Complexité du jeu de Sokoban

TIPE pour les Écoles Normales Supérieures

Jonathan Laurent

18 Juin 2012

Résumé

Introduit dans les années 1980, le jeu de sokoban nous propose d'assister un magasinier débordé dans le rangement de son entrepôt. Malgré une simplicité descriptive remarquable, ce casse tête donne lieu à une richesse stratégique édifiante et aucune intelligence artificielle n'est encore parvenue, à ce jour, à égaler les réflexes d'un bon joueur.

Nous nous intéresserons au problème qui consiste à déterminer si un niveau de sokoban admet ou non une solution. Plus particulièrement, nous apporterons une démonstration originale d'une limitation théorique mise à jour par Culberson en 1997 : la pspace-complétude du sokoban [1].

Table des matières

1 Quelques rudiments de théorie de la complexité

Dans cette partie, on établit un rapide inventaire des points théoriques et des notations sur lesquels s'appuient les démonstrations qui suivront. Une présentation plus détaillée est disponible en [3].

1.1 Machines de Turing et problèmes de décision

Un problème de décision est totalement déterminé par le langage constitué des chaînes binaires codant pour une instance positive du problème. Dans un souci de concision, on identifiera par la suite *problèmes* et *langages*, ainsi qu'*instance* d'un problème et *encodage* de cette instance sur l'alphabet binaire $\{0,1\}$, sur lequel l'ensemble des chaînes sera noté $\mathbb{B}$.

On introduit naturellement le langage noté SOK, qui décrit l'ensemble des niveaux de sokoban admettant une solution, dans un sens que l'on précisera par la suite. Résoudre le problème SOK, c'est disposer d'un moyen algorithmique permettant de décider en temps fini de l'appartenance d'une chaîne binaire à ce langage. On formalise la notion de *procédure algorithmique* à l'aide d'un modèle de calcul. Nous utiliserons celui des *machines de Turing déterministes*. Avec les notations classiques :

une machine de Turing est un sextuplet $(Q, \Sigma, \Gamma, q_0, F, \delta)$ où
— Q est un ensemble fini d'états
— Σ est un alphabet fini dit *alphabet d'entrée*. On prendra $\Sigma = \{0,1\}$
— $\Gamma = \Sigma \cup \{\sqcup\}$ est l'alphabet de travail. Le caractère spécial $\sqcup$ est dit *blanc*
— q_0 est l'état initial
— $F \subset Q$ est l'ensemble des états bloquants
— $\delta : \Gamma \times (Q \setminus F) \longrightarrow \Gamma \times Q \times \{\leftarrow, \rightarrow\}$ est dite *fonction de transition*

En outre, on se limite sans perte de généralité à des machines pour lesquelles $F = \{q_+, q_-\}$, où q_+ est l'*état acceptant* et q_- l'*état rejetant*. Un langage L est dit *décidé* par une machine de Turing M lorsque M s'arrête en temps fini pour toute chaîne binaire qui lui est fournie en entrée, acceptant les chaînes de L et rejetant les autres.

1.2 Complexité en temps et en espace

À une machine de Turing M on associe sa fonction de complexité en temps $T_M : n \mapsto \sup_{|x| \leqslant n} t_M(x)$ où $t_M(x)$ représente le nombre de transitions effectuées par M avant arrêt sur l'entrée x. Une machine de Turing M est dite polynomiale en *temps* s'il existe $k \in \mathbb{N}$ tel que $T_M(n) = O(n^k)$. De même, on dit d'une machine de Turing qu'elle est polynomiale en *espace* si elle s'arrête pour toute entrée x après consommation d'une longueur de bande majorée par une fonction polynomiale en $|x|$.

On note P la classe des problèmes décidés par une machine de Turing polynomiale en temps et PSPACE la classe des problèmes décidés par une machine polynomiale en espace.

1.3 Relation de difficulté sur les langages

On définit sur l'ensemble des langages la relation $\prec_P$. On note $L \prec_P L'$ lorsque il existe $f : \mathbb{B} \longrightarrow \mathbb{B}$ *fonction calculable* polynomiale telle que $\forall x \in \mathbb{B},\ x \in L \iff f(x) \in L'$. On dit également que L est réductible en temps polynomial à L', ou encore que L est plus facile que L'. Un langage L est dit *difficile* pour une classe de complexité C si $\forall M \in C,\ M \prec_P L$. Si, de surcroît, $L \in C$, on dit que L est *complet* pour C.

Dans son article fondateur [1], Joseph C. Culberson montre que le sokoban est PSPACE-difficile en proposant une construction directe qui à tout couple composé d'une machine de Turing M et d'une entrée x associe un niveau de sokoban qui admet une solution si et seulement si M accepte x. Nous nous proposons de procéder autrement, en explicitant une réduction polynomiale d'un problème PSPACE de référence vers SOK. Nous utiliserons le problème QSAT de la satisfiabilité des CNF quantifiées sous forme prénexe.

2 SOK est PSPACE complet

2.1 Principe du sokoban et position du problème

Un niveau de sokoban est un labyrinthe dans lequel sont entreposées des caisses. Pour résoudre le casse tête, il faut les placer toutes sur des emplacements spéciaux dits *objectifs*. Le magasinier peut se déplacer d'une case à la fois, selon quatre directions. Il peut avancer sur un emplacement libre, ou encore pousser une caisse dont il prend alors la place. Ce dernier mouvement est autorisé si et seulement si ladite caisse est suivie d'un emplacement libre, et il est donc impossible d'en pousser deux simultanément.

La structure du labyrinthe laissée invariante par les mouvements du joueur est nommée *grille*. On la représente naturellement à l'aide d'une matrice de cellules de type *mur*, *sol*, ou *objectif*. Pour une grille fixée, une *configuration* est la donnée d'une répartition de caisses et de la position du magasinier.

Un niveau de sokoban est ainsi totalement déterminé par une grille et une configuration initiale. On se pose alors naturellement la question de savoir s'il admet ou non une solution, c'est à dire s'il existe une séquence de mouvements autorisés conduisant à une configuration dite acceptante, où toutes les caisses sont placées sur un objectif.

Un exemple de niveau est donné FIGURE 1. Dans tout ce document, les objectifs sont représentés par des points noirs, les caisses placées par des disques d'intérieur gris et les caisses non placées par des disques d'intérieur blanc. La position initiale du magasinier est indiquée par la croix.

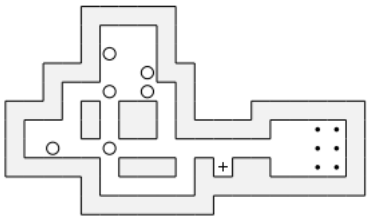

FIGURE 1 – Le premier niveau du jeu original, par *Thinking Rabbit Inc.*

2.2 Le sokoban est PSPACE

Avec le vocabulaire introduit, notre problème se ramène naturellement à un problème d'accessibilité dans le graphe orienté pour lequel chaque sommet est une configuration et chaque arête indique une transition réalisable en un unique mouvement autorisé. Notons qu'au plus quatre arêtes partent de chaque sommet de ce graphe. Donnons nous un niveau de sokoban à p caisses, dont la grille comprend n cellules. Pour un encodage raisonnable, on considère la taille d'un tel niveau comme étant en $O(n)$. Soit $G = (V, E)$ son graphe des configurations. La configuration initiale est notée c_0 et l'ensemble des configurations gagnantes F. On définit une procédure ACCESS qui prend pour paramètres deux configurations c et c' et un entier t. Elle retourne *vrai* si et seulement si il existe un chemin de c à c' de longueur au plus 2^t.

ACCESS(c, c', t) =
  if $t = 0$ **then**
    return $c = c'$ **or** $(c, c') \in E$
  else
    for all $c'' \in V$ **do**
      if ACCESS$(c, c'', t-1)$ **and** ACCESS$(c'', c', t-1)$ **then**
        return true
      end if
    end for
    return false
  end if

Analysons la complexité en espace du calcul de ACCESS(c, c', t) : le nombre d'appels récursifs imbriqués n'excède pas t et chaque appel utilise l'espace nécessaire pour stocker deux configurations et l'entier t. L'espace nécessaire au stockage d'une configuration n'excède pas $O(n)$, et t peut être stocké en binaire, en espace $\log_2(t)$. L'espace utilisé est donc en $O(t \cdot (\log(t) + n))$.

Pour déterminer si le niveau admet une solution, on effectue une série d'appels à la procédure ACCESS $(c_0, f, \lceil \log_2 |V| \rceil)$ pour $f \in F$. Or, $|V| \leqslant p \cdot C_n^p \leqslant n \cdot 2^n$. On utilise donc un espace en $O(n^2)$. Ainsi, SOK $\in$ PSPACE.

2.3 Le problème QSAT

Définition (CNF)**.** *Soient $x_1, x_2, \cdots, x_n$ variables booléennes. On appelle littéral une expression du type x_i ou $\neg x_i$ avec $1 \leqslant i \leqslant n$. Une clause est une disjonction de littéraux. Une* CNF *est une conjonction de clauses. Par exemple, $(\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1)$ est une* CNF *à deux clauses.*

Définition (QCNF)**.** *Si ψ* CNF *et $Q_i \in \{\forall, \exists\}$ pour $1 \leqslant i \leqslant n$, alors $Q_n x_n \cdots Q_1 x_1 \ \psi(x_1, \cdots, x_n)$ est dite* CNF *quantifiée sous forme prénexe, que l'on abrège* QCNF*.*

Une telle expression s'évalue en 0 ou en 1 de la manière suivante, par récurrence :

Pour $F = Q_n x_n \cdots Q_1 x_1 \ \psi(x_1, \cdots, x_n)$, on pose $F_0 = Q_{n-1} x_{n-1} \cdots Q_1 x_1 \ \psi(x_1, \cdots, x_{n-1}, 0)$ et $F_1 = Q_{n-1} x_{n-1} \cdots Q_1 x_1 \ \psi(x_1, \cdots, x_{n-1}, 1)$. Si $Q_n = \forall$, on a alors $F = F_0 \wedge F_1$. Si $Q_n = \exists$, on a $F = F_0 \vee F_1$.

On note QSAT le problème associé à l'évaluation de telles formules. La définition précédente nous donne un algorithme d'évaluation qui s'exécute en espace linéaire. On a donc QSAT $\in$ PSPACE. On montre en annexe que ce problème est PSPACE-complet. Ainsi, pour démontrer notre résultat principal, il suffit d'expliciter une réduction polynomiale de QSAT vers SOK. En d'autres termes, pour une QCNF F fixée, nous devrons donner en temps polynomial un niveau de sokoban qui admet une solution si et seulement si F est vraie. On assemblera pour cela des briques élémentaires agissant comme des unités fonctionnelles autonomes : les *gadgets*.

2.4 Le sokoban est PSPACE difficile

2.4.1 Construction des gadgets de base

Un *gadget* est un assemblage de cellules doté d'un certain nombre de bornes d'accès, et qui stocke un état courant dans la répartition des caisses qui le constituent. On nomme état *résolu* l'état dans lequel toutes les caisses du gadget sont placées.

On dit qu'un *blocage local* est crée lorsque une caisse qui ne se trouve pas sur un objectif n'est plus en état d'être déplacée. Il n'est alors plus possible de gagner à partir d'une telle configuration. Les gadgets orientent le parcours du magasinier en limitant implicitement ses choix aux mouvements qui ne créent pas de blocages locaux, et dont le nombre se trouve restreint.

En notant E l'ensemble des états d'un gadget G et B celui de ses bornes, G est entièrement caractérisé par sa *fonction de transition* $f_G : E \times B \to \mathcal{P}(E \times B)$. A un état courant et une borne d'entrée, f_G associe l'ensemble des sorties possibles, associées à l'état dans lequel le gadget est alors laissé. On présente dans la suite les gadgets utilisés pour les besoins de la démonstration principale.

La diode Le passage n'est permis que de A vers B et n'altère pas l'état du *gadget*, qui reste résolu.

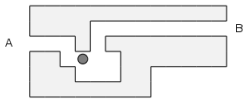

FIGURE 2 – Une diode

L'inverseur Ce gadget dispose de deux états. Dans l'état résolu, le magasinier ne peut aller que de A vers B, passant le gadget dans l'état irrésolu, dans lequel le passage n'est possible que de B vers A au prix d'un retour à l'état résolu.

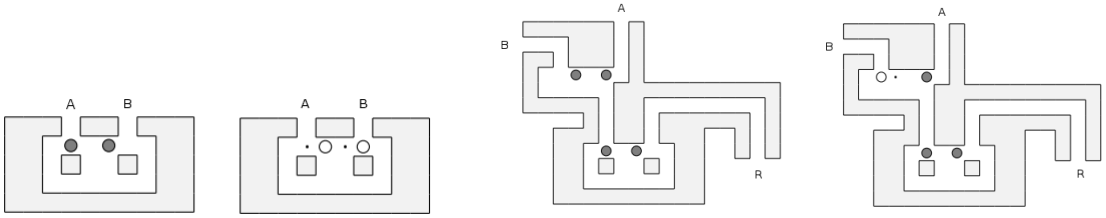

FIGURE 3 – L'inverseur et le transistor, dans leurs deux états respectifs

Le transistor On associe toujours deux diodes à ce gadget afin d'empêcher l'entrée en B et la sortie en A. Dans l'état résolu, le passage de A à B est bloqué. Il est possible de le déverrouiller pour un unique passage en allant au point R, passant ainsi le gadget dans l'état irrésolu.

On représente ces gadgets par les symboles de la FIGURE 4. Par souci de clarté, on a ajouté en annexes le détail de leur fonction de transition. Ils sont reliés les uns aux autres par des couloirs étroits que l'on représente par des fils.

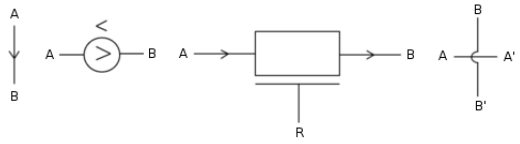

FIGURE 4 – De gauche à droite : la diode, l'inverseur, le transistor, et le pont plan

Le pont plan Dans la conception de circuits, il peut être nécessaire de faire se chevaucher plusieurs fils. On utilise pour cela une construction que nous devons à Culberson : le *pont plan*, et qui est détaillée en annexes.

Le verrou Enfin, on a besoin d'un dernier gadget qui permet de bloquer et débloquer un point du circuit pour un nombre illimité de passages. Ce gadget est présenté FIGURE 5.

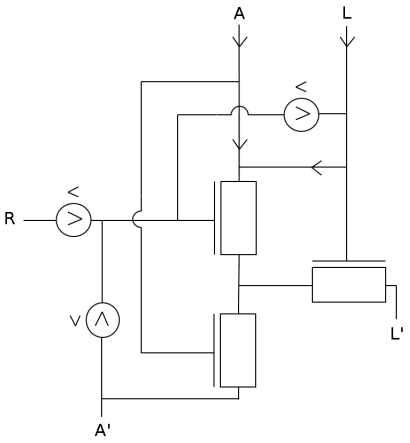

FIGURE 5 – Un verrou

Son état (ie. *verrouillé* ou *déverrouillé*) est codé par l'état du transistor central. En entrant par la borne L, le joueur est obligé de verrouiller le gadget, avant de pouvoir sortir en L'. Il est possible de le déverrouiller en accédant à la borne R. Le passage de A vers A' n'est possible que lorsque le gadget est déverrouillé, et le magasinier choisit alors de le verrouiller à nouveau ou non en le quittant.

2.4.2 Réduction polynomiale de QSAT à SOK

Théorème. SOK $\prec_P$ QSAT

Démonstration. Donnons-nous une QCNF $F = Q_n x_n \cdots Q_1 x_1 \ \psi(x_1, \cdots, x_n)$. On note $C_1, \cdots, C_p$ l'ensemble des clauses de ψ. L'objectif est de construire un niveau de sokoban qui admet une solution si et seulement si F s'évalue en 1. On commence par réaliser un gadget G^{SAT} qu'il est possible de traverser si et seulement si ψ est satisfiable :

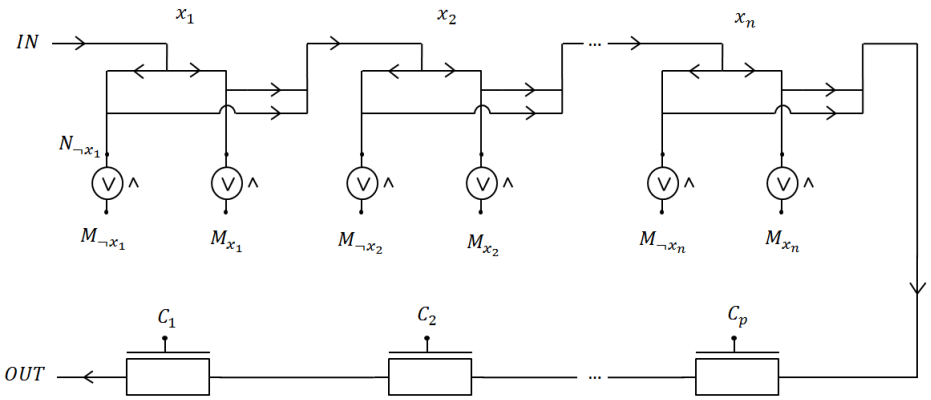

FIGURE 6 – Le circuit de satisfiabilité G^{SAT}

En bas du circuit décrit FIGURE 6, on a placé un transistor pour chaque clause. En haut, un même motif est répété pour chaque variable, et on a associé à chaque littéral l un point du circuit M_l. Enfin, pour toute clause C_i et l un littéral de C_i, on ajoute un fil reliant C_i à M_l. Après être entré en IN, le magasinier choisit une valeur de vérité pour chaque variable et il peut débloquer toutes les clauses contenant le littéral choisi. Ainsi, l'étage inférieur peut être traversé si et seulement si la valuation sélectionnée rend vraies toutes les clauses de ψ. Il est donc possible de relier IN à OUT si et seulement si ψ est satisfiable.

A ce stade, on a en fait démontré la NP-difficulté du sokoban par réduction polynomiale du problème de la satisfiabilité des CNF à SOK. On va maintenant intégrer un à un les quantificateurs de F pour compléter le circuit précédent en un circuit qui permettra d'évaluer complètement F.

On pose, pour k tel que $0 \leqslant k \leqslant n$,

$$F^k(x_{k+1}, \cdots, x_n) = Q_k x_k \, Q_{k-1} x_{k-1} \ \cdots \ Q_1 x_1 \ \psi(x_1, \cdots, x_n)$$

Pour tout k, on va construire par récurrence un gadget G^k qu'il est possible de traverser si et seulement si F^k est satisfiable. Le gadget G^{SAT} aurait parfaitement convenu pour réaliser G^0. Cependant, on préfère imposer au magasinier de choisir une valuation des x_i avant d'entrer par la borne IN, en la codant dans l'état du gadget. Ainsi, G^k comptera $2 + 3\,(n-k)$ bornes. Parmi elles, deux bornes IN et OUT, et pour chaque littéral $l \in \bigcup_{k+1 \leqslant i \leqslant n} \{x_i, \neg x_i\}$ des portes LO_l, LI_l et R_l servant à sélectionner la valuation des variables encore libres.

G^0 reprend la structure de G^{SAT}, mais on a disposé un verrou sur chaque N_l. Dans l'état résolu, tous sont bloqués. Pour débloquer le point associé au littéral l, le joueur utilise la borne R_l. On peut le contraindre à bloquer ce point en le forçant à entrer en LI_l. Il est alors obligé de sortir en LO_l en laissant N_l bloqué, conformément à la description faite précédemment d'un verrou.

L'hérédité exploite l'identité suivante :

$$F^k(x_{k+1}, \cdots, x_n) = \begin{cases} F^{k-1}(0, x_{k+1}, \cdots, x_n) \vee F^{k-1}(1, x_{k+1}, \cdots, x_n) & \text{si } Q_k = \exists \\ F^{k-1}(0, x_{k+1}, \cdots, x_n) \wedge F^{k-1}(1, x_{k+1}, \cdots, x_n) & \text{si } Q_k = \forall \end{cases}$$

Supposons G^{k-1} construit pour k fixé, $1 \leqslant k \leqslant n$. Pour construire G^k à partir de G^{k-1}, on impose au magasinier le parcours suivant :

— Si $Q_k = \exists$, il choisit d'abord une valeur de vérité pour x_k et altère G^{k-1} en conséquence. S'il choisit 0, il bloque le point M_{x_k}. S'il choisit 1, il bloque $M_{\neg x_k}$. Il tente ensuite de relier IN à OUT et il peut bien entendu y parvenir si et seulement si F^k
— Si $Q_k = \forall$, on impose au magasinier deux passages dans G^{k-1}, après blocages respectifs de M_{x_k} et de $M_{\neg x_k}$

Explicitons maintenant cette construction : si $Q_k = \exists$, on construit G^{k+1} à partir de G^k de la manière présentée FIGURE 7. Les bornes de G^k apparaissent à gauche et celles de G^{k-1} à droite. Toutes les bornes qui n'apparaissent pas sont simplement *transmises* d'un gadget à l'autre.

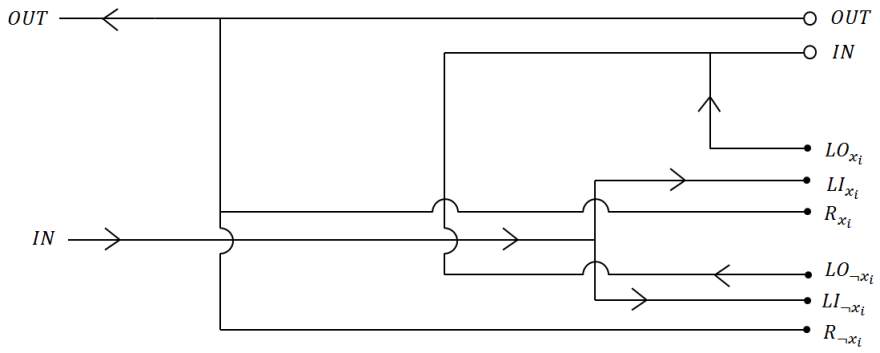

FIGURE 7 – Réalisation de G^k à partir de G^{k-1} pour $Q_k = \exists$

Si $Q_k = \forall$, on procède selon la FIGURE 8.

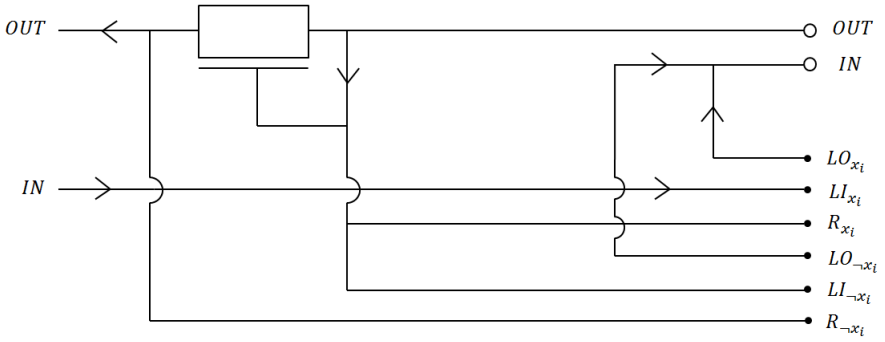

FIGURE 8 – Réalisation de G^k à partir de G^{k-1} pour $Q_k = \forall$

Le gadget G^n est ainsi un gadget à deux bornes IN et OUT et un unique état, qu'il est possible de traverser si et seulement si F est vraie. Pour achever notre réduction, il suffit alors d'insérer G^n dans un niveau où le magasinier part de la borne IN et où la seule caisse non placée l'attend en OUT.

Il reste seulement à montrer que notre construction est réalisable en temps polynomial. Il n'est pas difficile de se convaincre que le temps nécessaire au calcul du niveau à partir de F est proportionnel à la taille de ce dernier. Or, la taille de G^0 est en $O(n + p + np) = O(np)$. En outre, le nombre de cellules supplémentaires permettant de construire G^k à partir de G^{k-1} est en $O(n)$. La taille du niveau construit est ainsi en $O(np + n^2)$, ce qui termine la démonstration.

□

3 Bibliographie

Ci dessous se trouve une liste des principaux articles et ouvrages exploités. L'encyclopédie libre *Wikipedia* en langue anglaise a été abondamment consultée, en particulier aux articles *Turing Machine*, *Computational complexity theory*, *P = NP*, *NP*, *PSPACE*, *NP complete* ...

Références

[1] Joseph C. Culberson, *Sokoban is PSPACE-complete*, 4 Avril 1997

[2] Dorit Dor et Uri Zwick , *SOKOBAN and other motion planning problems (extended abstract)*, 28 Juin 1995

[3] Olivier Carton, *Langages formels, calculabilité et complexité*, Vuibert

[4] Christos H. Papadimitriou, *Computational complexity*, Addison Wesley

[5] D. Beauquier, J. Berstel, Ph. Chretienne, *Éléments d'algorithmique*, Masson, 1992

[6] Andreas Junghanns : Rapport de thèse *Pushing the limits : new developments in single-agent search*, 1999

4 Annexes

4.1 QSAT est PSPACE complet

Théorème. QSAT *est* PSPACE *complet.*

Démonstration. On a déjà montré que QSAT $\in$ PSPACE. Il suffit donc de montrer que QSAT est PSPACE difficile. Prenons $L \in$ PSPACE décidé par $M = (Q, \Sigma, \Gamma, q_0, F, \delta)$, P un polynôme qui borne la complexité de M en espace, et w une entrée de taille n. On travaille ici, sans perte de généralité, avec une machine dont la bande est indicée sur $\mathbb{N}$ et qui possède deux états finaux q_+ et q_-. En outre, on impose à M d'effacer la bande avant de s'arrêter (inscrire partout le symbole $\sqcup$).

La donnée de l'état courant, de la position de la tête, et du contenu de la bande à un instant donné constitue une *configuration* de M, que l'on peut représenter par un mot sur l'alphabet $Q \cup \Gamma$. En supposant Q et Γ disjoints, il suffit en effet d'intercaler un symbole de Q qui code pour l'état courant dans un mot sur Γ qui représente le contenu de la bande jusqu'au point où elle ne contient plus que le caractère blanc, à la position qui correspond à celle de la tête de M. Ainsi, de par les hypothèses faites sur M et pour une entrée w, la configuration initiale est $q_0 w$, et l'unique configuration acceptante est q_+.

À une configuration codée par le mot c on associe $\{c_{i,s} \mid 0 \leqslant i \leqslant P(n), s \in Q \cup \Gamma\}$ où $c_{i,s}$ est vrai si et seulement si $((i < |c|$ et $c_i = s)$ ou $s = \sqcup)$. Par hypothèse, toute configuration accessible depuis $q_0 w$ est de taille inférieure à $P(n)$. Un tel ensemble caractérise donc complètement une configuration accessible.

On va maintenant expliciter, par récurrence, une expression booléenne quantifiée $\phi_{c,d,i}$, vraie si et seulement si il est possible à M de passer de la configuration c à la configuration d en au plus 2^i transitions. w est alors accepté si et seulement si $\phi_{q_0 w, q_+, \lfloor \log_2(T_M(n)) \rfloor + 1}$.

Deux configurations c et d sont égales si et seulement si

$$\bigwedge_{i=0}^{P(n)} \bigwedge_{s \in Q \cup \Gamma} (c_{i,s} \Leftrightarrow d_{i,s})$$

ce que l'on abrège désormais en $(c \Leftrightarrow d)$.

Si $\delta(q, a) = (q', b, \leftarrow)$, on définit $\phi_{c,d,0}^{(q,a,q',b,\rightarrow)}$, vraie si et seulement si il est possible de passer de c à d par cette transition :

$$\phi_{c,d,0}^{(q,a,q',b,\rightarrow)} =$$

$$\bigvee_{j=1}^{P(n)} \left(c_{j,q} \wedge c_{j+1,a} \wedge d_{j-1,q'} \wedge d_{j+1,b} \wedge \left(\bigvee_{e \in \Gamma} c_{j-1,e} \wedge d_{j,e} \right) \wedge \left(\bigwedge_{\substack{i \in \{0, \cdots, j-2, j+2, \cdots P(n)\} \\ s \in Q \cup \Gamma}} (c_{i,s} \Leftrightarrow d_{i,s}) \right) \right)$$

On procède de même pour les déplacements de la tête à droite. On a alors

$$\phi_{c,d,0} = (c_{i,s} \Leftrightarrow d_{i,s}) \vee \left(\bigvee_{\delta(q,a) = (q', b, \mathcal{D})} \phi_{c,d,0}^{(q,a,q',b,\mathcal{D})} \right)$$

et

$$\phi_{c,d,i+1} = \exists m \, \forall c' \, \forall d' \, ((c \Leftrightarrow c' \wedge m \Leftrightarrow d') \vee (m \Leftrightarrow c' \wedge d \Leftrightarrow d')) \Rightarrow \phi_{c',d',i}$$

Les $\phi_{c,d,i}$ sont donc de taille linéaire en i. En outre, comme M s'arrête pour toute entrée, M ne peut passer deux fois par la même configuration sans quoi il y aurait création d'un cycle d'exécution. On a donc $T_M(n) \leqslant |\Gamma \cup Q|^{s(n)}$, d'où l'existence de K tel que $T_M(n) \leqslant 2^{K \cdot s(n)}$. Ainsi, $\phi_{q_0 w, q_+, \lfloor \log_2(T_M(n)) \rfloor + 1}$ est de taille polynomiale en n, et il en va de même pour le temps nécessaire à la produire.

La démonstration n'est cependant pas encore achevée, dans la mesure où $\phi_{c,d,i}$ n'est pas une QCNF. On propose donc un algorithme permettant de convertir une expression booléenne quantifiée sans variables libres en une QCNF équivalente, en temps polynomial.

On élimine d'abord tous les connecteurs $\Leftrightarrow$ et $\Rightarrow$ en faisant intervenir $\wedge$ et $\vee$. On met ensuite l'expression sous forme prénexe en déplaçant les quantificateurs grâce aux identités suivantes :

$$\neg(\forall x\, \varphi) = \exists x(\neg\varphi)$$
$$\neg(\exists x\, \varphi) = \forall x(\neg\varphi)$$
$$(Qx\ \varphi) * \psi = Qx\ (\varphi * \psi) \quad \text{pour } Q \in \{\forall, \exists\}, * \in \{\wedge, \vee\}$$

La dernière identité est valable si la variable x n'intervient pas dans ψ, ce à quoi on peut toujours se ramener, quitte à renommer les variables liées. A ce stade, notre expression booléenne s'écrit sous la forme $Q_n x_n \ldots Q_1 x_1\ \psi(x_1, \ldots, x_n)$ où ψ n'est pas nécessairement une CNF. On va maintenant construire une CNF φ des variables $x_1, \cdots, x_n, y_1, \cdots, y_m$ telle que

$$Q_n x_n \ldots Q_1 x_1\ \psi(x_1, \ldots, x_n) = Q_n x_n \ldots Q_1 x_1\, \exists y_n \ldots \exists y_1\ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

Pour cela, on se représente l'arbre syntaxique de ψ. Les variables x_i en occupent les feuilles, et chaque nœud interne est étiqueté par un opérateur. On numérote les nœuds internes de 1 à m et on associe à un nœud i la variable y_i. On note $*_i$ l'opérateur associé au nœud i, et on désigne par g_i et d_i les variables associées respectivement au fils gauche et droit du nœud i si $*_i$ est binaire, f_i la variable associée au fils de i si $*_i$ est unaire.

Pour tout $1 \leqslant i \leqslant m$, on note

$$\varphi_i = \begin{cases} y_i \wedge (y_i \Leftrightarrow g_i *_i d_i) & \text{si } *_i \in \{\vee, \wedge\} \\ y_i \wedge (y_i \Leftrightarrow \neg f_i) & \text{sinon} \end{cases}$$

Chaque φ_i ne comptant que trois variables au plus, il est possible mettre chacune sous forme normale conjonctive en temps constant. On pose enfin

$$\varphi = \bigwedge_{i=0}^{m} \varphi_i$$

ce qui achève la construction, qui est bien réalisée en temps polynomial.

□

4.2 Fonctions de transition des gadgets utilisés

Les fonctions de transition des différents gadgets étudiées sont présentées ici. Elles permettent de les décrire sans ambiguïté et d'en vérifier méthodiquement la construction. Les bornes sont systématiquement représentées par des lettres, en suivant le schéma de la FIGURE 4 et les états par des chiffres. L'état résolu est toujours désigné par 0.

	A	B
0	B0	-

Diode

	A	B
0	B1	-
1	-	A0

Inverseur

	A	B	R
0	-	-	R1
1	B0	-	R1

Transistor

	A	A'	L	L'	R
0	-	-	L'0	-	R1
1	A'0 A'1	-	L'0	-	R1

Verrou

La fonction de transition du pont plan est donnée dans la section suivante.

4.3 Construction du pont plan

Le gadget du *pont plan* permet d'avoir recours à des croisements de fils lors de la conception de circuits. Il est plus facile d'en concevoir une version *polarisée*, modélisant le croisement de deux couloirs sur lesquels sont placées des diodes. On peut aisément réaliser un pont plan en associant quatre ponts plans polarisés.

	A	B	A'	B'
0	A'0	B'0	A0	B0

Pont plan

	A	B	A'	B'
0	A'0	B'0	-	-

Pont plan polarisé

La première construction d'un pont plan polarisé est due à Culberson, et les détails permettant de comprendre son fonctionnement peuvent être trouvés dans son article [1],.

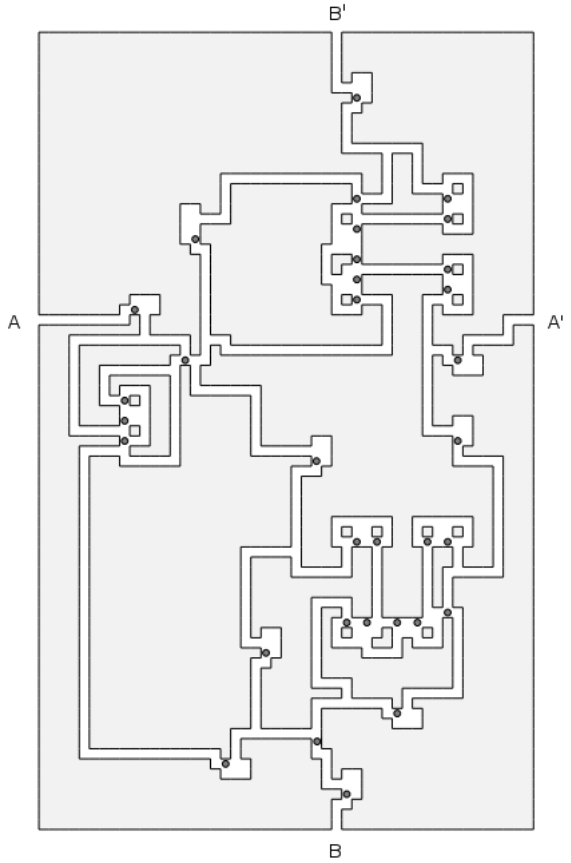

FIGURE 9 – Le pont plan polarisé