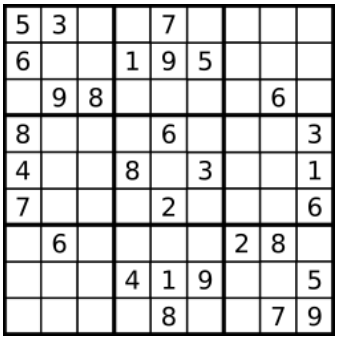
Résolution de puzzles à l'aide d'un solveur SAT

F. Herbreteau

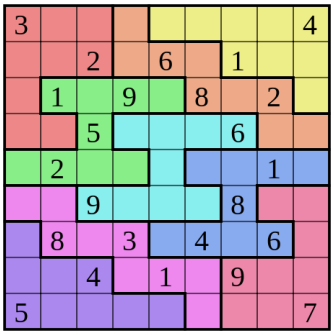
2013

Ce projet a pour but la résolution de puzzles de type « sudoku » à l'aide d'un solveur SAT. Il comprend essentiellement quatre parties. Premièrement, la mise en équation du puzzle en logique propositionnelle qui réduit la résolution du puzzle à un problème de satisfaisabilité (SAT). Deuxièmement, la génération des équations au format DIMACS, puis l'obtention d'une solution (s'il y en a) grâce à un solveur SAT. Troisièmement, l'étude des limites de cette approche : la taille maximale des puzzles qu'il est possible de résoudre. Enfin, la quatrième partie sera consacrée à la simplification des équations par une résolution partielle des puzzles par des méthodes *ad hoc* de votre choix, dans le but d'augmenter la taille des puzzles qu'il est possible de résoudre.

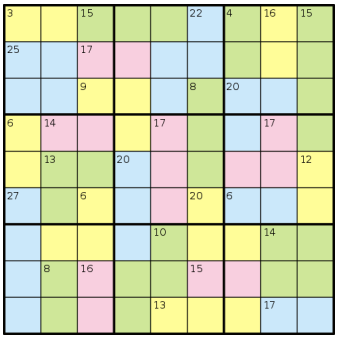
Un « sudoku »



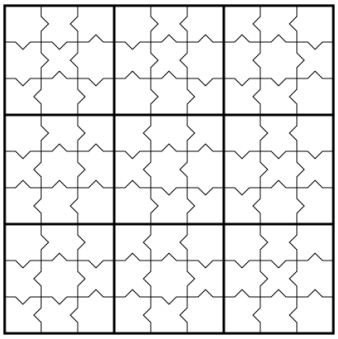
Un « jigsaw sudoku »



Un « killer-sudoku »



Un « comparison sudoku »



1 Quelques puzzles de type « sudoku »

Nous connaissons tous le classique « sudoku » de taille 9×9, comme celui-ci :

8	7				3	2	1	
4				8	7			
			4				9	
	9	4						
		7				8		
						1	4	
	6				9			
			3	5				7
	1	8	6				5	2

L'objectif est de remplir la grille avec des symboles de 1 à 9 de telle sorte que chaque symbole apparaisse exactement une fois dans chaque ligne, chaque colonne et chaque carré 3×3 de la grille.

Il existe de nombreuses variantes de puzzles de type « sudoku » (voir `http://en.wikipedia.org/wiki/Sudoku`). La plus simple consiste à faire varier la taille de la grille. Par exemple, en passant à des grilles 16×16 (symboles $1, \ldots, 9, A, \ldots F$) ou plus.

Certaines variantes modifient la géométrie de la grille. Par exemple, les « jigsaw sudoku » considèrent des grilles où les carrés 3×3 sont remplacés par des zones irrégulières.

Les puzzles peuvent également être définis par d'autres types de contraintes. Le « samunamupure » (ou « killer sudoku ») s'appuie sur une grille 9×9 classique, augmentée de contraintes sur la somme des nombres apparaissant dans certaines zones de la grille. Les « comparison sudoku » sont définis par des grilles 9×9 classiques augmentées de relation d'ordre entre les cases de la grille.

Afin de résoudre ces problèmes algorithmiquement, on peut envisager soit d'écrire des algorithmes spécifiques pour chaque jeu, soit d'utiliser une approche universelle. L'approche universelle la plus simple (dite « brute-force ») consiste à énumérer toutes les grilles possibles et à vérifier si chacune d'entre elles résout le puzzle. Cette approche est évidemment peu utilisable en pratique. Ce projet cherche à développer une approche universelle utilisable sur des puzzles complexes en s'appuyant sur un solveur SAT.

2 SAT : logique propositionnelle et satisfaisabilité

L'approche développée dans ce projet consiste à formaliser les puzzles de type « sudoku » en logique propositionnelle. Les contraintes seront exprimées en équations ou **formules** de la logique propositionnelle. La résolution du puzzle sera alors ramenée à un problème de satisfaisabilité (SAT). Un solveur SAT sera utilisé pour résoudre le puzzle.

2.1 Modéliser en logique propositionnelle

La logique propositionnelle est un langage mathématique qui permet d'exprimer des faits et d'en étudier la véracité. Les éléments du langages sont les **propositions** : des variables ayant pour valeur soit 1 (« vrai »), soit 0 (« faux »). Par exemple, pour modéliser la grille de sudoku précédente, on peut introduire les propositions p_{ijk}, avec $i, j, k \in \{1, \ldots, 9\}$. Dans ce modèle, la proposition p_{ijk} est vraie quand la case de coordonnées (i, j), c'est à dire en ligne i et colonne j, contient le chiffre k. Ainsi, p_{118}, p_{127} et p_{389} sont vraies. Inversement, p_{112} et p_{324} sont fausses.

On note $\neg$ la négation. On a donc $\neg p_{112}$ et $\neg p_{324}$ qui sont des formules vraies.

La conjonction est notée $\wedge$. La formule $p_{118} \wedge p_{127}$ est vraie puisque la case $(1, 1)$ contient 8 et la case $(1, 2)$ contient 7.

La disjonction est notée $\vee$. La formule $p_{127} \vee p_{125}$ est également vraie puisque la case $(1, 2)$ contient bien un 7 ou un 5. On ne sait pas dire si la formule $p_{132} \vee p_{135}$ est vraie : il faudrait résoudre la grille de sudoku pour savoir si la case $(1, 3)$ contient 2 ou 5.

Le symbole $\implies$ représente l'implication. La formule $p \implies q$ se lit « p implique q ». Elle signifie que si p est vraie, alors q est également vraie. En d'autres termes, la formule $p \implies q$ est vraie lorsque p est fausse et lorsque p et q sont vraies. Par exemple, la formule $p_{214} \implies \neg p_{224}$ est vraie, car puisque la case $(2, 1)$ contient un 4, il ne peut pas y avoir un 4 en case $(2, 2)$. La formule $p_{215} \implies p_{225}$ signifie que s'il y a un 5 en case $(2, 1)$, alors il y a également un 5 en case $(2, 2)$. Cette formule est également vraie puisqu'il n'y a pas de 5 en case $(2, 1)$. La formule $p_{429} \implies p_{431}$ est fausse : il y a un 9 en case $(4, 2)$ mais il n'y a pas de 1 en case $(4, 3)$.

La sémantique des connecteurs logiques s'exprime sous la forme de tables de vérité :

p	$\neg p$
0	1
1	0

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

p	q	$p \implies q$
0	0	1
0	1	1
1	0	0
1	1	1

Une **valuation** associe à chaque variable propositionnelle p une valeur 0 ou 1. Étant donnée une valuation, on peut déterminer la valeur d'une formule à l'aide des tables de vérité. Par exemple, la formule $p \wedge (q \vee r)$ a la valeur 1 (« vrai ») dans la valuation $p = 1$, $q = 1$, $r = 0$.

Le problème de **satisfaisabilité** consiste à déterminer si, pour une formule donnée Φ, il existe une valuation v telle que Φ a la valeur 1 dans v (voir `http://en.wikipedia.org/wiki/Boolean_satisfiability_problem`). On a vu par exemple que $p \wedge (q \vee r)$ est satisfaisable.

2.2 Les solveurs SAT et le format DIMACS

Un solveur SAT est un programme informatique qui résout le problème de satisfaisabilité. À partir d'une formule donnée, il indique si celle-ci est satisfaisable ou non. Si elle

est satisfaisable, il fournit une valuation qui satisfait la formule.

Les solveurs SAT prennent en entrée des formules propositionnelles en **forme normale conjonctive (CNF)**. Une formule est en forme normale conjonctive si elle est une conjonction de clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Chaque clause C_i étant elle-même une disjonction $l_1 \vee l_2 \vee \cdots \vee l_k$ de littéraux : soit une proposition atomique $l_j = p$, soit la négation d'une proposition atomique $l_j = \neg p$. Par exemple :

$$(p \vee \neg q) \wedge r \qquad p \qquad (\neg p \vee q \vee r) \wedge (p \vee \neg q) \wedge (\neg q \vee r)$$

sont en forme normale conjonctive. Par contre, les formules suivantes ne le sont pas :

$$\neg(p \wedge q) \qquad p \wedge (q \vee (r \wedge s))$$

Toute formule propositionnelle peut être mise sous forme normale conjonctive par application des lois de « de Morgan » et des lois de distributivité :

$$\neg(p \wedge q) \equiv \neg p \vee \neg q \qquad \neg(p \vee q) \equiv \neg p \wedge \neg q \qquad p \Longrightarrow q \equiv \neg p \vee q$$

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r) \qquad p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

La formule résultante peut être exponentiellement plus grande que la formule de départ. La page wikipedia `http://fr.wikipedia.org/wiki/Forme_normale_conjonctive` présente une mise en forme normale conjonctive de complexité linéaire.

La plupart des solveurs SAT admettent comme format d'entrée le format DIMACS. Les variables propositionnelles sont nommée $p_1, p_2, \ldots, p_n$. Dans ce format la formule $(\neg p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee \neg p_2) \wedge (\neg p_2 \vee p_3) \wedge p_2$ s'écrit :

```
c Une ligne qui commence par c est un commentaire
p cnf 3 4
-1 2 3 0
1 -2 0
-2 3 0
2 0
```

La ligne `p cnf 3 4` signifie que la formule est donnée en forme normale conjonctive avec 3 variables propositionnelles et 4 clauses. Chaque clause est écrite sur une ligne : `-1` signifie $\neg p_1$ et `2` signifie p_2. Chaque ligne se termine par `0` précédé d'un espace.

3 Travail à réaliser

L'objectif de ce projet est de programmer un solveur de sudoku, et de certaines de ses variantes. Étant donné une grille de jeu, votre solveur devra générer une formule de la logique propositionnelle telle que la formule est satisfaisable si et seulement si la grille admet une solution. Votre programme utilisera alors un solveur SAT pour résoudre le problème se satisfaisabilité. Lorsque la formule est satisfaisable, votre programme affichera la grille solution construite à partir du résultat donné par le solveur SAT.

1. Vous formaliserez en logique propositionnelle le jeu de sudoku (tout d'abord en version 9×9 puis en version $N \times N$). Vous formaliserez au moins les variantes suivantes : « killer sudoku », « comparison sudoku » et « jigsaw sudoku ». **Vous veillerez à ce que vos formules soient satisfaisables si et seulement si la grille a une solution.**

2. Vous écrirez un programme qui lit une grille de jeu depuis un fichier et qui génère la formule propositionnelle correspondante. Votre programme lira les fichiers au format suivant :

```
nsymbols nrows ncolumns
g sudoku
u (1,1) (1,2) (1,3) *
o (2,1) > (4,5) *
s 12 (1,1) (1,2) (2,1) *
```

   La première ligne contient trois nombres : `nsymbols` : le nombre de symboles, puis `nrows` : le nombre de lignes, et `ncolumns` : le nombre de colonnes.

   La deuxième ligne commence par `g` et indique le jeu décrit ensuite. Quatre jeux devront être supportés : « sudoku », « killer sudoku », « comparison sudoku » et « jigsaw sudoku ». En fonction du type de jeu, certaines contraintes (occurrence unique de chaque symbole par ligne, par colonne, ...) seront générées automatiquement par votre programme.

   Ensuite, suivront des lignes de type `u`, `o` et `s` dans un ordre quelconque. Toutes ces lignes se terminent par le symbole `*`.

   Les lignes `u` indiquent que chaque symbole a au plus une occurrence dans le groupe de cellules dont les coordonnées suivent.

   Les lignes `o` indiquent une relation d'ordre entre les cellules : `<`, `<=`, `=`, `>=` ou `>`.

   Enfin, les lignes `s` sont composées d'un nombre suivi d'une liste de cellules. Elles indiquent la somme des valeurs se trouvant dans cet ensemble de cellules. On pourra spécifier la valeur d'une cellule particulière par une ligne : `s 8 (1,1) *`.

3. Votre programme générera la formule qui correspond au jeu au format DIMACS, la fournira au solveur SAT, lira la sortie produite par celui-ci et affichera la grille résultat (s'il y en a une). Votre programme devra vérifier que la solution est bien consistante avec la grille de départ.

4. Vous étudierez les limites des solveurs SAT pour résoudre ces problèmes : quelle est la taille $N \times N$ maximale des grilles de sudoko que votre programme peut résoudre ? Vous proposerez des méthodes de résolution partielle de votre choix permettant de simplifier les formules fournies au solveur SAT.