Réduction et Encodage des Contraintes Ensemblistes en SAT

Frédéric Lardeux, Eric Monfroy

▶ **To cite this version:**

Frédéric Lardeux, Eric Monfroy. Réduction et Encodage des Contraintes Ensemblistes en SAT. Douzièmes journées Francophones de Programmation par Contraintes (JFPC), 2016, Montpellier, France. hal-02709514

HAL Id: hal-02709514
https://univ-angers.hal.science/hal-02709514

Submitted on 13 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Réduction et Encodage des Contraintes Ensemblistes en SAT

Frédéric Lardeux[1] **Eric Monfroy**[2]

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

Résumé

D'un côté, les problèmes de satisfaction de contraintes (CSP) procurent une méthode déclarative et expressive pour modéliser les problèmes. D'un autre côté, les solveurs pour les problèmes de satisfiabilité de formules logique propositionnelle (SAT) peuvent manipuler des instances énormes jusqu'à des millions de clauses et variables. Dans cet article, nous présentons une approche bénéficiant de la modélisation CSP et de la résolution SAT. Notre technique consiste à modéliser, de façon expressive, des problèmes de contraintes ensemblistes en en CSP qui sont ensuite automatiquement réduits afin de retirer les valeurs des variables qui ne participent à aucune solution. Ces CSPs réduits sont ensuite encodés en de "bonnes" instances SAT qui peuvent être résolues par des solveurs SAT standards. Nous illustrons notre technique par divers problèmes standards : le Sudoku, le Social Golfer Problem et le Sports Tournament Scheduling Problem.

Notre technique est plus simple, plus expressive et moins sensible aux erreurs qu'une modélisation directe en SAT. De plus, les instances SAT automatiquement générées sont généralement plus petites que celles directement écrite pour un problème particulier (comme par exemple pour le Social Golfer Problem [18]) et peuvent être évaluées efficacement même pour des instances énormes. Enfin, la phase de réduction nous permet de repousser les limites et de traiter des problèmes encore plus gros.

Abstract

On the one hand, Constraint Satisfaction Problems (CSP) are a declarative and expressive approach for modeling problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances up to millions of variables and clauses. In this article, we present an approach for taking advantage of both CSP modeling and SAT solving. Our technique consists in expressively modeling set constraint problems as CSPs that are automatically treated by some reduction rules to remove values that do not participate in any solution. These reduced CSPs are then encoded into "good" SAT instances that can be solved by standard SAT solvers. We illustrate our technique on various well-known problems such as Sudoku, the Social Golfer problem, and the Sports Tournament Scheduling problem. Our technique is simpler, more expressive, and less error-prone than direct SAT modeling. The SAT instances that we automatically generate are rather small (even w.r.t. direct-written SAT instances for the Social Golfer problem [18]) and can efficiently be solved up to huge instances. Moreover, the reduction phase enables to push back the limits and treat even bigger problems.

1 Introduction

Une méthode classique consiste à formuler les problèmes combinatoires comme des problèmes de satisfaction de contraintes (CSP) [21]. Un CSP est défini par des variables et des contraintes reliant ces variables. Résoudre un CSP consiste à trouver une affectation des variables qui satisfait toutes les contraintes. L'expressivité est l'une des forces des CSP : les variables et les contraintes peuvent être de types variés. De plus, les contraintes globales améliorent non seulement la résolution mais aussi l'expressivité : elles apportent de nouveaux constructeurs et relations tels que Alldifferent (pour forcer toutes les variables d'une liste à avoir des valeurs différentes), Cardinalité (pour relier un ensemble à sa taille), ...

Une autre façon de formuler les problèmes combinatoires est d'utiliser le problème satisfiabilité d'une formule logique propositionnelle (SAT) [12]. Un problème SAT est une formule booléenne sous forme normale conjonctive, c'est à dire une conjonction de clauses.

Une clause est une disjunction de litéraux et un litéral est une variable ou sa négation. Quand toutes les clauses peuvent être satisfaites, le problème est dit satisfiable. En terme d'expressivité SAT est donc réduit à des variables booléennes et des formules propositionnelles. Coder des contraintes directement en SAT est une tâche pénible (par exemple [22] ou[13]). De plus, lorsque l'on essaie d'optimiser son modèle (en terme de variables et clauses), cela conduit rapidement à des modèles très complexes et illisibles dans lesquels apparaissent facilement des erreurs. Cependant, les solveurs SAT peuvent maintenant manipuler des instances énormes (des millions de variables). Il est donc intéressant de 1) encoder les CSP en SAT (e.g., [4, 6]) afin de bénéficier de l'expressivité des CSP et de la puissance de SAT, et 2) d'introduire plus d'expressivité en SAT, e.g., avec des contraintes globales telles que Alldifferent[17] ou Cardinalité[5].

Divers systèmes de contraintes ensemblistes ([19],[14],[1]) ont été réalisés et il a été montré que de nombreux problèmes peuvent facilement être modélisés par des contraintes ensemblistes.

Nous nous intéressons ici à l'encodage de contraintes ensemblistes en instances SAT. Dans [18], nous avons présenté des règles d'encodage qui sont appliquées directement sur les contraintes ensemblistes CSP. Cependant, nous avons remarqué que ces contraintes ensemblistes ne sont pas toujours aussi petites qu'elles pourraient l'être : des éléments possiblement dans les ensembles peuvent être retirés sans perte de solutions. Ainsi, les instances SAT générées peuvent aussi être réduites.

Il est inconcevable d'assurer que le programmeur écrira des modèles CSP "minimaux", donc notre approche consiste à procurer :

- un ensemble de contraintes simples mais complètes et expressives (intersection, union, cardinal d'ensembles, ...)
- un ensemble de règles de réduction ($\Rightarrow_{red}$) réduisant les modèles CSP par propagation de contraintes [2]. La consistance calculée porte sur les bornes et cardinalité des ensembles (similaire à celle présentée dans [3]) qui est plus forte que la consistance de borne [20].
- un ensemble de règles d'encodage ($\Leftrightarrow_{enc}$) convertissant des contraintes CSP en instances SAT.

Dans ce papier, nous illustrons notre approche avec le Social Golfer Problem[18], le Sports Tournament Scheduling problem [23] ainsi que le Sudoku. Nous avons aussi appliqué notre approche à d'autres problèmes (tels que le Car-sequencing et le WhoWithWhom, ...) et les instances SAT générées automatiquement ne sont pas plus complexes que celles générées et optimisées directement et leur résolution avec un solveur SAT classique (dans notre cas Minisat [10]) est efficace.

Comparé à [18], les règles de réduction $\Rightarrow_{red}$ nous permettent :

- d'obtenir un encodage simplifié ;
- d'obtenir des instances plus petites ; ces problèmes sont résolus plus rapidement ;
- d'aborder et de résoudre des problèmes plus grands que nous étions incapable d'encoder auparavant pour des raisons de taille.

Les travaux sur les techniques d'encodage telles que [4] et [6] établissent une relation entre la résolution CSP et SAT en terme de propriétés telles que les consistances. Nous nous intéressons ici à un autre type de contraintes (i.e., contraintes ensemblistes) et nous tentons d'obtenir de petites instances SAT aux propriétés adaptées aux solveurs SAT. De plus, [4] et [6] ne considèrent pas de règles de réduction comme nos règles $\Rightarrow_{red}$. Notre approche est similaire à [17] où les contraintes globales Alldifferent et des superpositions de contraintes Alldifferent sont manipulées de manières expressives avant d'être encodées automatiquement en SAT par des techniques de réécritures.

Il est à noter que nous utilisons les travaux présentés dans [5] à propos de la contrainte globale Cardinalité afin d'améliorer l'encodage de la cardinalité d'un ensemble. Notre phase de réduction est plus forte que celle du solveur Conjunto [14] et similaire à celle de [8]. Dans [14], la propagation est réalisée à partir des bornes inférieures et supérieures des ensembles. Dans nos règles $\Rightarrow_{red}$ ainsi que dans [8], la propagation est basée sur les bornes inférieure et supérieure mais aussi sur les cardinalités minimum et maximum des ensembles. Il faut aussi noter que dans notre cas, la réduction est utilisée comme pré-traitement avant l'encodage en instances SAT alors que dans Conjunto et dans [8], la réduction est utilisée conjointement à l'énumération afin de produire un solveur complet. Notre objectif n'est pas de résoudre le modèle mais de le traiter afin d'obtenir un meilleur encodage que sera résolu par un solveur SAT. Nous ne cherchons pas à obtenir de meilleurs résultats que les solveurs CSP ensemblistes mais à introduire les contraintes ensemblistes dans SAT.

2 Problème de Satisfaction de Contraintes avec Ensembles

2.1 CSP Ensembliste

Definition 1 (CSP-Ensembliste)
Un CSP-Ensembliste est défini par :

- *un univers $\mathcal{U}$ d'entiers.*

— *un ensemble X de variables entières telles qu'un domaine $D_x \subseteq \mathcal{U}$ est associé à chaque variable entière $x \in \mathcal{X}$.*
— *un ensemble $\mathbb{F}$ de variables ensemblistes telles que pour chaque variable ensembliste $F \in \mathbb{F}$ sont associées :*
  — *une borne inférieure $\mathfrak{f} \subseteq \mathcal{U}$*
  — *une borne supérieure $\mathfrak{F} \subseteq \mathcal{U}$*
  — *une cardinalité minimum $\#_{\downarrow}F \in \mathbb{N}$*
  — *une cardinalité maximum $\#_{\uparrow}F \in \mathbb{N}$*

  Nous avons comme propriétés que : $\mathfrak{f} \subseteq F \subseteq \mathfrak{F}$, $\#_{\downarrow}F \leq \#F \leq \#_{\uparrow}F$, $\#\mathfrak{f} \leq \#_{\downarrow}F$, et $\#\mathfrak{F} \geq \#_{\uparrow}F$ où $\#F$ représente la cardinalité de F.
— *un ensemble de contraintes C qui sont des relations définies sur $\mathcal{D}^{|X|} \times \mathcal{U}^{|\mathbb{F}|}$.*

Il est à noter que la borne inférieure d'une variable ensembliste $F \in \mathbb{F}$ représente des valeurs (entiers) qui sont effectivement dans F ; la borne supérieure représente des valeurs qui sont possiblement (ou sont effectivement si elle sont également dans la borne inférieure) dans F. Nous avons donc $\mathfrak{f} \subseteq F \subseteq \mathfrak{F}$ et les valeurs possibles de F sont les éléments de l'ensemble des parties $2^{\mathfrak{F}}$ contenant $\mathfrak{f}$ et tels que leur cardinalité soit comprise dans $[\#_{\downarrow}F..\#_{\uparrow}F]$.

2.2 Contraintes Ensemblistes de Base

Les variables sont déclarées et initialisées de la manière suivante :
— un intervalle I d'entiers est noté $n..m$ où n et m sont des entiers ; $I = n..m$ représente tous les entiers compris entre n et m ; la borne inférieure de l'intervalle I est notée $\underline{I}$ et la borne supérieure $\overline{I}$.
— un s_intervalle est une séquence ordonnée d'intervalles disjoints [1] :

$$Li = (L_1, \ldots, L_l)$$

Le s_intervalle vide est noté $\perp$. Un s_intervalle $sI = (I_1, \ldots, I_n)$ est inclu dans un s_intervalle $sJ = (J_1, \ldots, J_m)$ si chaque entier apparaissant dans sI apparait aussi dans sJ :

$$sI \subseteq sJ \;\Leftrightarrow\; \forall I_i, \forall v \in [\underline{I_i}..\overline{I_i}], \exists J_k, \underline{J_k} \leq v \leq \overline{J_k}$$

La cardinalité $\#sI$ d'un s_intervalle $sI = (I_1, \ldots, I_n)$ est donnée par : $\#sI = \sum_{i=1}^{n}(\overline{I_i} - \underline{I_i} + 1)$. Les autres opérations (comme $\cup$, $\cap$, ...) sur les s_intervalles peuvent être obtenues de manière similaire.
— l'univers $\mathcal{U}$ est déclaré comme $\boldsymbol{\mathcal{U}} :: \top$ où $\top$ est un s_intervalle ;
— une variable entière x est déclarée comme $\boldsymbol{x :: D_x}$ où son domaine D_x est un s_intervalle ;
— une variable ensembliste F est déclarée comme $\boldsymbol{F :: \mathfrak{f}, \mathfrak{F}, \#_{\downarrow}F, \#_{\uparrow}F}$ où la borne inférieure $\mathfrak{f}$ et sa borne supérieure $\mathfrak{F}$ sont des s_intervalles, et la cardinalité minimum $\#_{\downarrow}F$ et la cardinalité maximum $\#_{\uparrow}F$ sont des entiers.

Une variable ensembliste particulière est celle représentant l'ensemble vide. Elle est notée $\emptyset$ et est définie par $\emptyset :: \perp, \perp, 0, 0$

Considérons F, G, H et F_i (i compris entre 1 et n) étant des variables ensemblistes et x étant une variable entière. Nous énumérons ici plusieurs contraintes ensemblistes courantes que nous avons traitées :

element (dis)equality	$x = y$	$(x \neq y)$
(non)membership	$x \in F$	$(x \notin F)$
set (dis)equality	$F = G$	$(F \neq G)$
intersection	$H = F \cap G$	
union	$H = F \cup G$	
disjoint union	$H = F \sqcup G$	
inclusion	$F \subseteq G$	$(F \not\subseteq G)$
difference	$H = F \setminus G$	
multi-intersection	$F = \bigcap_{i=1}^{n} F_i$	
multi-union	$F = \bigcup_{i=1}^{n} F_i$	
partition	$F = \bigsqcup_{i=1}^{n} F_i$	
cardinality	$x = \|F\|$	
minimum $<$	$x = min(F)$	
maximum $>$	$x = max(F)$	

3 Règles de Réduction

Les règles de réduction $\Rightarrow_{red}$ ont pour but de réduire l'espace de recherche. Elles peuvent donc ajouter des valeurs aux bornes inférieures, supprimer des valeurs aux bornes supérieures, augmenter la cardinalité minimale ou diminuer la cardinalité maximale des variables ensemblistes traitées. La consistance calculée par nos règles est similaire à la consistance calculée par le filtrage de [3] pour les ensembles.

Pour les variables entières, nos règles ne peuvent que supprimer des valeurs de leur domaine. Enfin, certaines règles peuvent aussi amener à un échec.

3.1 Variables Entières

Si une variable entière x a un domaine vide, le CSP n'a pas de solution :

$$D_x = \perp \;\Rightarrow_{red}\; echec \qquad (1)$$

Quand une variable entière x est déclarée deux fois, les deux déclarations sont groupées en une seule :

$$x :: D_x, x :: D' \;\equiv\; x :: D_x \cap D'_x \qquad (2)$$

1. Soit $I = a..b$ et $J = c..d$. $I \prec J$ si $b < c$.

Notons que l'application de la règle 2 remplace $x :: D_x$ et $x :: D'_x$ par $x :: D_x \cap D'_x$.

3.2 Variables Ensemblistes

$$\#_{\downarrow}F > \#_{\uparrow}F \;\Rightarrow_{red}\; echec \tag{3}$$
$$\mathfrak{f} \not\subseteq \mathfrak{F} \;\Rightarrow_{red}\; echec \tag{4}$$
$$\#\mathfrak{f} > \#_{\uparrow}F \;\Rightarrow_{red}\; echec \tag{5}$$
$$\#\mathfrak{F} < \#_{\downarrow}F \;\Rightarrow_{red}\; echec \tag{6}$$

La règle 4 est très importante car elle garantit que toute valeur présente dans $\mathfrak{f}$ est dans $\mathfrak{F}$ et que dans le cas contraire un cas d'échec est constaté. Les règles 4, 5 et 6 sont très utiles quand une contrainte de cardinalité modifie les cardinalités minimum et maximum d'une variable ensembliste.

La règle 7 signifie que F est l'ensemble vide (si $\mathfrak{f} \neq \bot$ alors cela conduira à un cas d'échec avec la règle 4) :

$$\#_{\uparrow}F = 0, \;\; \mathfrak{F} \neq \bot \;\Rightarrow_{red}\; \mathfrak{F} = \bot \tag{7}$$

Les règles 8 et 9 rendent la variable ensembliste F constante quand la taille de sa borne supérieure est égale à sa cardinalité minimum ou quand la taille de sa borne inférieure est égale à sa cardinalité maximum :

$$\#_{\downarrow}F = \#\mathfrak{F}, \;\; \mathfrak{f} \subset \mathfrak{F}, \;\; \#_{\downarrow}F \leq \#_{\uparrow}F \;\Rightarrow_{red}\; \left\{ \begin{array}{l} \mathfrak{f} \leftarrow \mathfrak{F} \\ \#_{\uparrow}F \leftarrow \#_{\downarrow}F \end{array} \right. \tag{8}$$

$$\#_{\uparrow}F = \#\mathfrak{f}, \;\; \mathfrak{f} \subset \mathfrak{F}, \;\; \#_{\downarrow}F \leq \#_{\uparrow}F \;\Rightarrow_{red}\; \left\{ \begin{array}{l} \mathfrak{F} \leftarrow \mathfrak{f} \\ \#_{\downarrow}F \leftarrow \#_{\uparrow}F \end{array} \right. \tag{9}$$

Les règles suivantes ne peuvent être déclenchées que s'il y a une erreur dans la déclaration de la variable ensembliste :

$$\#\mathfrak{f} > \#_{\downarrow}F \;\Rightarrow_{red}\; \#_{\downarrow}F \leftarrow \#\mathfrak{f} \tag{10}$$
$$\#\mathfrak{F} < \#_{\uparrow}F \;\Rightarrow_{red}\; \#_{\uparrow}F \leftarrow \#\mathfrak{F} \tag{11}$$

3.3 Contraintes Ensemblistes

Nous n'avons pas assez d'espace pour présenter les réductions ainsi que les règles de transformation pour toutes les contraintes ensemblistes. Nous nous contentons donc de ne présenter que les règles de réduction pour l'union de deux d'ensembles.

$H = F \cap G$
$\Rightarrow_{red}$

$$\left\{ \begin{array}{lcl} \mathfrak{H}' & \leftarrow & \mathfrak{H} \cap \mathfrak{F} \cap \mathfrak{G} \\ \mathfrak{h}' & \leftarrow & \mathfrak{h} \cup (\mathfrak{f} \cap \mathfrak{g}) \\ \mathfrak{f}' & \leftarrow & \mathfrak{f} \cup \mathfrak{h} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \cup \mathfrak{h} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus (\mathfrak{g} \setminus \mathfrak{H}) \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus (\mathfrak{f} \setminus \mathfrak{H}) \\ \#_{\downarrow}F' & \leftarrow & max\{\#_{\downarrow}F, \#\mathfrak{f}', \#_{\downarrow}H\} \\ \#_{\downarrow}G' & \leftarrow & max\{\#_{\downarrow}G, \#\mathfrak{g}', \#_{\downarrow}H\} \\ \#_{\downarrow}H' & \leftarrow & max\{\#_{\downarrow}H, \;\#\mathfrak{h}'\} \\ \#_{\uparrow}F' & \leftarrow & min\{\#_{\uparrow}F, \#\mathfrak{F}'\} \\ \#_{\uparrow}G' & \leftarrow & min\{\#_{\uparrow}G, \#\mathfrak{G}'\} \\ \#_{\uparrow}H' & \leftarrow & min\{\#_{\uparrow}H, \#_{\uparrow}F, \#_{\uparrow}G, \#\mathfrak{H}'\} \end{array} \right. \tag{12}$$

Quand les 3 ensembles sont fixés, i.e., $\mathfrak{h} = \mathfrak{f} \cap \mathfrak{g} = \mathfrak{H} = \mathfrak{F} \cap \mathfrak{G}$ alors la contrainte $H = F \cap G$ peut être supprimée.

La règle suivante est appliquée quand H est fixé. Pour simplifier la notation, nous notons $\bar{\mathfrak{f}}$ la variable ensembliste correspondant à $\mathfrak{F} \setminus \mathfrak{f}$.

$$\begin{array}{l} H = F \cap G, \\ \mathfrak{h} = \mathfrak{H}, \\ \#_{\downarrow}H = \#_{\uparrow}H \end{array} \Rightarrow_{red} \left\{ \begin{array}{lcl} \mathfrak{H}' & \leftarrow & \mathfrak{H} \\ \mathfrak{h}' & \leftarrow & \mathfrak{h} \\ \mathfrak{f}' & \leftarrow & \mathfrak{f} \cup \mathfrak{h} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \cup \mathfrak{h} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus (\bar{\mathfrak{f}}' \cap \mathfrak{g}') \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus (\bar{\mathfrak{g}}' \cap \mathfrak{f}') \\ \#_{\downarrow}F' & \leftarrow & max\{\#_{\downarrow}F, \#\mathfrak{f}'\} \\ \#_{\downarrow}G' & \leftarrow & max\{\#_{\downarrow}G, \#\mathfrak{g}'\} \\ \#_{\downarrow}H' & \leftarrow & \#_{\downarrow}H \\ \#_{\uparrow}F' & \leftarrow & min\{\#_{\uparrow}F, \#\mathfrak{F}'\} \\ \#_{\uparrow}G' & \leftarrow & min\{\#_{\uparrow}G, \#\mathfrak{G}'\} \\ \#_{\uparrow}H' & \leftarrow & \#_{\uparrow}H \end{array} \right. \tag{13}$$

Nous supposons maintenant que les règles suivantes ont été appliquées pour la contrainte d'intersection. Nous pouvons donc garantir que :

- $\#_{\downarrow}F - \#\mathfrak{f}$ est le nombre minimum de variables entières requis dans F ;
- $\#_{\uparrow}F - \#\mathfrak{f}$ est le nombre maximum de variables entières encore acceptable dans F ;
- $\#(\bar{\mathfrak{f}} \setminus \mathfrak{G})$ est le nombre de variables entières potentiellement dans F n'influençant pas H ;
- $\#(\bar{\mathfrak{f}} \setminus \mathfrak{g})$ est le nombre de variables entières potentiellement dans F influençant H ;
- $\#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}})$ est le nombre de variables entières potentiellement dans F pouvant influencer H ;
- $\#_{\downarrow}F - \#\mathfrak{f} - \#(\bar{\mathfrak{f}} \setminus \mathfrak{G}) - \#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}})$ est le nombre minimum de variables entières requis dans F et qui effectivement modifie H ;
- $\#_{\uparrow}H - \#\mathfrak{h}$ est le nombre maximum de variables entières encore acceptable dans H.

Nous avons ajouté une règle pour le cas spécial de la variable ensembliste vide. Cette règle n'est pas obligatoire mais elle est plus simple et plus lisible que la

précédente. De plus, elle est utile pour la contrainte d'union disjoint :

$$\begin{array}{l}\emptyset = F \cap G \\ \Rightarrow_{red}\end{array}$$

$$\left\{\begin{array}{lcl} \mathfrak{f}' & \leftarrow & \mathfrak{f} \\ \mathfrak{g}' & \leftarrow & \mathfrak{g} \\ \mathfrak{F}' & \leftarrow & \mathfrak{F} \setminus \mathfrak{g} \\ \mathfrak{G}' & \leftarrow & \mathfrak{G} \setminus \mathfrak{f} \\ \#_{\downarrow}F' & \leftarrow & \#_{\downarrow}F \\ \#_{\downarrow}G' & \leftarrow & \#_{\downarrow}G \\ \#_{\uparrow}F' & \leftarrow & \min\{\#_{\uparrow}F, \#(\mathfrak{F} \setminus \mathfrak{g})\} \\ \#_{\uparrow}G' & \leftarrow & \min\{\#_{\uparrow}G, \#(\mathfrak{G} \setminus \mathfrak{f})\} \end{array}\right. \quad (14)$$

4 Règles d'Encodage

Les règles d'encodage $\Leftrightarrow_{enc}$ ont pour but de transformer les contraintes ensemblistes d'un CSP en clauses SAT. Nos règles fonctionnent à la fois sur des instances réduites et non réduites de contraintes ensemblistes.

4.1 Variables Entières

Cette règle d'encodage force chaque variable entière à n'avoir qu'une et une seule valeur de son domaine.

$$\begin{array}{c} VariableEntiere(v, D_v) \\ \Leftrightarrow_{enc} \\ \forall x \in D_v,\ x_v \rightarrow \bigwedge_{y \in D_v, x \neq y} \neg y_v \ \ and \ \ \bigvee_{x \in D_v} x_v \\ \Leftrightarrow \\ (\bigwedge_{x \in D_v} \bigwedge_{y \in D_v, y > x} (\neg x_v \vee \neg y_v)) \wedge \bigvee_{x \in D_v} x_v \end{array}$$

Chaque encodage de variable entière génère $|D_v|.(|D_v|-1)/2$ cl. binaires et 1 $|D_v|aire$ clause.

4.2 Variables Ensemblistes

Pour une variable ensembliste $F :: (\mathfrak{f}, \mathfrak{F}, \#_{\downarrow}F, \#_{\uparrow}F)$, l'encodage consiste à créer des variables booléennes pour chaque valeur de la borne supérieure $\mathfrak{F}$ et de mettre à vrai celles appartenant à la borne inférieure $\mathfrak{f}$. Si nous considérons une constante x de l'univers $\mathcal{U}$, nous notons $?x_{\mathcal{F}}$ la création de la variable booléenne $x_{\mathcal{F}}$ représentant l'appartenance de x à la variable ensembliste F.

$$\begin{array}{c} F :: (\#_{\downarrow}F, \#_{\uparrow}F, \mathfrak{f}, \mathfrak{F}) \\ \Leftrightarrow_{enc} \\ \left\{\begin{array}{l} \forall x \in \mathfrak{F},\ ?x_{\mathcal{F}} \\ \forall x \in \mathfrak{f},\ x_{\mathcal{F}} \end{array}\right. \end{array}$$

4.3 Intersection d'Ensembles

Afin d'être complet, nous considérons tous les cas par rapport à $\mathfrak{H}, \mathfrak{h}, \mathfrak{F}, \mathfrak{f}, \mathfrak{G}, \mathfrak{g}$, bien que de nombreux cas soient impossibles. La table 1 détaille cette règle d'encodage avec en face de chaque cas le nombre et la forme des clauses générées.

5 Résultats expérimentaux

Toutes les expériences ont été réalisées sur un processeur Intel® Xeon® E5-2670 avec 2.3GHz et 230 Go RAM. Les règles d'encodage ont été implémentées en C^{++} et celles de réduction comme des Constraint Handling Rules (CHR[11]). Nous avons utilisé Minisat [10] comme solveur SAT pour toutes nos expérimentations.

5.1 Problèmes encodés

5.1.1 Sudoku

Le problème du Sudoku 9×9 [2] est un puzzle où chaque ligne, chaque colonne et chaque bloc doivent contenir toutes les valeurs entre 1 et 9. Des instances difficiles sont accessibles sur un site web [3] sur lequel elles sont référencées par un numéro de grille.

5.1.2 Social Golfer Problem

Le problème du Social Golfer (problème numéro 10 de la CSPLib[16]) est le suivant : q golfeurs jouent toutes les semaines durant w semaines en se séparant en g groupes de p golfeurs ($q = p.g$). Comment organiser la partie de ces golfeurs afin qu'aucun golfeur ne joue dans le même groupe qu'un autre golfeur plus d'une fois ? Une instance de ce problème est donc caractérisée par un triplet $g - p - w$. De nombreuses instances du Social Golfer sont encore ouvertes et ce problème est attractif puisqu'il est en relation avec des problèmes de cryptage et de couverture. Nous avons testé deux modélisations possibles pour la *contrainte de socialisation* du problème [4]. La première utilise des contraintes de cardinalité (le nom des instances est suffixée par "_C") et la seconde des contraintes d'implication (le nom des instances est suffixée par "_I"). Les differents modèles peuvent être trouvés dans [18].

5.1.3 Sports Tournament Scheduling

Ce problème a été proposé par Toby Walsh (problème numéro 26 de la CSPLib [23]) de la manière suivante : Il faut organiser un tournoi avec n équipes sur $n-1$ semaines où chaque semaine est divisée en $n/2$ périodes et où chaque période est divisée en deux créneaux. La première équipe de chaque créneau

2. http ://en.wikipedia.org/wiki/Sudoku
3. http ://www.e-sudoku.fr/jouer-sudoku-solo.php
4. Deux joueurs ne peuvent jouer plus d'une fois dans un même groupe

$$
\begin{array}{l}
H = F \cap G \\
\Leftrightarrow_{enc}
\end{array}
$$

$$
\forall x \in \mathcal{U}
\begin{cases}
x \in \mathfrak{h} & \begin{cases}
x \in \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & true \\
x \in \mathfrak{G} \setminus \mathfrak{g} & x_{\mathcal{G}} \quad \#(\mathfrak{h} \cap \mathfrak{f} \cap \bar{\mathfrak{g}}) \text{ cl. unitaires} \\
x \notin \mathfrak{G} & false
\end{cases} \\
x \in \mathfrak{F} \setminus \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & x_{\mathcal{F}} \quad \#(\mathfrak{h} \cap \bar{\mathfrak{f}} \cap \mathfrak{g}) \text{ cl. unitaires} \\
x \in \mathfrak{G} \setminus \mathfrak{g} & x_{\mathcal{F}} \wedge x_{\mathcal{G}} \quad \#(\mathfrak{h} \cap (\mathfrak{F} \setminus \mathfrak{f}) \cap \bar{\mathfrak{g}}) \; 2\times \text{ cl. unitaires} \\
x \notin \mathfrak{G} & false
\end{cases} \\
x \notin \mathfrak{F} & \begin{cases}
x \in \mathfrak{g} & false \\
x \in \mathfrak{G} \setminus \mathfrak{g} & false \\
x \notin \mathfrak{G} & false
\end{cases}
\end{cases} \\
x \in \mathfrak{H} \setminus \mathfrak{h} & \begin{cases}
x \in \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{f} \cap \mathfrak{g}) \text{ cl. unitaires} \\
x \in \mathfrak{G} \setminus \mathfrak{g} & x_{\mathcal{H}} \leftrightarrow x_{\mathcal{G}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{f} \cap \bar{\mathfrak{g}}) \times 2 \text{ cl. binaires} \\
x \notin \mathfrak{G} & \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap (\mathfrak{f} \setminus \mathfrak{G})) \text{ cl. unitaires}
\end{cases} \\
x \in \mathfrak{F} \setminus \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & x_{\mathcal{H}} \leftrightarrow x_{\mathcal{F}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{f}} \cap \mathfrak{g}) \times 2 \text{ cl. binaires} \\
x \in \mathfrak{G} \setminus \mathfrak{g} & x_{\mathcal{H}} \leftrightarrow x_{\mathcal{F}} \wedge x_{\mathcal{G}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{f}} \cap \bar{\mathfrak{g}}) \times 2 \text{ cl. binaires } et \text{ 1 cl. ternaire} \\
x \notin \mathfrak{G} & \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap (\bar{\mathfrak{f}} \setminus \mathfrak{G})) \text{ cl. unitaires}
\end{cases} \\
x \notin \mathfrak{F} & \begin{cases}
x \in \mathfrak{g} & \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \mathfrak{g} \setminus \mathfrak{F}) \text{ cl. unitaires} \\
x \in \mathfrak{G} \setminus \mathfrak{g} & \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \cap \bar{\mathfrak{g}} \setminus \mathfrak{F}) \text{ cl. unitaires} \\
x \notin \mathfrak{G} & \neg x_{\mathcal{H}} \quad \#(\bar{\mathfrak{h}} \setminus \mathfrak{F} \setminus \mathfrak{G}) \text{ cl. unitaires}
\end{cases}
\end{cases} \\
x \notin \mathfrak{H} & \begin{cases}
x \in \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & false \\
x \in \mathfrak{G} \setminus \mathfrak{g} & \neg x_{\mathcal{G}} \quad \#(\mathfrak{f} \cap (\bar{\mathfrak{g}} \setminus \mathfrak{H})) \text{ cl. unitaires} \\
x \notin \mathfrak{G} & true
\end{cases} \\
x \in \mathfrak{F} \setminus \mathfrak{f} & \begin{cases}
x \in \mathfrak{g} & \neg x_{\mathcal{F}} \quad \#(\bar{\mathfrak{f}} \cap \mathfrak{g} \setminus \mathfrak{H}) \text{ cl. unitaires} \\
x \in \mathfrak{G} \setminus \mathfrak{g} & \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}} \quad \#(\bar{\mathfrak{f}} \cap \bar{\mathfrak{g}} \setminus \mathfrak{H}) \text{ cl. binaires} \\
x \notin \mathfrak{G} & true
\end{cases} \\
x \notin \mathfrak{F} & \begin{cases}
x \in \mathfrak{g} & true \\
x \in \mathfrak{G} \setminus \mathfrak{g} & true \\
x \notin \mathfrak{G} & true
\end{cases}
\end{cases}
\end{cases}
$$

TABLE 1 – Règle d'encodage pour l'intersection de deux ensembles

joue à domicile alors que la seconde joue à l'extérieur. Un tournoi doit satisfaire les 3 contraintes suivantes : toutes les équipes jouent une fois par semaine ; toutes les équipes jouent au plus deux fois sur la même période au cours du tournoi ; toutes les équipes rencontrent toutes les autres équipes. La valeur n permet de définir totalement une instance du problème. Ce problème peut être vu comme un le problème du Round Robin Tournament auquel une contrainte sur les périodes à été ajoutée.

Voici un modèle ensembliste où les $n-1$ semaines sont notées w et les $n/2$ périodes sont notées p :

- Univers et ensembles d'équipes : $\mathcal{U} :: 1..n, \mathcal{T} :: \mathcal{U}, \mathcal{U}, n, n$
- Matches de 2 équipes pour chaque semaine et chaque période : $\forall i \in [1..w], \forall j \in [1..p], \;\; G_{i,j} :: \perp, \mathcal{U}, 2, 2$
- Chaque équipe joue chaque semaine : $\forall i \in [1..w], \; \mathcal{T} = \bigcup_{j=1}^{p} G_{i,j}$
- Chaque équipe joue au plus 2 fois dans une période : $\forall q \in [1..p], \; \forall i \in [1..w-2], \; \forall j \in [i+1..w-1], \; \forall k \in [j+1..w], \emptyset = G_{i,p} \cap G_{j,p} \cap G_{k,p}$
- Chaque équipe rencontre toutes les autres équipes : puisque chaque équipe joue chaque semaine, il est suffisant de forcer que chaque paire de matches partage au plus une équipe : $\forall i \in [1..w-1], \; \forall j \in [i+1..w], \; \forall p_1, p_2 \in [1..p], G_{i,p_1,j,p_2} :: \perp, \mathcal{U}, 0, 1 \wedge G_{i,p_1,j,p_2} = G_{i,p_1} \cup G_{j,p_2}$
- Symétrie 1. La première semaine est simplement remplie : l'équipe 1 joue contre la 2 dans la 1ère période, la 3 contre la 4 dans la 2ème, etc $\forall i \in [1..n], \;\; i \in G_{1,((i-1)div2)+1}$
- Symétrie 2. La première équipe est placée pendant p semaines (en diagonale, en partant de la seconde semaine) : $\forall i \in [1..p], \;\; 1 \in G_{i+1,i}$

5.2 Pré-traitement SAT

L'utilisation d'un pré-traitement est recommandé pour minimiser la taille des instances CNF. Il est aussi connu que les instances réduites ne sont pas forcément plus simples à résoudre. En effet, il est possible que des solutions facilement atteignables soient retirées par le pré-traitement et que seules les solutions plus difficiles à atteindre soient conservées. Nous utilisons SatElite [9] comme "CNF minimizer". Nous pouvons l'utiliser de manière complète (CM_{Sat}) avec subsomption, symétries, ... ou uniquement pour appliquer la propagation unitaire comme pré-traitement (UP_{Sat}). Les comparaisons sont réalisées pour les 3 problèmes présentés précédemment et les résultats apparaissent dans le tableau 2. Le modèle $unrefined$ (i.e., sans pré-traitement SAT) est obtenu après l'unique application de l'encodage $\Leftrightarrow_{enc}$. Le modèle réduit (avec $\Rightarrow_{red}$) est également calculé pour chaque instance. Le suffixe "_R" est ajouté au nom de l'instance réduite.

Le tableau 2 montre que les instances non-raffinées sont résolues plus facilement que les instances ayant subit un pré-traitement SAT.

Pour les petites instances (toutes celles du Sudoku, et les premières du SGP et STS), les règles de réduction n'améliorent pas les résultats : en effet, elles nécessitent trop de temps par rapport au gain obtenu par la suite avec Minisat. Pour toutes les instances SGP avec le modèle "implication", et les instances STS (exceptée la première), l'application des règles de réduction améliore à la fois la taille et le temps de résolution.

5.3 Résultats pour de grandes instances

La section précèdente a montré que le problème du Sudoku est très simple à résoudre. Nous nous concentrons maintenant sur des problèmes plus complexes : le SGP et STS.

Dans [18], seule la modélisation et l'encodage sans réduction a été étudié. Les instances générées étaient plus petites (variables et clauses) et résolues plus rapidement que les instances générées directement. Le tableau 3 montre que le modèle à base d'implication associé aux réductions permet une résolution plus rapide. Des solutions sont calculées pour les instances SGP 8_4_8 et 9_4_9, et à notre connaissance, c'est le premier modèle SAT qui permet de résoudre ces instances. Il est à noter que les instances 8_4_11, 8_4_12 et 9_4_12 n'ont pas de solution ; dans ce cas, le solveur doit prouver l'insatisfiabilité des instances.

Le tableau 4 montre l'impact important qu'a la réduction sur notre encodage SAT. En effet, sans réduction les instances STS sont résolues jusqu'à la taille 12 alors qu'avec réduction, des solutions sont trouvées jusqu'à la taille 66. Comme précisé en section 5.2, un procédé de réduction du modèle pourrait compliquer la résolution (par exemple en retirant solution symétriques). Nous pouvons observer que ce n'est pas le cas ici. Nos règles de réduction simplifient la recherche : elles gardent la structure du problème, sans retirer de solutions (même symétriques), et réduisent l'espace de recherche. La taille des instances n'est donc pas l'unique critère : la structure du problème et la taille de l'espace de recherche est également primordial. Par exemple, des instances plus hautes mais réduites, peuvent être plus grandes (nombres de variables et clauses) que des instance plus simples non réduites ; mais les instances réduites (bien que plus grandes en variables et clauses) peuvent être résolues alors que les non-réduites ne le sont pas. Par exemple, l'instance 14 non réduite du STS est plus petite (variables, clauses) que l'instance 20 réduite ; mais l'instance 20 réduite est résolue alors que l'instance 14 non-réduite n'y est pas. Pour les instances 68 à 72, Minisat stoppe immédiatement la résolution, le nombre de clauses et de variables étant trop grand. Des solveurs spécifiques [15] ont été réalisés pour résoudre ce problème. Les problèmes eux-mêmes sont sur-contraints par l'ajout de contraintes n'apparaissant pas dans le problème initial. Ainsi, l'instance résolue la plus grande est 70, mais la moitié des instances n'ont plus de solutions (dû aux sur-contraintes).

6 Conclusion

Nous avons présenté une technique pour l'encodage des contraintes ensembliste CSP en SAT : le processus de modélisation est effectué grâce à des contraintes ensemblistes expressives ; ces contraintes sont ensuite réduites par nos règles $\Rightarrow_{red}$ avant d'être converties automatiquement ($\Leftrightarrow_{enc}$) en variables et clauses SAT. Nous avons illustré notre approche par divers problèmes (Sudoku, le problème du Social Golfer, et le problème du Sports Tournament Scheduling) et avons obtenus de bons résultats en appliquant nos règles de réduction et d'encodage. Les avantages de notre technique sont les suivants :

- la modélisation est simple, expressive et lisible. De plus, les modèles sont indépendants des solveurs SAT ou CSP ;
- la technique est beaucoup moins sensible aux erreurs qu'un encodage direct en SAT ;
- les instances SAT générées automatiquement sont plus petites en terme de nombres de variables et clauses ;
- finalement, l'ajout d'une phase de réduction permet de réduire le temps total de résolution (réduction+encodage+résolution) ;
- les instances SAT générées semblent aussi être

TABLE 2 – Efficiency of SAT pre-processes

	Instances	$\Rightarrow_{red}$	Model characteristics						Encoding time			Resolving time					
			Unrefined		UP_{Sat}		CM_{Sat}		$\Leftrightarrow_{enc}$	UP_{Sat}	CM_{Sat}	Unrefined		UP_{Sat}		CM_{Sat}	
		sec.	#cl	#var	#cl	#var	#cl	#var	sec.	sec.	sec.	sec.	sum	sec.	sum	sec.	sum
Sudoku	empty	-	19 728	4 122	13 527	2 997	12 474	2 430	0.04	0.02	0.17	0.47	0.51	0.18	**0.24**	0.14	0.35
	empty_R	0.03	14 031	3 339	13 527	2 997	12 474	2 430	0.03	0.01	0.16	0.18	**0.25**	0.18	0.26	0.14	0.37
	523262	-	19 757	4 122	8 684	1 924	8 008	1 560	0.04	0.03	0.11	0.14	**0.18**	0.16	0.22	0.21	0.36
	523262_R	0.12	7 260	1 936	6 814	1 594	5 901	1 223	0.02	0.00	0.06	0.03	**0.20**	0.08	0.22	0.09	0.30
	527796	-	19 756	4 122	8 851	1 961	8 162	1 590	0.04	0.03	0.11	0.08	0.12	0.04	**0.10**	0.06	0.20
	527796_R	0.20	6 648	1 828	6 202	1 486	4 998	1 043	0.02	0.00	0.06	0.01	**0.24**	0.02	**0.24**	0.01	0.30
	53151	-	19 754	4 122	9 185	2 035	8 470	1 650	0.04	0.03	0.11	0.12	**0.16**	0.11	0.18	0.13	0.28
	53151_R	0.16	7 461	1 994	7 009	1 652	5 908	1 207	0.02	0.01	0.08	0.05	**0.23**	0.05	0.24	0.05	0.30
	55112	-	19 757	4 122	8 684	1 924	8 008	1 560	0.04	0.03	0.11	0.06	**0.07**	0.03	0.10	0.05	0.20
	55112_R	0.11	7 897	2 049	7 451	1 707	6 789	1 387	0.02	0.01	0.06	0.02	**0.15**	0.02	0.16	0.02	0.21
	58657	-	19 760	4 122	8 183	1 813	7 546	1 470	0.04	0.03	0.10	0.03	**0.07**	0.02	0.08	0.02	0.17
	58657_R	0.17	6 651	1 804	6 211	1 462	5 070	1 036	0.02	0.00	0.06	0.01	**0.20**	0.01	**0.20**	0.01	0.26
	128214	-	19 762	4 122	7 849	1 739	7 238	1 410	0.04	0.03	0.10	0.01	**0.05**	0.02	0.08	0.01	0.15
	128214_R	0.19	6 279	1 721	5 843	1 379	4 972	1 025	0.01	0.00	0.06	0.00	**0.20**	0.01	0.21	0.00	0.26
SGP	8_4_5_C	-	526 333	29 696	214 370	14 787	146 332	3 474	0.97	0.73	11.11	0.08	**1.05**	0.04	1.75	0.02	12.11
	8_4_5_C_R	0.92	216 751	16 361	193 582	14 060	139 919	3 109	0.43	0.18	7.72	0.05	**1.40**	0.04	1.57	0.02	9.09
	8_4_6_C	-	745 338	41 760	337 665	22 684	230 357	4 346	1.38	1.00	21.68	0.12	**1.50**	0.08	2.46	0.04	23.10
	8_4_6_C_R	1.19	334 591	24 422	305 647	21 570	220 847	3 889	0.67	0.29	15.18	0.09	**1.94**	0.07	2.22	0.04	17.08
	8_4_5_I	-	464 893	9 216	191 410	3 811	176 004	3 601	3.13	5.16	5.91	0.06	**3.19**	0.02	8.31	0.03	9.07
	8_4_5_I_R	-	193 987	5 861	170 858	3 600	168 994	3 220	1.70	0.26	0.75	0.06	**1.76**	0.02	1.98	0.03	2.48
	8_4_6_I	-	653 178	11 040	300 145	4 764	276 245	4 503	4.64	8.80	10.52	0.13	**4.77**	0.07	13.51	0.06	15.22
	8_4_6_I_R	-	297 921	7 292	269 037	4 500	266 414	4 024	2.67	0.38	1.31	0.04	**2.71**	0.05	3.09	0.05	4.03
STS	8	-	23 812	5 528	13 794	3 867	6 422	1 040	0.04	0.04	0.28	0.01	**0.05**	0.01	0.09	0.01	0.33
	8_R	0.08	16 606	4 549	13 345	3 902	2 649	426	0.03	0.02	0.26	0.00	**0.11**	0.00	0.13	0.00	0.37
	10	-	77 135	17 170	46 917	12 530	29 233	5 194	0.14	0.12	0.91	3.37	3.51	1.18	**1.44**	5.32	6.37
	10_R	0.11	58 235	14 788	45 918	12 650	13 729	2 352	0.11	0.07	0.74	0.01	**0.23**	0.01	0.30	-	0.97
	12	-	198 198	42 612	124 980	32 297	85 581	15 576	0.35	0.40	3.08	143.96	**144.31**	509.24	509.99	266.26	269.70
	12_R	0.22	148 364	35 188	112 467	29 541	40 188	6 624	0.39	0.22	2.22	0.04	**0.65**	0.03	0.86	0.01	2.84

TABLE 3 – Results for large SGP instances using cardinality and implication models with reduction rules.

Inst.	cardinality model						implication model					
	$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution		$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution	
	sec.	#cl	#var	sec.	sec.	sum	sec.	#cl	#var	sec.	sec.	sum
8_4_5	0.92	216 751	16 361	0.03	0.05	1.00	0.03	193 987	5 861	0.01	0.03	**0.07**
8_4_6	1.19	334 591	24 422	0.04	0.15	1.37	0.31	297 921	7 292	0.21	0.10	0.62
8_4_7	1.56	477 865	34 085	0.06	0.71	2.33	0.06	424 003	8 723	0.03	0.16	**0.25**
8_4_8	1.95	646 573	45 350	0.09	-	-	0.08	572 233	10 154	0.07	0.26	**0.41**
8_4_9	2.46	840 715	58 217	0.13	-	-	0.12	742 611	11 585	0.05	-	-
8_4_10	2.93	1 060 291	72 686	0.15	-	-	0.18	935 137	13 016	0.08	-	-
8_4_11	3.45	1 305 301	88 757	0.18	-	-	0.21	1 149 811	14 447	0.13	-	-
8_4_12	4.10	1 575 745	106 430	0.2	-	-	0.26	1 386 633	15 878	0.08	-	-
9_4_6	1.58	523 471	34 079	0.07	0.17	1.82	0.04	468 991	9 489	0.03	0.09	**0.16**
9_4_7	2.09	750 568	47 818	0.09	0.23	2.42	0.12	670 306	11 356	0.07	0.26	**0.45**
9_4_8	2.65	1 018 441	63 875	0.13	1.37	4.15	0.11	907 435	13 223	0.09	0.38	**0.58**
9_4_9	3.24	1 327 090	82 250	0.17	-	-	0.23	1 180 378	15 090	0.09	6.37	**6.69**
9_4_10	3.90	1 676 515	102 943	0.22	-	-	0.26	1 489 135	16 957	0.08	-	-
9_4_11	4.79	2 066 716	125 954	0.27	-	-	0.31	1 833 706	18 824	0.20	-	-
9_4_12	5.43	2 497 693	151 283	0.33	-	-	0.33	2 214 091	20 691	0.18	-	-

TABLE 4 – Résultats pour de grandes instances de STS

Inst.	Initial					reduced					
	Unrefined		$\Leftrightarrow_{enc}$	Resolution		$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution	
	#cl $\times 10^3$	#var$\times 10^3$	sec.	sec.	sum	sec.	#cl$\times 10^3$	#var$\times 10^3$	sec.	sec.	sum
8	24	5 528	0.02	0.04	**0.06**	0.08	15	4	0.02	0.02	0.12
10	77	17 170	0.01	9.30	9.31	0.11	54	13	0.01	0.04	**0.16**
12	198	43	0.03	387.85	387.88	0.22	148	35	0.02	0.14	**0.38**
14	437	91	0.04	-	-	0.41	343	78	0.03	0.22	**0.66**
16	861	175	0.09	-	-	0.66	700	154	0.05	0.45	**1.16**
18	1 569	314	0.17	-	-	0.98	1 307	281	0.14	0.89	**2.01**
20	2 676	528	0.32	-	-	1.34	2 275	479	0.24	1.69	**3.28**
22	4 331	842	0.49	-	-	1.92	3 742	773	0.41	3.43	**5.76**
24	6 713	1 289	0.75	-	-	2.65	5 879	1 194	0.66	4.82	**8.13**
26	10 041	1 905	1.15	-	-	3.71	8 890	1 779	0.98	9.24	**13.93**
28	14 567	2 735	1.62	-	-	5.34	13 020	2 569	1.37	12.37	**19.08**
30	20 591	3 827	2.26	-	-	6.13	18 551	3 616	2.03	20.20	**28.36**
32	28 453	5 241	3.18	-	-	8.11	25 813	4 974	2.92	31.93	**42.96**
34	38 581	7 059	4.00	-	-	10.62	35 203	6 719	3.89	36.17	**50.68**
36	51 396	9 343	5.54	-	-	12.67	47 151	8 925	5.04	83.38	**101.09**
38	67 397	12 178	7.26	-	-	14.96	62 129	11 668	6.64	124.25	**145.85**
40	87 144	15 655	9.29	-	-	12.78	80 678	15 041	8.44	114.74	**135.96**
50	266 070	46 637	27.93	-	-	36.93	250 325	45 254	27.38	880.50	**944.81**
52	323 676	56 497	33.62	-	-	46.99	305 266	54 901	31.79	1240.79	**1319.57**
54	390 835	67 948	40.61	-	-	69.64	369 437	66 116	39.13	1492.06	**1600.83**
56	468 687	81 175	48.36	-	-	60.58	443 952	79 083	46.24	1803.75	**1910.57**
58	558 459	96 375	67.49	-	-	60.19	530 012	93 996	55.61	2362.21	**2478.01**
60	661 467	113 760	70.66	-	-	78.28	628 906	111 067	64.76	2952.74	**3095.78**
62	779 123	133 556	79.58	-	-	87.63	742 017	130 519	77.08	3367.36	**3532.07**
64	912 933	156 005	95.06	-	-	100.48	870 823	152 592	92.14	4903.03	**5095.65**
66	1 064 779	181 500	109.51	-	-	115.36	1 017 067	177 625	105.45	5247.73	**5468.54**
68	1 236 149	210 203	126.70	-	-	135.61	1 182 405	205 875	122.12	-	-
70	1 428 864	242 406	145.71	-	-	154.44	1 368 535	237 589	139.81	-	-
72	1 644 854	278 418	172.92	-	-	182.35	1 577 355	273 071	167.62	-	-

bien adaptées à Minisat.

Nous prévoyons d'utiliser de combiner nos contraintes ensemblistes avec l'arithmétique sur les domaines finis, et de fournir également l'encodage SAT de ces nouvelles contraintes. A ces fins, nous devrons ajouter de nouvelles contraintes et compléter nos ensembles de règles $\Leftrightarrow_{enc}$ et $\Rightarrow_{red}$. Nous avons notre propre format pour nos modèles (XML-like) mais nous prévoyons de passer au standard XCSP3 [7].

Références

[1] CHOCO. http ://www.emn.fr/z-info/choco-solver/.

[2] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[3] Francisco Azevedo. Cardinal : A finite sets constraint solver. *Constraints*, 12(1) :93–129, 2007.

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

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

[6] 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.

[7] Frédéric Boussemart, Christophe Lecoutre, Cédric Piette, and Vincent Perradin. XCSP3 an integrated format for benchmarking combinatorial constrained problems. http ://www.xcsp.org/.

[8] Francisco de Moura e Castro Ascensão de Azevedo. *Constraint Solving over Multi-valued Logics - Application to Digital Circuits*. PhD thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2002.

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

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

[11] T. Früwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1979.

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

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

[15] Jean-Philippe Hamiez and Jin-Kao Hao. A note on a sports league scheduling problem. *CoRR*, abs/1410.2721, 2014.

[16] Warwick Harvey. CSPLib problem 010 : Social golfers problem. http ://www.csplib.org/Problems/prob010.

[17] F. Lardeux, E. Monfroy, F. Saubion, B. Crawford, and C. Castro. Sat encoding and csp reduction for interconnected alldiff constraints. In *Proc. of MICAI 2009*, pages 360–371, 2009.

[18] Frédéric Lardeux, Eric Monfroy, Broderick Crawford, and Ricardo Soto. Set constraint model and automated encoding into SAT : application to the social golfer problem. *Annals OR*, 235(1) :423–452, 2015.

[19] B. Legeard and E. Legros. Short overview of the clps system. In *Proc. of PLILP'91*, volume 528, pages 431–433. Springer, 1991.

[20] A. Mackworth. *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley, 1987.

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

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

[23] Toby Walsh. CSPLib problem 026 : Sports tournament scheduling. http ://www.csplib.org/Problems/prob026.