combinatorial problems, axiomatization of set theory, analysis of programs,…They have shown that some problems can easily be modeled with set constraints.

Coding set constraints directly into SAT is a tedious tasks (see for example Triska and Musliu 2012 or Gent and Lynce 2005). 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 intersection, union, membership, cardinal of sets) into the corresponding SAT clauses and variables. The advantage is that the modeling language (i.e., standard set constraints) is expressive, simple, and readable. We have tried this technique on various problems, and the SAT instances which are automatically 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 problem (problem number 10 of the CSPLib Gent and Walsh 1999). The problem is the following: 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. An instance of the problem is then given by a triple $g - p - w$. Various instances of the social golfer problem are still open, and the problem is attractive since it is related to problems such as encryption and covering problems. Compared to direct encodings (such as the one of Triska and Musliu 2012), the instances we generate are smaller (less clauses), and also contain less variables using unit propagation. The introduction of symmetry breaking is simplified with our technique and can be done by adding constraints to the initial model or by refining the initial model. Using Minisat Eén and Sörensson (2003), our automatically generated instances (with or without symmetry breaking) are solved faster than the ones of Triska and Musliu (2012).

We can compare our work with works of different types, first of all with SAT encoding techniques such as Bacchus (2007) and Bessière et al. (2004). These works make a relation between CSP solving and SAT solving in terms of properties such as consistencies for finite domain variables and constraints. In this article, we focus on expressiveness for SAT and on a different type of constraints, i.e., set constraints.

Concerning applications, i.e., the social golfer problem, the closest work is Triska and Musliu (2012) which is a revision and improvement of Gent and Lynce (2005). Whereas these works are direct modeling of the social golfer problem directly in SAT, we are concerned with a higher-level model language which is automatically transformed into SAT instances. Triska and Musliu (2012) also proposes various symmetry breaking techniques to improve the model; some of these symmetries naturally disappear using our set constraint model (for example, we do not have any permutations due to the numbering of players inside a group). Other symmetry breakings can easily be introduced in our model, by adding constraints or by refining the initial model.

In Cotta et al. (2006), the social golfer problem is modeled with a combination of set constraints and arithmetic constraints. However, this model is not directly used but it is transformed into CSP before being solved by mimetic algorithms.

Finally, our approach is similar to Lardeux et al. (2009) in which alldifferent global constraints and overlapping alldifferent constraints are handled expressively before being encoded automatically in SAT using rewrite rules.

Note also that we use the work of Bailleux and Boufkhad (2003) about the *cardinality* global constraint in order to perform the encoding of set cardinality.

In the next section (Sect. 2), we present our set constraint language and the rule-based system for encoding set constraints into SAT; we consider standard set constraints. To get a comparison basis, we then (Sect. 3) give a direct SAT model of the social golfer problem, and some variants of this model. We then present how to model the social golfer problem with set constraints, and show the interest of our system in terms of expressiveness. In Sect. 4, we show how to introduce symmetry breaking techniques (that can be found in the literature) with our set constraint language: by adding new constraints or by refining the initial model. In Sect. 5, we compare various SAT instances, either direct or automatically generated with our encoding rule: this analysis is made with respect to instance structures (e.g., number of variables and clauses). In the next section, we compare the solving time of these instances. Section 7, discusses various points related to our technique: structure of instances, usefullness of unit propagation, difference with work about set constraints in constraint programming, …We finally conclude in Sect. 8.

2 Set constraint encoding

We present here the encoding of usual (CSP) set constraints (such as $\in$, $\cup$, $\cap$, …) into SAT clauses. More constraints could be defined, but they can be deduced from these basic constraints.

2.1 Universe and supports

We consider two notions: *universe* and *support*. Unformally, 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 $\mathcal{U} \backslash \mathcal{F}$ cannot be element of F. In the following, we denote sets by uppercase letters (e.g., F) and their supports by calligraphic uppercase letters (e.g., $\mathcal{F}$). When there is no confusion of model, 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 $\mathcal{F}$.

Example 1 Let $\mathcal{U} = \{x, y, z, t\}$ be the universe of a model M, and $\mathcal{F} = \{x, y, t\}$ be the support of a set F of M. Then, we have 3 Boolean variables $x_{\mathcal{F}}$, $y_{\mathcal{F}}$, and $t_{\mathcal{F}}$ corresponding respectively to x, y, and z to represent F. By definition, $z \notin F$ and there is no $z_{\mathcal{F}}$ variable; and x, y, t can possibly be in F. Consider now that $F = \{x, y\}$. Then, $x_{\mathcal{F}} = true$, $y_{\mathcal{F}} = true$, and $t_{\mathcal{F}} = false$.

In the following, we write $x_{\mathcal{F}}$ for $x_{\mathcal{F}} = true$ and $\neg x_{\mathcal{F}}$ for $x_{\mathcal{F}} = false$.

2.2 The $\Leftrightarrow_{enc}$ encoding rule

We can now define the encoding of the various CSP set constraints into SAT. 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.

Note that we do not force the supports to be minimal: for example, for the equality constraint $F = G$, the sets $\mathcal{F} \backslash \mathcal{G}$ and $\mathcal{G} \backslash \mathcal{F}$ can be non empty whereas $F \backslash G$ and $G \backslash F$ must be empty. We thus consider these cases in the $\Leftrightarrow_{enc}$ encoding rule. Allowing the supports to be non minimal eases the modeling process: indeed, one does not have to compute the minimal support and can use a superset of it or the universe. This is practical when sets are built from many other sets using numerous set constraints. Note also that using smaller 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 $\mathcal{F}_i$.

In the following, we propose several set constraint encodings with: first the set constraint, then its encoding in SAT, and finally, the number of generated clauses.

2.3 Membership constraint

This constraint enforces the membership of an element x to a set F:

– if $x \in \mathcal{F}$ (x is in the support of F), then the corresponding support variable must be true, i.e., $x_{\mathcal{F}}$.
– if $x \notin \mathcal{F}$ (x is not in the support of F), then the constraint $x \in F$ must generate a failure since the problem does not have any solution.

$$x \in F \Leftrightarrow_{enc} \begin{cases} x \in \mathcal{F},\ x_{\mathcal{F}} & \text{1 unit clause} \\ x \notin \mathcal{F},\ false & \text{1 empty clause} \end{cases}$$

The constraint $x \notin F$ can be similarly defined:

$$x \notin F \Leftrightarrow_{enc} \begin{cases} x \in \mathcal{F},\ \neg x_{\mathcal{F}} & \text{1 unit clause} \\ x \notin \mathcal{F},\ true & \text{1 empty clause} \end{cases}$$

2.4 Set equality constraint

Two sets G and F are equal if and only if:

– for the elements of $\mathcal{F} \cap \mathcal{G}$: the support variables of G have the same values as the support variables of F;
– for the elements of $\mathcal{F} \backslash \mathcal{G}$: the support variables of F must be false. Indeed, an element of the universe which is not in the support of a set is not part of this set; thus, an element of $\mathcal{F} \backslash \mathcal{G}$ cannot be in F.
– for the elements of $\mathcal{G} \backslash \mathcal{F}$: the support variables of G must be false.

$$F = G \Leftrightarrow_{enc} \begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{G}} & 2.|\mathcal{F} \cap \mathcal{G}| \text{ binary clauses} \\ \forall x \in \mathcal{F} \backslash \mathcal{G},\ \neg x_{\mathcal{F}} & |\mathcal{F} \backslash \mathcal{G}| \text{ unit clauses} \\ \forall x \in \mathcal{G} \backslash \mathcal{F},\ \neg x_{\mathcal{G}} & |\mathcal{G} \backslash \mathcal{F}| \text{ unit clauses} \end{cases}$$

The constraint $F \neq G$ can be similarly defined by considering the negation of the conjunction of formulae of the previous encoding.

2.5 Intersection constraint

Let H be the intersection of two sets G and F:

– for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: a support variable of H is true if and only if this variable is in F and G;
– for the elements of $(\mathcal{F} \cap \mathcal{G}) \backslash \mathcal{H}$: since such an element cannot be in H, it must not be in F and G;
– for the elements of $\mathcal{H} \backslash (\mathcal{F} \cap \mathcal{G})$: a support variable of H which is not in the support of F and G cannot be true

$$F \cap G = H$$
$$\Leftrightarrow_{enc}$$
$$\begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & \begin{array}{l} |\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses} \end{array} \\ \forall x \in (\mathcal{F} \cap \mathcal{G}) \backslash \mathcal{H},\ \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}} & |(\mathcal{F} \cap \mathcal{G}) \backslash \mathcal{H}| \text{ binary clauses} \\ \forall x \in \mathcal{H} \backslash (\mathcal{F} \cap \mathcal{G}),\ \neg x_{\mathcal{H}} & |\mathcal{H} \backslash (\mathcal{F} \cap \mathcal{G})| \text{ unit clauses} \end{cases}$$

Note that if $H = \emptyset$ (e.g., we want to force the intersection to be empty), then the encoding can be simplified into $\forall x \in U, \neg x_F \vee \neg x_G$, and thus, reduce its size to $|U|$ clauses.

2.6 Union constraint

More cases are to be considered for this constraints:

- for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: a support variable of H is true if and only if this variable is in F or in G; this is the trivial case;
- for the elements of $(\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G}$: this case is a reduction of the previous one but it is however equivalent; since such an element x is not in the support of G then $x_{\mathcal{G}}$ does not exist, and x is in H if and only if it is in F; note that the generated clauses are exactly the same removing $x_{\mathcal{G}}$;
- for the elements of $(\mathcal{G} \cap \mathcal{H})\backslash\mathcal{F}$: this is the symmetrical case for G;
- for the elements of $\mathcal{H}\backslash(\mathcal{F} \cup \mathcal{G})$: the support variables of H that are not in F or in G must be false;
- for the elements of $\mathcal{F}\backslash\mathcal{H}$: elements of the support of F that are not in the support of H cannot be in F;
- for the elements of $\mathcal{G}\backslash\mathcal{H}$: symmetrical case for G.

$$F \cup G = H$$
$$\Leftrightarrow_{enc}$$
$$\begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \vee x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & \begin{array}{l}|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses}\end{array} \\ \forall x \in (\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}} & 2.|(\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G}| \text{ binary clauses} \\ \forall x \in (\mathcal{G} \cap \mathcal{H})\backslash\mathcal{F},\ x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & 2.|(\mathcal{G} \cap \mathcal{H})\backslash\mathcal{F}| \text{ binary clauses} \\ \forall x \in \mathcal{H}\backslash(\mathcal{F} \cup \mathcal{G}),\ \neg x_{\mathcal{H}} & |\mathcal{H}\backslash(\mathcal{F} \cup \mathcal{G})| \text{ unit clauses} \\ \forall x \in \mathcal{F}\backslash\mathcal{H},\ \neg x_{\mathcal{F}} & |\mathcal{F}\backslash\mathcal{H}| \text{ unit clauses} \\ \forall x \in \mathcal{G}\backslash\mathcal{H},\ \neg x_{\mathcal{G}} & |\mathcal{G}\backslash\mathcal{H}| \text{ unit clauses} \end{cases}$$

2.7 Inclusion constraint

- for the elements of $\mathcal{F} \cap \mathcal{G}$: such an element is in G if it is in F,
- for the elements of $\mathcal{F}\backslash\mathcal{G}$: since these elements cannot be in G, they cannot be in F;

$$F \subseteq G \Leftrightarrow_{enc} \begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}} & |\mathcal{F} \cap \mathcal{G}| \text{ binary clauses} \\ \forall x \in \mathcal{F}\backslash\mathcal{G},\ \neg x_{\mathcal{F}} & |\mathcal{F}\backslash\mathcal{G}| \text{ unit clauses} \end{cases}$$

2.8 Difference constraint

- for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: such elements are in H if and only if they are in F and not in G;
- for the elements of $\mathcal{F}\backslash(\mathcal{G} \cup \mathcal{H})$: such elements cannot be in F;
- for the elements of $\mathcal{H}\backslash\mathcal{F}$: such elements cannot be in H;
- for the elements of $(\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G}$: such elements are in H if and only if they are in F;
- for the elements of $(\mathcal{F} \cap \mathcal{G})\backslash\mathcal{H}$: since such elements cannot be in H, if they are in F they also must be in G;

$$H = F\backslash G$$
$$\Leftrightarrow_{enc}$$
$$\begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge \neg x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & |\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ & +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses} \\ \forall x \in \mathcal{F}\backslash(\mathcal{G} \cup \mathcal{H}),\ \neg x_{\mathcal{F}} & |\mathcal{F}\backslash(\mathcal{G} \cup \mathcal{H})| \text{ ternary clauses} \\ \forall x \in \mathcal{H}\backslash\mathcal{F},\ \neg x_{\mathcal{H}} & |\mathcal{H}\backslash\mathcal{F}| \text{ unit clauses} \\ \forall x \in (\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}} & 2.|(\mathcal{F} \cap \mathcal{H})\backslash\mathcal{G}| \text{ binary clauses} \\ \forall x \in (\mathcal{F} \cap \mathcal{G})\backslash\mathcal{H},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}} & |(\mathcal{F} \cap \mathcal{G})\backslash\mathcal{H}| \text{ binary clauses} \end{cases}$$

2.9 Multi-union constraint

The multi-union constraint $H = \bigcup_{i=1}^{n} F_i$ is equivalent to the n constraints expressed as $H = F_1 \cup (F_2 \cup (\ldots (F_{n-1} \cup F_n) \ldots)$. It is not only a short-hand, but it also significantly reduces the number of generated clauses. Indeed, elements of $\bigcap_{i=1}^{n} \mathcal{F}_i$ are considered once in the multi-union constraint whereas it is considered n times in the corresponding n union constraints. We do not detail the encoding since this is an extension of the union constraint. In the next formulae, the set $\{1, \ldots, n\}$ is noted N.

$$H = \bigcup_{i=1}^{n} F_i$$
$$\Leftrightarrow_{enc}$$
$$\begin{cases} \forall I, J \in \mathcal{P}(N), I \neq \emptyset, I \cup J = N, & \\ \forall x \in \mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i)\backslash(\bigcup_{j \in J} \mathcal{F}_j), \bigvee_{i \in I} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}} & (I) \\ \forall x \in \mathcal{H}\backslash(\bigcup_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}} & (II) \\ \forall i \in [1..n],\ \forall x \in \mathcal{F}_i\backslash\mathcal{H},\ \neg x_{\mathcal{F}_i} & (III) \end{cases}$$

(I) generates

$$\sum_{\substack{I,J \in \mathcal{P}(N), \\ I \neq \emptyset, \\ I \cup J = N}} \big(|\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i)\backslash(\bigcup_{j \in J} \mathcal{F}_j)|.(|I| + 1)\big) \text{ binary clauses}$$

and

$$\sum_{\substack{I,J \in \mathcal{P}(N), \\ I \neq \emptyset, \\ I \cup J = N}} \big(|\mathcal{H} \cap (\bigcap_{i \in I} \mathcal{F}_i)\backslash(\bigcup_{j \in J} \mathcal{F}_j)|\big) \ (|I| + 1)\text{-ary clauses}$$

(II) generates $|\mathcal{H}\backslash(\bigcup_{i=1}^{n} \mathcal{F}_i)|$ unit clauses
(III) generates $\sum_{i=1}^{n} |(\mathcal{F}_i\backslash\mathcal{H}|$ unit clauses

Note also that in our implementation that generates SAT instances, the result of an union must be stored in a set: thus, $H = \bigcup_{i=1}^{n} F_i$ is equivalent to $H = F_1 \cup H_1$, $H_1 = F_2 \cup H_2$, …,$H_{n-1} = F_{n-1} \cup F_n$. The multi-union constraint thus also significantly reduce the number of variables (variables necessary for the intermediate sets H_i).

2.10 Multi-intersection constraint

Similarly, we define the multi-intersection constraints. As for the multi-union, the advantage is the gain of clauses, and of variables in our implementation of the encoding.

$$H = \bigcap_{i=1}^{n} F_i \Leftrightarrow_{enc} \begin{cases} \forall x \in \mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i),\ \ \bigwedge_{i=1}^{n} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}} & (I) \\ \forall x \in \bigcap_{i=1}^{n} \mathcal{F}_i\backslash\mathcal{H},\ \ \bigvee_{i=1}^{n}(\neg x_{\mathcal{F}_i}) & (II) \\ \forall x \in \mathcal{H}\backslash(\bigcap_{i=1}^{n} \mathcal{F}_i),\ \neg x_{\mathcal{H}} & (III) \end{cases}$$

(I) generates $2.|\mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i)|$ $(n+1)$-ary clauses
(II) generates $|\bigcap_{i=1}^{n} \mathcal{F}_i \backslash \mathcal{H}|$ n-ary clauses
(III) generates $|\mathcal{H} \backslash (\bigcap_{i=1}^{n} \mathcal{F}_i)|$ unit clauses

2.11 Cardinality constraint

This constraint is interesting to enforce the size of a set, or to compute the size of a set. We denote by $k = |G|$ the cardinality constraint linking the cardinal of G to the finite domain number (or variable) k. This constraint has been studied for the encoding of global constraints, see for example Bailleux and Boufkhad (2003).

The very intuitive encoding of this constraint is quite simple. If we have a support $\mathcal{G}$ of size n and we want to obtain a set G of k elements ($k \leq n$) we have to verify that:

- All the sets of $k+1$ variables have at least one false variable.
- All the sets of $n-k+1$ variables have at most one true variable.

$$|G| = k \quad \Leftrightarrow_{enc}$$
$$\forall \{x_1, \dots, x_{k+1}\} \subseteq \mathcal{V}, \bigvee_i \neg x_i, \forall \{x_1, \dots, x_{n-k+1}\} \subseteq \mathcal{V}, \bigvee_i x_i$$

The weakness of this encoding is the number of generated clauses:

$$\frac{n!}{(k+1)! + (n-k-1)!} + \frac{n!}{(k-1)! + (n-k+1)!}$$

A more efficient encoding (but less intuitive) for this constraint is the use of the unary representation of integers (an integer $k \in [0..n]$ is represented by 1 k times followed by 0 $n-k$ times). This encoding is presented in Bailleux and Boufkhad (2003) with two main components: the *totalizer* and the *comparator*. Note that we have chosen this encoding for the unit clauses it generates (see Sect. 3.3.2).

The totalizer corresponds to a balanced binary tree structure. It is used to associate an auxiliary variable (output variable) for each variable of the cardinality constraint (input variable) and to sort these new variables such that the true variables are placed before the false variables. Internal variables used to linked input and output variables are called linking variables. The main property of the binary tree is that each non-leaf node corresponds to the union of the two children. The leaves are the input variables and the seed is the set of the output variables. Each node N has two child nodes C^1 and C^2 that are sets of Boolean variables. We denote C^1_α the α-th variable of the set C^1.

The totalizer is encoded by generating for each node the next clauses:

$$\bigwedge_{\substack{0 \leq \alpha \leq |C^1| \\ 0 \leq \beta \leq |C^2| \\ 0 \leq \gamma \leq |N| \\ \alpha+\beta=\gamma}} (\neg C^1_\alpha \vee \neg C^2_\beta \vee N_\gamma) \wedge (C^1_{\alpha+1} \vee C^2_{\beta+1} \vee \neg N_{\gamma+1})$$

with

- $C^1_0 = C^2_0 = N_0 = 1$
- $C^1_{|C^1|+1} = C^2_{|C^2|+1} = N_{|N|+1} = 0$

The comparator enforces the cardinal k of the set simply by assigning the true value to the first k output variables (noted s_i) of the totalizer. Its encoding is very simple:

$$\bigwedge_{1\leq i\leq k} s_i \bigwedge_{k+1\leq j\leq n} \neg s_j$$

In total, if G is over the support $\mathcal{G}$ of size n, then the set constraint $|G| = k$ generates:

- $n + \sum_{i=1}^{n} 2u_i^n \left(\lfloor \frac{u_i^n}{2} \rfloor + 1\right)\left(\lceil \frac{u_i^n}{2} \rceil + 1\right) - \left(\frac{u_i^n}{2} + 1\right)$ clauses
- $\sum_{i=1}^{n} u_i^n$ variables.

with $u_n^n = 1$,$u_1^n = n$ and $u_i^n = u_{2i-1}^n + 2u_{2i}^n + u_{2i+1}^n$.

3 Models for the social golfer problem

In this section we describe various SAT related models for the social golfer problem.

3.1 Direct encoding

In order to present (and then compare) a SAT model for the social golfer problem which does not use set constraints, we give here a model, similar to the one of Triska and Musliu (2012) (which was already a revision of Gent and Lynce 2005) without auxiliary variables.

The Boolean variables to be considered are denoted $g_{q',p',g',w'}$ meaning (when $g_{q',p',g',w'}$ is true) that player q' is the p'-th player of the group number g' of week w' with:

- p' ranging from 1 to p, p being the number of players in each group;
- g' ranging from 1 to g, g being the number of groups each week;
- q' ranging from 1 to q, $q = g.p$ being the total number of players;
- and w ranging from 1 to w, w being the number of weeks considered.

With the $q.p.g.w$ variables of type $g_{q',p',g',w'}$, the constraints are:

- each golfer plays once per week;
- there is p players in each group;
- two players never play twice in the same group.

Each golfer plays at least once per week To enforce that each golfer plays at least once per week, we need the following $g.p.w$ clauses:

$$\bigwedge_{q'=1}^{q} \bigwedge_{w'=1}^{w} \bigvee_{p'=1}^{p} \bigvee_{g'=1}^{g} g_{q',p',g',w'} \tag{1}$$

meaning that for each week w', each player q' is at least the p'-th player in one group g'.

Each players plays at most once per week Enforcing that each players plays at most once per week is done in two steps, first enforcing that each golfer plays at most once per group in each week: on week w', group g', the same player cannot play both on position p' of g' and position p'' of g':

$$\bigwedge_{q'=1}^{q} \bigwedge_{w'=1}^{w} \bigwedge_{p'=1}^{p} \bigwedge_{g'=1}^{g} \bigwedge_{p''=p'+1}^{p} \neg g_{q',p',g',w'} \vee \neg g_{q',p'',g',w'} \tag{2}$$

Formula (2) consists in $q.w.g.p.(p-1)/2$ clauses.

Then, the following $q.w.p.(p-1).g.(g-1)/4$ clauses ensure than a player does not play in more than a group each week:

$$\bigwedge_{q'=1}^{q} \bigwedge_{w'=1}^{w} \bigwedge_{p'=1}^{p} \bigwedge_{g'=1}^{g} \bigwedge_{g''=g'+1}^{g} \bigwedge_{p''=p'+1}^{p} \neg g_{q',p',g',w'} \vee \neg g_{q',p'',g'',w'} \tag{3}$$

Groups are correct The same has to be done for groups to ensure that they are correct: one and only one player per position in each group, each week. There is at least a golfer playing at position p' in the group g' on week w'; this gives $w.p.g$ clauses:

$$\bigwedge_{w'=1}^{w} \bigwedge_{p'=1}^{p} \bigwedge_{g'=1}^{g} \bigvee_{q'}^{q} g_{q',p',g',w'} \tag{4}$$

And at most one golfer plays at position p' in the group g' on week w':

$$\bigwedge_{w'=1}^{w} \bigwedge_{p'=1}^{p} \bigwedge_{g'=1}^{g} \bigwedge_{q'}^{q} \bigwedge_{q''=q'+1}^{q} \neg g_{q',p',g',w'} \vee \neg g_{q'',p',g',w'} \tag{5}$$

which results in $q.(q-1).w.p.g/2$ clauses.

The socialization constraint The only remaining constraint (named the socialization constraint) states that two players cannot play twice in the same group, i.e., if a player q' plays in the same group g' on the same week w' as player q'', and that q' plays in another group g'' another week w'', then q'' cannot play on group g'' on week w'' at whatever position:

$$\bigwedge_{w'=1}^{w} \bigwedge_{g'=1}^{g} \bigwedge_{w''=w'+1}^{w} \bigwedge_{g''=1}^{g} \bigwedge_{q'=1}^{q} \bigwedge_{p_1=1}^{p} \bigwedge_{p_1'=1}^{p} \bigwedge_{q''=q'+1}^{q} \bigwedge_{p_2=1}^{p} \bigwedge_{p_2'=1}^{p}$$

$$g_{q',p_1,g',w'} \wedge g_{q'',p_2,g',w'} \wedge g_{q',p_1',g'',w''} \rightarrow \neg g_{q'',p_2',g'',w''} \tag{6}$$

Formula (6) is the hard point of the direct model with a complexity of $w.(w-1).g^2.q.(q-1).p^4/4$ clauses.

Complexity of the direct encoding The complexity of the direct encoding DE which contains Formulae (1)–(6) is thus: $\mathcal{O}(w^2.g^4.p^6)$ in terms of clauses with $p^2.g^2.w$ variables.

3.2 Variants of the direct encoding

3.2.1 The ladder matrix structure

In Gent and Lynce (2005) a ladder matrix is used: the ladder matrix, which was first presented in Gent and Prosser (2002), introduces a set of auxiliary variables $g'_{i,k,l} \leftrightarrow \bigvee_{p'=1}^{p} g'_{i,p',k,l}$. Intuitively, these new variables abstract the positions of the players in the group. These new variables together with the characteristics of the ladder matrix are then used to model the socialization constraint. The resulting constraints are a bit less complex than the socialization constraint given above, but the ladder matrix introduces an "intermediate level" in the model which is not so simple to handle and not expressive. Moreover, it also results from this model more variables and more clauses.

3.2.2 Intermediate variables

In Triska and Musliu (2012), $q.g.w$ intermediate variables $g'_{i,k,l}$ are introduced:

$$\forall i \in [1..q], \forall k \in [1..g], \forall l \in [1..w], g'_{i,k,l} \leftrightarrow \bigvee_{p'=1}^{p} g_{i,p',k,l} \tag{7}$$

As for the ladder matrix, these variables abstract the positions of players in the groups. These variables simplify the socialization constraint by abstracting positions as follows:

$$\begin{gathered} \bigwedge_{w'=1}^{w} \bigwedge_{g'=1}^{g} \bigwedge_{w''=w'+1}^{w} \bigwedge_{g''=1}^{g} \bigwedge_{q'}^{q} \bigwedge_{q''=q'+1}^{q} \\ (\neg g'_{q',g',w'} \vee \neg g'_{q'',g',w'}) \vee (\neg g'_{q',g'',w''} \vee \neg g'_{q'',g'',w''}) \end{gathered} \tag{8}$$

This introduces $q.w.g$ new intermediate variables $g'_{i,k,l}$ and $q.w.g.(p+1)$ clauses in $g'_{i,k,l} \leftrightarrow \bigvee_{p'=1}^{p} g'_{i,p',k,l}$, but this significantly reduces the complexity of the new socialization constraint from $w.(w-1).g^2.q.(q-1).p^4/4$ to $w.(w-1).g^2.q.(q-1)/4$.

The complexity of the Triska–Musliu encoding (2012) (Formulae 1–5, 7, and 8) is thus $\mathcal{O}(w^2.g^4.p^2)$ in terms of clauses. In the following we call this encoding TME. A more complete analysis in terms of variables and clauses is given in Sect. 5.2.

3.3 SAT encoding for set constraint model

We propose a model for the social golfer problem using set constraints in a solver independent way. These constraints are then encoded into SAT using our $\Leftrightarrow_{enc}$ rules.

3.3.1 Set constraints model

An instance of the problem is thus given by a triple $g-p-w$:

- p is the number of players per group;
- g is the number of groups per week;
- w is the number of weeks;

The universe for this model is the set of players $\mathcal{P} = \{p_1, \ldots, p_q\}$ with $q = g.p$ being the total number of players. We need the following $w.g$ set variables to model the groups $G_{1,1}$, …, $G_{w,g}$. The set $G_{i,j}$ is the group number j of week i and is over the support $\mathcal{G}_{i,j} = \mathcal{P}$. Each $G_{i,j}$ will contain p players from $\mathcal{P}$. Note that the supports are minimal and cannot be reduced without loosing solutions (or symmetric solutions). We now give the constraints of the social golfer problem.

p players per group every weeks:

$$\forall i \in [1..w], \forall j \in [1..g], \quad |G_{i,j}| = p \tag{9}$$

Every golfer plays every weeks:

$$\forall i \in [1..w] \bigcup_{j=1..g} G_{i,j} = \mathcal{P} \tag{10}$$

No golfer plays in two groups the same week:

$$\forall i \in [1..w] \bigcap_{j=1..g} G_{i,j} = \emptyset \tag{11}$$

However, Constraints (11) are not required since they are implied by Constraints (9) and Constraints (10).

Two players cannot play twice together in the same group : The simplest formulation is: $\forall p_1, p_2 \in \mathcal{P}, \forall w_1, w_2 \in [1..w], \forall g_1, g_2 \in [1..g], p_1 \neq p_2 \wedge (g_1 \neq g_2 \vee w_1 \neq w_2) \wedge p_1 \in G_{g_1,w_1} \wedge p_2 \in G_{g_1,w_1} \wedge p_1 \in G_{g_2,w_2} \rightarrow p_2 \notin G_{g_2,w_2}$ meaning : if two different golfers play in the same group g_1, if p_1 plays in another group g_2 then p_2 cannot play in this group g_2. However, due to the permutations p_1, p_2, w_1, w_2, and g_1, g_2, this constraint introduces redundancies that can be removed using the following constraint:

$$\begin{gathered} \forall w_1, w_2 \in [1..w], p_i, p_j \in \mathcal{P}, g_1, g_2 \in [1..g], \\ w_1 > w_2 \wedge i > j \wedge \\ p_i \in G_{w_1,g_1} \wedge p_j \in G_{w_1,g_1} \wedge p_i \in G_{w_2,g_2} \rightarrow p_j \notin G_{w_2,g_2} \end{gathered} \tag{12}$$

Another formulation of these constraints can be given using the cardinality constraints:

$$\forall w_1, w_2 \in [1..w], g_1, g_2 \in [1..g], w_1 > w_2 \wedge |G_{w_1,g_1} \cap G_{w_2,g_2}| \leq 1 \tag{13}$$

3.3.2 SCE: set constraint encoding

From the set constraint model proposed previously, our $\Leftrightarrow_{enc}$ encoding rule automatically generates SAT instances as describe in Sect. 2. For each type of the above constraints we give the number of clauses generated in the SAT instance:

p players per group every weeks : Constraints (9) generates

$$w.g.w.\left(g.p + \sum_{i=1}^{g.p}\left[2u_i^{g.p}\left(\lfloor\frac{u_i^{g.p}}{2}\rfloor + 1\right)\left(\lceil\frac{u_i^{g.p}}{2}\rceil + 1\right) - \left(\frac{u_i^{g.p}}{2} + 1\right)\right]\right)$$

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 (9) is $\mathcal{O}(w^2.g^3.p^2)$.

Every golfer plays every week : Constraints (10) generates $w.g.p$ clauses.

Two players cannot play twice together in the same group : Two formulations are possible:

- with implication formulation, Constraints (12) generates $w.(w-1).g.(g+1).q.(q-1)/2)$ clauses ($\mathcal{O}(w^2.g^4.p^2)$).
- with cardinality formulation, Constraints (13) 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 generated SAT instances Complexity of Constraints (12) is $\mathcal{O}(w^2.g^4.p^2)$ whereas complexity of Constraints (13) is $\mathcal{O}(w^2.g^5.p^3)$. Thus in the following we will only focus on the implication formulation (Constraints 12). To summarize, the complexity of the SAT instances generated by the SCE model (Set Constraint Encoding model) made from

Constraints (9), (10), and (12) is $\mathcal{O}(w^2.g^4.p^2)$. In Sect. 5.2, we show the exact numbers of clauses that are required for specific instances of the social golfer problem.

Post-treatment by Unit Propagation Unit propagation is a simply process corresponding to constraint propagation. The idea is to eliminate unit clauses (clauses with only false literals and one free literal) by valuing the free literal to $true$. This valuation can produce new unit clauses and then the process is achieved until there is no longer any unit clause. In term of complexity, algorithms for unit propagation is in polynomial time; however, in practice, this process is insignificant compared to solving time and may significantly reduce:

- instances size,
- number of variables,
- and solving time.

Note also that the cardinality constraint encoding that we have chosen generates a lot of unit clauses that vanish using unit propagation.

4 Symmetry breaking for the social golfer problem

The idea of symmetry breaking is to remove symmetric solutions and to ease 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 these symmetries by adding new constraints or modifying the model. Gent and Lynce (2005) proposes some clauses to remove symmetries among players, to order groups within a week with respect to their first player, to order lexicographically the weeks with respect to the second player in the first group of each week, … However, these clauses become more and more complicated and mistakes can easily be introduced. Indeed, Triska and Musliu (2012) revised the clauses for symmetry breaking of Gent and Lynce (2005) in order to correct the ranges of the various $\bigvee$ and $\bigwedge$ appearing in these clauses.

More symmetries can be broken, such as in Frisch et al. (2002) or Flener et al. (2002). All symmetries can be broken, such as shown in Crawford et al. (1996), but this is often at the cost of a super exponential number of constraints. Thus, this cannot be considered in practice.

4.1 Symmetry breaking for TME

In Triska and Musliu (2012), three types of symmetry breaking are added to the TME encoding. Note that this is done by adding constraints. The first one consists in breaking the symmetry among players within each group.

$$\bigwedge_{i=1}^{p.g} \bigwedge_{j=1}^{p-1} \bigwedge_{k=1}^{g} \bigwedge_{l=1}^{w} \bigwedge_{m=1}^{i} \neg G_{i,j,k,l} \vee \neg G_{m,(j+1),k,l} \tag{14}$$

The second one consists in ordering all groups within a single week by their first players.

$$\bigwedge_{i=1}^{p.g} \bigwedge_{k=1}^{g-1} \bigwedge_{l=1}^{w} \bigwedge_{m=1}^{i-1} \neg G_{i,1,k,l} \vee \neg G_{m,1,(k+1),l} \tag{15}$$

The last one consists in strictly ascending second players in the first group of each week.

$$\bigwedge_{i=1}^{p.g} \bigwedge_{l=1}^{w} \bigwedge_{m=1}^{i} \neg G_{i,2,1,l} \vee \neg G_{m,2,1,(l+1)} \tag{16}$$

The encoding $\text{TME}^{\text{SB(p)}}$corresponding to the Triska–Musliu encoding breaking symmetries among players is thus defined by Formulae (1)–(5), (7), (8), (14). The encoding $\text{TME}^{\text{SB(p,g,w)}}$corresponding to TME (breaking symmetries among players, groups, and weeks) is thus Formulae (1)–(5), (7), (8), (14)–(16).

Note that symmetry (14) does not apply to the SCE model: using sets there is no permutation of players inside a group.

4.2 Symmetry breaking with set constraint model

With our set constraint language, we have two possibilities to break symmetries. The first one consists in adding some constraints to the initial model; the second one consists in refining/modifying the model itself by modifying the supports of sets and the constraints.

Since our model is different from the ones of Gent and Lynce (2005) and Triska and Musliu (2012), we do not obtain exactly the same symmetries: we do not have symmetries due to the numbering of players inside a group, but we have symmetric weeks and symmetric groups in a week. In the following, we break symmetries by completely fixing the first week (f1), and then by fixing the first player of the p first groups of each week as in Gent and Lynce (2005) and Triska and Musliu (2012) (f2).

The first group of symmetry breaking (f1) 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 to the second group of the first week; and so on.

We consider a second group f2 of symmetry breaking which completes f1. f2 consists in spreading the first p players (who already played together the first week in the first group due to f1) in different groups each week: the first player in the first group of each week (except the first week); the second one in the second group of each week; and so on. This approximately corresponds to group (23) of constraints of Triska and Musliu (2012).

We first consider the following fact to simplify the following models: when p (the number of players per group) becomes greater than g (the number of groups per week) we can rather obviously see that the problem has no solution. Indeed, consider the p players of the first group of the first week; for the second week, they all must play in different groups; thus, the number of groups needs to be greater or equal to the number of players per group, otherwise, there is no solution. In the following, we thus consider $g \geq p$. However, if one does not want to make this simplification, it is sufficient to change p by $min(g, p)$ in the following, and to add the constraints "Two players cannot play twice together in the same group" between $G_{1,1}$ and the other groups. Indeed, these constraints make immediately the model unsatisfiable for $g < p$.

4.2.1 Symmetry breaking for the set constraint model by adding constraints

In this section constraints are added to the initial model in order to break symmetries. For f1, we only have to add the following simple constraints to the model of the SCE.

$$\forall i \in [1..q],\ p_i \in G_{1,((i-1)\ div\ p)+1} \tag{17}$$

For the second group f2 of symmetry breaking, the required constraints are also simple:

$$\forall i \in [2..w], \forall j \in [1..p],\ p_j \in G_{i,j} \tag{18}$$

We can note that these constraints add clauses to the set model and its SAT encoding, but all these extra constraints are unit clauses that will produce unit propagation: thus, they will vanish.

The SAT encoding of the set model with symmetry breaking by adding constraints to the model is named $\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$and consists in Constraints (9), (10), (12), (17), and (18).

Symmetry breaking f1 and f2 can be added to the TME model:

- in TME, (17) corresponds to:

$$\bigwedge_{i=1}^{p.g} G_{i,((i-1)\ mod\ p)+1,((i-1)\ div\ p))+1,1} \tag{19}$$

- and (18) corresponds to:

$$\bigwedge_{l=2}^{w} \bigwedge_{k=1}^{g} G_{k,1,k,l} \tag{20}$$

4.2.2 *Symmetry breaking for the set constraint model by modifying the model*

Modifying the model is more tedious. However, the gain is to reduce the supports of sets and cardinality constraints. These modified models will thus significantly reduce the size of the generated SAT instances.

The only modification for f1 consists in both modifying the supports of the groups of the first week and to fix these groups:

$$\forall i \in [1..g], \mathcal{G}_{1,i} = \{p_{1+(i-1).g}, \ldots p_{p+(i-1).g}\}$$

and

$$\forall i \in [1..g], G_{1,i} = \mathcal{G}_{1,i} \tag{21}$$

The other sets, variables, and constraints remain unchanged.

To introduce f2, we change the group variables. Instead of the $G_{i,j}$, we now consider the sets $G'_{1,1}, \ldots, G'_{w,g}$ 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 $j \le p$, $G_{i,j} = G'_{i,j}$ otherwise.

The support of the $G'_{1,i}$ (i.e., the groups of the first week) are defined as with $SB1$. Since the $min(p, g)$ first player are spread on the $min(p, g)$ first groups of each week, the supports of the other groups can be reduced. Let $\mathcal{P}' = \{p_{min(p,g)+1}, \ldots, p_q\}$ be the set of golfers except the first ones. The supports can thus be defined by:

$$\forall i \in [2..w], \forall j \in [1..g], \mathcal{G}_{i,j} = \mathcal{P}'$$

Constraints are modified as follows.

P players per group every weeks : Constraints (9) must be replaced by Constraints (22)–(24).

$$\forall i \in [1..g], \quad |G'_{1,i}| = p \tag{22}$$

$$\forall j \in [2..w], \forall i \in [1..p], \quad |G'_{j,i}| = p - 1 \tag{23}$$

$$\forall j \in [2..w], \forall i \in [p+1..g], \quad |G'_{j,i}| = p \tag{24}$$

Every golfer plays every week : Constraints (25) replace Constraints (10).

$$\forall j \in [2..w] \bigcup_{i=1..g} G_{j,i} = \mathcal{P}' \tag{25}$$

Two players cannot play twice together in the same group: Constraints (12) are replaced by Constraints (26)–(29).

We recall here that we are working on $G'_{i,j}$ which has the following relation with the intial set $G_{i,j}$ of the model without symmetry breaking: if $j \leq p$ and $i > 1$, then $G_{i,j} = G'_{i,j} \cup \{p_j\}$. 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:

$$\begin{gathered}\forall w_1, w_2 \in [2..w], \ p_i \in \mathcal{P}, \ g_1 \in [1..p], \ w_1 > w_2, \\ p_i \in G'_{w_1,g_1} \ \rightarrow \ p_i \notin G'_{w_2,g_1}\end{gathered} \tag{26}$$

The relation between other two groups is not changed as shown below.

Constraints between a group of the first week (except the first group) and groups of other weeks:

$$\begin{gathered}\forall w_1 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [2..g], g_2 \in [1..g], \ i > j, \\ p_i \in G'_{1,g_1} \ \wedge \ p_j \in G'_{1,g_1} \ \wedge \ p_i \in G'_{w_1,g_2} \ \rightarrow \ p_j \notin G'_{w_1,g_2}\end{gathered} \tag{27}$$

Note that if one does not consider the simplification $p \leq g$, 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).

Constraints between two groups (except of the first week) equally numbered with an index greater than p:

$$\begin{gathered}\forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1 \in [p+1..g], \ w_1 > w_2, \ i > j, \\ p_i \in G'_{w_1,g_1} \ \wedge \ p_j \in G'_{w_1,g_1} \ \wedge \ p_i \in G'_{w_2,g_1} \ \rightarrow \ p_j \notin G'_{w_2,g_1}\end{gathered} \tag{28}$$

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

$$\begin{gathered}\forall w_1, w_2 \in [2..w], \ p_i, p_j \in \mathcal{P}, \ g_1, g_2 \in [1..g], \ w_1 > w_2, \ g_1 \neq g_2, \ i > j, \\ p_i \in G'_{w_1,g_1} \ \wedge \ p_j \in G'_{w_1,g_1} \ \wedge \ p_i \in G'_{w_2,g_2} \ \rightarrow \ p_j \notin G'_{w_2,g_2}\end{gathered} \tag{29}$$

The SAT encoding of the set model with symmetry breaking by modifying the model is named $\text{SCE}^{\text{SBM(f1,f2)}}$ and consists in Constraints (21)–(29).

5 Comparisons of models

We now define an order over models with respect to their solutions. Let $\sigma(m)$ denotes the complete set of solutions of the model m. Then, we define the $\succeq$ order as follows:

$$m_1 \succeq m_2 \quad \text{iff} \quad \sigma(m_2) \subseteq \sigma(m_1)$$

and

$$m_1 \cong m_2 \quad \text{iff} \quad \sigma(m_1) = \sigma(m_2)$$

$\succeq$ thus enables us to compare models in terms of solutions. Consequently, we obtain the following correspondences between models:

$$\begin{array}{ccccccc}
\text{TME} & \succeq & \text{TME}^{\text{SB(p)}} & \succeq & \text{TME}^{\text{SB(p,f1,f2)}} & \succeq & \text{TME}^{\text{SB(p,g,w,f1,f2)}} \\
 & & \cong & & \cong & & \\
 & & \text{SCE} & \succeq & \text{SCE}^{\text{SBM(f1,f2)}} & & \\
 & & & & \cong & & \\
 & & & & \text{SCE}^{\text{SBC(f1,f2)}} & &
\end{array}$$

and

$$\text{TME}^{\text{SB(p,g,w)}} \succeq \text{TME}^{\text{SB(p,g,w,f1,f2)}}$$

where SB(x) denotes the x broken symmetries:

- p: symmetries inside a group are broken by ordering players w.r.t. their numbers
- g: symmetries in a week are broken by ordering groups inside a week w.r.t. their first players
- w: symmetries between weeks are broken by ordering weeks with respect to the second players of the first groups.
- f1 and f2: f1 fixes the first week; and f2 fixes the first player of the p first group of the next weeks.

Unit propagation does not change the set of solutions, and thus does not modify the above order. The $\succeq$ order will help us to better compare models that are equivalent with respect to their solutions.

Table 1 summarizes the various encodings that we will compare in the following sections. These encodings have been described in previous sections. NAME_{UP} denotes the encoding NAME after unit propagation.

5.1 Expressiveness

We compare here the models in terms of expressiveness. Comparisons in terms of structures (number of clauses and variables) are given in the next section.

The first remark is that the variables we use in the set model are much simpler. Indeed, we have only two indices instead of 4, making them more readable. This is due to the fact that we do not have to number the positions in a group (groups are sets), and we do not have to add an index for the number of players (players are members of the groups).

The second difference to be noticed is the simplicity and expressiveness of constraints. Indeed, set constraints are more expressive than pure SAT clauses. Then, the encoding in SAT is performed using the encoding rules $\Leftrightarrow_{enc}$. The advantage is double:

Table 1 List of the encoding names, descriptions and the corresponding constraints or formulae

Encoding name	Description	Corresponding constraints or formulae
DE	Direct encoding	(1)–(6)
TME	Triska–Musliu encoding	(1)–(5), (7), (8)
$\mathrm{TME}^{\mathrm{SB(p)}}$	TME with sym. breaking p	(1)–(5), (7), (8), (14)
$\mathrm{TME}^{\mathrm{SB(p,f1,f2)}}$	TME with sym. breaking p, f1, f2	(1)–(5), (7), (8), (14), (19)–(20)
$\mathrm{TME}^{\mathrm{SB(p,g,w)}}$	TME with sym. breaking p, g, w	(1)–(5), (7), (8), (14)–(16)
$\mathrm{TME}^{\mathrm{SB(p,g,w,f1,f2)}}$	TME with sym. breaking p, g, w, f1, f2	(1)–(5), (7), (8), (14)–(16),(19)–(20)
SCE	SAT encoding of the set constraint model	(9), (10), (12)
$\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$	SCE with sym. breaking f1, f2 by adding constraints	(9), (10), (12), (17), (18)
$\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$	SCE with sym. breaking f1, f2 by modifying the model	(21)–(29)
$\mathrm{NAME}_{\mathrm{UP}}$	Encoding after unit propagation treatment	

– first, constraints are readable, expressive, easy to modify, resulting in a much understandable model;
– second, less mistakes are introduced since the modeling process is much simpler.

Last, but not least, the set encoding is solver independent: the same model (changing the syntax) could be used in a CSP solver with set constraints[1] or in a SAT solver after applying the rule encoding $\Leftrightarrow_{enc}$ proposed above.

With the set model, symmetry breaking can be achieved by adding constraints or 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.

To summarize, 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 in the set model. However, all symmetries cannot be broken in the set model (e.g., 15 and 16), and techniques such as supersymmetric[2] modeling Prestwich (2003) cannot be applied for various problems modeled with sets (for example, for the social golfer problem with sets we cannot introduce symmetries inside a group by changing the order of players).

Each encoding produces specific SAT instances. We compare the direct encodings and the set constraint encoding in two ways: the size of the provided instances and the ease to solve them with a complete SAT solver.

[1] Indeed, we did it with MiniZinc (http://www.minizinc.org/) but did not obtain good results in terms of running time and instances that could be solved.

[2] This technique which consists in increasing the number of symmetries in order to obtain more solutions, sometimes gives even better results, especially with incomplete solvers such as local search.

Table 2 Size of instances generated using the direct encoding (DE), the Triska and Musliu encoding (TME) Triska and Musliu (2012), the set constraints encoding [(with unit propagation post-process (SCE_{UP}) and without (SCE)]

Prob.	DE		TME		SCE		SCE_{UP}	
	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls
5-3-6	1350	3,203,055	1800	60,255	8625	50,400	1410	43,905
5-3-7	1575	4,481,085	2100	79,485	11,110	67,985	1645	60,410
8-4-4	4096	48,850,176	5120	322,816	24,224	234,912	**3840**	**204,928**
8-4-5	5120	81,378,880	6400	482,880	34,752	372,992	**4800**	**335,520**
8-4-6	6144	121,896,960	7680	674,688	47,072	542,816	**5760**	**497,856**
8-4-7	7168	170,815,680	8960	898,240	61,184	744,384	**6720**	**691,936**
8-4-8	8192	227,723,776	10,240	1,153,536	77,088	977,696	**7680**	**917,760**
8-4-9	9216	292,552,704	11,520	1,440,576	94,784	1,242,752	**8640**	**1,175,328**
8-4-10	10,240	365,690,880	12,800	1,759,360	114,272	1,539,552	**9600**	**1,464,640**
9-4-6	7776	196,150,032	9720	1,047,762	117,324	858,366	**7344**	**792,882**
9-4-7	9072	274,564,584	11,340	1,400,994	157,284	1,180,026	**8568**	**1,103,634**
9-4-8	10,368	366,042,816	12,960	1,805,256	203,076	1,552,716	**9792**	**1,465,416**
9-4-9	11,664	470,584,728	14,580	2,260,548	254,700	1,976,436	**11,016**	**1,878,228**
9-4-10	12,960	588,190,320	16,200	2,766,870	312,156	2,451,186	**12,240**	**2,342,070**

5.2 Model structure

In order to compare our set constraint encoding, we generate a set of social golfer instances with: the direct encoding DE, the Triska–Musliu encoding (TME) proposed in Triska and Musliu (2012), and our set constraint encoding with unit propagation post-treatment (SCE_{UP}) and without (SCE). In Table 2, each instance is defined by the triple (groups, players per group, weeks) and for each encoding the number of variables and the number of (generated) clauses are provided. It is not possible to compare efficiency of an encoding only in terms of instance size; this is done in the next section. Nevertheless, big instances are intractable due to the limited size of computer memory. It is thus necessary to generate as small as possible instances. In Table 2, for each instance, encodings generating the smallest number of clauses and variables are in bold.

The DE encoding is clearly unsuitable when the number of players or groups increases: the number of clauses immediately blows up. With the introduction of auxiliary variables the number of clauses is less important for TME but the number of variables is increased. 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 variables: indeed, SCE generates a lot of unit and binary clauses (Sect. 3.1) than vanish using unit propagation.

5.3 Impact of the symmetry breaking

Social golfer problem has a lot of identical solutions modulo symmetries. In Table 3 we apply the symmetry breaking processes presented in Sect. 4.2 to the instances proposed in Table 2.

For TME, introducing symmetry breaking constraints only increases the number of clauses (around 10 % more clauses), the number of variables does not change. Note also that unit

Table 3 Size of SAT instances w.r.t. the various models and with/without UP as a pre-process

Prob.	TME		$TME^{SB(p)}$		$TME^{SB(p,g,w)}$		$TME^{SB(p,f1,f2)}$		$TME^{SB(p,g,w,f1,f2)}$		$TME_{UP}^{SB(p,f1,f2)}$		$TME_{UP}^{SB(p,g,w,f1,f2)}$	
	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls
5-3-6	1800	60,255	1800	67,455	1800	70,935	1800	67,485	1800	70,965	1020	28,950	1020	29,652
5-3-7	2100	79,485	2100	87,885	2100	91,965	2100	87,918	2100	91,998	1224	39,798	1224	40,656
8-4-4	5120	322,816	5120	373,504	5120	389,872	5120	373,548	5120	389,916	3024	164,976	3024	169,442
8-4-5	6400	482,880	6400	546,240	6400	566,832	6400	546,288	6400	566,880	4032	268,576	4032	274,666
8-4-6	7680	674,688	7680	750,720	7680	775,536	7680	750,772	7680	775,588	5040	396,480	5040	404,194
8-4-7	8960	898,240	8960	986,944	8960	1,015,984	8960	987,000	8960	1,016,040	6048	548,688	6048	558,026
8-4-8	10,240	1,153,536	10,240	1,254,912	10240	1,288,176	10,240	1,254,972	10,240	1,288,236	7056	725,200	7056	736,162
8-4-9	11,520	1,440,576	11,520	1,554,624	11,520	1,592,112	11,520	1,554,688	11,520	1,592,176	8064	926,016	8064	938,602
8-4-10	12,800	1,759,360	12,800	1,886,080	12,800	1,927,792	12,800	1,886,148	12,800	1,927,860	9072	1,151,136	9072	1,165,346
9-4-6	9720	1,047,762	9720	1,155,654	9720	1,190,952	9720	1,155,710	9720	1,191,008	6560	631,520	6560	644,192
9-4-7	11,340	1,400,994	11,340	1,526,868	11,340	1,568,160	11,340	1,526,928	11,340	1,568,220	7872	878,736	7872	894,048
9-4-8	12,960	1,805,256	12,960	1,949,112	12,960	1,996,398	12,960	1,949,176	12,960	1,996,462	9184	1,166,256	9184	1,184,208
9-4-9	14,580	2,260,548	14,580	2,422,386	14,580	2,260,548	14,580	2,422,454	14,580	2,475,734	10,496	1,494,080	10,496	1,514,672
9-4-10	16,200	2,766,870	16,200	2,946,690	16,200	3,005,964	16,200	2,946,762	16,200	3,006,036	11,808	1,862,208	11,808	1,885,440

Prob.	SCE		$SCE^{SBM(f1,f2)}$		$SCE^{SBC(f1,f2)}$		SCE_{UP}		$SCE_{UP}^{SBM(f1,f2)}$		$SCE_{UP}^{SBC(f1,f2)}$	
	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls
5-3-6	8625	50,400	5702	21,487	8625	50,430	1410	43,905	**860**	**17,680**	980	23,110
5-3-7	11,110	67,985	7734	30,243	11,110	68,018	1645	60,410	**1032**	**25,680**	1176	33,690
8-4-4	24,224	234,912	14,192	95,712	24,224	234,956	3840	204,928	**2376**	**77,700**	2580	91,548
8-4-5	34,752	372,992	22,476	173,180	34,752	373,040	4800	335,520	**3168**	**149,184**	3440	176,240

Table 3 continued

Prob.	SCE		$SCE^{SBM(f1,f2)}$		$SCE^{SBC(f1,f2)}$		SCE_{UP}		$SCE_{UP}^{SBM(f1,f2)}$		$SCE_{UP}^{SBC(f1,f2)}$	
	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls	#Vars	#Cls
8-4-6	47,072	542,816	32,552	273,440	47,072	542,868	5760	497,856	**3960**	**243,460**	4300	288,020
8-4-7	61,184	744,384	44,420	396,492	61,184	744,440	6720	691,936	**4752**	**360,528**	5160	426,888
8-4-8	77,088	977,696	58,080	542,336	77,088	977,756	7680	917,760	**5544**	**500,388**	6020	592,844
8-4-9	94,784	1,242,752	73,532	710,972	94,784	1,242,816	8640	1,175,328	**6336**	**663,040**	6,880	785,888
8-4-10	114,272	1,539,552	90,776	902,400	114,272	1,539,620	9600	1,464,640	**7128**	**848,484**	7740	1,006,020
9-4-6	117,324	858,366	46,344	447,832	117,324	858,422	7344	792,882	**5620**	**471,690**	5620	471,690
9-4-7	157,284	1,180,026	63,368	652,344	157,284	1,180,086	8568	1,103,634	**6008**	**561,712**	6744	700,830
9-4-8	203,076	1,552,716	82,984	895,176	203,076	1,552,780	9792	1,465,416	**7024**	**782,620**	7868	974,904
9-4-9	254,700	1,976,436	105,192	1,176,328	254,700	1,976,504	11,016	1,878,228	**8040**	**1,039,956**	8992	1,293,912
9-4-10	312,156	2,451,186	129,992	1,495,800	312,156	2,451,258	12,240	2,342,070	**9056**	**1,333,720**	10,116	1,657,854

For each instance, encodings generating the smallest number of clauses and variables are in bold propagation is not worth for TME instances nor for $\mathrm{TME}^{\mathrm{SB(p)}}$ or $\mathrm{TME}^{\mathrm{SB(p,g,w)}}$ instances: there is no unit clause and the size of the instance is not changed (both in terms of variables and clauses). For $\mathrm{TME}^{\mathrm{SB(p,f1,f2)}}$ and $\mathrm{TME}^{\mathrm{SB(p,g,w,f1,f2)}}$ unit propagation significantly reduces the size of the generated SAT instances.

For SCE, symmetry breaking by adding constraints adds a negligible amount of constraints (see $\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$). More interestingly, adding symmetry breaking by modifying the model ($\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$) 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 due to the reduction of supports and to the cardinality constraints: sets with $k-1$ elements instead of k, and less clauses are necessary when supports are smaller.

Without unit propagation, the instances of $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$ are always the smallest one generated with respect to the number of clauses.

Unit propagation has no impact at all on TME, $\mathrm{TME}^{\mathrm{SB(p)}}$ and $\mathrm{TME}^{\mathrm{SB(p,g,w)}}$. However, its impact is significant on $\mathrm{TME}^{\mathrm{SB(p,f1,f2)}}$, $\mathrm{TME}^{\mathrm{SB(p,g,w,f1,f2)}}$, SCE, $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$, and $\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$:

- for $\mathrm{TME}^{\mathrm{SB(p,f1,f2)}}$, unit propagation decreases the number of variables (around 40 %) and the number of clauses (up to 50 %).
- 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 $\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$, 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 $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$, unit propagation is less spectacular: indeed, the initial model itself is reduced by adding symmetry breaking. However, the number of variable is divided by 5 up to 15. The number of clauses is reduced of about 10 %.

To summarize, unit propagation is more beneficial to $\mathrm{SCE}^{\mathrm{SBC(f1,f2)}}$; however, $\mathrm{SCE}_{\mathrm{UP}}^{\mathrm{SBM(f1,f2)}}$ always gives the best instances in terms of number of clauses and number of variables.

6 Experimental Analysis

In the previous section we have shown that SCE enables us to obtain the smallest instances with unit propagation. The use of symmetry breaking also reduces the size of the SAT instances. It can happen that symmetry breaking makes more difficult the resolution: by changing the search landscape, an "easy" solution can disappear; with incomplete solvers (such as local search), symmetry breaking can partitions the search space and makes difficult a path to a solution.[3] In this section we will compare the efficiency of the encodings in terms of running time.

To compare our set constraints encoding with Triska–Musliu Triska and Musliu (2012) encoding, we use the well known solver Minisat Eén and Sörensson (2003). This solver won various competitions.[4] Since some few years, a pre-treatment named SatELite Eén and Biere (2005) has been added to Minisat in order to drastically reduce the number of clauses (e.g., by using subsumptions detections) and variables (e.g., eliminating pure literals). It is now included in Minisat but an option enables one to deactivate it. When we will use this pre-treatment, its running time will be included in the running time of Minisat.

[3] We have tested the incomplete solver Sparrow Balint and Fröhlich (2010) and no solution was found with a time out of one hour.

[4] http://www.satcompetition.org/.

Experimentations are realized on a 2.60 GHz Intel Core i5-2540M CPU and 4 GB RAM. For each experiment, the time-out is 300 s. Larger execution times were tested but no real differences were observed. Results for the direct encoding DE are not presented since, as supposed, no results are obtained in a reasonable time.

Tables 4 and 5 represent respectively the running time of Minisat with the use of SatElite as pre-treatment and without pre-treatment.

First of all, the two tables show that the use of SatElite is difficult to predict: for some instances, it significantly improves the results whereas for others, it significantly degrades the results. On average, it does not improve the results and the best running times are obtained without pre-treatment.

Moreover, symmetry breaking modifying the model ($\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$) provides the best results (or results very close to the best ones), with or without pre-treatment. The use of unit propagation seems to have a weak impact to the resolution time of $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$.

When unit propagation is applied, the new SAT instance is solved more quickly. However, the running time needed by unit propagation brings out a global running time higher than without the use of unit propagation.

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 $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$ when the pre-treatment is not applied. Finally, the best results are obtained without pre-treatment.

7 Discussion

Modeling Modeling a problem with set constraints and then automatically generating the corresponding SAT instances is much simpler than directly writing encodings such as DE or TME. Breaking symmetries can be rather tedious in direct encodings, very easy by adding constraints in the set model, and rather tedious by modifying the set constraint model. Using sets, some symmetries (such as 14) can vanish naturally. However, all symmetries that can be broken in set models can be broken in a direct model, whereas the opposite is not true. For example, we cannot add symmetry breaking for symmetries g and w corresponding to clauses (15) and (16) for the TME model.

Using a higher level formalism (such as our set constraint) is thus beneficial to the modeling phase: it simplifies the task, and avoid making errors (mainly errors in the numerous indices required by a direct encoding). The SAT encoding is then automatically done.

SAT instances We have shown that the SAT instances that are automatically produced by our encoding rules are of good quality:

– they always produce significantly less clauses (with or without symmetry breaking, and with or without unit propagation);
– with unit propagation, they also generate less variables;
– and finally, they are solved faster with Minisat (if we do not take into account the unit propagation running time), without "tuning parameters", with or without pre-treatment with SatElite.

Table 4 Minisat **with** SatElite: Running time for the SCE encoding and the TME encoding. Formulations with symmetry breaking and UP are compared

Prob.	TME	$TME^{SB(p)}$	$TME^{SB(p,g,w)}$	$TME^{SB(p,f1,f2)}$	$TME^{SB(p,g,w,f1,f2)}$	$TME_{UP}^{SB(p,f1,f2)}$			$TME_{UP}^{SB(p,g,w,f1,f2)}$		
						UP	Min.	Tot.	UP	Min.	Tot.
5-3-6	8.92	2.16	0.69	0.23	0.22	0.44	0.07	0.51	0.44	0.08	0.52
5-3-7	98.28	3.62	13.37	0.37	0.49	0.67	1.21	1.88	0.68	0.46	1.14
8-4-4	1.04	1.24	1.33	1.29	1.44	3.72	0.26	3.98	3.75	0.27	4.02
8-4-5	2.26	2.47	2.64	2.55	2.65	7.02	0.64	7.66	6.99	0.67	7.66
8-4-6	4.44	5.48	5.16	4.92	4.96	11.57	1.34	12.91	11.66	1.36	13.02
8-4-7	34.25	51.60	94.68	8.34	12.48	17.90	4.58	22.48	17.85	4.08	21.93
8-4-8	–	–	–	–	–	26.04	–	–	26.02	–	-
8-4-9	–	–	–	–	–	38.44	–	–	36.60	–	-
8-4-10	–	–	–	–	–	49.01	–	–	48.89	–	-
9-4-6	8.45	8.22	10.52	8.70	8.73	22.03	2.57	24.6	22.25	3.29	25.54
9-4-7	13.69	17.46	27.16	16.04	15.98	33.76	4.78	38.54	34.38	6.70	41.08
9-4-8	–	117.45	31.87	37.40	52.00	49.30	17.15	66.45	50.18	30.35	80.53
9-4-9	–	–	–	–	–	68.66	–	–	69.34	–	-
9-4-10	–	–	–	–	–	93.95	–	–	93.32	–	-

Prob.	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}			SCE_{UP}^{SBM}			SCE_{UP}^{SBC}		
				UP	Min.	Tot.	UP	Min.	Tot.	UP	Min.	Tot.
5-3-6	0.18	0.06	0.12	0.41	0.12	0.53	3.1	0.07	3.17	0.41	0.04	0.45
5-3-7	1.42	0.13	1.21	0.54	5.09	5.63	0.48	0.09	0.57	0.58	0.08	0.66
8-4-4	0.97	0.32	1.19	1.63	0.90	2.53	1.41	0.29	1.7	1.64	0.27	1.91
8-4-5	1.93	0.86	2.51	2.67	1.89	4.56	2.39	0.84	3.23	2.73	0.78	3.51
8-4-6	3.65	1.87	4.74	3.89	3.65	7.54	3.56	1.82	5.38	4.01	1.71	5.72

Table 4 continued

Prob.	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}			SCE_{UP}^{SBM}			SCE_{UP}^{SBC}		
				UP	Min.	Tot.	UP	Min.	Tot.	UP	Min.	Tot.
8-4-7	8.66	3.59	8.52	5.40	7.52	12.92	5.09	3.64	8.73	5.57	3.46	9.03
8-4-8	–	–	–	7.23	–	–	6.70	–	–	7.29	–	–
8-4-9	–	–	–	9.09	–	–	8.71	–	–	9.56	–	–
8-4-10	–	–	–	11.86	–	–	10.84	–	–	11.61	–	–
9-4-6	11.24	3.15	10.34	6.11	11.10	17.21	5.72	2.71	8.43	6.42	4.58	11.00
9-4-7	18.95	5.80	17.80	8.37	19.04	27.41	8.09	5.12	13.21	8.70	8.76	17.46
9-4-8	31.87	11.10	29.60	11.22	31.48	42.7	10.70	12.72	23,42	12.01	14.90	26.91
9-4-9	–	–	–	14.55	–	–	13.92	–	–	15.00	–	–
9-4-10	–	–	–	18.00	–	–	17.44	–	–	18.64	–	–

Table 5 Minisat **without** SatElite: Running time for the SCE and TME encodings. Formulations with symmetry breaking and UP are compared

Prob.	TME	$TME^{SB(p)}$	$TME^{SB(p,g,w)}$	$TME^{SB(p,f1,f2)}$	$TME^{SB(p,g,w,f1,f2)}$	$TME_{UP}^{SB(p,f1,f2)}$			$TME_{UP}^{SB(p,g,w,f1,f2)}$		
						UP	Min.	Tot.	UP	Min.	Tot.
5-3-6	9.37	1.97	0.30	0.09	0.10	0.44	0.03	0.47	0.44	0.06	0.50
5-3-7	97.47	20.80	24.86	0.22	0.42	0.67	0.47	1.14	0.68	0.24	0.92
8-4-4	0.05	0.10	0.23	0.08	0.09	3.72	0.03	3.75	3.75	0.03	3.78
8-4-5	0.08	0.16	0.58	0.15	0.24	7.02	0.08	7.1	6.99	0.12	7.11
8-4-6	0.25	0.46	3.58	0.38	0.24	11.57	0.31	11.88	11.66	0.30	11.96
8-4-7	27.05	7.43	25.88	1.95	8.82	17.90	0.73	18.63	17.85	7.17	25.02
8-4-8	–	–	–	–	–	26.04	–	–	26.02	–	-
8-4-9	–	–	–	–	–	38.44	–	–	36.60	–	-
8-4-10	–	–	–	–	–	49.01	–	–	48.89	–	-
9-4-6	0.23	0.47	3.72	0.37	0.36	22.03	0.33	22.36	22.25	0.38	22.63
9-4-7	0.31	6.15	6.61	1.30	3.04	33.76	1.15	34.91	34.38	1.08	35.46
9-4-8	247.83	149.27	–	17.39	15.81	49.30	19.35	68.65	50.18	37.16	87.34
9-4-9	–	–	–	–	–	68.66	–	–	69.34	–	-
9-4-10	–	–	–	–	–	93.95	–	–	93.32	–	-

Prob.	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}			SCE_{UP}^{SBM}			SCE_{UP}^{SBC}		
				UP	Min.	$Tot.$	UP	Min.	$Tot.$	UP	Min.	Tot.
5-3-6	1.05	0.01	0.01	0.41	0.26	0.67	3.1	0.01	3.11	0.41	0.01	0.42
5-3-7	9.19	0.06	0.13	0.54	5.67	6.21	0.48	1.79	2.27	0.58	0.28	0.86
8-4-4	0.09	0.03	0.07	1.63	0.07	1.7	1.41	0.03	1.44	1.64	0.03	1.67
8-4-5	0.13	0.06	0.11	2.67	0.06	2.73	2.39	0.05	2.44	2.73	0.07	2.8
8-4-6	0.27	0.14	0.18	3.89	0.19	4.08	3.56	0.08	3.64	4.01	0.09	4.1

Table 5 continued

Prob.	SCE	SCE^{SBM}	SCE^{SBC}	SCE_{UP}			SCE_{UP}^{SBM}			SCE_{UP}^{SBC}		
				UP	Min.	Tot.	UP	Min.	Tot.	UP	Min.	Tot.
8-4-7	3.53	0.48	1.71	5.40	1.94	7.34	5.09	0.56	5.65	5.57	0.98	6.55
8-4-8	–	–	–	7.23	–	–	6.70	–	–	7.29	–	–
8-4-9	–	–	–	9.09	–	–	8.71	–	–	9.56	–	–
8-4-10	–	–	–	11.86	–	–	10.84	–	–	11.61	–	–
9-4-6	0.37	0.13	0.29	6.11	0.25	6.36	5.72	0.11	5.83	6.42	0.13	6.55
9-4-7	0.31	6.61	0.58	8.37	0.36	8.73	8.09	0.14	8.23	8.70	0.24	8.94
9-4-8	14.66	5.03	1.10	11.22	20.93	32.15	10.70	2.62	13.32	12.01	0.68	12.69
9-4-9	–	–	–	14.55	–	–	13.92	–	–	15.00	–	–
9-4-10	–	–	–	18.00	–	–	17.44	–	–	18.64	–	–

Symmetry breaking We have shown that 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 for modifying the model is more important than the effort for adding constraints. This extra work is very beneficial for the size of the generated SAT instances, but not so much worth for the solving time (it is depending on instances, and pre-treatment). Thus, one has to make the trade-off between solving time and modeling time. The size of the generated instances can be the deciding factor: larger problems can be modeled and generated introducing symmetry breaking into the model as in $\mathrm{SCE}^{\mathrm{SBM(f1,f2)}}$.

Set constraints in constraint programming The expressiveness of set constraints in constraint programming (such as in Gervet 1994; Legeard and Legros 1991, or MiniZinc (http://www.minizinc.org/)) is more or less the same as the one of our set constraints in terms of sets: that was our goal. However, our approach is different: in systems such as Gervet (1994) or http://www.emn.fr/z-info/choco-solver/, sets constraints are not the only constraints, but a special set solver has to be designed to solve these models. For example, the mechanism of Gervet (1994) consists in reducing the domain of the sets by working on lower and upper bounds of the sets and to combine this process with search. Note that the domain of a set is similar to our notion of support, and lower and upper bounds of sets are the smallest and largest elements of a set with respect to a given ordering. Our approach is 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 model written with set constraints into a good quality (in terms of size and solving time) SAT instance that is efficiently solved by an existing multi-purpose SAT solver.

Note that in the future, we want to add a pre-process to reduce support sizes. Indeed, the size of the SAT instances depends on the size of the supports. For the social golfer problem, supports are minimal: they cannot be reduced without loosing solutions. But for some other problems, supports can be reduced by a deduction process (withtout loosing solution), and thus, generated SAT instances can be reduced. Such a process could be to enforce generalized arc consistency (GAC) for sets (i.e., similar to one application of the first phase of the mechanism of Gervet (1994) for bounds without search). A next step would also be to compare generalized arc consistency for sets and unit propagation.

Note also that in Azevedo (2006) some comparisons of set constraint solvers in constraint programming are given for the social golfer problem. Most of the results reported are obtained by giving special (dynamic) search heuristics or special solving mechanisms. The approach is thus very different from ours.

The social golfer problem in constraint programming We have tried to solve several CSP models with the MiniZinc (http://www.minizinc.org/) constraint system:

– 3 set constraint models: our SCE model, the model presented in the MiniZinc tutorial, and a model of the MiniZinc github;[5]
– 3 constraints models: 2 from the MiniZinc github based on two 2D matrices and one on a 3D matrix of player numbers; one from the webpage of Hakan Kjellerstrand.[6]

[5] https://github.com/MiniZinc/minizinc-benchmarks/tree/master/golfers.

[6] http://www.hakank.org/minizinc/social_golfers1.mzn.

In terms of declarativeness, these models are readable and rather easy to understand. However, in terms of efficiency, we could only solve small instances of the SGP. That is the reason why we did not present them in the previous tables.

8 Conclusion

We have presented a technique for encoding set constraints into SAT: the modeling process is achieved using some very expressive set constraints which are then automatically transformed into SAT variables and clauses using our $\Leftrightarrow_{enc}$ encoding rules. This technique has been applied successfully to model and encode the social golfer problem, and to study some symmetry breaking on this problem.

The advantages of our technique are the following:

- the modeling process is simple, expressive, and readable. Moreover, it is solver independent and independent from CSP or SAT;
- the technique is less error-prone than direct SAT encodings;
- breaking symmetry can be achieved by just adding new constraints or by refining/modifying the model (this cannot be done so easily with direct encodings such as DE or TME);
- the SAT instances which are automatically generated are smaller than the ones of Triska and Musliu (2012); with unit propagation, our instances also contain less variables than the ones of Triska and Musliu (2012);
- 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 SatElite (the pre-treatment mechanism of Minisat).

We have tested our technique to model and solve other problems (such as n-queen problem, Sudoku, WhoWithWhom, car sequencing, ...). Each time we obtained very readable and simple set models. The generated SAT instances also appeared to be well-suited for Minisat.

In the future, we plan to use our set constraints encoding for formalizing domain variables and sequences of elements. To this end, we will need to add some new constraints and to complete our $\Leftrightarrow_{enc}$ encoding rule.

We want to refine the notion of supports and reduce their sizes. As said before, this does not have any impact on a problem such as the social golfer problem for which supports are already "minimal". But for many problems (in which supports are not clear at the principle), it is important to reduce the size of the supports (using a pre-treatment) before generating the SAT instances.

Finally, we also plan to combine set constraints with some arithmetic constraints, and we want to define the corresponding SAT encoding.

References

Azevedo, F. (2006). An attempt to dynamically break symmetries in the social golfers problem. In F. Azevedo, P. Barahona, F. Fages, & F. Rossi (Eds.), *CSCLP, Lecture Notes in Computer Science, vol. 4651* (pp. 33–47). Springer.

Bacchus, F. (2007). Gac via unit propagation. In *Proceedings of CP 2007, LNCS, vol. 4741* (pp. 133–147). Springer.

Bailleux, O., & Boufkhad, Y. (2003). Efficient cnf encoding of boolean cardinality constraints. In *Proceedings of CP 2003, vol. 2833* (pp. 108–122). Springer.

Balint, A., & Fröhlich, A. (2010). Improving stochastic local search for SAT with a new probability distribution. In *Proceedings of Theory and Applications of Satisfiability Testing—SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11–14, 2010* (pp. 10–15).

Bessière, C., Hebrard, E., & Walsh, T. (2004). Local consistencies in sat. In *Selected Revised Papers of SAT 2003, LNCS, vol. 2919* (pp. 299–314). Springer.

Choco. http://www.emn.fr/z-info/choco-solver/.

Cotta, C., Dotú, I., Fernández, A. J., & Hentenryck, P. V. (2006). Scheduling social golfers with memetic evolutionary programming. In *Proceedings of HM 2006, LNCS, vol. 4030* (pp. 150–161). Springer.

Crawford, J. M., Ginsberg, M. L., Luks, E. M., & Roy, A. (1996). Symmetry-breaking predicates for search problems. In *Proc. of KR'96* (pp. 148–159). Morgan Kaufmann.

Eén, N., & Biere, A. (2005). Effective preprocessing in sat through variable and clause elimination. In *Proceedings of SAT 2005, vol. 3569* (pp. 61–75).

Eén, N., & Sörensson, N. (2003) An extensible sat-solver. In *Proceedings of SAT 2003, vol. 2919* (pp. 502–518).

Flener, P., Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., & Walsh, T. (2002). Breaking row and column symmetries in matrix models. In *Proceedings of CP 2002, vol. 2470* (pp. 462–476). Springer.

Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., & Walsh, T. (2002). Global constraints for lexicographic orderings. In *Proceedings of CP 2002, vol. 2470* (pp. 93–108). Springer.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability, a guide to the theory of NP-completeness*. San Francisco: W.H. Freeman & Company.

Gent, I., & Lynce, I. (2005). A sat encoding for the social golfer problem. In *IJCAI'05 workshop on modelling and solving problems with constraints*.

Gent, I. P., & Prosser, P. (2002). An empirical study of the stable marriage problem with ties and incomplete lists. In *Proceedings of ECAI'2002* (pp. 141–145). IOS Press.

Gent, I. P., & Walsh, T. (1999). CSPLib: A benchmark library for constraints. In *Proceedings of CP 1999, LNCS, vol. 1713* (pp. 480–481). Springer.

Gervet, C. (1994). Conjunto: Constraint propagation over set constraints with finite set domain variables. In *Proceedings of ICLP'94* (p. 733). MIT Press.

Lardeux, F., Monfroy, E., Saubion, F., Crawford, B., & Castro, C. (2009). Sat encoding and csp reduction for interconnected alldiff constraints. In *Proceedings of MICAI 2009* (pp. 360–371).

Legeard, B., & Legros, E. (1991). Short overview of the clps system. In *Proceedings of PLILP'91, vol. 528* (pp. 431–433). Springer.

MiniZinc. http://www.minizinc.org/.

Prestwich, S. D. (2003). Negative effects of modeling techniques on search performance. *Annals OR*, *118*(1–4), 137–150.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Amsterdam: Elsevier.

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