

Rapport Projet de stage : Limitedness

BREBANT Alexandre
XUE Juedong

10 juillet 2014

Table des matières

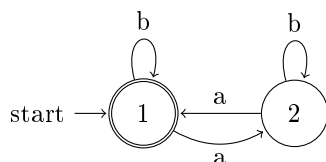
1	Introduction au problème	2
I	Mise en place théorique de l'algorithme	5
2	De l'automate aux matrices	6
3	Matrices idempotentes	9
4	Matrices stables	11
II	Exécution détaillée du programme	12
5	Réutilisation du programme d'Adrien Boussicault	13
6	Création et manipulation des matrices	14
7	Structure de stockage des matrices	16
8	Résolution du problème de limitedness	17

Chapitre 1

Introduction au problème

Durant ce stage, nous avons voulu approfondir nos connaissances dans une branche de l'informatique théorique que nous avons commencé à étudier lors de ce dernier semestre de licence. Ce sont donc les automates et les interrogations qui gravitent autour de cette structure qui nous ont intéressés.

Un automate est une machine abstraite qui va accepter ou rejeter des mots dans un alphabet fini. Voici l'exemple d'un automate très basique qui accepte tous les mots sur l'alphabet $\{a, b\}$ qui comportent un nombre pair de a .



Pour représenter graphiquement un automate, on utilise donc un graphe orienté dans lequel les sommets du graphe sont les états de l'automate et où chaque arrête correspond à une transition de l'automate, c'est à dire à la lecture d'une lettre. Un mot est accepté par un automate \mathcal{A} , s'il existe un chemin dans celui-ci, partant d'un état initial et allant jusqu'à un état final en lisant les lettres une par une. Un tel chemin est appelé *chemin acceptant* et l'ensemble des mots acceptés par un automate est appelé *Langage*, on le note $L(\mathcal{A})$.

Formellement, un automate est défini, par 5 ensembles, de la façon suivante :

$$\mathcal{A} = \{A, Q, I, F, \delta\} \text{ avec,}$$

A : l'alphabet

Q : l'ensemble des états

I : l'ensemble des états initiaux

F : l'ensemble des états finaux

δ : l'ensemble des transitions

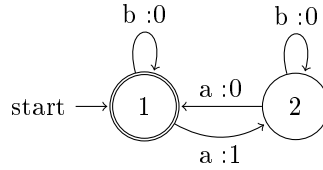
Nous pouvons donc définir l'automate ci-dessus de cette manière :

$$\mathcal{A} = \left(A = \{a, b\}, Q = \{1, 2\}, I = \{1\}, F = \{1\}, \delta = \begin{cases} (1, a) \rightarrow 2 \\ (2, a) \rightarrow 1 \\ (1, b) \rightarrow 1 \\ (2, b) \rightarrow 2 \end{cases} \right)$$

La théorie des automates et des langages est un domaine très actif depuis le milieu du 20e siècle et il existe aujourd'hui plusieurs modèles d'automates nés pour répondre à certaines problématiques.

Ainsi les automates permettent de caractériser certains langages, par exemple, les automates finis qui sont le modèle de base des automates, définissent ce qu'on appelle les langages rationnels ([1]). Mais il existe des extensions à ce modèle, comme par exemple les automates à pile qui permettent de définir les langages algébriques ([1]), qui sont grandement utilisés en analyse syntaxique. Et c'est une autre extension de l'automate fini qui va nous intéresser dans ce rapport : les automates à distance (distance automata). Ce sont des automates finis, non déterministes, dans lesquels les transitions sont pondérées. On associe donc un coût pour chaque mot accepté par l'automate. Le coût d'un mot correspond au minimum des coûts de tous ses chemins acceptants dans l'automate .

Reprenons l'automate représenté plus haut en y ajoutant des coûts sur les transitions :



Avant d'aller plus loin dans la description de cet automate, nous devons définir une notion élémentaire : le déterminisme. Un automate est déterministe si et seulement si il possède un unique état initial et que pour tout état, il existe au plus, une transition par lettre de l'alphabet. Cependant, la déterminisation des automates à distance est encore une question ouverte aujourd'hui. On dit alors qu'un automate à distance est déterministe si l'automate sans les poids l'est.

Notre automate à distance est donc déterministe. Cela signifie qu'il n'existe qu'un seul chemin acceptant pour tout les mots reconnus par l'automate et donc qu'il n'existe également qu'un seul coût pour ce mot. On peut observer dans cet exemple que le coût correspondra au nombre de paires de a dans le mot.

Ce type d'automate a été présenté par Hashiguchi dans [3] pour la résolution d'une célèbre problématique, la question de hauteur d'étoile (star height problem) [2]. Mais les automates à distance posent également un autre problème, la limitation des automates à distance (Limitedness problem).

Le problème de limitedness est très simple à comprendre, il est simplement question de savoir si il existe une majoration des coûts pour tous les mots acceptés par un automate à distance.

La décidabilité de ce problème n'est pas évidente. En effet il existe des problèmes similaires dont on a prouvé leur indécidabilité, c'est à dire la preuve qu'il n'existe pas d'algorithme capable de résoudre le problème. Il est par exemple prouvé que la question suivante est indécidable :

Soit \mathcal{A}_1 et \mathcal{A}_2 deux automates à distance distincts. Est ce que le coût de w dans \mathcal{A}_1 est égale au coût de w dans \mathcal{A}_2 , $\forall w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$?

Le travail d'Hashiguchi étant considéré comme difficile à comprendre, d'autres chercheurs, comme Simon [5] ou Leung [4], ont étudié le problème et apporté d'autres solutions sur le sujet.

Notre travail a donc consisté, dans un premier temps, à lire et comprendre les différents travaux de ces chercheurs afin d'être capable d'implémenter un algorithme permettant la détection de limite dans un automate à distance.

Première partie

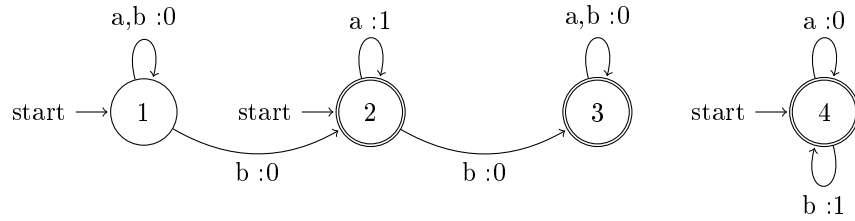
Mise en place théorique de l'algorithme

Chapitre 2

De l'automate aux matrices

Le papier dont nous nous sommes le plus servi afin de comprendre et résoudre le problème de limitedness, est [5] mais il nous a d'abord fallu comprendre la complexité du problème.

Pour simplifier les calculs de coûts, il fallait utiliser des structures mathématiques pour représenter la lecture d'une lettre ou d'un mot. Les matrices permettent de créer des "tables de coût" pour la lecture d'une lettre sur les différents états. Pour la suite du rapport nous utiliseront l'automate suivant en tant que référence pour nos exemples :



Automate \mathcal{A}_{ref}

Cet automate est-il limité ? Plus formellement :

$$\exists ?B \in \mathbb{N} \text{ tel que } \forall w \in L(\mathcal{A}_{ref}), \text{ cost}(w) \leq B$$

La réponse n'est évidemment pas trivial. C'est pourquoi nous allons utiliser des matrices pour représenter les différents coûts pour une lecture de lettres. La matrice $M(a)$ va par exemple contenir le coût de la lecture de la lettre a depuis tout les états de l'automate de la manière suivante :

$$M_{i,j} = \begin{cases} n, & \text{s'il existe une transition } (i, a : n) \rightarrow j \\ \infty, & \text{sinon} \end{cases}$$

Pour \mathcal{A}_{ref} , on a donc par exemple :

$$M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

$\begin{matrix} 1 & 2 & 3 & 4 \end{matrix}$

et

$$M(b) = \begin{pmatrix} 0 & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 \end{pmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

$\begin{matrix} 1 & 2 & 3 & 4 \end{matrix}$

Ces matrices sont donc définies sur $\{\mathbb{N} \cup \infty\}$, nous les appellerons matrices de coûts. Leur fonctionnement est très simple : la numérotation de chaque ligne et de chaque colonne correspond aux états de l'automate. Par exemple, dans la matrice de coûts pour la lettre b , on a $M(b)_{(2,3)} = 0$. Cela signifie que le coût pour aller de l'état 2 à l'état 3 en lisant un b est de 0, ce qu'on peut vérifier sur l'automate.

Avant d'aller plus loin il est nécessaire de définir la structure algébrique qui est au cœur de l'algorithme de résolution du problème de limitedness, les semi-anneaux.

Un semi-anneau est formé d'un ensemble E muni de deux lois associatives, \oplus et \otimes , et d'un élément neutre pour chaque loi. De plus, la première loi est commutative et la seconde est distributive sur la première.

Exemple : Les entiers naturels, munis de l'addition et de la multiplication, forment un semi-anneau : $\{\mathbb{N}, +, *\}$. Les éléments neutres sont 0 pour l'addition et 1 pour la multiplication. Ces deux lois sont bien associatives, l'addition est commutative et la multiplication est distributive sur l'addition. On remarque que la multiplication est également commutative, on dit que \mathbb{N} est un semi-anneau commutatif.

Cette structure va nous servir pour définir la multiplication de deux matrices de coûts. Le premier semi-anneau que nous allons définir est celui formé par l'ensemble $\{\mathbb{N} \cup \infty\}$ tel que :

- $a \oplus b = \min(a, b)$, avec élément neutre ∞
- $a \otimes b = a + b$, avec élément neutre 0

Nous avons maintenant la possibilité de multiplier deux matrices de coûts entre elles afin d'obtenir la matrice correspondant à la concaténation des mots des matrices. On a donc :

$$M(ab) = M(a) \times M(b) = \begin{pmatrix} 0 & 0 & \infty & \infty \\ \infty & \infty & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 \end{pmatrix}$$

Il est ainsi possible d'obtenir les matrices de coûts pour n'importe quel mots composé de lettres de l'alphabet de l'automate. Et pour obtenir le coût du mot, il faut prendre le minimum des coûts situés dans l'intersection des états initiaux en ligne et des états finaux en colonne. Nous représentons les coûts compris dans cette intersection en rouge.

On peut donc dire que $cost(ab) = M(ab)_{(1,2)} = 0$ et il s'obtient en partant de l'état 1 vers l'état 2.

Cela ne nous permet cependant pas de déterminer si l'automate est limité ou pas. En effet deux facteurs rendent le nombre de matrices infini : l'ensemble de définition des matrices $\{\mathbb{N} \cup \infty\}$ et le nombre de mots acceptés par l'automate qui est lui aussi infini.

Chapitre 3

Matrices idempotentes

Dans ce chapitre, nous allons faire évoluer nos matrices afin de rendre leur nombre limité. Tout d'abord, il est nécessaire de modifier le semi-anneaux qui est défini sur un ensemble infini. Dans le problème de limite, la valeur du coût en elle-même n'a pas vraiment d'utilité. Ce qui nous intéresse, c'est de savoir si cette valeur est fixe ou si elle va croître avec la longueur du mot. Si elle croît, ce signifie qu'il existe une boucle dans l'automate dont le poids est strictement supérieur à 0.

Nous allons donc définir un ensemble de 4 valeurs qui suffiront à représenter l'intégralité des coûts possibles :

- 0 qui correspond au coût gratuit
- 1 qui correspond à un coût potentiellement grand mais fixe
- ω qui correspond à un coût instable, qui va croître
- ∞ qui a toujours la même signification, à savoir correspondre aux coûts impossibles à payer

Et on a donc $0 < 1 < \omega < \infty$, il reste maintenant à définir les deux lois associatives du nouveau semi-anneau. La première est identique car nous souhaitons toujours obtenir le minimum parmi tous les chemins : $a \oplus b = \min(a, b)$. Mais l'addition de coûts ne nous intéresse plus, nous voulons maintenant connaître le plus gros coût parmi toutes les transitions du chemin de lecture d'un mot. On prend donc $a \otimes b = \max(a, b)$.

Nous formons ainsi le semi-anneau qui est le sujet principal de l'article de Simon, le semi-anneau tropical que nous nommeront également \mathcal{R} .

On remarque tout de fois qu'à partir des matrices de base dont on dispose et de ces deux nouvelles lois, il est impossible d'obtenir la valeur ω par simple multiplication matricielle dans \mathcal{R} . Il nous est donc impossible de détecter les coûts instables, en effet certaines positions contenant la valeur 1 vont rester à inchangées après une multiplication alors que le coût réel va croître d'une certaine valeur pour chaque lecture du mot représenté par la matrice.

Reprenons par exemple notre matrice $M(a)$:

$$M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

Calculons maintenant $M(aa)$ dans \mathcal{R} :

$$M(aa) = M(a) \times M(a) = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

On remarque que $M(aa) = M(a)$. Ce qui implique que $M(aaaa...) = M(a)$ et donc que $M(aaaa...)_{(2,2)} = M(a)_{(2,2)} = 1$.

Or on peut observer sur l'automate que la lecture répétée de la lettre a en bouclant sur l'état 2 va causer un coût croissant dépendant du nombre de a lus. Pour distinguer ce genre de coût de ceux qui ne causent pas ce problème de croissance infinie, il faut donc ajouter un traitement supplémentaire aux matrices dont la multiplication avec elles-mêmes dans \mathbf{R} donne la même matrice. On appelle ces matrices des matrices idempotentes.

Le dernier chapitre de cette partie théorique va donc consister à la détection des coût instables dans les matrices idempotente.

Chapitre 4

Matrices stables

Nous avons donc vu que lorsqu'une matrice est idempotente, il y a un risque que certaines positions voient leur coût exploser à chaque lecture du mot de cette matrice. Il s'avère donc nécessaire de représenter l'itération d'un mot d'une matrice.

Au lieu d'écrire $M(aaaaa...)$ comme nous l'avons fait dans le chapitre précédent, nous allons introduire un nouveau symbole afin de simplifier la notation et écrire ce genre de matrices de la façon suivante : $M(a^\#)$.

Soit w un mot, tel que $M(w)$ idempotente, la matrice $M(w^\#)$ va donc contenir les coûts dans \mathcal{R} d'un nombre quelconque de lectures successives du mot w . Dans ces matrices que nous appellerons simplement matrice dièse, il reste donc à déterminer la valeur de chaque position. Dans son article, Simon définit le terme "ancrée" pour définir les positions dont le coût est stable ou va se stabiliser au bout d'un nombre fini d'itération.

Lemme : *Soit w un mot tel que $M(w)$ idempotente dans \mathcal{R} . Si $M(w)_{(i,j)} = 0$ alors la position (i, j) est ancrée. De plus, si il existe $k \in [1, n]$ tel que $M(w)_{(k,k)} = 0 \leq M(w)_{(i,j)} = \max(M(w)_{(i,k)}, M(w)_{(k,j)})$ alors la position (i, j) est ancrée.*

Deuxième partie

Exécution détaillée du
programme

Chapitre 5

Réutilisation du programme d'Adrien Boussicault

(présentation des différents modules et fonctions disponible + détails des ajouts pour la gestion des coût, de l'affichage et autres fonctions additionnelles que nous avons ajouté)

Chapitre 6

Création et manipulation des matrice

(créer_matrice_transition, multiplication, etc...)

Dans notre sujet, la matrice représente la matrice d'incidence. Les indices dans la matrice sont des étiquettes d'état. Du coup, dans notre cas, toutes les matrices sont de la forme carrée $n \times n$, dont n est le nombre d'états.

D'après ce fait, on construit la matrice avec la fonction `créer_matrice(int taille)`. La matrice contient la taille le mot et un tableau de deux dimensions du type `int`. alors, on alloue le mémoire successivement pour la structure de la matrice et le tableau d'int de deux dimensions.

Également, pour la destruction, si la matrice est déjà `NULL`, on la laisse tranquille, sinon on des-alloue la structure de la matrice ainsi pour le tableau de deux dimensions.

Pour accéder dans la matrice, on a trois successeurs pour obtenir les informations sur le tableau, le mot et la taille de la matrice.

vous voyez bien que dans notre structure de la matrice il y a un "mot" qui est du type `tree`, ce mot est pour exprimer la matrice sous forme d'expression rationnel. le mot contient trois genres de choses :

- l'alphabet (dans `R a,b,c,etc.`)

- la dièse '#' (la répétition comme `*` dans l'expression rationnel)

- le point "." ("." signifie la concaténation)

Car on construit le mot par la manipulation de concaténation en collant les sous mot, on include `arbresyntaxique.h` pour bien résoudre le problème de construction du mot.

La fonction `print_matrice_in_R` permet d'afficher la matrice de façon mathématique. en précise, on définit `INFINI` est 3 et `OMÉGA` est 2 dans `Matrice.h`, ici on reconverti `INFINI` et `OMÉGA` comme ∞ et ω .

la fonction `multiplication` fais pas la multiplication normale de matrice. en fait on prend deux matrices de la meme taille comme parametres, en suite, on cree un nouveau matrice en attend a retourner.on initialise tous les elelment dans la tableau a -1.puis on parcours toutes les case des deux tableaux de chacun.

Chapitre 7

Structure de stockage des matrices

(mautomate, description des fonctions de création)

Chapitre 8

Résolution du problème de limitedness

Bibliographie

- [1] J. Berstel, D. Beauquier, and P. Chrétienne. *Elements d'algorithmique*. 2005.
- [2] L. Eggan. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal*, 10 :385–397, 1963.
- [3] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 5201 :233–244, 1982.
- [4] H. Leung and V. Podolskiy. The limitedness problem on distance automata : Hashiguchi's method revisited. *Theoret. Comput. Sci.*, 310(1-3) :147–158, 2004.
- [5] I. Simon. On semigroups of matrices over the tropical semiring. *RAIRO Inform. Théor. Appl.*, 28(3-4) :277–294, 1994.