Projet de programmation par contraintes avancée

The Social Golfer Problem

Sullivan Bitho - Valentin Deguil - Théo Hermegil - Nicolas Mathey

Université de Nantes — UFR Sciences et Techniques
Master informatique parcours "optimisation en recherche opérationnelle" (ORO)
Année académique 2022-2023

Nantes Université

Avril 2022

Table des matières

Introduction

Le Social Golfer Problem (SGP) est un problème de mathématiques combinatoire $\mathcal{NP}$-dur. g golfeurs jouent toutes les semaines pendant s semaines répartis en k groupes de p golfeurs ($g = k.p$). Le but est de trouver une répartition des joueurs dans les groupes, tel que chaque joueur joue chaque semaine, sur un maximum de semaine, tout en respectant deux contraintes :

- Contrainte d'affectation par semaine :
  Chaque joueur doit jouer dans exactement un groupe chaque semaine
- Contrainte de sociabilité :
  Si $j1$ et $j2$ jouent dans un même groupe en semaine $s1$, alors ils ne peuvent pas se retrouver dans un même groupe en semaine $s2$.

Ce problème peut être modélisé sous la forme d'un problème d'optimisation, où l'objectif est de trouver une affectation des golfeurs qui maximise le nombre de semaines jouées. Dans cette exercice, nous allons étudier le problème de décision associé.
Le SGP a été étudié de manière approfondie et a de nombreuses applications pratiques, notamment pour la planification et l'allocation de ressources.

Dans ce projet, nous avons dans un premier temps étudié comment un modèle ensembliste SET peut différer d'un modèle SAT sur la résolution du SGP. Dans un second temps, nous avons construit notre propre solveur ensembliste, afin d'explorer différentes stratégies de résolution pour un problème entrée quelconque. En particulier, les problèmes du SGP, du Sudoku et des n Reines.
Un travail particulier a été mené concernant l'impact des cassages de symétries sur les résolutions du SGP.

Inventaire des fichiers

Vous trouverez dans cette archive :

`Readme.txt` : contient la liste des commandes possibles

Docs > : divers documents nous ayant aidé pour le projet

Solveurs_Pro > : Etude des modeles SAT et SET. Les solveurs utilisés sont des solveurs professionnels (Gecode et Glucose3).

- `Modeles_MZN_SET` *modèles SET*
- `Modeles_MZN_SAT` *modèles SAT*
- `res` *résultats des experimentations*
- `src` *construction des clauses SAT*
  - `graphs` *contient les resultats graphiques des exp (plots)*
  - `SET` *contient les resultats relatifs aux modèles SET*
  - `SAT` *contient les resultats relatifs aux modèles SAT*
- `Exp_SET.py` *experimentations SET*
- `Exp_SAT.py` *experimentations SAT*

Solveurs_Vanilla > : dossiers contenant l'implémentation de notre solveur personnel.

- `src` *contient le code source du solveur*
- `modeles` *modèles*
- `res` *résultats des experimentations*
  - `graphs` *contient les resultats graphiques des exp (plots)*

- `Exp_SGP.jl` *experimentations solveur perso SGP*
- `Exp_Sudoku.jl` *experimentations solveur perso Sudoku*
- `Exp_N_Reines.jl` *experimentations solveur perso N Reines*

Acronymes

Vous trouverez dans ce rapport les acronymes suivants :

- *AL*, *AM*, *sbs* *bs*, *a*, *o* : AtLeast, AtMost, Sans Brise Symmétrie, Brise Symmétrie, Assumption, Ordre.

Ce document est divisé en deux parties. La première partie présente l'étude des modèles SET et SAT et l'impact des cassages de symétries sur leur résolution. La deuxième partie présente l'implémentation de notre solveur personnel et le compare aux solveurs professionnels.

Protocole d'expérimentation :

Toutes les expérimentations présentées dans ce document ont été effectuées dans les conditions suivantes :

- Environnement matériel :
  - **Mémoire** : 8GB (DDR4)
  - **CPU** : Intel(R) Core(TM) i3-6300 CPU @ 3.80GHz 3.79 GHz
  - **SSD** Kingston 112Go
- Solveurs :
  - **Gecode** pour les modèles SET
  - **Glucose3** pour les modèles SAT
- Budget :
  - **200 secondes**.
    Ce budget est alloué pour l'encodage d'un modèle et sa résolution.
    Pour les modèles SET, le timeout a été ajouté via l'option `timeout=BUDGET` de la fonction `solve`.
    Pour les modèles SAT, le timeout sur la création des clauses a été instauré via la fonction `alarm` du module `signal` de python. Le timeout sur la résolution par Glucose3 est effectué via l'option `expect_interrupt=True` de la fonction `solve_limited`

1 Modélisation

1.1 Modèle SET

Le SGP peut être modélisé via le modèle ensembliste suivant :
— Soit S l'ensemble des semaines de la solution.
— Soit G l'ensemble des joueurs.
— Soit K l'ensemble des groupes de joueurs dans une semaine.
— Soit p le nombre de joueurs par groupe.
— Soit k_i le ième groupe d'une semaine.

$$|k_i| = p, \quad (\forall s \in S)(\forall k_i \in K) \tag{1}$$
$$k_i \cap k_j = 0, \quad (\forall s \in S), (\forall k_i, k_j \in K | k_i \neq k_j) \tag{2}$$
$$g_i \cap g_j \leq 1, \quad (\forall s_1, s_2 \in S)(\forall g_i \in s_1, g_j \in s_2) \tag{3}$$

La première contrainte assure que tous les joueurs sont répartis en groupes de p joueurs chaque semaine. La deuxième contrainte assure qu'aucun joueur ne joue dans deux groupes différents. La troisième contrainte est la contrainte de sociabilité : elle assure qu'aucun joueur ne peut jouer une seconde fois avec un même joueur.

1.2 Modèle SAT

Le SGP peut être modélisé via le modèle SAT suivant :

— Soit S l'ensemble des semaines de la solution.
— Soit G l'ensemble des joueurs.
— Soit K l'ensemble des groupes de joueurs dans une semaine.
— Soit g le nombre de golfeurs
— Soit p le nombre de joueurs par groupe
— Soit x_{sjq} le littéral qui vaut 1 si en semaine s, le joueur g joue en position q.
— Soit `lit(g,q,sem,grp,j,pos)` la fonction qui retrouve le numéro du littéral associé à la semaine sem, groupe grp, joueur j, position pos.

$$\bigwedge_{s \in S} \bigvee_{j \in G} \bigvee_{q \in G} x_{sjq} \tag{4}$$
$$\bigwedge_{s \in S} \bigwedge_{i \in G} \bigwedge_{j \in G} \neg x_{si} \vee \neg x_{sjq} \tag{5}$$

$$\bigwedge_{j1,j2} \bigwedge_{s1} \bigwedge_{g1} \bigwedge_{s1p1,s1p2} \bigwedge_{s2} \bigwedge_{g2} \bigwedge_{s2p1,s2p2} \Big(\neg \text{lit}(g, p, s1, g1, j1, s1p1) \tag{6}$$
$$\vee \neg \text{lit}(g, p, s1, g1, j2, s1p2) \tag{7}$$
$$\vee \neg \text{lit}(g, p, s2, g2, j1, s2p1) \tag{8}$$
$$\vee \neg \text{lit}(g, p, s2, g2, j2, s2p2) \Big) \tag{9}$$

Les clauses (4) et (5) sont les clauses *AtLeast* et *AtMost*. Elles assurent qu'exactement un joueur doit jouer à une position d'un groupe dans chaque semaine. Les clauses (6) assurent la sociabilité : aucun joueur j1 ayant joué avec un joueur j2 en semaine s1 ne peut se retrouver à jouer contre lui en semaine s2, quelque soit s1, s2, j1, j2.

1.3 Comparaison des modèles SET et SAT

Afin de comparer l'efficacité des modèles SET et SAT sur la résolution du SGP, nous avons choisi les instances présentes dans le papier Cardinal :
(2,5,4),(2,6,4),(2,7,4),(2,8,5),(3,5,4),(3,6,4),(3,7,4),(4,5,4),(4,6,5),(4,7,4),
(4,9,4),(5,4,3),(5,5,4),(5,7,4),(5,8,3),(6,4,3),(6,5,3),(6,6,3),(7,5,3),(7,5,5)

Les instances (5,4,3), (6,4,3) et (7,5,5) sont UNSAT.

Vous trouverez les résultats détaillés des experimentations en Annexe, et les plots présentés ci-après au format SVG dans `Solveurs_Pro/res/graph/Save`

1.3.1 Experimentations SET vs SAT

Nous considérons la résolution en deux temps :
— le temps de génération du modèle
— le temps de résolution du solveur

Le modèle SET est généré très rapidement, alors que la génération des clauses pour le modèle SAT, en particulier les clauses de sociabilité (7 boucles for imbriquées) coûtent cher (cf Annexe B). Bien que la résolution SAT soit extrêmement rapide une fois les clauses ajoutées, nous nous attendions à ce que le modèle SET domine significativement le modèle SAT. Voici les résultats des temps totaux obtenus par les modèles SAT et SET (sans cassage de symétrie) :

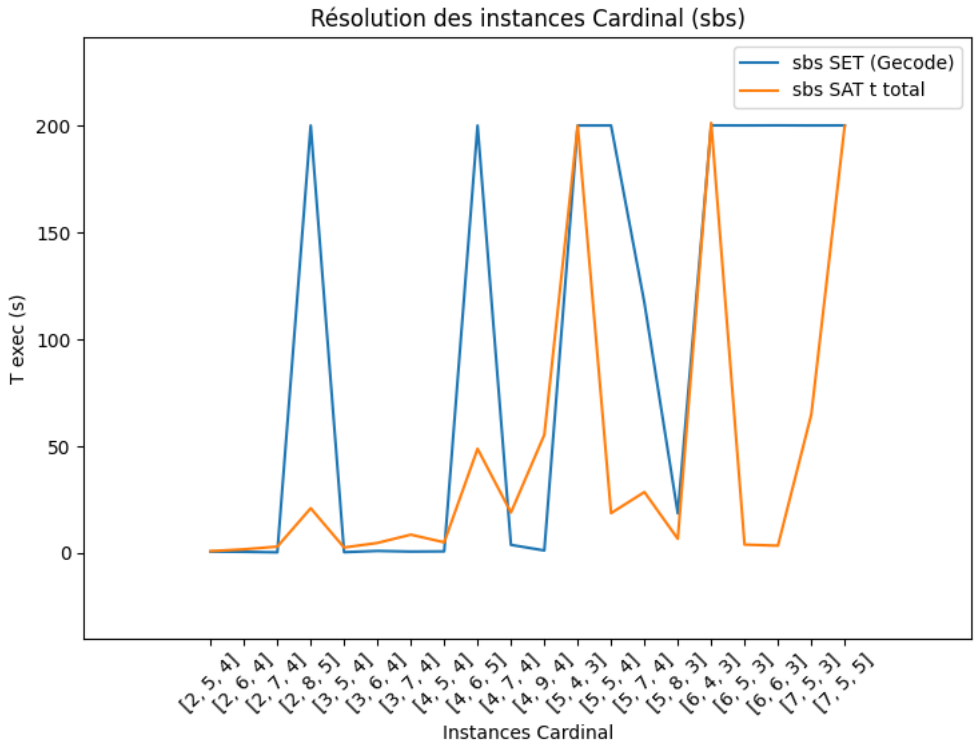

FIGURE 1 – Temps de résolution du modèle SET (Gecode) vs modèle SAT (Glucose3) sbs

Nous remarquons que notre modèle SAT ne "s'en sort pas si mal". Notre modèle SAT domine le modèle SET sur 8 instances sur 20. Pour les instances (2,8,5), (4,6,5), (5,5,4), (5,7,4) et (7,5,3) on observe un "retour sur investissement" du temps passé à générer les clauses. Pour les instances (5,8,3), (6,5,3), (6,6,3) le modèle SAT est non seulement rapide à résoudre mais

également plus rapide pour générer ses clauses.
Le modèle SET semble dominer sur les premières instances ”faciles”, sauf les (2,8,5) et (4,6,5). Aucune caractéristique particulière ne semble se dégager de ces 2 instances comparées aux autres.
Les instances UNSAT (5,4,3), (6,4,3) et (7, 5, 5) sont difficiles à résoudre pour les deux modèles.

Bien sûr, si on compare à présent les temps de résolution des solveurs seuls, SAT a un fort avantage :

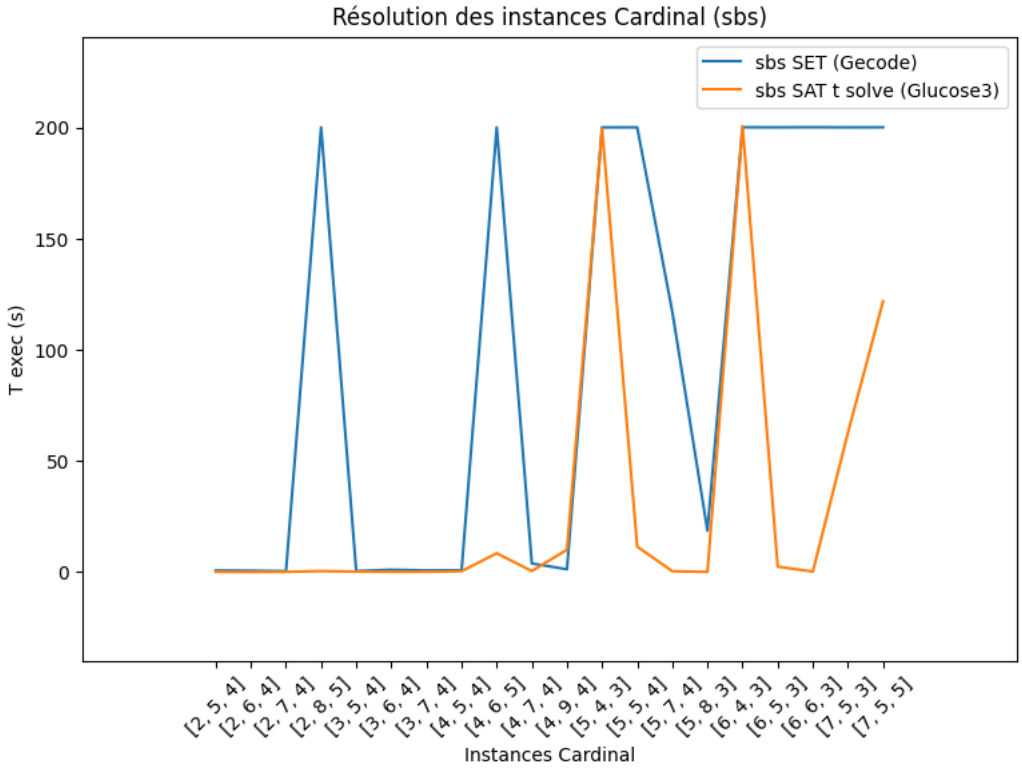

FIGURE 2 – Temps de résolution du modèle SET (Gecode) vs modèle SAT (Glucose3) sbs par les solveurs seuls

La résolution du modèle SAT seul domine la résolution du modèle SET seul dans 16 instances sur 20.

1.3.2 Conclusion SET vs SAT

En l'état, le temps total pris par la construction de notre modèle SET et sa résolution par Gecode semble, hormis pour quelques instances ”ovnis”, plus rapide pour résoudre le SGP que la construction de notre modèle SAT et sa résolution par Glucose3. Néanmoins, il est à noter que la résolution SAT seul (Glucose3) domine généralement la résolution SET seul (Gecode). En effet, la presque totalité du temps pris par l'approche SAT se retrouve régulièrement dans la modélisation des clauses de sociabilité. En implémentant ces clauses de manière plus efficace, l'approche SAT pourrait surpasser l'approche SET. Nous n'avons pas encore trouvé de meilleure implémentation pour les clauses de sociabilité. Sur les instances UNSAT, le modèle génère les clauses très rapidement mais le solveur consomme la totalité du budget restant.

D'une manière générale, il apparaît pour l'instant qu'il nous faille choisir entre une création rapide du modèle pour une résolution plus lente (SET), ou inversement une création plus lente

du modèle pour une résolution rapide (SAT).

D'autres expérimentations (voir instances Balayage dans le fichier `Readme.txt`) nous donnent aussi des informations sur l'impact des paramètres s, k, p : plus ils sont grands, plus les temps de résolution en "pâtissent". Mais les paramètres n'ont pas tous le même poids dans l'impact du temps de résolution. Nous les avons comparé un à un en fixant les deux autres. Voici l'ordre (déduit de notre étude) des paramètres classés selon leur impact sur les temps de résolution :

$$s < p < k$$

Le paramètre s est celui qui ralentit le plus les temps de résolution, jusqu'à ce qu'il dépasse complètement la transition de phase et que le problème devienne trivialement infaisable pour le solveur. Le paramètre p est le second paramètre qui a le plus d'impact. Enfin le paramètre k à un impact moindre sur les temps de résolution. A noter que, contrairement à s et p, k peut grandir à l'infini sans rendre le problème infaisable.

1.4 Symétries

Le SGP possède de nombreuses symétries de structure. En effet, les joueurs d'un même groupe peuvent être intervertis, de même que les groupes à l'intérieur d'une semaine, ainsi que les semaines à l'intérieur d'une même solution. Afin d'accélerer la résolution, il est important d'écarter un maximum de ces symétries en fixant ou en ordonnant une partie des variables. Dans cette section, nous étudions l'effet de casser les symétries sur les temps de résolution pris par Gecode pour le modèle SAT, et pris par Glucose3 pour le modèle SAT.

Nous avons identifié deux grands types de cassage de symétrie :

- Les brise symétries par *assertions* :
  On fixe une partie des variables afin d'avancer un maximum dans la résolution. Nous indiquerons ces différentes assertions par la lettre 'a'.
- Les brise symétries par définition d'*ordre* :
  On ordonne une partie des variables afin de réduire l'espace de recherche. Nous indiquerons ces différents ordres par la lettre 'o'.

L'acronyme 'bs' signifie brise-symétrie.

1.4.1 Cassage des symétries sur le modèle SET

Le modèle SET présente l'avantage d'être de très haut niveau, facile à implémenter y compris pour les cassages de ses symétries.

Vous trouverez en annexe les résultats détaillés des temps de résolution selon les cassages des symétries pour le modèle SET.

Brise-symétrie assertion semaine 1 (bs_a1)
Pour fixer une semaine, nous avons simplement ajouté une contrainte dans le modèle MiniZinc qui fixe le joueur en position i au joueur d'indice i.

Brise-symétrie assertion du premier joueur des p premiers groupes (bs_a2)
Cette bs consiste simplement à fixer le premier joueur des p premiers groupes de chaque semaine à son numéro de groupe.

Brise-symétrie inter-groupes (*bs_o*1)
Afin d'éviter les symétries inter-groupes, nous avons ordonné les groupes selon leur joueur d'indice minimal croissant. Par exemple si la semaine $s1$ est composée des groupes [{8,9,7} {2,1,3} {6,5,4}], l'ordre imposé fixera la semaine à [{2,1,3} {6,5,4} {8,9,7}]

Brise-symétrie inter-semaines (*bs_o*2)
Pour briser les symétries entre les semaines, nous avons ordonné les semaines selon le joueur de second indice minimal de leur premier groupe. Par exemple si la semaine $s1$ est [{7,4,1} {2,5,8} {9,3,6}] et que la semaine $s2$ est [{2,1,3} {6,5,4} {8,9,7}], 4 est le joueur de second indice minimal pour le premier groupe de $s1$, et 2 est le joueur de second indice minimal pour le premier groupe de $s2$. Comme $2 < 4$, alors $s2$ apparaîtra avant $s1$ dans la liste des semaines de la solution.

1.4.2 Cassage des symétries sur le modèle SAT

Le modèle SAT est de très bas niveau, et se résout rapidement. L'implémentation des cassages de symétries demande plus de réflexion et d'efforts d'implémentation. De plus, le nombre de clauses générées par les cassages de symétries explose de manière exponentielle en la taille de l'instance. Ces générations de clauses ralentissent le temps d'exécution total.

Veuillez trouver en annexe le détail des temps de résolution selon les cassages des symétries pour le modèle SAT.

Brise-symétrie assertion semaine 1 (*bs_a*1)
Pour fixer une semaine, nous fixons chaque littéral correspondant à la $i$ème position du joueur i. Par exemple, pour fixer le joueur 4 de la semaine 1 d'une instance $s = 2, k = 3, p = 3$, il nous faut fixer son littéral associé $x31$ $((4-1) * p * k + 4 = 31)$ à Vrai. Pour fixer les littéraux à Vrai, nous utilisons l'option *assumptions*=[litterals] de la fonction solve() de Glucose3.

Brise-symétrie assertion premiers joueurs des p premiers groupes (*bs_a*2)
Pour fixer le premier joueur des p premiers groupe, nous calculons l'indice de son littéral associé comme précédemment et le fixons à Vrai.

Brise-symétrie intra-groupe (*bs_o*0)
Les joueurs étant encodés comme des entiers, nous pouvons briser les symétries à l'intérieur d'un groupe en imposant un ordre (par exemple croissant) entre les joueurs. Ainsi, si le groupe $g1$ est composé des joueurs [4,2,3], l'ordre imposé le fixera à [2,3,4]. Pour imposer cet ordre dans le modèle SAT, nous ajoutons des clauses de telle sorte que si j1 est en position p1 dans g1, alors aucun joueur d'indice plus petit ne doit apparaître dans une position postérieure dans g1. Par exemple, avec l'instance $s = 2, k = 3, p = 3$, si le joueur 4 est en position 3 ($x30$=Vrai), alors on doit avoir l'expression logique suivante Vrai :

$$x28 \implies (\neg x1 \wedge \neg x2) \wedge (\neg x10 \wedge \neg x11) \wedge (\neg x19 \wedge \neg x20)$$

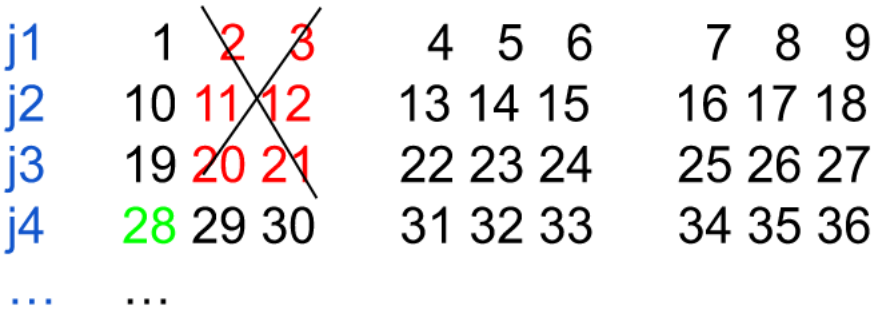

FIGURE 3 – Illustration de l'ordre intra groupe ($o0$) en SAT

Pour implémenter cette idée en SAT, nous l'avons d'abord exprimée sous la forme d'une expression logique puis convertie au format CNF en utilisant un convertisseur.
(Vous trouverez cette brise-symétrie dans la fonction `Intra_groupe_constraints` du fichier `Solveurs_Pro/src/Fonctions_SAT.py`).
A noter que cette brise-symétrie n'est pas applicable sur un modèle utilisant des variables ensemblistes comme le modèle SET ci-dessus.

Brise-symétrie inter-groupes (bs_o1)
Pour briser les symétries inter-groupes en SAT, nous générons des clauses de telle sorte que si j1 a le plus petit indice de son groupe, alors aucun joueur d'indice plus petit ne doit apparaître dans les groupes suivants. Pour exprimer cette idée sous la forme d'une expression logique, nous utilisons l'implication. A gauche de l'implication se trouve le cas où le joueur i est le minimum de son groupe (aucun joueur d'indice j plus petit que i ne se trouve dans son groupe). A droite de l'implication se trouve la négation de l'existence d'un groupe possédant un joueur d'indice plus petit que i. Illustrons le cas où 4 en position 2 du groupe 1 est le plus petit de son groupe pour l'instance $s = 2, k = 3, p = 3$:

$$x29 \wedge \neg(x1 \vee x2 \vee x10 \vee x12 \vee x19 \vee x21) \implies \neg(x4 \vee x5 \vee x6 \vee x7 \vee x8 \vee x9) \wedge \neg(x13 \vee x14 \vee x15 \vee x16 \vee x17 \vee x18) \wedge \neg(x22 \vee x23 \vee x24 \vee x25 \vee x26 \vee x27)$$

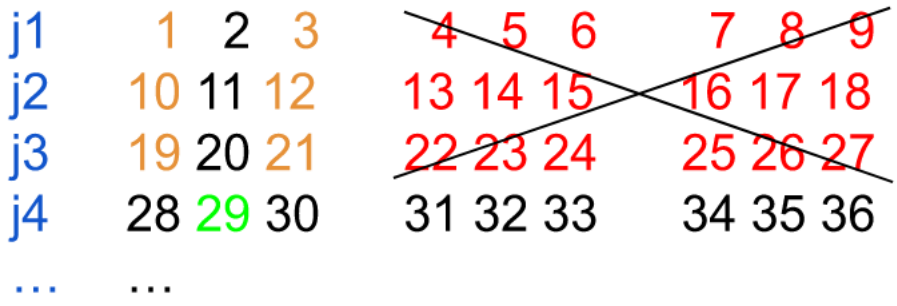

FIGURE 4 – Illustration de l'ordre inter-groupe ($o1$) en SAT

Brise-symétrie inter-semaines (bs_o2)
Pour briser les symétries inter semaines en SAT, nous générons des clauses de telle sorte que

pour chaque paire de semaine (s1,s2) avec s1 < s2, si en semaine s1, j1 est le deuxieme plus petit de son groupe, alors en semaine s2, aucun joueur d'indice plus petit ne doit apparaître dans les groupes suivants. Il s'agit là de l'expression qui nous a demandé le plus de travail. Malheureusement, l'idée d'implémentation que nous avons eu génère beaucoup trop de clauses pour être exploitable. En effet, pour exprimer le fait qu'un joueur est d'indice second minimum, on s'assure qu'exactement un joueur d'indice plus petit se trouve dans son groupe grâce à des clauses At Least et At Most. La partie gauche de l'implication est donc déjà une chaine de boucle for conséquente. A la suite de cette chaine de boucle for, on rajoute la chaine de boucle correspondant au fait qu'au maximum un joueur d'indice plus petit peut apparaître dans ce groupe les semaines suivantes, ici aussi à l'aide de clauses At Least et At Most. Le nombre de clauses générées n'est pas exploitable. Nous n'avons pas encore trouvé d'implémentation astucieuse de cette brise symétrie en SAT, mais nous sommes convaincus qu'en cherchant davantage une implémentation plus efficace peut être trouvée. (Vous trouverez notre implémentation dans la fonction `Inter_semaines_constraints` du fichier `Solveurs_Pro/src/Fonctions_SAT.py`)

1.4.3 Sur-contraintes

Les sur-contraintes, quand elles amènent à une solution faisable, permettant d'accélérer grandement les temps de résolution. Nous avons étudié l'impact de deux de ces sur-contraintes. La première qui fixe le premier groupe de la deuxième semaine. La seconde qui fixe les premiers joueurs de chaque groupe de chaque semaine. Cette seconde sur-contrainte s'avère particulièrement efficace pour résoudre certaines instances "challenge" satisfiables. L'idée de faire une première résolution du modèle sur-contraint avant de résoudre le modèle bien contraint si le modèle sur-contraint est infaisable est intéressante si l'on cherche à résoudre le problème le plus rapidement possible. Mais notre étude ici concerne davantage l'influence des symétries et la qualité des solveurs.

1.4.4 Expérimentation : comparaison des modèles selon les cassages de symétries

Vous trouverez les résultats graphiques suivants au format svg dans `Solveurs_Pro/res/graph/Save`.

Notation des brises symétries (bs)

- a1 : on fixe la premiere semaine
- a2 : on fixe le premier joueur des p premiers groupes à chaque semaine
- o1 : les groupes d'une semaine doivent être ordonnés par joueur d'indice minimum croissant
- o2 : les premiers groupes de chaque semaine doivent être ordonnés par joueur d'indice second minimum croissant
- a12 : a1 + a2 (à noté que a2 ne doit être appliqué qu'à partir de la deuxième semaine si a1 est appliqué)
- a12_o1 : a1 + a2 + o1
- etc

Un modèle regroupant un sous ensemble de bs est appelé une *variante*.

Cassage des symmétries sur le modèle SET (Gecode)
Pour le modèle SET nous nous attendions à ce que plus on casse de symétries plus les temps de résolutions accélèrent.

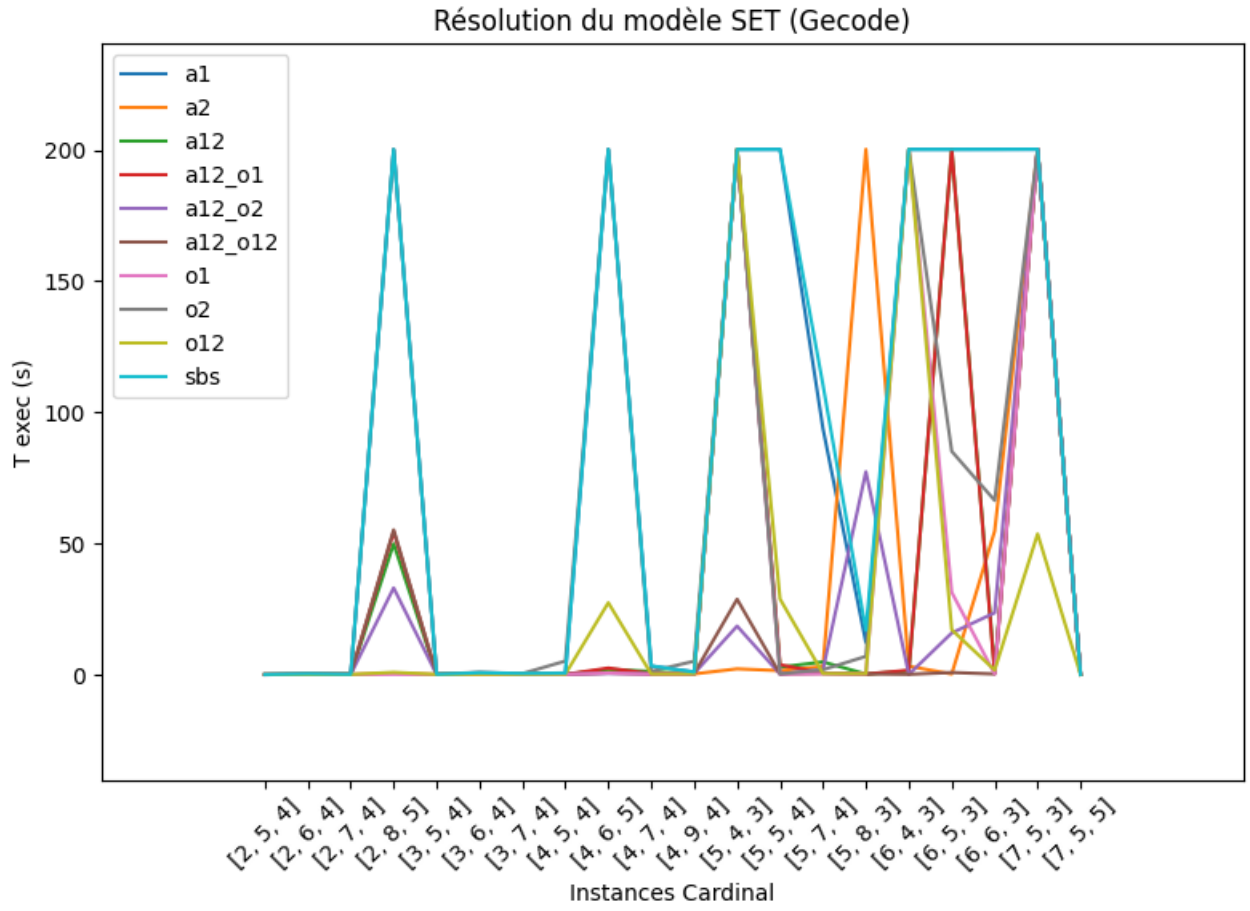

FIGURE 5 – Cassage de symétries sur les instances Cardinal modèle SET (Gecode)

On confirme expérimentalement que l'ajout de symétrie semble globalement améliorer les temps de résolution mais les pires temps sont parfois obtenus par des combinaisons de symétries. Les pires temps sont en général obtenus par la variante sbs, mais sur certaines instances il s'agit des variantes o12 ou a12_o1 ou a12_o2 qui combinent plusieurs bs. Notons qu'il est difficile de juger d'une meilleure bs qui dominerait globalement toutes les autres car d'une instance à l'autre, les tendances peuvent s'inverser.
On note tout de même que la brise symétrie a12_o12 (fixe semaine 1 et premier joueur des p premiers groupes) est constamment parmi les meilleures.

Cassage des symmétries sur le modèle SAT (Glucose3)
Le solveur Glucose3 étant très rapide même pour un nombre important de clauses, nous nous attendions à ce que notre modèle SAT soit ralenti par les bs qui ajoutent beaucoup de clauses.

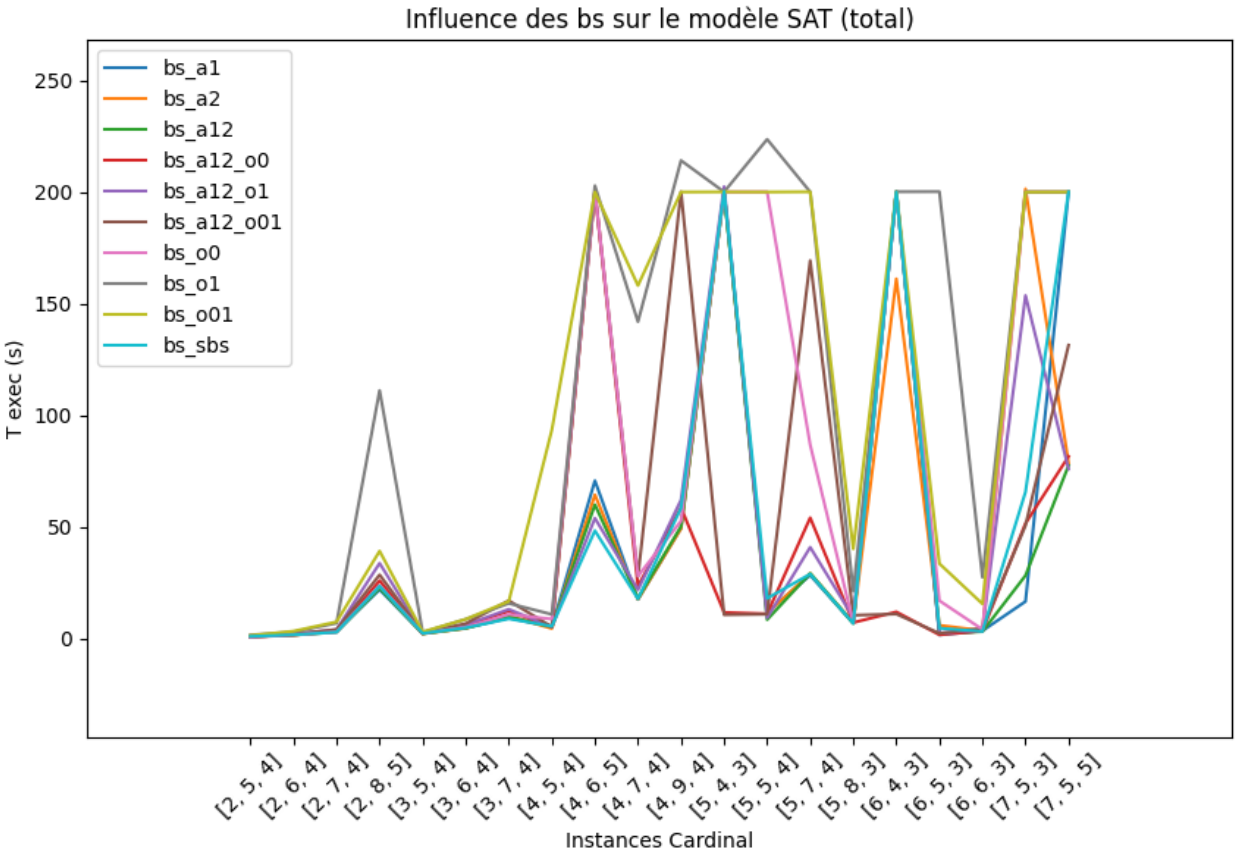

FIGURE 6 – Cassage de symétries sur les instances Cardinal modèle SAT (Glucose3)

Le cassage de symétries sur le modèle SAT donne des résultats plus homogènes que ceux du modèle SET. En particulier les variantes du même type a qui se regroupent. Nous expliquons cela en partie par la non génération de clauses supplémentaires pour ces variantes.

Il est ici plus aisé de dégager des tendances : les bs a semblent globalement plus rapides que les bs o. Pour casser les symétries par assumption, notre modèle SAT ne fait que fixer des littéraux. Pour casser les symétries d'ordre, nous créons de nouvelles clauses relativement nombreuses, en particulier pour `o2`, dont le temps doit s'ajouter au temps total de résolution.

On note que la bs_a12_o0 est systématiquement parmi les meilleures. La a12_o2 est également intéressante. La variante sbs semble être avantagée dans le modèle SAT comparé au modèle SET. Les bs o1 et o01 sont elles régulièrement dans les pires.
Enfin, il est intéressant de noter que l'ajout ou le retrait d'une bs peut significativement changer les temps d'une bs : par exemple o0 qui seule n'est pas intéressante vient aider a12 sur l'instance (6,4,3) alors que ni a12 ni o0 ne passe cette instance dans le budget imparti.

Comparaison des meilleures variantes SET vs SAT :
Bien que dominée par a1 sur certaines instances, a12_o0 présente l'avantage d'être plus robuste en finissant dans le budget imparti pour chaque instance Cardinal sauf 1. C'est donc l'instance a12_o0 que nous retenons comme "meilleure" pour l'instance SAT. Pour la variante SET nous choisissons a12_o12.

(Pour effectuer vos propres comparaisons, rendez-vous dans le fichier Readme.txt)

Voici le graphique obtenu :

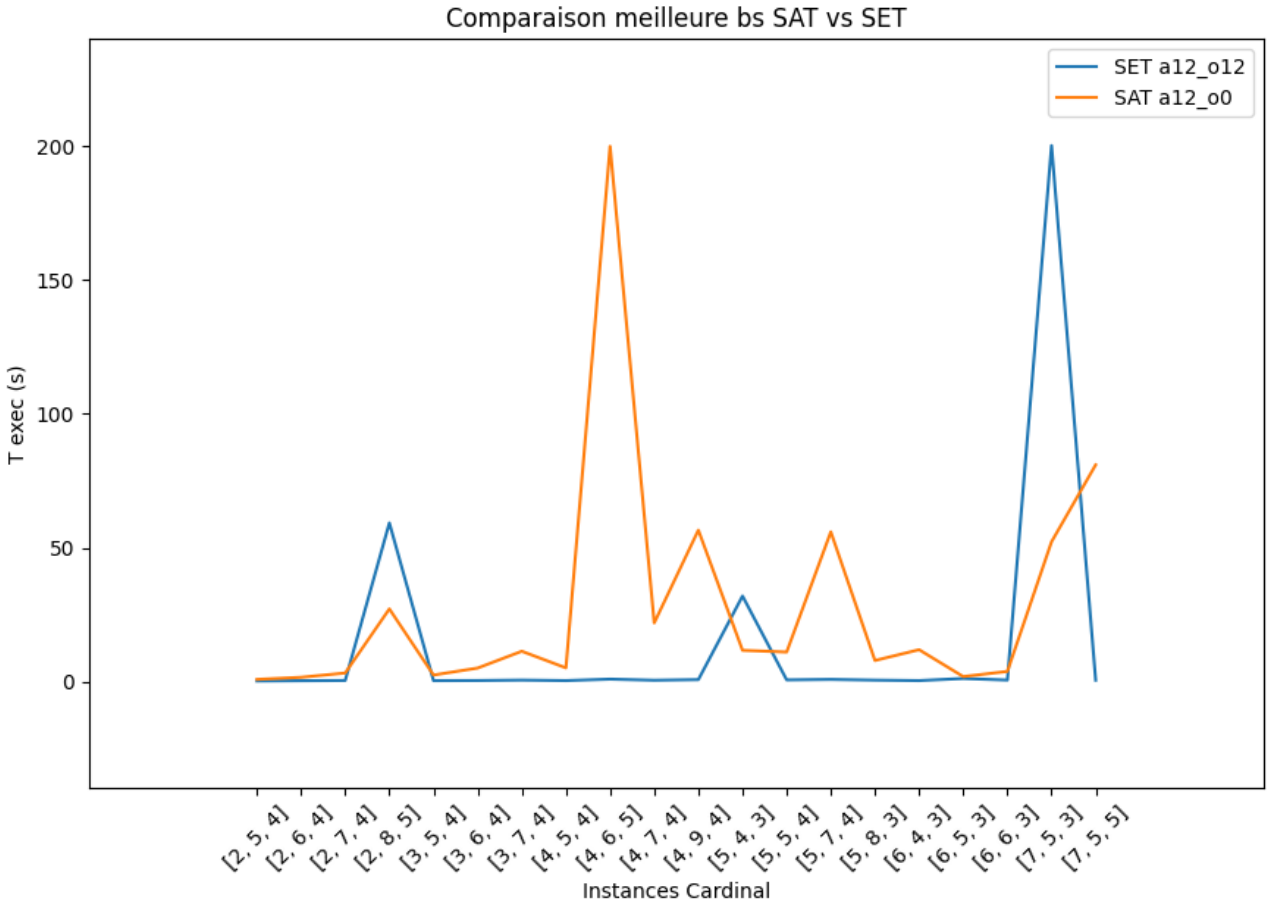

FIGURE 7 – Meilleures variantes SET vs SAT

1.4.5 Conclusion de l'étude des symétries

En cassant les symétries nous arrivons à la même conclusion que sans casser les symétries : le modèle SET semble plus approprié pour résoudre le SGP. Il est toutefois possible qu'une meilleure implémentation du modèle SAT puisse remettre en question ce résultat, ce que nous pourrons voir dans une prochaine étude.

2 Implémentation d'un solveur SET

Nous avons implémenté un solveur ensembliste que nous avons nommé "Vanilla". Vanilla prend en entrée un modèle et retourne vrai si une solution a été trouvée, faux sinon. Nous avons choisi d'utiliser un arbre de recherche type Branch & Prune pour générer de l'information (énumération) lorsqu'un point fixe a été atteint et qu'il reste des variables non fixées. Le Branch & Prune est composé d'un propagateur que nous présentons ci-après.

2.1 Propagateur

Le propagateur est un algorithme de point fixe qui propage l'information nouvellement acquise (via la propagation précédente ou l'énumération du B&P) en réveillant les variables concernées. Une fois la variable réveillée, on met à jour ses domaines min et max via un algorithme de filtrage. Voici l'algorithme de notre propagateur :

Algorithme 1 PROPAGATOR !

Entrée: un modèle
Sortie: Vrai si la propagation conserve la cohérence des variables, Faux sinon.

```
copie_variables ← copie(modele.variables)
propager ← Vrai
Tant que propager = Vrai Faire
    Si Propagate !(modele) = Faux Alors
        Retourne Faux
    Fin Si
    Si copie_variables ≠ modele.variables Alors
        copie_variables ← modele.variables
    Sinon
        propager ← Faux
    Fin Si
Fin Tant que
Si modele.variables_non_fixées = ∅ Alors
    modele.status ← "SAT"
Fin Si
Retourne Vrai
```

Algorithme 2 PROPAGATE !

Entrée: un modèle
Sortie: Vrai si la propagation conserve la cohérence des variables, Faux sinon.

```
Pour chaque variable du modèle Faire
    Pour chaque contrainte du modèle où elle est impliquée Faire
        Si Filtrage !(modele, contrainte) = Faux Alors
            Retourne Faux
        Fin Si
    Fin Pour
Fin Pour
Retourne Vrai
```

2.2 Filtrage

Afin de mettre à jour les variables en cohérence avec l'information nouvelle, nous rappelons les fonctions de filtrages associées aux contraintes auxquelles elles appartiennent. Nous avons implémenté quelques filtrages élémentaires dans le solveur comme l'intersection vide entre deux variables et la cardinalité max de l'intersection de deux variables afin d'accélérer le travail de modélisation de l'utilisateur. Nous aimerions, dans une prochaine version, ajouter beaucoup plus de fonctions de filtrage élémentaires, avec de la surcharge d'opérateurs comme $\cap$ pour l'intersection et $\cup$ pour l'union. Voici l'algorithme de notre fonction de filtrage :

Algorithme 3 FILTRAGE

Entrée: un modèle, une contrainte
Sortie: Vrai si le filtrage conserve la cohérence des variables, Faux sinon.

```
variables ← variables de la contrainte
Si contrainte.fonction_filtrage !(variables) = False Alors Retourne Faux
Fin Si
Pour chaque variable des variables Faire
    Si la mise à jour de la variable après filtrage est incohérente Alors Retourne Faux
    Fin Si
Fin PourRetourne Vrai
```

2.3 Consistence de bornes

Nous nous sommes rendu compte un peu tard qu'une composante clé nous manquait dans notre algorithme de filtrage : la consistance de borne. Elle consiste à réduire les bornes via un raisonnement logique global sur l'ensemble des variables participant à un type de contrainte. Par exemple pour le SGP, si la cardinalité de l'intersection des domaines max des variables non fixées est inférieure au nombre de joueurs par groupe p, alors on peut conclure que la branche explorée est infaisable. Admettons que nous ayons les informations suivantes sur les variables d'une semaine de l'instance s=4 k=3 p=3 :

min	{1,2,3}	{4}	{5}
max	{1,2,3}	{4,7,8,9}	{5,7,8,9}

FIGURE 8 – inconsistence de bornes

Nous n'avons ici pas assez de joueurs possibles pour les groupes g2 et g3 :
$|g2.max \cap g3.max| = 3 < (3-1)+(3-1) = 4$. Nous pouvons alors immédiatement déduire que la branche explorée est UNSAT. Cette consistance de bornes apparaît essentielle dans un algorithme de filtrage et pourrait nous permettre de grandement améliorer l'efficacité de la recherche dans l'arbre.

2.4 Branch and Prune

Notre Branch and Prune est un algorithme récursif qui explore des branches en créant une information nouvelle dans les variables du modèle. Il existe plusieurs stratégies de branchement pour l'arbre de recherche, aussi appelée heuristique de branchement dans la littérature. Nous avions initialement implémenté une stratégie naïve qui consistait à prendre la première variable non fixée dans l'ordre des id des variables. Néanmoins nous avons remarqué que d'autres heuristiques plus efficaces pourraient convenir davantage. L'heuristique *First Fail* nous est venue rapidement : on cherche à fixer les variables les plus proches d'être fixées. Par curiosité nous avons également essayé de fixer en priorité les variables dont le domaine max était le plus proche de sa cardinalité max. Et d'après nos tests, cette heuristique est souvent meilleure que la première. C'est donc celle-ci que nous avons choisi d'implémenter. (Notons ici que, normalement, un solveur doit implémenter chaque variable avec comme propriété une cardinalité min et max requise. Notre solveur actuel n'implémente qu'une unique cardinalité requise, car nous avons voulu commencer le plus simplement possible, et que les problèmes testés requiert des cardinalités max et min égales. Les cardinalités min et max requises seront ajoutées dans une

prochaine version du solveur.)

Pour l'énumération des valeurs du domaine min de la variable non fixée choisie, nous avions là aussi commencé par prendre les valeurs du domaine max selon l'ordre naturel. Puis, nous avons essayé de brancher sur la valeur qui fournit le plus d'information immédiates : la valeur qui apparaît dans un maximum de domaines min des autres variables. Il s'agit ici d'un pari, car rien ne nous garantit que les fonctions de filtrage des contraintes nous fournissent des informations venant d'elles, et il est possible que le solveur se perde sur de mauvais chemin. Néanmoins il nous semble que ces variables *informatrices* sont prometteuses.

Finalement voici l'algorithme du Branch and Prune :

Algorithme 4 BRANCH & PRUNE

Entrée: un modèle
Sortie: Vrai si le modèle est SAT, Faux sinon.

```
1: sat ← Faux
2: Si Propagator!(modele) = Faux Alors
3:     Retourne Faux
4: Fin Si
5: Si modele.status gets "SAT" Alors Retourne Vrai
6: Fin Si
7: Si modele.vars_non_fixees = ∅ Alors
8:     var ← variable non fixée avec le domaine minimal le plus petit
9:     vals_candidats ← var.dom_max \ var.dom_min
10:    Tant que vals_candidats ≠ ∅Et  sat ≠ Faux Faire
11:        vars_copie ← copie(modele.variables)
12:        modele_copie ← copie(modele)
13:        val_enum ← valeur présent dans le plus de domaines min
14:        vals_candidats ← vals_candidats \ val_enum)
15:        ajouter_dom_min!(modele_copie, var, val_enum)
16:        sat ← Branch_and_Prune!(modele_copie)
17:        Si sat = False Alors
18:            vals_candidats ← vals_candidats \ val_enum)
19:        Sinon
20:            modele ← = copie(modele_copie.variables)
21:        Fin Si
22:    Fin Tant que
23: Fin Si
24: Retourne sat
```

2.5 Expérimentations et analyses

Le solveur fournit en archive à été testé sur le SGP et sur le Sudoku. Ci-dessous est présenté nos tests sur le SGP. Vous trouverez quelques capture d'écran de résolution du Sudoku en Annexe C.2 Pour reproduire les expérimentations ci-dessous, veuillez vous référer au fichier Readme.txt présent à la racine de l'archive.

2.5.1 Analyse du Solveur SET sur le SGP

Les instances choisies pour l'expérience présentée restent celles de Cardinal. Le détail des temps de résolution sont présenté en Annexe C.1.

Vous trouverez les résultats graphiques suivants au format SVG dans `Solveur_Vanilla/res/graph/Save`.

Afin d'avoir des comparaisons pertinentes avec les résultats des instances Cardinal, nous avons systématiquement retiré les "pré-tests" qui évaluent si l'instance est trivialement infaisable (de par sa structure) avant le lancement du solveur. Le but étant de comparer le temps que met le solveur a trouver par lui-même que l'instance est infaisable (cf fonction `Solve` dans `Solveurs_Pro/Exp_SAT.py`).

Voici les temps obtenu de notre solveur (sans cassage de symmétrie) :

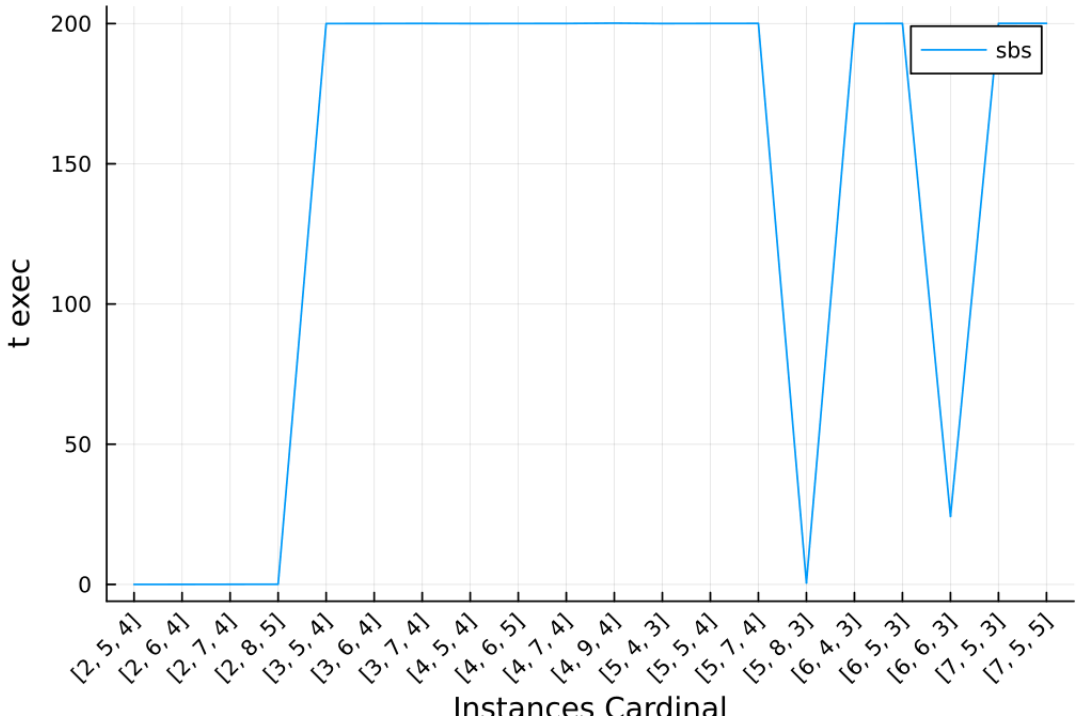

FIGURE 9 – Temps de résolution des instances Cardinal du modèle SGP SET sbs (Vanilla)

Sur le SGP, notre Solveur semble soit très rapide soit très lent selon les instances. Nous pensons qu'en améliorant le filtrage avec, entre autre, la consistance de borne, ces temps peuvent être significativement améliorés. Vanilla est rapide sur les instances "faciles" (celles résolu rapidement par les solveurs professionnels). Pour les autres instances, notre hypothèse est que Vanilla a encore du mal avec les instances demandant beaucoup d'exploration par le B&P.

2.5.2 Analyse du Solveur SET sur le SGP avec cassage de symmétries

Comme pour le modèle SET résolu par Gecode, nous nous attendions à ce que plus on casse de symétries, plus les temps de résolution s'améliorent.

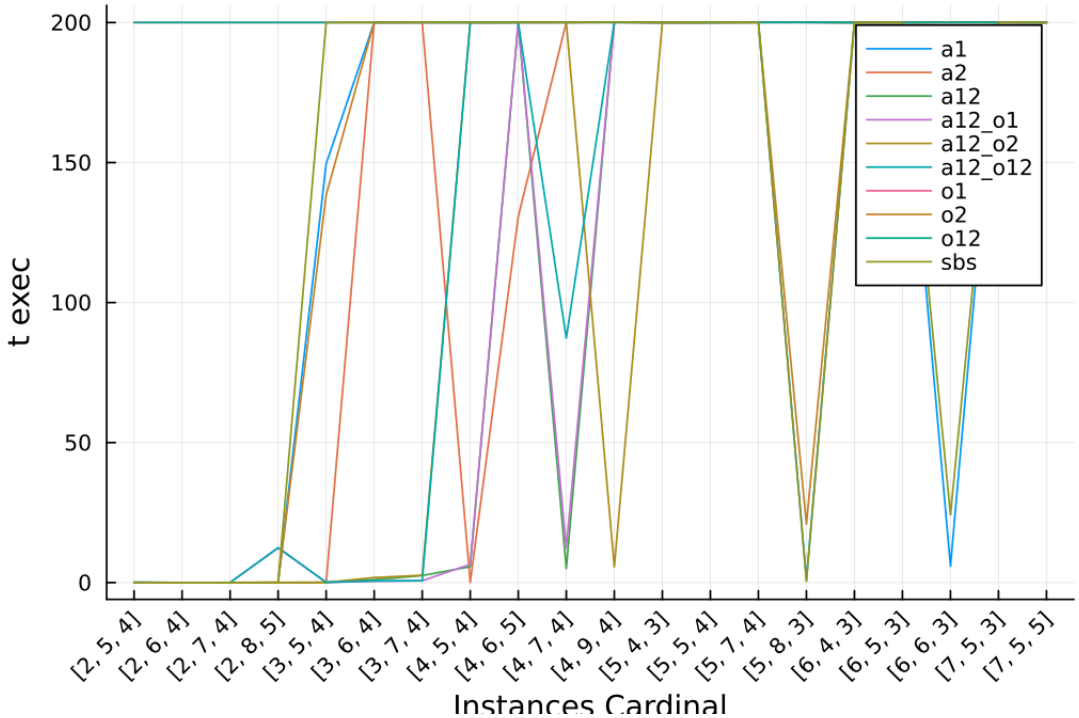

FIGURE 10 – Cassage de symétries sur les instances Cardinal du modèle SGP SET (Vanilla)

Nous confirmons cette hypothèse expérimentalement : en particulier a12_o1, a12_o2 démontre de très bonne performances mais sur des instances différentes. On remarque finalement que presque toutes les variantes ont un sous ensemble d'instance où elles performent mieux que les autres. Notons que la sbs est la plus rapide sur l'instance (5,8,3).

2.5.3 Comparaisons des solveurs Gecode, Glucose3 et Vanilla

Nous choisissons de comparer la variante bs_a12_o1 avec les autres. Cette comparaison n'apporte pas d'information général sur la qualité du solveur mais permet un début de positionnement.

Voici le graphique de comparaison :

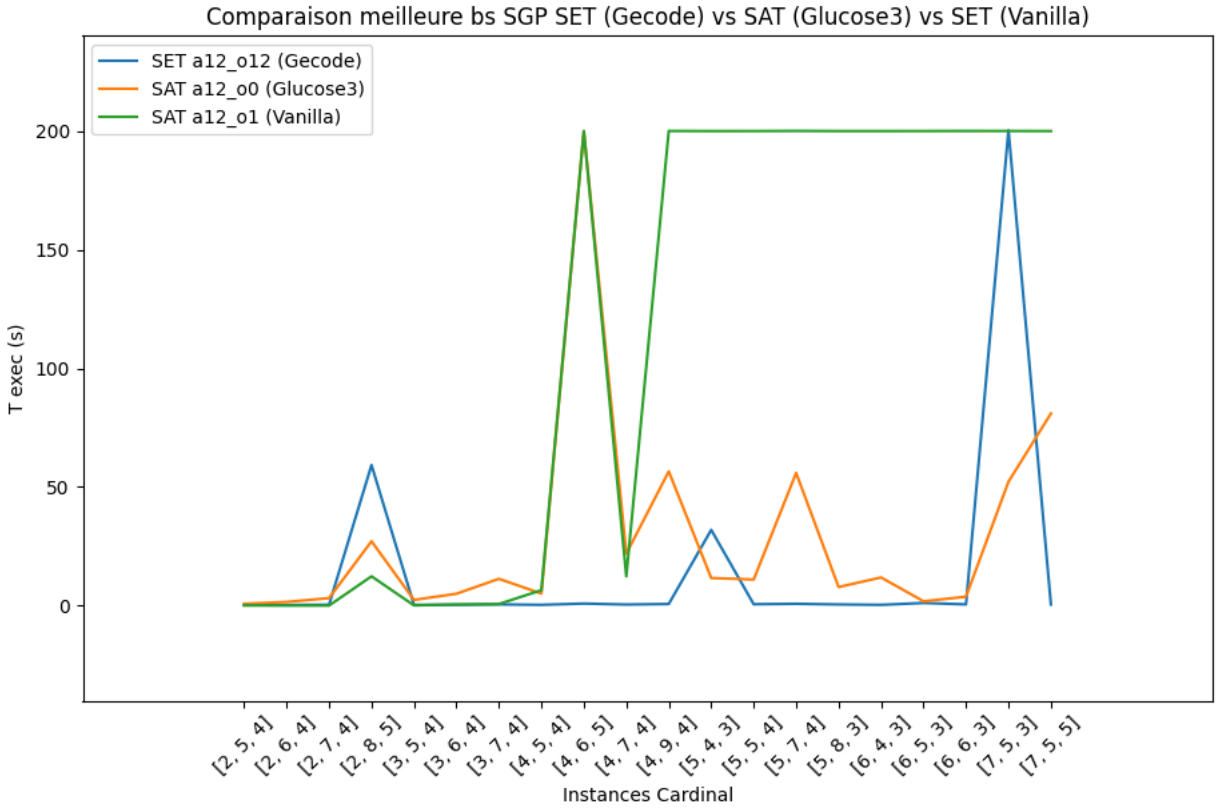

FIGURE 11 – Meilleure variante modèle SET (Gecode) vs SAT (Glucose3) vs SET (Vanilla)

Vanilla domine les autres solveurs sur les premières instances ”faciles” puis se fait dominer sur les autres. Notre hypothèse est que Vanilla étant un solveur encore très rudimentaire comparer aux solveurs professionnel, il présente l'avantage d'être ”léger” en ressource.

2.5.4 Comparaisons avec les temps présentés dans le papier cardinal

Comparons à présent tous les solveurs entre eux, en utilisant la même bs que le papier, c'est à dire a12_bs_o1 (ordre entre les groupes) :

w-g-s	*Cardinal*	*ic_sets*	*ROBDD*	**Gecode**	**Glucose3**	**Vanilla**
2-5-4	0.83	5.3	0.1	0.157	1.101	0.092
2-6-4	1.75	35.5	0.2	0.174	2.192	0.016
2-7-4	2.82	70.3	0.6	0.221	4.129	0.026
2-8-5	—	—	3.1	57.667	33.821	12.353
3-5-4	1.89	9.3	0.5	0.174	2.606	0.094
3-6-4	4.62	59.2	2.3	0.235	6.279	0.495
3-7-4	6.37	113.6	3.5	0.345	13.209	0.665
4-5-4	3.13	10.5	1.3	0.216	5.361	6.511
4-6-5	—	—	171.5	2.763	53.998	-
4-7-4	12.46	135.8	21.8	0.519	21.954	12.355
4-9-4	42.45	22.7	338.4	0.52	62.165	-
5-4-3	165.63	—	44.4	-	-	-
5-5-4	28.65	267.3	4.4	3.964	9.38	-
5-7-4	17.18	—	54.7	0.704	40.972	-
5-8-3	1.01	4.1	6.6	0.333	9.519	-
6-4-3	94.67	—	29.6	1.818	-	-
6-5-3	—	—	2.0	-	2.553	-
6-6-3	1.20	2.7	2.5	0.554	4.067	-
7-5-3	—	—	28.4	-	153.692	-
7-5-5	—	—	0.4	0.111	75.999	-

FIGURE 12 – Comparaison des solveurs sur l'instance cardinal variante bs_a12_o1

Etant donnée que le papier Cardinal date de 2007, cette comparaison n'est pas pertinente. En effet les solveurs ont certainement beaucoup évolué depuis, comme le suggère les publications de Peter stuckey et Guido Tack sur la site de Minizinc. D'autres explications pourraient être les conditions d'experimentation (processeur de l'époque), ou une mauvaise compréhension de notre part concernant les modèles utilisés. Nous sommes cependant satisfait de constater la dominance de Vanilla sur la première partie des instances.

2.6 Conclusion du travail d'implémentation

Ces premiers résultats sont encourageant pour poursuivre le développement de Vanilla. Ce solveur est encore naissant et il n'est pas garanti qu'il soit exempt de bug. Il nous faudrait effectuer davantage de tests et bénéficier de retours utilisateurs pour valider cette version 1.

L'un de nos objectif d'implémentation concernait une approche *ROBDD* pour notre B&P. Nous y avons réfléchit mais n'avons pas trouvé d'implémentation pertinente : en effet d'après nos algorithmes d'essais, la binarisation se prête mal à l'énumération de l'ensemble des valeurs max d'une variable. Dans un prochain travail, nous pourrions investiguer davantage cette approche en étudiant la littérature associée.

3 Conclusion

Ce projet fût passionnant et particulièrement enrichissant. Le Social Golfer Problem est un problème truffé de faux semblants : en effet nous pensions par exemple qu'en accumulant les cassages de symétries, n'importe quel instance se résoudrait plus rapidement, or, nos expérimentations nous ont montré que ce n'est pas toujours vrai. Nous sommes néanmoins conscients que des erreurs d'implémentation ont pu se glisser ce qui fausserait des parties de nos analyses, mais l'étude du SGP demande de rassembler beaucoup de recherches et de travaux dont les conclusions ne peuvent s'arrêter à cette étude seule. Il nous reste beaucoup de tests que nous aimerions approfondir, comme l'implémentation de la bs_o2 pour le modèle SAT, l'amélioration potentielle de certaines de nos formules SAT ou l'impact des surcontraintes pour les modèles SET et SAT. Nous sommes également très satisfaits d'avoir les bases d'un solveur fonctionnel qui n'attend qu'à être amélioré. En particulier, nous aimerions pour la version 1 de Vanilla ajouter la consistance de borne à la fonction de filtrage, ainsi que des contraintes pré-faites pour faciliter l'implémentation d'un modèle Vanilla pour l'utilisateur. En extension de ce projet, nous aimerions également tester l'implémentation d'un solveur SAT pour le comparer avec Vanilla et les autres solveurs.

Références

[1] Carmen Gervet. « Conjunto : constraint logic programming with finite set domains ». In : *ILPS*. 1994.

[2] Francisco Azevedo. « Cardinal : A finite sets constraint solver ». In : *Constraints* 12.1 (2007), p. 93-129.

[3] Olivier Bailleux et Yacine Boufkhad. « Efficient CNF encoding of boolean cardinality constraints ». In : *International conference on principles and practice of constraint programming*. Springer. 2003, p. 108-122.

[4] J Correas, S Estévez Martin et Fernando Sáenz-Pérez. « Enhancing set constraint solvers with bound consistency ». In : *Expert Systems with Applications* 92 (2018), p. 485-494.

[5] Frederic Lardeux et al. « Set constraint model and automated encoding into SAT : application to the social golfer problem ». In : *Annals of Operations Research* 235.1 (2015), p. 423-452.

A Modèle SET : Résultat expérience instances Cardinal

Les temps d'exécution sont en secondes.

Instance	Modèle	temps exec
2 5 4	a1	0.188
	a2	0.135
	a12	0.133
	a12_o1	0.157
	a12_o2	0.162
	a12_o12	0.188
	o1	0.169
	o2	0.201
	o12	0.166
	sbs	0.19

TABLE 1 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
2 6 4	a1	0.466
	a2	0.139
	a12	0.142
	a12_o1	0.174
	a12_o2	0.16
	a12_o12	0.185
	o1	0.169
	o2	0.471
	o12	0.181
	sbs	0.447

TABLE 2 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
2 7 4	a1	0.255
	a2	0.144
	a12	0.177
	a12_o1	0.221
	a12_o2	0.194
	a12_o12	0.237
	o1	0.18
	o2	0.278
	o12	0.201
	sbs	0.247

TABLE 3 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
2 8 5	a1	200.077
	a2	4.049
	a12	52.674
	a12_o1	57.667
	a12_o2	35.141
	a12_o12	58.374
	o1	0.215
	o2	200.098
	o12	1.032
	sbs	200.082

TABLE 4 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
3 5 4	a1	0.286
	a2	0.148
	a12	0.143
	a12_o1	0.174
	a12_o2	0.165
	a12_o12	0.194
	o1	0.173
	o2	0.29
	o12	0.217
	sbs	0.261

TABLE 5 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
3 6 4	a1	0.866
	a2	0.159
	a12	0.225
	a12_o1	0.235
	a12_o2	0.241
	a12_o12	0.26
	o1	0.2
	o2	1.009
	o12	0.234
	sbs	0.87

TABLE 6 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
3 7 4	a1	0.563
	a2	0.168
	a12	0.467
	a12_o1	0.345
	a12_o2	0.659
	a12_o12	0.387
	o1	0.242
	o2	0.542
	o12	0.276
	sbs	0.583

TABLE 7 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
4 5 4	a1	0.783
	a2	0.373
	a12	0.169
	a12_o1	0.216
	a12_o2	0.19
	a12_o12	0.274
	o1	0.23
	o2	5.497
	o12	0.321
	sbs	0.701

TABLE 8 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
4 6 5	a1	200.088
	a2	40.758
	a12	2.264
	a12_o1	2.763
	a12_o2	0.597
	a12_o12	0.755
	o1	0.807
	o2	200.117
	o12	28.933
	sbs	200.092

TABLE 9 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
4 7 4	a1	3.564
	a2	0.835
	a12	1.349
	a12_o1	0.519
	a12_o2	0.459
	a12_o12	0.363
	o1	0.305
	o2	1.463
	o12	0.359
	sbs	3.607

TABLE 10 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
4 9 4	a1	1.206
	a2	0.403
	a12	0.394
	a12_o1	0.52
	a12_o2	0.613
	a12_o12	0.584
	o1	0.387
	o2	5.552
	o12	0.44
	sbs	1.127

TABLE 11 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
5 4 3	a1	200.089
	a2	200.088
	a12	200.08
	a12_o1	200.097
	a12_o2	19.205
	a12_o12	30.091
	o1	200.129
	o2	200.155
	o12	200.103
	sbs	200.079

TABLE 12 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
5 5 4	a1	200.086
	a2	125.962
	a12	2.977
	a12_o1	3.964
	a12_o2	0.315
	a12_o12	0.359
	o1	0.375
	o2	0.419
	o12	30.053
	sbs	200.087

TABLE 13 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
5 7 4	a1	97.463
	a2	5.111
	a12	4.954
	a12_o1	0.704
	a12_o2	1.978
	a12_o12	0.606
	o1	0.434
	o2	1.942
	o12	0.498
	sbs	110.417

TABLE 14 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
5 8 3	a1	12.321
	a2	29.277
	a12	0.3
	a12_o1	0.333
	a12_o2	77.351
	a12_o12	0.365
	o1	0.325
	o2	7.067
	o12	0.436
	sbs	16.892

TABLE 15 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
6 4 3	a1	200.088
	a2	3.255
	a12	1.574
	a12_o1	1.818
	a12_o2	0.186
	a12_o12	0.217
	o1	200.104
	o2	200.103
	o12	200.114
	sbs	200.086

TABLE 16 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
6 5 3	a1	200.096
	a2	200.096
	a12	200.095
	a12_o1	200.126
	a12_o2	15.888
	a12_o12	0.902
	o1	31.462
	o2	85.08
	o12	17.153
	sbs	200.095

TABLE 17 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
6 6 3	a1	200.109
	a2	0.442
	a12	1.667
	a12_o1	0.554
	a12_o2	23.576
	a12_o12	0.339
	o1	0.351
	o2	66.332
	o12	1.816
	sbs	200.11

TABLE 18 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
7 5 3	a1	200.108
	a2	200.108
	a12	200.108
	a12_o1	200.143
	a12_o2	200.132
	a12_o12	200.155
	o1	200.144
	o2	200.147
	o12	47.861
	sbs	200.105

TABLE 19 – Influence des brise-symétries sur les temps de résolution sous Gecode

Instance	Modèle	temps exec
7 5 5	a1	0.117
	a2	0.114
	a12	0.11
	a12_o1	0.111
	a12_o2	0.112
	a12_o12	0.114
	o1	0.111
	o2	0.116
	o12	0.113
	sbs	0.131

TABLE 20 – Influence des brise-symétries sur les temps de résolution sous Gecode

B Modèle SAT : Résultat expérience instances Cardinal

Les temps d'exécution sont en secondes.

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
2 5 4	a1	691640	99.0	1.0	0.772
	a2	691640	97.0	3.0	0.97
	a12	691640	98.0	2.0	0.919
	a12_o0	703040	98.0	2.0	0.859
	a12_o1	752440	97.0	3.0	1.101
	a12_o01	763840	96.0	4.0	1.015
	o0	703040	95.0	5.0	0.807
	o1	752440	67.0	33.0	1.721
	o01	763840	85.0	15.0	1.46
	sbs	691640	91.0	9.0	1.07

TABLE 21 – Instance 2 5 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
2 6 4	a1	1444080	98.0	2.0	1.939
	a2	1444080	97.0	3.0	1.655
	a12	1444080	99.0	1.0	1.742
	a12_o0	1463952	98.0	2.0	1.607
	a12_o1	1576560	96.0	4.0	2.192
	a12_o01	1596432	91.0	9.0	2.158
	o0	1463952	87.0	13.0	1.859
	o1	1576560	66.0	34.0	3.044
	o01	1596432	59.0	41.0	3.413
	sbs	1444080	98.0	2.0	1.759

TABLE 22 – Instance 2 6 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
2 7 4	a1	2688392	99.0	1.0	3.335
	a2	2688392	98.0	2.0	3.267
	a12	2688392	99.0	1.0	2.958
	a12_o0	2720144	97.0	3.0	3.354
	a12_o1	2942408	95.0	5.0	4.129
	a12_o01	2974160	95.0	5.0	4.106
	o0	2720144	98.0	2.0	2.954
	o1	2942408	52.0	48.0	6.976
	o01	2974160	51.0	49.0	7.582
	sbs	2688392	98.0	2.0	2.977

TABLE 23 – Instance 2 7 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
2 8 5	a1	20030480	98.0	2.0	22.014
	a2	20030480	94.0	6.0	23.518
	a12	20030480	98.0	2.0	22.66
	a12_o0	20155280	84.0	16.0	26.037
	a12_o1	21122480	83.0	17.0	33.821
	a12_o01	21247280	95.0	5.0	28.65
	o0	20155280	97.0	3.0	23.791
	o1	21122480	25.0	75.0	111.187
	o01	21247280	74.0	26.0	39.266
	sbs	20030480	98.0	2.0	23.596

TABLE 24 – Instance 2 8 5 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
3 5 4	a1	2063460	97.0	3.0	2.343
	a2	2063460	99.0	1.0	2.284
	a12	2063460	99.0	1.0	2.295
	a12_o0	2080560	97.0	3.0	2.356
	a12_o1	2154660	98.0	2.0	2.606
	a12_o01	2171760	95.0	5.0	2.599
	o0	2080560	85.0	15.0	2.565
	o1	2154660	83.0	17.0	2.874
	o01	2171760	78.0	22.0	3.1
	sbs	2063460	93.0	7.0	2.417

TABLE 25 – Instance 3 5 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
3 6 4	a1	4312296	98.0	2.0	4.777
	a2	4312296	98.0	2.0	4.65
	a12	4312296	98.0	2.0	4.793
	a12_o0	4342104	94.0	6.0	5.989
	a12_o1	4511016	96.0	4.0	6.279
	a12_o01	4540824	77.0	23.0	6.814
	o0	4342104	89.0	11.0	5.141
	o1	4511016	57.0	42.0	8.767
	o01	4540824	59.0	41.0	8.739
	sbs	4312296	98.0	2.0	4.933

TABLE 26 – Instance 3 6 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
3 7 4	a1	8033340	99.0	1.0	9.343
	a2	8033340	99.0	1.0	10.262
	a12	8033340	99.0	1.0	9.812
	a12_o0	8080968	84.0	16.0	12.364
	a12_o1	8414364	98.0	2.0	13.209
	a12_o01	8461992	73.0	27.0	17.094
	o0	8080968	95.0	5.0	10.939
	o1	8414364	77.0	23.0	15.854
	o01	8461992	68.0	32.0	16.752
	sbs	8033340	99.0	1.0	8.964

TABLE 27 – Instance 3 7 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
4 5 4	a1	4119280	93.0	7.0	5.307
	a2	4119280	98.0	2.0	4.594
	a12	4119280	98.0	2.0	5.345
	a12_o0	4142080	85.0	15.0	5.433
	a12_o1	4240880	97.0	3.0	5.361
	a12_o01	4263680	92.0	8.0	5.358
	o0	4142080	54.0	46.0	8.842
	o1	4240880	52.0	48.0	10.93
	o01	4263680	5.0	95.0	93.522
	sbs	4119280	94.0	6.0	5.642

TABLE 28 – Instance 4 5 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
4 6 5	a1	37636320	72.0	28.0	70.776
	a2	37636320	72.0	28.0	64.454
	a12	37636320	74.0	26.0	59.997
	a12_o0	37740720	-	-	200.041
	a12_o1	38288820	92.0	8.0	53.998
	a12_o01	38393220	-	-	200.161
	o0	37740720	-	-	200.343
	o1	38288820	-	-	202.845
	o01	38393220	-	-	200.021
	sbs	37636320	84.0	16.0	48.384

TABLE 29 – Instance 4 6 5 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
4 7 4	a1	16045456	99.0	1.0	17.615
	a2	16045456	99.0	1.0	17.627
	a12	16045456	99.0	1.0	18.168
	a12_o0	16108960	85.0	15.0	23.65
	a12_o1	16553488	97.0	3.0	21.954
	a12_o01	16616992	79.0	21.0	27.903
	o0	16108960	66.0	34.0	28.081
	o1	16553488	14.0	86.0	141.95
	o01	16616992	12.0	88.0	158.145
	sbs	16045456	97.0	3.0	17.863

TABLE 30 – Instance 4 7 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
4 9 4	a1	44180784	98.0	2.0	49.596
	a2	44180784	98.0	2.0	49.331
	a12	44180784	98.0	2.0	50.261
	a12_o0	44316864	85.0	15.0	58.735
	a12_o1	45632304	87.0	13.0	62.165
	a12_o01	45768384	-	-	200.089
	o0	44316864	93.0	7.0	52.741
	o1	45632304	-	-	214.113
	o01	45768384	-	-	200.001
	sbs	44180784	82.0	18.0	58.556

TABLE 31 – Instance 4 9 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
5 4 3	a1	384180	-	-	200.017
	a2	384180	-	-	200.033
	a12	384180	-	-	200.062
	a12_o0	388140	4.0	96.0	11.798
	a12_o1	402000	-	-	202.412
	a12_o01	405960	5.0	95.0	10.761
	o0	388140	-	-	200.002
	o1	402000	-	-	200.056
	o01	405960	-	-	200.055
	sbs	384180	-	-	200.306

TABLE 32 – Instance 5 4 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
5 5 4	a1	6859100	67.0	33.0	11.891
	a2	6859100	65.0	35.0	11.408
	a12	6859100	85.0	15.0	8.568
	a12_o0	6887600	69.0	31.0	11.177
	a12_o1	7011100	86.0	14.0	9.38
	a12_o01	7039600	71.0	29.0	11.005
	o0	6887600	-	-	200.078
	o1	7011100	-	-	223.604
	o01	7039600	-	-	200.003
	sbs	6859100	40.0	60.0	18.129

TABLE 33 – Instance 5 5 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
5 7 4	a1	26724740	99.0	1.0	28.45
	a2	26724740	96.0	4.0	29.238
	a12	26724740	99.0	1.0	29.267
	a12_o0	26804120	55.0	45.0	54.171
	a12_o1	27359780	84.0	16.0	40.972
	a12_o01	27439160	18.0	82.0	169.303
	o0	26804120	33.0	67.0	86.804
	o1	27359780	-	-	200.044
	o01	27439160	-	-	200.093
	sbs	26724740	99.0	1.0	28.969

TABLE 34 – Instance 5 7 4 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
5 8 3	a1	6392280	99.0	1.0	6.95
	a2	6392280	99.0	1.0	7.271
	a12	6392280	99.0	1.0	6.898
	a12_o0	6425400	96.0	4.0	7.257
	a12_o1	6740040	89.0	11.0	9.519
	a12_o01	6773160	78.0	22.0	10.543
	o0	6425400	97.0	3.0	7.136
	o1	6740040	36.0	64.0	21.256
	o01	6773160	20.0	80.0	40.26
	sbs	6392280	99.0	1.0	6.984

TABLE 35 – Instance 5 8 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
6 4 3	a1	575064	-	-	200.06
	a2	575064	0.0	100.0	161.137
	a12	575064	-	-	200.273
	a12_o0	579816	5.0	95.0	12.041
	a12_o1	596448	-	-	200.079
	a12_o01	601200	7.0	93.0	11.091
	o0	579816	-	-	200.005
	o1	596448	-	-	200.138
	o01	601200	-	-	200.041
	sbs	575064	-	-	200.02

TABLE 36 – Instance 6 4 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
6 5 3	a1	1427040	94.0	6.0	1.746
	a2	1427040	26.0	74.0	6.049
	a12	1427040	66.0	34.0	2.386
	a12_o0	1436490	87.0	13.0	1.79
	a12_o1	1483740	62.0	38.0	2.553
	a12_o01	1493190	67.0	33.0	2.443
	o0	1436490	9.0	91.0	17.067
	o1	1483740	-	-	200.179
	o01	1493190	6.0	94.0	33.658
	sbs	1427040	39.0	61.0	4.738

TABLE 37 – Instance 6 5 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
6 6 3	a1	2990952	99.0	1.0	3.572
	a2	2990952	98.0	2.0	3.664
	a12	2990952	99.0	1.0	3.232
	a12_o0	3007476	93.0	7.0	3.476
	a12_o1	3114882	90.0	10.0	4.067
	a12_o01	3131406	76.0	24.0	4.66
	o0	3007476	80.0	20.0	4.188
	o1	3114882	13.0	87.0	27.538
	o01	3131406	23.0	77.0	15.636
	sbs	2990952	95.0	5.0	3.269

TABLE 38 – Instance 6 6 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
7 5 3	a1	1995630	12.0	88.0	16.707
	a2	1995630	-	-	201.392
	a12	1995630	8.0	92.0	28.32
	a12_o0	2006655	4.0	96.0	51.641
	a12_o1	2061780	2.0	98.0	153.692
	a12_o01	2072805	5.0	95.0	50.767
	o0	2006655	-	-	200.066
	o1	2061780	-	-	200.0
	o01	2072805	-	-	200.004
	sbs	1995630	3.0	97.0	65.758

TABLE 39 – Instance 7 5 3 : influence des bs sur les temps de résolution de Glucose3

Instance	Modèle	nb clauses	% t clauses	% t solve	t total (s)
7 5 5	a1	63052675	-	-	200.001
	a2	63052675	97.0	3.0	78.791
	a12	63052675	92.0	8.0	77.566
	a12_o0	63157675	87.0	13.0	81.671
	a12_o1	63577675	94.0	6.0	75.999
	a12_o01	63682675	63.0	37.0	131.43
	o0	63157675	-	-	200.01
	o1	63577675	-	-	200.012
	o01	63682675	-	-	200.034
	sbs	63052675	-	-	200.056

TABLE 40 – Instance 7 5 5 : influence des bs sur les temps de résolution de Glucose3

C Solveur

C.1 SGP instances Cardinal résolues avec Vanilla

Instance	Modèle	t total (s)
2 5 4	bs_a1	0.147
	bs_a2	0.031
	bs_a12	0.008
	bs_a12_o1	0.092
	bs_a12_o2	0.081
	bs_a12_o12	0.011
	bs_o1	200.002
	bs_o2	0.015
	bs_o12	200.002
	bs_sbs	0.013

TABLE 41 – Instance 2 5 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
2 6 4	bs_a1	0.015
	bs_a2	0.025
	bs_a12	0.01
	bs_a12_o1	0.016
	bs_a12_o2	0.023
	bs_a12_o12	0.015
	bs_o1	200.007
	bs_o2	0.024
	bs_o12	200.001
	bs_sbs	0.023

TABLE 42 – Instance 2 6 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
2 7 4	bs_a1	0.022
	bs_a2	0.032
	bs_a12	0.033
	bs_a12_o1	0.026
	bs_a12_o2	0.017
	bs_a12_o12	0.028
	bs_o1	200.003
	bs_o2	0.031
	bs_o12	200.003
	bs_sbs	0.046

TABLE 43 – Instance 2 7 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
2 8 5	bs_a1	0.04
	bs_a2	0.064
	bs_a12	0.044
	bs_a12_o1	12.353
	bs_a12_o2	0.029
	bs_a12_o12	12.472
	bs_o1	200.003
	bs_o2	0.083
	bs_o12	200.004
	bs_sbs	0.071

TABLE 44 – Instance 2 8 5 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
3 5 4	bs_a1	149.545
	bs_a2	0.039
	bs_a12	0.097
	bs_a12_o1	0.094
	bs_a12_o2	0.073
	bs_a12_o12	0.105
	bs_o1	200.002
	bs_o2	138.442
	bs_o12	200.002
	bs_sbs	200.009

TABLE 45 – Instance 3 5 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
3 6 4	bs_a1	200.011
	bs_a2	200.014
	bs_a12	1.019
	bs_a12_o1	0.495
	bs_a12_o2	1.795
	bs_a12_o12	0.61
	bs_o1	200.003
	bs_o2	200.015
	bs_o12	200.004
	bs_sbs	200.016

TABLE 46 – Instance 3 6 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
3 7 4	bs_a1	200.016
	bs_a2	200.021
	bs_a12	2.517
	bs_a12_o1	0.665
	bs_a12_o2	2.601
	bs_a12_o12	0.69
	bs_o1	200.006
	bs_o2	200.025
	bs_o12	200.009
	bs_sbs	200.046

TABLE 47 – Instance 3 7 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
4 5 4	bs_a1	200.009
	bs_a2	0.148
	bs_a12	5.687
	bs_a12_o1	6.511
	bs_a12_o2	200.01
	bs_a12_o12	200.013
	bs_o1	200.004
	bs_o2	200.021
	bs_o12	200.005
	bs_sbs	200.014

TABLE 48 – Instance 4 5 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
4 6 5	bs_a1	200.02
	bs_a2	130.723
	bs_a12	200.022
	bs_a12_o1	200.025
	bs_a12_o2	200.024
	bs_a12_o12	200.028
	bs_o1	200.022
	bs_o2	200.051
	bs_o12	200.024
	bs_sbs	200.033

TABLE 49 – Instance 4 6 5 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
4 7 4	bs_a1	200.026
	bs_a2	200.035
	bs_a12	5.081
	bs_a12_o1	12.355
	bs_a12_o2	200.05
	bs_a12_o12	87.326
	bs_o1	200.013
	bs_o2	200.055
	bs_o12	200.021
	bs_sbs	200.042

TABLE 50 – Instance 4 7 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
4 9 4	bs_a1	200.09
	bs_a2	200.121
	bs_a12	200.085
	bs_a12_o1	200.059
	bs_a12_o2	5.624
	bs_a12_o12	200.08
	bs_o1	200.021
	bs_o2	200.111
	bs_o12	200.014
	bs_sbs	200.155

TABLE 51 – Instance 4 9 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
5 4 3	bs_a1	200.013
	bs_a2	200.006
	bs_a12	200.007
	bs_a12_o1	200.005
	bs_a12_o2	200.006
	bs_a12_o12	200.004
	bs_o1	200.012
	bs_o2	200.015
	bs_o12	200.007
	bs_sbs	200.012

TABLE 52 – Instance 5 4 3 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
5 5 4	bs_a1	200.013
	bs_a2	200.022
	bs_a12	200.015
	bs_a12_o1	200.02
	bs_a12_o2	200.023
	bs_a12_o12	200.034
	bs_o1	200.023
	bs_o2	200.035
	bs_o12	200.017
	bs_sbs	200.047

TABLE 53 – Instance 5 5 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
5 7 4	bs_a1	200.055
	bs_a2	200.073
	bs_a12	200.049
	bs_a12_o1	200.079
	bs_a12_o2	200.063
	bs_a12_o12	200.091
	bs_o1	200.042
	bs_o2	200.011
	bs_o12	200.036
	bs_sbs	200.065

TABLE 54 – Instance 5 7 4 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
5 8 3	bs_a1	1.141
	bs_a2	200.102
	bs_a12	200.075
	bs_a12_o1	200.013
	bs_a12_o2	200.072
	bs_a12_o12	200.014
	bs_o1	200.053
	bs_o2	20.928
	bs_o12	200.098
	bs_sbs	0.51

TABLE 55 – Instance 5 8 3 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
6 5 3	bs_a1	200.036
	bs_a2	200.044
	bs_a12	200.025
	bs_a12_o1	200.024
	bs_a12_o2	200.023
	bs_a12_o12	200.036
	bs_o1	200.034
	bs_o2	200.033
	bs_o12	200.024
	bs_sbs	200.045

TABLE 56 – Instance 6 5 3 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
6 6 3	bs_a1	5.921
	bs_a2	200.066
	bs_a12	200.038
	bs_a12_o1	200.06
	bs_a12_o2	200.06
	bs_a12_o12	200.055
	bs_o1	200.012
	bs_o2	200.079
	bs_o12	200.01
	bs_sbs	24.226

TABLE 57 – Instance 6 6 3 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
7 5 3	bs_a1	200.048
	bs_a2	200.02
	bs_a12	200.03
	bs_a12_o1	200.042
	bs_a12_o2	200.013
	bs_a12_o12	200.01
	bs_o1	200.074
	bs_o2	200.067
	bs_o12	200.051
	bs_sbs	200.063

TABLE 58 – Instance 7 5 3 : influence des bs sur les temps de résolution de Vanilla

Instance	Modèle	t total (s)
7 5 5	bs_a1	200.041
	bs_a2	200.02
	bs_a12	200.018
	bs_a12_o1	200.015
	bs_a12_o2	200.012
	bs_a12_o12	200.021
	bs_o1	200.039
	bs_o2	200.015
	bs_o12	200.033
	bs_sbs	200.078

TABLE 59 – Instance 7 5 5 : influence des bs sur les temps de résolution de Vanilla

C.2 Sudoku

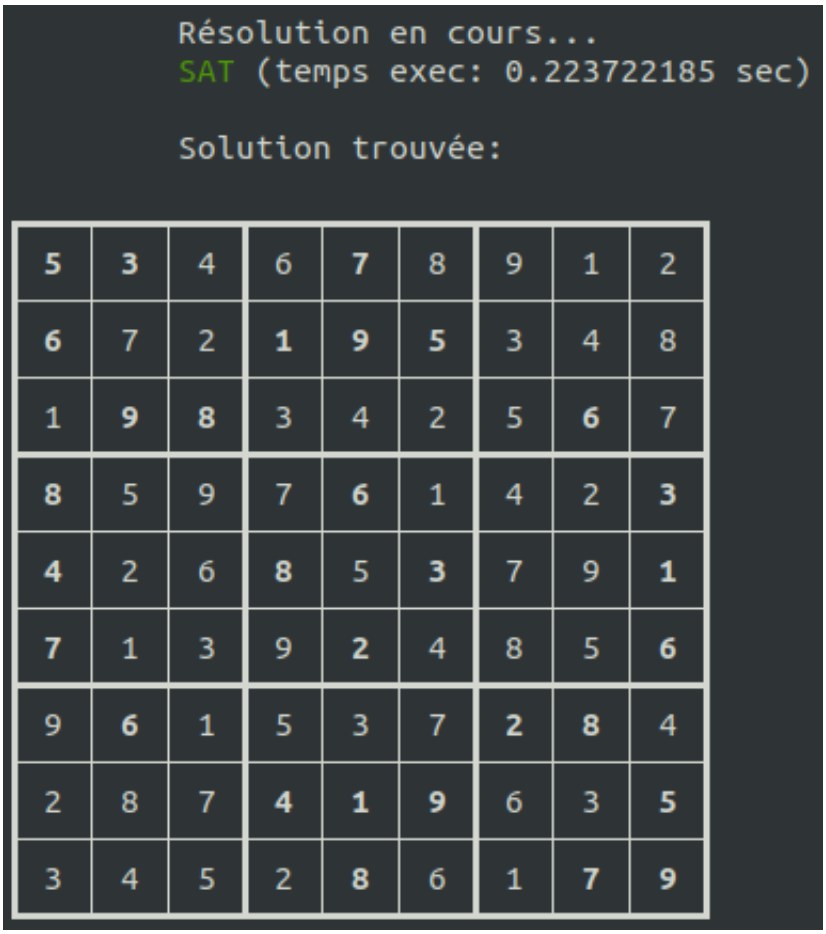

FIGURE 13 – Résolution d'une grille de sudoku (1)

FIGURE 14 – Résolution d'une grille de sudoku (2)