Résolution du Sudoku

Thomas Hofer

sous la direction de Dr. Alain Prodon et Christophe Weibel

10 février 2006

1 Introduction

Le Sudoku est un jeu de logique. Le but du jeu est de remplir une grille de 9x9 cases en ajoutant un chiffre par case, quelques chiffres étant déjà donnés au début. Les règles à respecter : le même chiffre de 1 à 9 ne peut figurer qu'une seule fois par colonne, qu'une seule fois par ligne et qu'une seule fois par petit carré de neuf cases. Figure 1 montre un exemple d'un Sudoku. Le jeu Sudoku apparaissait pour la première fois en 1979 dans un magasin de puzzle américain sous le nom "number place". Il avait son premier succés au Japon est depuis 2005 il est populaire dans le monde entier.

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

Fig. 1 – Exemple d'un Sudoku

1.1 Terminologie

Une région d'un Sudoku est un petit carré 3x3. Une unité dans un Sudoku est une ligne, une colonne ou une région. Un Sudoku donné est appelé symétrique si l'emplacement des chiffres initiales est symétrique par rapport au centre. Le Sudoku donné en figure 1 est symétrique. Les candidats d'une case sont tous les chiffres qu'on peut mettre dans la case sans violer les règles.

2 Mathématiques du Sudoku

2.1 Nombre de chiffres donnés

Le nombre maximal de chiffres donnés tel que la grille admet encore plus qu'une solution est 77. Si deux fois deux chiffres égaux manquent, les cases vides sont dans les coins d'un rectangle orthogonal et exactement deux de ces cases sont dans le même petit carré 3x3, alors il y a deux possibilités de mettre les chiffres. Le nombre minimal de chiffres donnés tel que la grille admet une unique solution n'est pas connu. Si on n'impose pas que la grille soit symétrique, le nombre minimum de donnés trouvé jusqu'à maintenant est 17, avec la restriction de symétrie il est 18.

2.2 Nombre de solutions

Le nombre de grilles solution d'un Sudoku est $6'670'903'752'021'072'936'960 \approx 6.67 \cdot 10^{21}$. Ce nombre a été trouvé par brute force, la méthode est décrite dans [1]. Le nombre de grilles essentiellement différentes a aussi été calculé, il est $5'472'730'538$. La méthode qui à conduit à ce résultat est décrite dans [2]. Essentiellement différent veut dire qu'on ne peut pas transformer une solution en une autre par

- interchangement des chiffres
- reflexion
- rotation
- permutation d'un bloc de colonnes 1-3, 4-6 et 7-9
- permutation d'un bloc de lignes 1-3, 4-6 et 7-9
- permutation des colonnes 1-3, 4-6 ou 7-9
- permutation des lignes 1-3, 4-6 ou 7-9.

Le groupe formé par les solutions essentiellement différentes peut être vu comme un sous-groupe du groupe symétrique $\mathcal{S}_{81}$.

2.3 Sudoku comme problème de coloration d'un graphe

Résoudre des Sudokus peut être exprimé comme problème de coloration d'un graphe. Le but est de construire une 9-coloration d'un graphe pour une 9-coloration partielle donnée. Le graphe a 81 sommets, un pour chaque case de la grille. On peut nommer les sommets avec les paires ordonnées (x, y), où x et y sont des entiers entre 1 et 9. Dans ce cas, deux sommets différents (x, y) et (x', y') sont rejoints par une arête si et seulement si :

- $x = x'$ ou
- $y = y'$ ou
- $\lceil x/3 \rceil = \lceil x'/3 \rceil$ et $\lceil y/3 \rceil = \lceil y'/3 \rceil$.

Le problème est alors résolu en associant un entier entre 1 et 9 à chaque sommet tel que les sommets reliés par une arête n'ont pas le même nombre associé.

3 Modèle de base

J'ai utilisé le modèle suivant : Les cases sont numérotées comme les éléments d'une matrice. Donc A_{32} correspond à la case dans la troisième ligne et deuxième colonne. Associée à chaque case il y a neuf variables, x_{ij1} à x_{ij9}. x_{ijk} vaut 1 si la valeur du chiffre dans la case A_{ij} est k, sinon, elle vaut 0. Comme il y a un chiffre dans chaque case, on a que

$$\sum_{k=1}^{9} x_{ijk} = 1, \forall i, j \in \{1, \ldots, 9\}. \tag{1}$$

De plus, par les règles du Sudoku, chaque chiffre apparaît exactement une fois dans chaque ligne, une fois dans chaque colonne et une fois dans chaque région, ce qui donne les équations linéaires

$$\sum_{j=1}^{9} x_{ijk} = 1, \forall i, k \in \{1, \ldots, 9\}, \tag{2}$$

$$\sum_{i=1}^{9} x_{ijk} = 1, \forall j, k \in \{1, \ldots, 9\} \tag{3}$$

et

$$\sum_{i,j=1}^{3} x_{i+3r,j+3s,k} = 1, \forall r, s \in \{1, 2, 3\}, \forall k \in \{1, \ldots, 9\}. \tag{4}$$

En tout, il y a 729 variables et 324 équations. Le rang de la matrice variables-équations est 249, donc il n'y a que 249 équations linéairement indépendantes.

4 Résolution du Sudoku

Pour résoudre le Sudoku, j'utilise les programmes OPL et CPLEX. Avec ces programmes on peut résoudre des programmes linéaires. Comme fonction objective je prends 0, comme contraintes je prends les équations (1) - (4) et les variables des chiffres donnés fixées à 1. En plus, la solution doit être en nombre entiers. Alternativement, on peut prendre la somme des variables des chiffres donnés comme fonction objective et les équations (1) - (4) comme contraintes.
OPL et CPLEX n'ont aucun problème à resoudre un Sudoku, le temps de calcul est moins d'un second. Il y a deux problèmes avec les solutions obtenues : D'abord, d'après le programme elle est toujours unique, même si elle l'est pas. Puis, on ne sait pas grand chose sur comment elle a été obtenue. Lors du calcul le programme travaille avec des solutions intermédiaires fractionnaires, mais il n'est apparamment pas possible de savoir quelque chose de celles-ci.
Remarque : La numérotation des variables dans le programme commence à 0. Donc s'il y a un 5 dans la case A_{23} (deuxième ligne, troisième colonne), c'est la variable `Number[1,2,4]` qui vaut 1.

5 Création des grilles

L'algorithme de base pour créer des grilles de Sudoku est le suivant :

```
ensemble de toutes les variables: variableSet
grille vide des données: givenNumber
grille vide des chiffres à deviner: nonGivenNumber
tant qu'il y a des cases indéterminées par givenNumber et nonGivenNumber
   ajouter un chiffre aléatoire de variableSet à givenNumber
   supprimer toutes les variables déterminées par le chiffre aléatoire dans
    variableSet, et les ajouter si nécessaire à nonGivenNumber
return givenNumber
```

Le point crucial dans ce programme est de savoir quelles variables sont déterminées. Pour cela j'ai implementé plusieures méthodes. Quelques-unes ont des noms plus ou moins officielles, donc je les ai repris. Les méthodes "X-wing", "swordfish", "burma", "forcing chains" et "colouring" sont expliquées pour des raisons de complétude, je ne les ai pas utilisées.

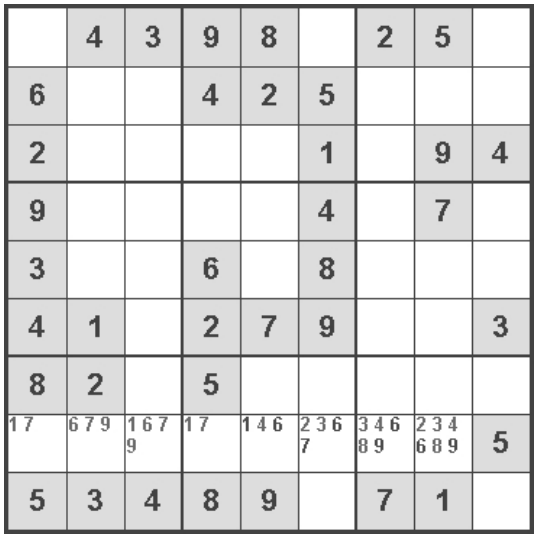

FIG. 2 – Illustration pour la méthode "disjoint set". Trouvé sur [3]

5.1 Méthode "single candidat"

Il y a deux possibilités : *S'il n'y a qu'un candidat restant pour une certaine case, alors le chiffre doit être là* (ce chiffre est parfois appelée "naked single" ou "sole candidat"). *Si une case est la seule qui peut contenir un certain chiffre dans une unité, alors le chiffre doit être là* (ce chiffre est aussi appelée "hidden single" ou "unique candidat"). Cette méthode est la base pour déterminer des variables. C'est presque la seule qui permet de dire que la valeur d'une variable est 1 (sauf "forcing chains", voir plus loin). Les autres méthodes permettent uniquement d'exlure certains chiffres dans certaines cases, donc de dire que les variables associées sont 0.

5.2 Méthode "disjoint subsets"

Le principe de la méthode est le suivant : *Si n cases dans une unité n'ont que n candidats, alors les n chiffres ne sont pas des candidats dans les autres cases de l'unité* (cette partie de la méthode s'appelle aussi "naked subset"). Une variation de ceci est : *Si tous les candidats pour n chiffres se trouvent dans exactement n cases d'une unité, alors tous les autres chiffres dans ces n cases ne sont pas des candidats* (cette deuxième partie de la méthode est aussi appelée "hidden subset").
Illustration (cf. figure 2) : On considère d'abord les quatre premières cases de l'avant dernière ligne. Dans ces cases, seulement les chiffres 1,6,7 et 9 sont candidats. Par le principe des sous-ensembles disjoints ("naked subset"), ces chiffres ne peuvent pas apparaitre dans les cases 5 à 8 de cette ligne.
Dans les cases 5 à 8 de l'avant dernière ligne on voit apparaitre des sous-ensembles disjoints par la variation du principe ("hidden subset") : Les chiffres 2,3,4 et 8 sont uniquement des candidats dans ces quatre cases, donc toutes les autres chiffres ne sont pas candidats dans ces cases.
J'ai implementé deux versions des sous-ensembles disjoints. La première prend une case qui a $n \geq 2$ candidats et essaie de trouver $n-1$ autres cases dans la même unité qui ont ces mêmes ou une partie de ces candidats. La deuxième version cherche n cases où il y exactement n différents candidats, mais où aucun des candidats apparaît dans toutes les n cases.

5.3 Méthode "nishio"

On se pose la question : *Est-ce qu'un certain chiffre dans une certaine case empecherait la grille d'être remplit ? Si la réponse est oui, le chiffre n'est pas un candidat pour cette case.* En utilisant cette méthode, il n'est en principe pas permis d'utiliser d'autres méthodes que "single candidat" pour montrer qu'un certain chiffre dans une case n'est pas possible.
La méthode implementé dans mon programme cherche des nishio dès qu'il y a 16 chiffres donnés et

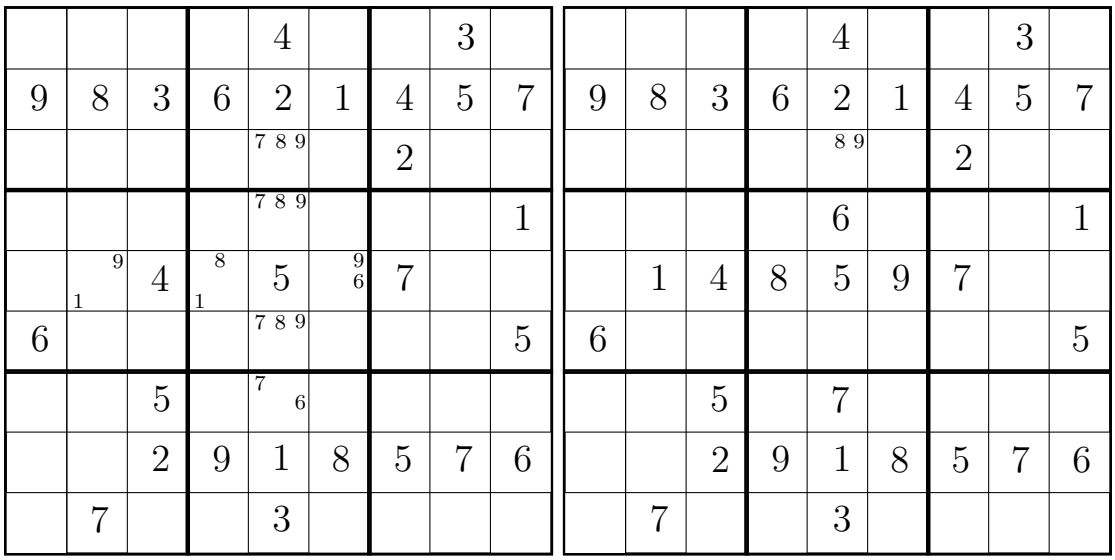

FIG. 3 – Illustration de la méthode "nishio"

seulement dans les cases où il n'y a que deux candidats. Elle utilise toutes les méthodes pour prouver qu'un chiffre ne va pas, donc ce n'est pas vraiment la méthode "nishio", mais aussi une sorte de trial and error (T&E). En fait il y a des gens qui disent que "nishio" est déjà T&E et d'autres qui disent que ce n'est pas le cas parce que même avec la méthode "single candidat", ce qu'on fait finalement est de regarder si un certain chiffre peut être mis ou non, donc nishio fait exactement la même chose un peu plus sophistiquée. Par contre, si toutes les méthodes sont permises pour prouver qu'un chiffre ne peut pas être dans une case, c'est considéré comme T&E (sinon, il n'existait pas de T&E). Le nom de cette méthode provient du constructeur de Sudoku Tetsuya Nishio.
Illustration : Si on choisit de mettre un 7 dans la case A_{75} (5-ème colonne de droite, 3-ème ligne de bas) du Sudoku montré à gauche de la figure 3, on trouve une contradiction : Dans la case A_{65}, il n'y a plus de chiffre qui peut être mis , dans la case A_{35} on devrait mettre le 8 et le 9 (cf. figure 3 à droite).

5.4 Méthode "X-wing"

Règle : *S'il y a seulement deux cases possibles pour un certain chiffre dans chacune de deux colonnes différentes et si ces cases se trouvent sur la même ligne, alors toute autre case dans ces deux lignes ne contient pas le chiffre en question.* De même pour deux lignes avec deux colonnes en commun.
Illustration : Dans la figure 4 il faut qu'il y ait un 9 dans deux des cases marquées par *, donc il n'y a pas de 9 dans les autres cases des colonnes 2 et 9.
Généralisation : Au lieu de prendre uniquement des colonnes et des lignes, on peut aussi prendre les régions. La règle devient : *S'il y a seulement deux cases possibles pour un certain chiffre dans chacune de deux unités différentes de la même sorte et si ces cases se trouvent dans deux autres unités de même sorte, alors toute autre case dans les deux dernières unités ne contient pas le chiffre en question.* Illustration : Dans la figure 4, la valeur 9 doit se trouver dans les cases marquées par *, donc il n'y a pas de 9 dans les cases marquées par /.

5.5 Méthodes "swordfish" et "burma"

Ces méthodes sont encore des généralisations de la méthode "X-wing". La règle est la suivante : *S'il y a N colonnes avec seulement deux cases possibles pour un certain chiffre, si ces cases se trouvent sur exactement N lignes communes et chacune des lignes en a au moins deux, alors le chiffre ne peut pas apparaitre dans une autre case des N lignes.* On peut inverser les rôles des colonnes et des lignes. Si $N = 2$, c'est la méthode "X-wing". Pour $N = 3$, la configuration décrite s'appelle "swordfish", "jellyfish" pour $N = 4$ et "squirmbag" pour $N = 5$.

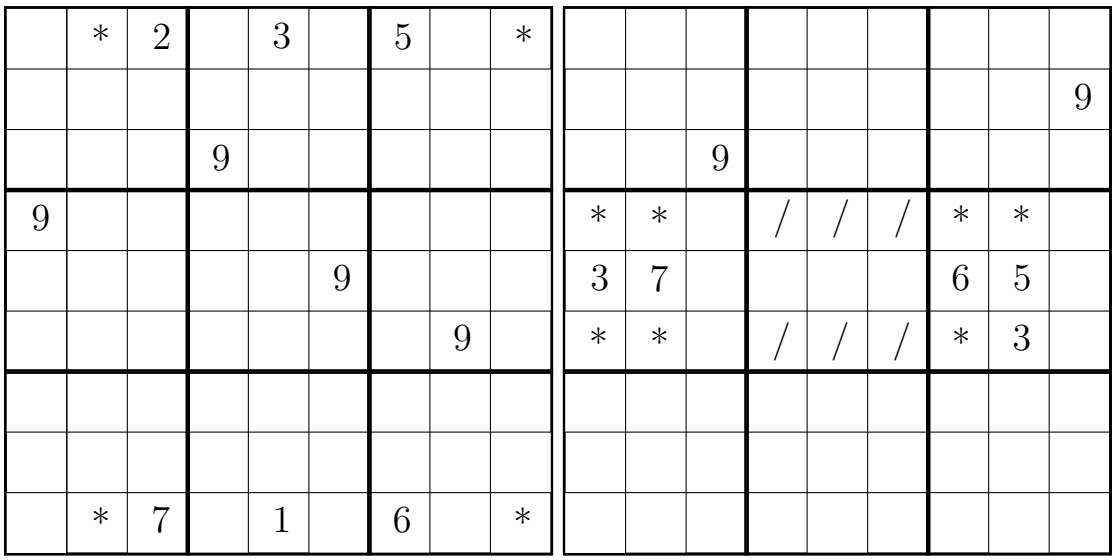

FIG. 4 – Illustration de la méthode “X-wing” à gauche, de la généralisation à droite

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

FIG. 5 – Illustration de la méthode “forcing chains”

Il existe encore une généralisation de ceci. Au lieu de chercher deux cases qui permettent contenir un certain chiffre, on cherche après $C \leq N$ cases. Cette méthode s’appelle “burma”.

5.6 Méthode “forcing chains”

Cette méthode permet de déduire le chiffre d’une case et non seulement d’exclure certains chiffres. Elle est aussi appelée “double-implication chains”. On considère une case où il y $N \geq 2$ candidats restants. On met le premier candidat et déduit quelques chiffres dans des cases voisines (par “single candidat”), puis on fait la même chose avec les autres candidats. A la fin on compare les chiffres déduits et regarde s’il y a des cases où toujours le même chiffre apparaissait. Ce chiffre doit alors être là. Illustration : Dans le Sudoku de la figure 5, on met d’abord le 2 dans la case A_{12} et on remarque qu’on obtient 2 dans la case A_{51}. Puis on met le 7 dans la case A_{12} et on remarque qu’on obtient toujours 2 dans la case A_{51}. Comme il n’y a pas plus de candidats dans A_{12}, on déduit que A_{51} contient 2.

2	6	5	$_{1}$	1^{*}	$_{1}$	9	8	3
7	4	8	6	3	9	$_{1}$	$_{1}$	2
3	1	9	5	8	2	4	7	6
6		1^{*}	3			$1/$	9	8
$1/$		4	1^{*}	2	8	6	3	5
8	5	3	9	$1/$	6	$_{1}$	$_{1}$	4
		6	8					1
$_{1}$	8	$1/$	X	9	$_{1}$		6	7
$_{1}$	3	7	2	6	$_{1}$	8	4	9

FIG. 6 – Illustration de la méthode “colouring”

5.7 Méthode “colouring”

Si une unité n’a que deux cases avec un certain candidat, ces cases sont appelées conjuguées ou fortement liées. Ceci parce que si on sait que le candidat apparait ou n’apparait pas dans l’une des cases, on le sait automatiquement aussi pour l’autre. “Colouring” est basée sur des cases fortement liées. On cherche des chaînes de cases fortement liées, c’est-à-dire deux chaînes de cases où soit toutes les cases de l’une contiennent le chiffre et aucune de l’autre, soit l’envers. Toutes les cases qui ont une case de chacune des chaînes en même unité ne peuvent pas contenir le chiffre en question. La méthode s’appelle “colouring” parce que si on colorise les différentes chaînes, on voit rapidement les chaînes fortement liées et les cases qui ont une case de chaque chaîne en commune. Illustration : Les cases marquées avec $1*$ dans la figure 6 forment une chaîne, les cases marquées avec $1/$ une autre qui est fortement liée. La case A_{84} se trouve sur la même colonne comme une case $1*$ et sur la même ligne comme une case $1/$, donc elle ne peut pas contenir 1. Dans l’illustration, les $1*$ et les $1/$ sont des candidats et les autres chiffres candidats ne sont pas écrits.

5.8 Autres méthodes

Si on sait qu’un chiffre doit se trouver dans une certaine colonne ou ligne d’une certaine région, alors il ne sera pas dans les autres cases de la colonne ou ligne. Illustration : Dans la figure 7, le 7 doit être dans la colonne 2. Donc on peut éliminer le 7 de toutes les cases marquées avec *. Pour trouver de telles structures lors de la création d’une grille, je regarde si une équation est une partie d’une autre. En équations, l’illustration donnée précédemment se traduit comme suit :
Dans la région :

$$x_{127} + x_{327} = 1$$

Dans la deuxième colonne :

$$x_{127} + x_{327} + x_{427} + x_{527} + x_{627} + x_{727} + x_{827} + x_{927} = 1$$

Donc on trouve que les variables $x_{427}, x_{527}, x_{627}, x_{727}, x_{827}$ et x_{927} sont toutes égales à 0.
Pour plus d’exemples des méthodes données dans cette section, le lecteur est invité à consulter [4].
Mon programme ne réussit pas à déterminer toutes les variables qui pourraient être déterminées. Donc il peut arriver que la variable choisie aléatoirement a déjà été déterminée. Dans ce cas, le programme remarque l’erreur, met la variable choisie à 0 et prend une autre. Je pourrais éviter ceci en considérant dans l’application de la méthode “nishio” non seulement les cases à deux candidats mais toutes. Le programme devient alors très lourd, et en plus, ceci n’aboutit que rarement à la détermination d’une

3		6						
				7				
4		5						
	*							
	*							
	*							
	*							
	*							
	*							

FIG. 7 – Le 7 peut être éliminé des cases marquées avec *.

variable et la grille ainsi créée est difficile à résoudre. Remarque : les variables dans le programme sont numérotées de 0 à 728. $x_{111} = 0, x_{112} = 1, x_{121} = 9$ et $x_{211} = 81$, le reste par logique.

6 Définition de la difficulté d'un Sudoku

Sur l'internet, j'ai trouvé essentiellement deux propositions pour définir la difficulté d'un Sudoku. La première consiste à écrire un programme qui n'utilise que des méthodes très simples, puis à compter le nombre de fois qu'on doit deviner pour remplir la grille. Le plus grand problème avec cette méthode est que le nombre de chiffres qu'il faut deviner dépend de l'emplacement de ces chiffres. Il y a des cases qui sont plus importantes pour la suite de la résolution que d'autres. En plus, on ne sait rien sur la difficulté de trouver le chiffre qui à été deviné.
La deuxième possibilité pour définir la difficulté regarde, combien de méthodes différentes on doit utiliser pour résoudre le Sudoku. On fait un classement des difficultés des méthodes, puis on utilise toujours la méthode la plus simple pour trouver le prochain chiffre. On note toutes les méthodes utilisées et le nombre de méthodes utilisées et leur difficulté décide après sur la difficulté générale du Sudoku considéré. Il y a deux problèmes : D'abord il n'existe pas d'ordre général des difficultés des méthodes. Certaines gens trouvent la méthode A plus simple que la méthode B, chez d'autres c'est à l'envers. Puis, il est parfois possible d'utiliser plusieures méthodes une fois ou une méthode plusieures fois pour trouver les mêmes chiffres. Et parfois, il n'y a qu'une seule méthode pour trouver le prochain chiffre, donc même si cette méthode n'est pas très difficile, il faut déjà la trouver.

7 Résoudre des Sudoku en floating point

Le programme linéaire qui résolu des Sudoku utilise des nombres entiers. Après avoir remplacé les entiers par des floats, j'ai fait résoudre CPLEX le Sudoku donné comme exemple en figure 1. Dans la solution fractionnaire, il y avait alors des cases où deux chiffres apparaissaient à 50%. Le résultat est dans la figure 8 à gauche. Deux chiffres dans la même case veut dire que les deux chiffres sont à 50% là. Les chiffres entourés sont faux comparés ceux de la solution entière (qui se trouve en figure 9). En imposant la valeur 2 dans la case A_{11}, on obtient la solution dans la figure 8 à droite. Par contre, en imposant la valeur 6 dans la case A_{75}, on obtient la solution entière.
Sur un autre Sudoku, j'ai fait la même chose et j'ai obtenu une solution qui contient des valeurs apparaissant à 33% ou à 66%. Le résultat est sur la figure 10 à gauche. Dans cette solution fractionnaire il n'y a pas de chiffre "faux", donc de chiffre qui apparait à 100% mais qui ne se trouve pas là dans

27	26	17	58	4	59	16	3	89
9	8	3	6	2	1	4	5	7
45	45	16	③	89	⑦	2	16	89
57	35	89	47	68	24	36	29	1
23	19	4	18	5	69	7	68	23
6	12	78	12	79	③	89	④	5
1	69	5	27	67	24	38	89	34
34	34	2	9	1	8	5	7	6
8	7	69	45	3	56	19	12	24

2	56	17	78	4	59	16	3	89
9	8	3	6	2	1	4	5	7
57	4	16	16	89	37	2	16	89
57	25	78	23	68	34	69	49	1
3	19	4	18	5	69	7	68	2
6	12	89	14	78	27	3	48	5
⑧	69	5	47	67	24	19	12	3
4	3	2	9	1	8	5	7	6
①	7	69	25	3	56	⑧	29	4

FIG. 8 – Solutions fractionnaires du Sudoku en figure 1. Les chiffres entourés sont faux par rapport à la solution entière (figure 9).

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

FIG. 9 – Solution entière du problème donné en figure 1.

113	2	6	449	335	7	149	559	8
8	9	5	6	2	114	144	7	3
133	7	4	399	8	155	266	225	169
4	5	7	1	9	3	8	6	2
9	8	3	2	4	6	5	1	7
6	1	2	5	7	8	339	399	4
2	366	9	334	1	445	7	8	556
5	4	8	7	366	9	122	233	116
7	336	1	8	556	2	369	4	599

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

FIG. 10 – Sur la grille à gauche on voit une solution fractionnaire du problème donné par les chiffres entourés de la grille à droite. A droite on a aussi la solution entière.

la solution entière. A droite de la même figure on voit la solution entière, et les chiffres donnés sont entourés. En imposant par exemple 1 en A_{11} ou 4 en A_{14} on obtient toujours des solutions fractionnaires, en imposant 1 en A_{37} on obtient la solution entière.

8 Conclusion

La résolution du Sudoku à l'aide de CPLEX s'est avérée beaucoup plus facile qu'on pensait. A cause de ceci j'ai eu plus de temps pour écrire un programme qui génère des Sudoku. Finalement, ce programme sait aussi résoudre des Sudokus et il m'a montré qu'il n'est pas facile à aboutir à cela. Donc même si CPLEX arrive aisément à résoudre le problème donné comme programme linéaire la solution n'est pas du tout triviale, c'est plutôt CPLEX qui est bien fait. Ce qu'on pourrait faire est d'essayer de traduire toutes les méthodes en équations. J'ai essayé de faire ça lorsque je rencontrait une méthode plus difficile pour la première fois et je n'ai pas réussi, mais je crois qu'il est faisable. Je pense par contre que l'application de ces équations ne sera pas très efficace, qu'il faudrait tester trop de cas chaque fois qu'on a ajouté un chiffre.
Une autre possibilité pour générer des grilles de Sudoku est de remplir une grille complétement est d'enlever des chiffres après.
Comme une des dernières tentatives du projet j'ai essayé de visualiser le polyèdre de toutes les solutions possibles d'un Sudoku, donc si on impose seulement les règles de base. Pour ceci j'ai utilisé le programme scdd_gmp de cddlib. Malheureusement, même avec une grille 4x4 le programme était loin d'aboutir à la représentation des sommets qu'on aimerait avoir (il y a 288 sommets). La taille des entrées semble être beaucoup trop élevée.
Je remercie Christophe Weibel pour l'idée générale du modèle, ses conseils et son enthousiasme. Aussi à cause de ses "belles théories" et les choses qu'on a fait "parce que c'est marrant". Je remercie M. Prodon pour sa suggestion d'utiliser OPL et CPLEX, sa patience et ses propositions.

Références

[1] B. Felgenhauer and F. Jarvis. Enumerating possible sudoku grids. `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf`, 2005.

[2] E. Russel and F. Jarvis. There are 5472730538 essentially different sudoku grids... and the sudoku symmetry group. `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html`.

[3] http://users.pandora.be/vandenberghe.jef/sudoku/index.html?how2solve.

[4] http://www.simes.clara.co.uk/programs/sudokutechniques.htm.