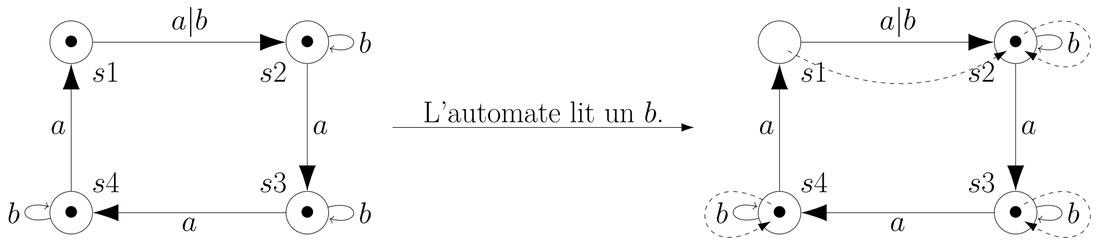
Application de la Recherche opérationnelle au calcul d'un mot synchronisant de taille minimum

Walid Benameur[1,2], Natalia Kushik[1,2], Davy Nguyen[3], Dimitri Watel[2,4]

Définition du problème



On considère un automate où un jeton est posé sur chaque état.

Les jetons se déplacent le long des transitions. Si deux jetons se rencontrent (ici en $s2$), ils fusionnent.

(Problème du mot synchronisant le plus court.) Connaissant un automate (S, T, Σ) déterministe complet, quel est le mot, noté SS, le plus court de Σ^* qui permet de fusionner tous les jetons posés sur les états de S?

Dans l'exemple ci-dessus, il s'agit de $SS = baaabaaab$.

Ce problème permet de réinitialiser un système. Même si on ne sait pas dans quel état se trouve le système, on sait dans quel état il se trouvera à l'issue de la séquence SS.

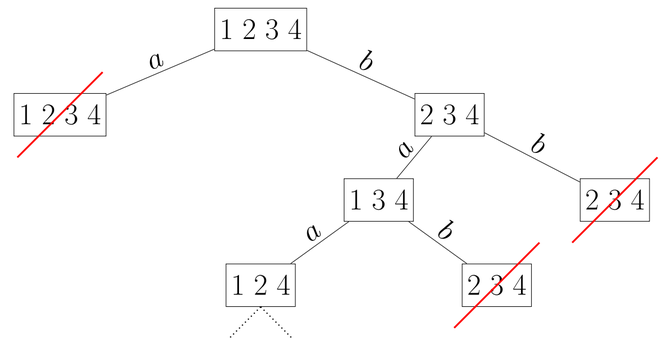
Ce problème est NP-Complet [1].

Algorithmes de résolution existants

Recherche arborescente

Chaque nœud de l'arbre représente les positions des jetons.

Les successeurs indiquent où se trouvent les jetons si l'automate lit un a ou un b. Si un nœud est identique à un de ses ancêtres, on le barre, car une séquence optimale ne répète pas le placement des jetons.



On explore l'arbre en largeur jusqu'à barrer tous les nœuds explorables, ou jusqu'à tomber sur un nœud ne contenant qu'un jeton. Avec quelques améliorations et une bonne structure de donnée, cet algorithme est très efficace [2].

SAT Solver

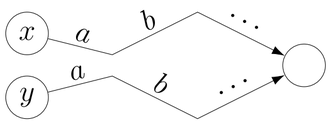
On fixe un entier k et on transforme l'automate en formule booléenne satisfiable si et seulement s'il existe une séquence synchronisante de taille au plus k. On utilise ensuite un SAT Solver pour la résoudre, puis on effectue une dichotomie sur k pour déterminer la plus petite valeur possible pour cet entier [3].

Amélioration de l'algorithme de recherche arborescente

- On utilise un algorithme de Branch and Bound, l'arbre d'exploration est le même que celui de la recherche arborescente. On peut explorer en profondeur à gauche, avec le meilleur d'abord, ...
- Il faut pour cela une borne inférieure de la taille de la séquence optimale SS. Toutes les bornes suivantes se calculent en temps polynomial.

Merging sequences

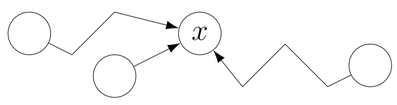
$MS(x, y)$ est le mot le plus court permettant de fusionner x et y.



$\max_{x,y} MS(x, y) \leq |SS|$

Rayons

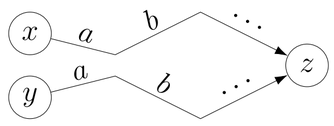
$RAD(x)$ est le plus long des plus courts chemins menant à x.



$\min_{x} RAD(x) \leq |SS|$

Rayons + merging sequences

$MS(x, y, z)$ est le mot le plus court permettant de fusionner x et y en z.



$\min_{z} \max_{x,y} MS(x, y, z) \leq |SS|$

Programmation linéaire

On modélise le problème avec un programme linéaire et on utilise la relaxation continue pour en déduire une borne inférieure (voir ci-après).

Modélisation par un programme linéaire

Attention, ce programme ne fonctionne qu'avec un alphabet binaire.

Soit k une borne *supérieure* de SS, k' la borne Rayons + merging sequences

Variables $x_t \in \{0, 1\}$, avec $t \leq k$. On veut $x_t = 1$ ssi le t^e symbole envoyé est un a.

Variables $s_t^i \in \{0, 1\}$ avec $i \in S$ et $t \leq k$. On veut $s_t^i = 1$ ssi, en envoyant le mot $x_1 x_2 \cdots x_t$, il y a un jeton dans l'état i. On fixe $s_0^i = 1$ pour tout $i \in S$.

Variables $z_t \in \{0, 1\}$ avec $t \leq k$. On veut $z_t = 1$ ssi $\sum_{i \in S} s_t^i > 1$. On fixe $z_t = 1$ si $t \leq k'$.

Objectif Minimiser $\sum_{t=0}^{k} z_t$

Contraintes

$\forall t \leq k$ $\quad \sum_{i \in S} s_t^i \leq 1 + (|S| - 1) \cdot z_t$

$\forall i \in S, t \leq k$ $\quad s_{ia}^t \geq s_j^t + x_t - 1$ où ia est le successeur de i avec le symbole a.

$\forall i \in S, t \leq k$ $\quad s_{ib}^t \geq s_j^t - x_t$ où ib est le successeur de i avec le symbole b.

La relaxation continue de ce PL donne une première borne. Celle-ci peut être améliorée.

Soit $P = CONV\left(\left\{\begin{pmatrix} s^{t-1} \\ s^t \\ x_{t-1} \end{pmatrix} \text{ vérifiant les contraintes précédentes.}\right\}\right)$

Le point de P minimisant l'objectif donne la meilleur borne inférieure possible avec ce modèle.

Soit $P_1 = CONV\left(\left\{\begin{pmatrix} s^{t-1} \\ s^t \end{pmatrix} \text{ points de } P \text{ restreints à } x_{t-1} = 1.\right\}\right)$

Soit $P_0 = CONV\left(\left\{\begin{pmatrix} s^{t-1} \\ s^t \end{pmatrix} \text{ points de } P \text{ restreints à } x_{t-1} = 0.\right\}\right)$

Alors $\begin{pmatrix} s^{t-1} \\ s^t \\ x_{t-1} \end{pmatrix} \in P \Leftrightarrow \exists h_1 \in P_1, h_0 \in P_0 | \begin{pmatrix} s^{t-1} \\ s^t \end{pmatrix} = x_{t-1} \cdot h_1 + (1 - x_{t-1}) \cdot h_0$

Posons $y_1^{t-1} = x_{t-1} \cdot h_1$ et $y_0^{t-1} = (1 - x_{t-1}) \cdot h_0$, ainsi $\begin{pmatrix} s^{t-1} \\ s^t \end{pmatrix} = y_1^{t-1} + y_0^{t-1}$. On rajoute au programme des inégalités sur ces variables qui renforcent sa relaxation. Par exemple:

$\forall i \in S, 1 \leq t \leq k$ $\quad (y_1^{t-1})_i^t \leq \sum_{j|i=ja} (y_1^{t_1})_j^{t-1}$

(si $x_{t-1} = 1$ et si on a un jeton sur i à l'itération t, alors on a un jeton sur un état j dont i est le successeur avec le symbole a)

Les notation $(y_1^{t-1})_i^{t-1}$ et $(y_1^{t-1})_i^t$ renvoient respectivement au i^e et $|S| + i^e$ éléments du vecteur (y_1^{t-1}).

Résultats préliminaires

On évalue le nombre de nœuds explorés par notre méthode par rapport à la recherche en largeur. On utilise 2 bornes, les 2 dernières, et les 2 explorations possibles (profondeur à gauche et meilleure borne). On a donc 4 méthodes à tester.

Chaque algorithme est testé sur 120 automates ; 30 automates de chaque taille parmi $\{5, 10, 15, 20\}$; l'alphabet est binaire ; les deux transitions sortante de chaque état vont vers un état sélectionné uniformément.

Le tableau ci-dessous à gauche indique combien d'instances où l'algorithme en ligne a exploré *strictement* moins de nœuds que l'algorithme en colonne. La valeur en gras indique quand un algo est meilleur que son concurrent. Le second tableau indique la moyenne du pourcentage de nœuds explorés en moins parmi toutes les instances où l'algorithme en ligne a strictement moins exploré que l'algorithme en colonne.

	BFS	*DMS*	*DPL*	*BMS*	*BPL*
BFS	0	23	22	0	0
DMS	**83**	0	0	20	15
DPL	**84**	**32**	0	21	17
BMS	**113**	**84**	**83**	0	0
BPL	**114**	**86**	**85**	**35**	0

	BFS	*DMS*	*DPL*	*BMS*	*BPL*
BFS	−−	20	20	−−	−−
DMS	50	−−	−−	22	24
DPL	50	8	−−	25	24
BMS	65	56	55	−−	−−
BPL	65	57	56	11	−−

BFS : Recherche en largeur (algorithme existant)
DMS : Exploration en profondeur, borne Rayons + Merging sequences
DPL : Exploration en profondeur, borne Programme linéaire
BMS : Exploration du meilleur d'abord, borne Rayons + Merging sequences
BPL : Exploration du meilleur d'abord, borne Programme linéaire

Formule : $\frac{\max - \min}{\max} \cdot 100$

où max et min sont respectivement le nombre de nœuds explorés par l'algorithme en colonne et en ligne.

Perspectives

- Tester des explorations mixtes entre largeur, meilleur et profondeur.
- Evaluer précisément le temps de calcul de la borne inférieure, pour déterminer quand le gain en nombre de nœuds explorés est rentable.

Références

[1] Eppstein, D., Reset Sequences for Monotonic Automata, SIAM Journal on Computing, vol. 19, no 3, 1990, p. 500–510

[2] Kisielewicz, A., Kowalski, J., Szykuła, M., A fast algorithm finding the shortest reset words, ICCC, 2013, p. 182-196

[3] Skvortsov, E., Tipikin, E., Experimental study of the shortest reset word of random automata, ICIAA, 2011, p. 290-298

[1] Télécom SudParis, 9 Rue Charles Fourier, 91000, Evry ; {prenom.nom}@telecom-sudparis.eu
[2] Laboratoire SAMOVAR, 9 Rue Charles Fourier, 91000, Evry
[3] Master ANDROIDE, Sorbonne université, Faculté des Sciences, 4 place Jussieu 75005 Paris
[4] ENSIIE, 1 place de la résistance, 91000, Evry ; {prenom.nom}@ensiie.fr

Équipe METHODES

Dimitri Watel