SAT-solitaire

Cordeiro Fonseca Loïc - Bourgeois Noé

November 2021

Contents

1 Introduction

Le Solitaire est un jeu de plateau classique à un seul joueur. Le plateau de jeu comporte des trous dans lesquels on peut mettre des billes. Pour jouer un coup au Solitaire, il faut choisir trois cases adjacentes, qui sont toutes les trois sur une même ligne ou une même colonne du plateau. De surcroît, exactement deux billes doivent être posées sur les trois cases, dont une posée sur la case du milieu. On fait "sauter" la bille qui se trouve sur une des extrémités par dessus la bille du milieu, pour la faire atterrir dans le trou qui se trouve à l'autre extrémité. On enlève alors la bille du milieu du plateau. Nous pouvons représenter une configuration du jeu par un entier positif n, et une matrice de taille n $\times$ n dont les éléments appartiennent à -1, 0, 1. Si la valeur de la case (i, j) de la matrice est égale à -1, alors il n'y a pas de trous sur le plateau à la position (i, j); si cette valeur est égale à 0, alors il y a un trou sur le plateau 'a la position (i, j), mais il n'y a pas de bille posée dessus; si cette valeur est égale à 1, alors il y a un trou sur le plateau à la position (i, j) avec une bille posée dessus.

2 Tests de satisfiabilité

2.1 Données générales

- Soit M une matrice de taille L (nombre de lignes) * C (nombre de colonnes) dont les éléments sont dans {-1, 0, 1}. Ceci est la matrice de départ.
- Soit M' une matrice de taille L (nombre de lignes) * C (nombre de colonnes) dont les éléments sont dans {-1, 0, 1}. Ceci est notre matrice finale souhaitée.
- S, le nombre d'étapes nécessaires pour atteindre la configuration désirée.
- s, l'étape actuelle, de la partie, $s \in \{1, ..., S\}$. Lors d'une partie, on joue forcément un seul coup par étape, et chaque coup élimine exactement une bille. S est donc égal à la différence de billes entre le plateau original et le plateau final (ou le plateau final désiré).
- Notons D, l'ensemble contenant les tuples directionnels, tel que $D = \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$
- d, un tuple dans D la direction du mouvement d'un coup, avec d_0 le premier élément du tuple et d_1 le second.
- f, est la fonction de transition. Elle prend en paramètres les coordonnées et le coup à jouer, ainsi que l'état du plateau pour lequel le coup doit être joué. Cette fonction vérifie alors la légalité du coup suivant les règles du jeu, renvoyant un nouveau tableau dans le cas d'un coup légal et 'None' sinon.

2.2 Variables Booléennes

On commence par définir les variables de nos formules.
La variable $E_{\iota,s}$ représente un état du tableau identifié par ι à l'étape s.
Elle vaut "Vrai" si on considère le tableau ι à l'étape s pour satisfaire notre formule.
On considère que $\iota \in I$, l'ensemble des identificateurs de tous les états du tableau possible et I_s l'ensemble des états du taleau possibles à l'étape s.
La variable $m_{i,j,d,s}$ représente un coup joué depuis (i, j) vers d à l'étape s.
Elle vaut "Vrai" si on considère la bille à déplacer dans la direction d, à la ligne i, colonne j à l'étape s pour satisfaire notre formule.

2.3 Clauses relatives au jeu

Nous avons 5 clauses générales définissant le fonctionnement du jeu. Une d'entre elles a des conditions spéciales.

2.3.1 L'Etat du plateau:

Il y a plusieurs clauses gérant uniquement l'état du plateau.

2.3.1.1 Etat initial et final: Il faut fixer quel état le tableau est sensé avoir au début ainsi qu'à la fin. Pour cela il faut fixer $E_{0,0}$ l'état initial associé à la matrice M et $E_{-1,S}$ l'état final associé à la matrice M'. Vu qu'on ne sait pas combien d'états intermédiaires seront construits on donne simplement l'id "-1" à l'état final, afin d'éviter tout chevauchement.
Pour chacun de ces états on rajoute simplement une clause confirmant l'état, soit:
$K_1 = E_{0,0} \wedge E_{-1,S}$

2.3.1.2 Maxmium un état par étape: En effet, on ne peut considérer qu'un seul état par étape, afin d'avoir une continuation logique lors de la solution. S'il y avait plusieurs états par étape il serait rapidement compliqué d'identifier les bons états menant à l'état final désiré. À cette fin nous avons la clause:

$$K_2 = \bigwedge_{\substack{s\in 1,\dots S \\ \iota\in\{1,\dots,I_s\} \\ \iota'\in\{1,\dots,I_s\} \\ \iota\neq\iota'}} (\neg E_{\iota,s} \vee \neg E_{\iota',s}),$$

2.3.2 Jouer des coups

Il nous faut une clause qui va vérifier la cohérence des coups à jouer à une étape s en fonction de l'état du tableau à cette même étape, ainsi que du tableau résultant de ce coup. Soit à une étape s, on ne peut jouer un coup $m_{i,j,d,s}$ que si l'état $E_{\iota,s}$ contient des billes et des cases libres aux endroits appropriés par rapport au coup (rappel: Pour jouer un coup il faut faire sauter une bille au-dessus d'une autre de telle sorte qu'elle atterrisse dans la case suivante qui doit être vide. Une bille peut se déplacer dans 4 directions: haut, bas, droite et gauche.). On vérifie également que le coup ne sort pas du tableau bien sûr.
À la base on dirait, que si à une étape $s-1$ on joue un coup $m_{i,j,d,s-1}$ sur un tableau $E_{\iota,s-1}$, alors l'état à l'étape suivante sera $E_{f(m_{i,j,d,s-1},E_{\iota,s-1}),s}$ où $f(m_{i,j,d,s-1}, E_{\iota,s-1})$ est la fonction de transition vérifiant la cohérence du coup. Ce qui nous donne la formule:

$$K_3 = \bigwedge_{\substack{s\in 1,\dots S \\ \iota\in I_s \\ i\in\{1,\dots,L\} \\ j\in\{1,\dots,C\} \\ d\in D}} ((m_{i,j,d,s-1} \wedge E_{\iota,s-1}) \rightarrow E_{f(m_{i,j,d,s-1},E_{\iota,s-1}),s})$$

$$K_3 = \bigwedge_{\substack{s\in 1,\dots S \\ \iota\in I_s \\ i\in\{1,\dots,L\} \\ j\in\{1,\dots,C\} \\ d\in D}} (\neg(m_{i,j,d,s-1} \wedge E_{\iota,s-1}) \vee E_{f(m_{i,j,d,s-1},E_{\iota,s-1}),s})$$

$$K_3 = \bigwedge_{\substack{s\in 1,\dots S \\ \iota\in I_s \\ i\in\{1,\dots,L\} \\ j\in\{1,\dots,C\} \\ d\in D}} (\neg m_{i,j,d,s-1} \vee \neg E_{\iota,s-1} \vee E_{f(m_{i,j,d,s-1},E_{\iota,s-1}),s})$$

2.3.2.1 Coups interdits: Toute combinaison de coup et d'état ne menant pas à une configuration légale doit être rendue impossible, éliminant les tableaux et coups obsolètes lors de la résolution de la formule, ce qui force la solution à emprunter le bon chemin jusqu'au tableau final désiré. Pour ce faire, on a la clause suivante:
$K_4 = \neg(m_{i,j,d,s} \wedge E_{\iota,s})$
$K_4 = \neg m_{i,j,d,s} \vee \neg E_{\iota,s}$
Il est important de préciser que cette clause n'est ajoutée que pour les couples de coup et état qui ne retournent rien par la fonction de transition f.

2.3.2.2 Au moins un coup par étape : Il faut jouer au moins un coup par étape, sinon la formule K_3 devient obsolète car en ne jouant aucun coup pour une étape, on peut aussi satisfaire la formule, or si on force un coup à être joué, il doit être cohérent. Cette clause s'écrit ainsi:
$K_5 = \bigwedge_{s\in\{1,\dots,S\}} \left(\bigvee_{\substack{i\in\{1,\dots,L\}\\ j\in\{1,\dots,C\}\\ d\in D}} m_{i,j,d,s} \right)$

2.4 Déroulement d'une partie - Formule FNC

Dans ce projet une partie est un cas général du jeu de table anglais (solitaire). Les clauses définies ci-dessus nous permettent de construire une formule en FNC qui, donnée à un solveur SAT, nous dit s'il est possible de passer d'une configuration M donnée à une autre configuration M' donnée. Joignons donc nos clauses :
$K = K_1 \wedge K_2 \wedge K_3 \wedge K_4 \wedge K_5$
$$K = E_{0,0} \wedge E_{-1,S} \wedge \bigwedge_{\substack{s\in 1,\dots S\\ \iota\in\{1,\dots,I_s\}\\ \iota'\in\{1,\dots,I_s\}\\ \iota\neq\iota'}} (\neg E_{\iota,s} \vee \neg E_{\iota',s}) \wedge \bigwedge_{\substack{s\in 1,\dots S\\ \iota\in I_s\\ i\in\{1,\dots,L\}\\ j\in\{1,\dots,C\}\\ d\in D}} (\neg m_{i,j,d,s-1} \vee \neg E_{\iota,s-1} \vee E_{f(m_{i,j,d,s-1},E_{\iota,s-1}),s}) \wedge$$
$$(\neg m_{i,j,d,s} \vee \neg E_{\iota,s}) \wedge \bigwedge_{s\in\{1,\dots,S\}} \left(\bigvee_{\substack{i\in\{1,\dots,L\}\\ j\in\{1,\dots,C\}\\ d\in D}} m_{i,j,d,s} \right)$$
Notez que la clause K_4 interdisant des coups a été noté une seule fois au sein de cette fonction. Cela est dû aux conditions un peu spéciales de l'ajout d'une telle clause, prenant en compte l'ajout passé d'une autre clause.

3 Code

Cette section porte sur le code utilisé pour construire les formules en FNC, et par extension aussi sur les réponses aux questions de l'énoncé. Comme demandé, le code est dans un fichier nommé `projet.py` et est en Python 3.
Le programme marche pour les 9 premiers et le dernier couple, mais a un problème de mémoire pour le 10ième et s'y plante.

3.1 Librairies utilisées

Les librairies utilisées dans les différents programmes sont les suivantes:

- pysat.solvers (Glucose4)
- pysat.formula
- colorama
- numpy
- copy

3.2 Dernière bille

(Question 3) Pour toute configuration de la Figure 1 (sauf la 6 et la 4), on veut savoir s'il est possible d'atteindre une configuration du même plateau mais qui contient une seule bille là où au départ il n'y avait pas de bille. Dû à un manque de mémoire de nos machines, nous avons décidés de faire ces tests sur des version raccourcies de ces matrices que nous donnerons ci-dessous.
Le code pour l'exécution de ces matrices se trouve dans le fichier `testQ3.py` se trouvant dans le même dossier que le reste des fichiers relevant au projet. Pour faire tourner les tests il faut faire de la même manière que pour le fichier `test.py` qui nous avait été donné.

Pour la première matrice, nous avons comme résultat qu'elle n'est pas solvable.

$$M_1 = \begin{pmatrix} -1 & -1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 \end{pmatrix}$$

Pour la seconde matrice, nous avons comme résultat qu'elle n'est pas solvable.

$$M_2 = \begin{pmatrix} -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 & -1 \end{pmatrix}$$

Pour la troisième matrice, nous avons comme résultat qu'elle [] solvable.

$$M_3 = \begin{pmatrix} -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \end{pmatrix}$$

Pour la quatrième et dernière matrice, nous avons comme résultat qu'elle [] solvable.

$$M_4 = \begin{pmatrix} -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 \end{pmatrix}$$

3.2.1 Peu importe où

(Question 4) Même question que précédemment mais on ne demande pas que la dernière bille soit placée là où il y avait le trou de départ: on demande simplement qu'il y ait dans la configuration finale exactement une bille, peu importe sa position. Tout comme pour la question précédente, pour des raisons de mémoire, nous avons décidés de raccourcir les matrices. Nous utiliserons donc pour ces tests-ci les mêmes matrices que précédemment et afin de gagner de la place nous n'allons pas le re-écrire ici, mais nous donnerons la position de la bille de fin et dirons si le problème est solvable avec cette configuration ou non.
Le code pour l'exécution de ces matrices se trouve dans le fichier `testQ4.py` se trouvant dans le même dossier que le reste des fichiers relevant au projet. Pour faire tourner les tests il faut faire de la même manière que pour le fichier `test.py` qui nous avait été donné.

3.2.2 Stratégie de placement de la première bille

(Question 5) On veut trouver une façon de ne pas placer une bille dans la matrice M (basée sur la matrice 1 de la figure 1 de l'énoncé), de manière à ce qu'on puisse trouver une solution telle que la matrice résultante soit la matrice M' de même taille où il n'y a qu'une bille dans la case où il n'y en avait pas à l'origine. Pour des raisons de capacités de calcul nous allons nous limiter à une matrice de taille 5x5, M ainsi que M', prendraient donc une forme comme ci-dessous.
Le code pour l'exécution de ces matrices se trouve dans le fichier `testQ5.py` se trouvant dans le même dossier que le reste des fichiers relevant au projet. Pour faire tourner les tests il faut faire de la même manière que pour le fichier `test.py` qui nous avait été donné.

$$M = \begin{pmatrix} -1 & -1 & 0 & -1 & -1 \\ -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 \end{pmatrix}$$

$$M' = \begin{pmatrix} -1 & -1 & 1 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & -1 & -1 \end{pmatrix}$$

Notons R la matrice de forme:

$$R = \begin{pmatrix} -1 & -1 & b_1 & -1 & -1 \\ -1 & b_2 & b_3 & b_4 & -1 \\ b_5 & b_6 & b_7 & b_8 & b_9 \\ -1 & b_{10} & b_{11} & b_{12} & -1 \\ -1 & -1 & b_{13} & -1 & -1 \end{pmatrix}$$

Contenant les résultats récupérés du solveur où b_i vaut 1 si le solveur a trouvé le problème satisfiable, et 0 sinon. Après avoir fait tourner le solveur la matrice R nous donne:

$$R = \begin{pmatrix} -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & -1 & -1 \end{pmatrix}$$

Soit pour aucune configuration de plateau avec une case laissée vide à l'origine, ne trouve-t-on une solution où aucune case de la matrice n'a de bille sauf celle originellement laissée vide.