Rapport INF402

Jizong Zhan/Dorian Thivolle/Timon Roxard/Lilian Russo

Mai 2020

Sommaire

1 Introduction

Ce document présentes le projet complet du solveur sat, réalisé en python réalisé pendant la deuxième partie du quatrième semestre de la licence Informatique

1.1 Rappel

Le jeu du takuzu consiste a remplir une grille carré avec des 0 et des 1 suivant 3 règles :

1. Il doit y avoir autant de 0 que de 1 sur chaque lignes et chaque colonnes
2. Il ne peut pas avoir plus de deux fois a la suite le même chiffre sur chaque ligne et chaque colonnes
3. Toutes les lignes et les colonnes sont différentes entre elles

2 Modélisation des règles en FNC

Pour obtenir une forme normale conjonctive, on peut utiliser une double négation. En effet les règles précise les formes de suites qui ne sont pas possibles. On peut donc spécifier tout les cas impossibles et faire la négation de tout. Par les lois de De Morgan on arrive a une forme normale conjonctive. On va répéter cette logique deux fois : une fois pour les lignes et une fois pour les colonnes.
Pour ce projet on prend le prédicat $P_{i,j}$ signifiant qu'il y a un 1 dans la case à la ligne i et à la colonne j et $\overline{P_{i,j}}$ signifiant qu'il y a un 0 dans la case à la ligne i et à la colonne j

2.1 Cas des règles 1 et 2

Ces deux règles peuvent être traitée ensembles étant donné qu'elles apporte une restriction sur la forme des assignassions (un nombre binaire codé sur n bit, n étant la longueur/largeur de la matrice) possibles. Le tableau stockant toutes ces assignassions représente donc toutes les possibilité de combinaisons de 0 et de 1 sur n bits. On peut simplifier cela en observant que ce tableau représente les nombres de 0 a 2^n en binaire.

De ce tableau doit être déduit toutes les combinaisons qui ne respectes pas les deux premières règles que l'on stocke dans un second tableau. Par exemple, avec une grille de 4x4 : 1110,1101,1000,...

Nous avons donc toutes les combinaisons fausse possibles pour notre grille, il faut maintenant l'écrire sous une forme compréhensible par le solveur sat.
Nous utiliseront pour cela une fonction permettant de donner le numéro de la case comme si la grille étais en une dimension (nommé index). On arrive a la forme suivante (en considérant m comme la longueur d'une ligne/colonne) :

Pour les lignes

$$\forall i \in \{1;n\} : (P_{i,1} + P_{i,2} + ... + P_{i,m}) \cdot (P_{i,1} + P_{i,2} + ... + \overline{P_{i,m}}) \cdot ... \cdot (\overline{P_{i,1}} + \overline{P_{i,2}} + ... + \overline{P_{i,m}})$$

Pour les colonnes

$$\forall j \in \{1;n\} : (P_{j,1} + P_{j,2} + ... + P_{j,m}) \cdot (P_{j,1} + P_{j,2} + ... + \overline{P_{j,m}}) \cdot ... \cdot (\overline{P_{j,1}} + \overline{P_{j,2}} + ... + \overline{P_{j,m}})$$

2.2 Cas de la règle 3

Cette dernière se différencie car elle concerne des comparaisons entres clauses. Ici on utilise toujours la double négation pour avoir une forme normale conjonctive. Pour cela, on itère pour chaque couple de ligne une formule qui comporte chaque possibilité de lignes identiques
Par exemple : pour deux lignes de longueur 4, on aura :
$(P_{i,1} + P_{i,2} + P_{i,3} + P_{i,4}) \cdot ... \cdot \overline{(P_{i,1} + P_{i,2} + P_{i,3} + P_{i,4})}$ pour les lignes (on remplace i par j pour les colonnes)

2.3 Cas des cases pré remplies

Pour inclure un littéral pré remplis dans la grille de Takuzu, il suffit de le mettre seul sur une ligne, il sera donc obligé d'être juste.

3 Format des fichier .tak

Les fichier ont 2 partie distinctes :

1. La première ligne représente la taille de la grille. Une valeurs suffit étant donné qu'elle est carrée
2. Les lignes d'en dessous sont du même nombre et de la même taille que le chiffre donné en première ligne. Sur chaque ligne on peut trouver :
   - un 0 : valeur prédéfinie
   - un 1 : valeur prédéfinie
   - un _ : valeur indéfinie

4 Algorithmes

4.1 Algorithme principal (écriture dans un fichier dimacs)

```
function ecrireDimacs(file_path: str){
    en ouvrant le fichier file_path en ecriture:
        file_path.write(Le nombres de variables et de lignes)
        pour chaque case de la guille donné: //On remplis d`abord le fichier avec les
    valeurs déjà donnée dans la grille
            si la case est un 0:
                on ajoute le numéro de l`index + 1 précédé d`un -
            si la case est un 1:
                on ajoute le numéro de l`index + 1
        pour chaque possibilité de combinaison : // On prend toutes les combinaisons
    impossibles
            si elle ne respecte pas la regle 1 ou la règle 2:
                la possibilité est ajoutée a la liste impossible

        on créer un tableau de toutes les combinaisons de valeurs possibles
        on créer deux tableau distinct : l`un pour les lignes et l`autre pour les colonnes (
    appelé tab_ligne et tab_colonne)
        on remplis ces tableaux avec toutes les combinaisons d`index possibles

        //Écriture des deux premières règles
        pour a dans la liste d`exclusion:
            pour b dans tab_ligne:
                pour n allais de 0 a la longueur/largeur de la grille:
                    si a[n] = "1":
                        on écrit - dans le fichier
                    on écrit b[n] + 1 dans le fichier
                on écrit 0 et un saut à la ligne dans le fichier

            pour b dans tab_colonne:
                pour n allais de 0 a la longueur/largeur de la grille:
                    si a[n] = "1":
                        on écrit - dans le fichier
                    on écrit b[n] + 1 dans le fichier
                on écrit 0 et un saut à la ligne dans le fichier

        //Écriture de la troisième règle
        pour a dans la liste des possibilité
            pour n allant de 0 a longueur(tab_ligne):
                pour m allant de n+1 a longueur(tab_ligne):
                    pour i allant de 0 a la longueur/largeur de la grille:
                        si a[j] = 0:
                            on écrit -(tab_ligne[n][i] +1) dans le fichier
                            on écrit -(tab_ligne[m][i] +1) dans le fichier
                        sinon :
                            on écrit tab_ligne[n][i] +1 dans le fichier
                            on écrit tab_ligne[m][i] +1 dans le fichier
                    on écrit 0 et un retour a la ligne dans le fichier

            pour n allant de 0 a longueur(tab_colonne):
                pour m allant de n+1 a longueur(tab_colonne):
                    pour i allant de 0 a la longueur/largeur de la grille:
                        si a[j] = 0:
                            on écrit -(tab_colonne[n][i] +1) dans le fichier
                            on écrit -(tab_colonne[m][i] +1) dans le fichier
                        sinon :
                            on écrit tab_colonne[n][i] +1 dans le fichier
                            on écrit tab_colonne[m][i] +1 dans le fichier
                    on écrit 0 et un retour a la ligne dans le fichier
```

```
}

//Fonction permettant de savoir si il y a autant de 0 que de 1 dans une chaîne passé en paramètre
//Exemple : verif_ligne_col("100101") -> True
//Exemple : verif_ligne_col("111100") -> False
function verif_ligne_col(tab:str){
    on déclare un compteur de 0 et un compteur de 1 (Z_count et O_count)

    pour chaque éléments i dans tab:
        si i = 0:
            on incrémente Z_count
        sinon:
            on incrémente O_count
    on retourne Z_count == O_count

}

//Permet de vérifier si il y a 3 caractère a la suite dans une chaine donnée en paramètre
//Exemple : verif_suite("00101") -> True
//Exemple : verif_suite("001110") -> False
function verif_suite(tab:str){
    pour i allant de 0 a longueur(tab)-2:
        si tab[i] = tab[i+1] et tab[i+1] = tab[i+2]:
            on renvois True
    on renvois False
}

//Renvois l'index d'une case se trouvant a la ligne j et a la colonne i
//Exemple (taille de 4) : index(2,3) -> 14
function index(i:int, j:int){
    on retourne i + j*longueur/largeur de la grille
}
```

4.2 Algorithme secondaire (lecture du fichier contenant la grille de Takuzu)

```
On initialise la taille a 0 (size)
On initialise un tableau vide (tab)
Si le fichier donné en paramètre n`est pas bon :
    on retourne un message d'erreur et on quitte le programme

tant que le fichier passé en paramètre est ouvert en lecture
    on alloue la première ligne a la variable size

    si size n'est pas un nombre ou que size est négatif :
        On quitte le programme avec un message d`erreur

    pour i allant de 0 a size - 1:
        On lit un nouvelle ligne que l`on stocke dans la variable ligne

        si la ligne est trop petite:
            On quitte le programme avec un message d`erreur

        si un caractère de la ligne n`est ni un 0 ni un 1 ni un _:
            On quitte le programme avec un message d`erreur
        On ajoute la ligne au tableau tab
On défini une liste de toutes les combinaisons possibles. Donc on créer un tableau
    list_possible qui contiendra les valeurs en binaires de 0 a 2^size
```

5 Implémentions des algorithmes

5.1 Choix du langage

Comme dit au début de ce document, nous avons choisit le python pour notre projet. Étant donné les nombreuses opération et affectation de tableau, un langage haut niveau nous semblais le plus approprié. Toutes les librairies incluse dans le projet sont des librairies standard de python. Veillez toutefois a bien posséder une version de python a jours (3.7 ou supérieur)

5.2 Choix du solveur

Nous utilisons minisat, qui est simple d'utilisation et facile à installer

```
sudo apt-get install -y minisat
```

6 Test sur le programme final

6.1 Test de performance

Un script python est dédié au test de performance. Pour l'utiliser il suffit de remplacer le nom du fichier main.py par performance.py. Par exemple

```
python3 main.py 1
```

deviens

```
python3 performance.py 1
```

Le programme met dans les 51 ms pour générer un fichier dimacs d'une grille 6x6, 166ms pour du 8x8 et on monte jusqu'à 20s pour du 13x13

6.2 Test de robustesse

Des fichiers erroné pour chaque cas ont été placé, commençant par la lettre W. Aucune erreur n'est à déploré

6.3 Vérification des résultats

Les fichiers dimacs créer sont ensuite passé dans le programme minisat, permettant de trouver ou non une solution au problème. Les fichiers sont stocké dans le dossier out/

7 Conclusion sur le projet

La réalisation de ce programme a été un bon challenge pour nous, surtout la 3ème règle qui nous a demandé plus de temps que prévu. Toutefois le programme est parfaitement fonctionnel et nous en sommes plus que content.