Université de Montréal

Programmes de branchement restreints pour un problème P-complet.

Par

Dominik Gehl

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
En vue de l'obtention du grade de
Maître ès science (M.Sc.)
En informatique

Août, 2001

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Programmes de branchement restreints pour un problème P-complet

présenté par :

Dominik Gehl

a été évalué par un jury composé des personnes suivantes :

Président-rapporteur
Monsieur Patrice Marcotte

Membre du jury
Monsieur Gilles Brassard

Directeur de recherche
Monsieur Pierre McKenzie

Mémoire accepté le 10 décembre 2001

Sommaire

Ce mémoire, intitulé *Programmes de branchement restreints pour un problème P-complet*, se veut une étude des tailles de divers types de programmes de branchement pour le problème de génération d'un groupoïde, communément appelé GEN.

Il est connu que le problème GEN est $\mathcal{P}$-complet. Par conséquent, il est improbable que GEN puisse être résolu par un programme de branchement de taille polynomiale. En gardant en tête l'hypothèse que tout programme de branchement résolvant GEN est de taille exponentielle, nous allons revisiter GEN en étudiant divers types de programmes de branchement le résolvant.

Le mémoire étudie d'abord en détail la complexité du problème GEN et de ses variantes. Par la suite, des algorithmes permettant de générer des programmes de branchement restreints pour GEN sont proposés. Tous ces programmes de branchement possèdent une taille exponentielle.

Finalement, plusieurs nouvelles bornes inférieures sur la taille des programmes de branchements pour GEN sont prouvées, sans toutefois être assez fortes pour arriver à prouver la conjecture initiale. Ce mémoire permet tout de même d'apporter un regard nouveau au problème posé, en particulier en prouvant que tout programme de branchement à lecture unique ainsi que tout *ordered binary decision diagram* (OBDD) pour GEN doit être de taille exponentielle.

Table des matières

Table des figures

Remerciements

Je voudrais tout d'abord exprimer ma grande reconnaissance à mon directeur, M. Pierre McKenzie, pour sa patience, son dévouement et ses encouragements constants.

Je tiens aussi à remercier mes parents, Theresia et Horst, pour leur support moral et leur amour.

Et finalement, j'aimerais dire merci du fond de mon cœur à Annabelle Franche pour sa patience lors de mes discours vagues et incompréhensibles et l'amour qu'elle m'a témoigné tout au long de la rédaction de ce mémoire.

Introduction

Ce mémoire traitera du problème de génération d'un groupoïde, GEN, qu'on sait $\mathcal{P}$-complet. Conceptuellement le problème GEN se résume à la question suivante : est-ce que, étant donnés une loi binaire interne pour les éléments 1 à n, et un ensemble de départ, n se trouve dans la fermeture de l'ensemble de départ sous l'opération interne ? Nous calculerons des bornes inférieures et supérieures pour plusieurs types de programmes de branchement pour GEN en tentant de fortifier l'hypothèse que GEN $\notin \mathcal{L}$. Nous essayerons de développer des méthodes de calcul, et surtout une intuition, pouvant servir à sa preuve formelle toujours inexistante. Au cours du premier chapitre, nous ferons un survol global et "pédagogique" des notions générales de l'informatique théorique, nous attardant particulièrement sur la définition de classes de complexité $\mathcal{P}$ et $\mathcal{L}$. Nous définirons aussi formellement le problème GEN et les divers types de programmes de branchement utilisés dans la suite du mémoire. Quiconque lira cet ouvrage pourra alors s'approprier les notions nécessaires qui lui permettront de suivre, et, je l'espère, de comprendre, les chapitres subséquents, ainsi que de bien saisir l'enjeu de la question si $\mathcal{P} = \mathcal{L}$. Le chapitre deux nous permettra d'explorer en détails la complexité du problème GEN en montrant tout d'abord que la probabilité que GEN_S = FAUX pour un groupoïde aléatoire tend vers zéro quand la taille du groupoïde tend vers l'infini. Nous montrerons ensuite que GEN, ainsi que ses restrictions GEN_C, GEN_S et

GEN_{SC} sont $\mathcal{P}$-complets. Une légère modification à l'énoncé du problème permet, par contre, d'obtenir un nouveau problème $\mathcal{NP}$-complet.

Nous montrerons par la suite, au chapitre trois, comment on peut créer, en pratique, des programmes de branchement pour GEN. Ceci nous permettra d'obtenir des bornes supérieures sur la taille de ces programmes de branchement.

Le dernier chapitre sera l'occasion d'établir des bornes inférieures sur la taille des programmes de branchement pour GEN. Nous prouverons en particulier des bornes inférieures exponentielles sur la taille d'un programme de branchement à lecture unique et d'un OBDD pour GEN, ce qui n'a jamais été prouvé auparavant. Les méthodes ayant permis d'obtenir ces résultats nous feront sans doute revoir le problème GEN sous un nouvel angle ; un nouvel angle qui, espérons-le, permettra dans l'avenir de prouver que $\text{GEN} \notin \mathcal{L}$.

Chapitre 1

Introduction, définitions et motivations

1.1 Machine de Turing

Definition formelle

Le modèle le plus courant d'un calcul séquentiel est la machine de Turing introduite par Alan Turing dans [Tur36].

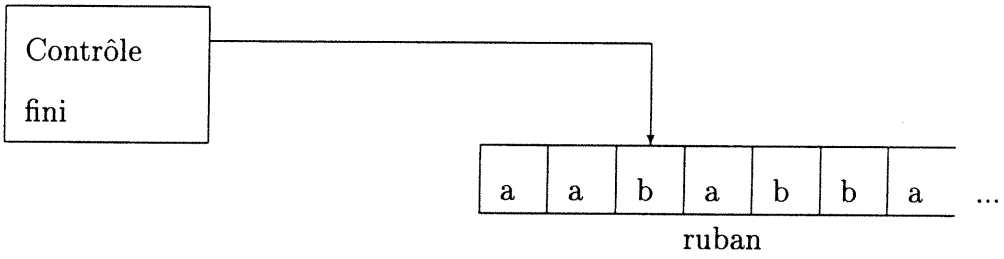


FIG. 1.1 – Machine de Turing

Une machine de Turing est composée d'un ruban semi-infini, d'une tête de lecture et d'écriture et d'un contrôle fini. Le travail de la machine de Turing consiste à accepter ou à rejeter un mot qui lui est présenté en entrée. Au début du calcul, le ruban de la machine de Turing ne contient que le mot en question. Pendant son travail, la machine peut lire et écrire à volonté des informations sur le ruban en

déplacant sa tête soit vers la gauche, soit vers la droite.

On peut définir, de façon rigoureuse, une machine de Turing et son calcul comme suit :

Définition 1 *Une machine de Turing est un septuplet* $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ *où*

- Q *est l'ensemble fini d'états,*
- Σ *est l'alphabet d'entrée avec* $\sqcup \notin \Sigma$,
- Γ *est l'alphabet du ruban avec* $\sqcup \in \Gamma$ *et* $\Sigma \subseteq \Gamma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{G, D\}$ *est la fonction de transition,*
- q_0 *est l'état initial,*
- q_a *est l'état acceptant,*
- q_r *est l'état rejetant.*

À tout moment, on peut décrire exactement l'avancement du calcul par la configuration de la machine de Turing, c'est-à-dire son état, la position de la tête et le contenu de son ruban. Comme la fonction de transition de la machine est déterministe, il est possible de déterminer la prochaine configuration à partir de la configuration actuelle.

Définition 2 *Une configuration d'une machine de Turing* $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ *est un triplet* (u, q, v) *avec* $q \in Q$ *et* $u, v \in \Gamma^*$. *Le ruban contient le mot* uv *et la tête se trouve au dessus du premier symbole de* v. ε *dénote la chaîne vide.*
La configuration (u, q, av) *mène à la configuration* (ub, q', v), *si* $\delta(q, a) = (q', b, D)$.
La configuration (uc, q, av) *mène à la configuration* (u, q', cbv) *si* $\delta(q, a) = (q', b, G)$.
La configuration (ε, q, av) *mène à la configuration* (ε, q, av) *si* $\delta(q, a) = (q', b, G)$.
On note $c \Rightarrow^* c'$ *si* c *mène à* c' *en* n *étapes où* $n \geq 0$.

Avec ces définitions, on peut maintenant définir formellement l'acceptation et le rejet d'un mot par une machine de Turing.

Définition 3 *Une machine de Turing M accepte un mot w s'il existe $u, u_1, \ldots, u_m$, $v, v_1, \ldots, v_m \in \Gamma^*$ et $q_{n_1}, \ldots, q_{n_m} \in Q - \{q_a, q_r\}$ tels que*

$$(\varepsilon, q_0, w) \Rightarrow (u_1, q_{n_1}, v_1) \Rightarrow \cdots \Rightarrow (u_m, q_{n_m}, v_n) \Rightarrow (u, q_a, v)$$

Une machine de Turing M rejette w s'il existe $u, u_1, \ldots, u_m, v, v_1, \ldots, v_m \in \Gamma^*$ et $q_{n_1}, \ldots, q_{n_m} \in Q - \{q_a, q_r\}$ tels que

$$(\varepsilon, q_0, w) \Rightarrow (u_1, q_{n_1}, v_1) \Rightarrow \cdots \Rightarrow (u_m, q_{n_m}, v_n) \Rightarrow (u, q_r, v)$$

Si ni l'une ni l'autre de ces conditions ne s'applique, on dit que M boucle sur w.

La machine de Turing est un modèle de calcul fondamental à cause de sa simplicité et surtout à cause de l'existence de la "thèse de Church-Turing" qui stipule que tout problème résoluble par un algorithme peut être résolu par une machine de Turing. Une importante partie de l'informatique théorique est bâtie sur cette notion de machine de Turing.

Il y a plusieurs variantes de machines de Turing. Il est, par exemple, parfois utile, parfois même nécessaire, de donner plusieurs rubans à la machine de Turing. Dans une de ces variantes, on sépare le ruban d'entrée, qui devient un ruban à lecture seulement, du ruban de travail qui contient des résultats intermédiaires du calcul. Il est évident que malgré cette modification, ce modèle de machine de Turing est équivalent, en termes de puissance de calcul, au modèle de base.

Il est possible de modifier la machine encore plus en ajoutant un troisième ruban oracle tel que l'explique la définition (tirée de [Weg00]) ci-bas.

Définition 4 *Une machine de Turing non-uniforme est une machine de Turing équipée d'un ruban oracle à lecture seulement. Pour des entrées w de longueur n,*

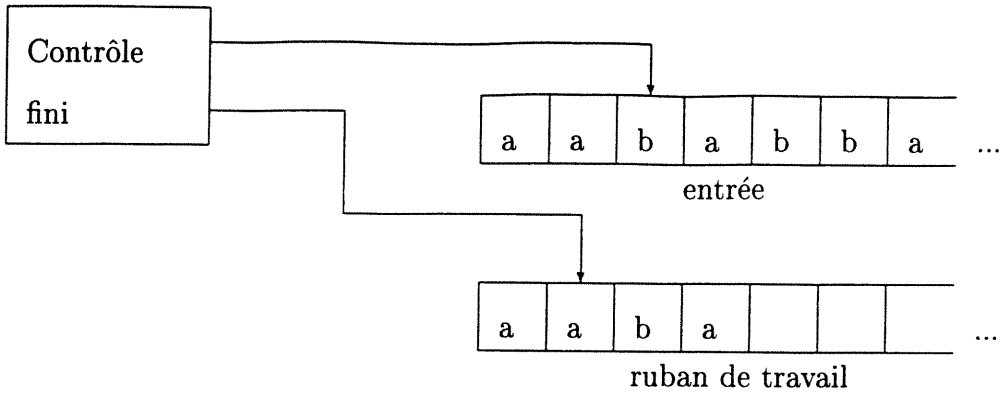


FIG. 1.2 – Machine de Turing avec ruban d'entrée et ruban de travail

le ruban oracle contient un oracle qui ne dépend que de la longueur de l'entrée et non pas de l'entrée elle-même.

Une autre variante très souvent utilisée est la machine de Turing non-déterministe dont la fonction de transition peut contenir, pour chaque transition, plusieurs choix. Avec ce changement, on se rend vite compte qu'il faut redéfinir l'acceptation et le rejet d'un mot tout simplement puisque le même mot peut potentiellement mener à la fois à un état acceptant (en choisissant un chemin de calcul approprié) et à un état rejetant (en suivant un autre chemin de calcul). On dit alors qu'une machine de Turing accepte un mot si et seulement s'il existe (au moins) un chemin de calcul qui mène à un état acceptant. Autrement, la machine de Turing rejette le mot.

Dans un premier temps, il est nécessaire de savoir si un problème donné possède un algorithme, et ainsi une solution sur une machine de Turing ou un ordinateur réel. Mais ceci est loin d'être suffisant. À quoi bon un algorithme qui permet de résoudre un problème dans 10^9 années ou nécessitant plus de mémoire, c'est-à-dire plus de cases sur le ruban de la machine de Turing, que le nombre d'atomes dans l'univers ?

En pratique, l'intérêt est d'obtenir des réponses à une question dans un temps

"raisonnable" et avec une machine de Turing qui n'utilise qu'un nombre "raisonnable" de cases sur son ruban. Il est alors primordial de définir ce que l'on entend par un temps et un espace "raisonnables". En effet, ces deux contraintes ont mené à deux parties de la théorie de la complexité : les classes de complexité de temps et les classes de complexité d'espace.

Temps de calcul

Définition 5 *Le temps de calcul d'une machine de Turing déterministe M sur w est le nombre de transitions avant l'arrêt de M sur w.*

Puisqu'il existe éventuellement plusieurs chemins de calcul pour une machine de Turing non-déterministe, il faut définir son temps de calcul autrement.

Définition 6 *Le temps de calcul d'une machine de Turing non-déterministe M sur w est le nombre maximal de transitions avant l'arrêt de M sur w.*

Définition 7 *Le temps de calcul de M est la fonction*

$$\begin{aligned} f \quad &: \mathbb{N} \to \mathbb{N} \\ &n \mapsto \max_{|w|=n} \{\textit{temps de calcul de } M \textit{ sur } w\} \end{aligned}$$

Cette définition permet de prévoir une borne supérieure sur le temps de calcul de M pour une instance de toute taille d'un problème donné. Il est important de noter que l'encodage du problème est essentiel. Le temps de calcul pour un problème ne peut être déterminé qu'une fois l'encodage du problème connu. Par exemple, savoir si un nombre est composé est certainement plus difficile si le nombre est présenté uniquement dans sa représentation binaire que si le nombre est encodé comme la suite de ses facteurs premiers.

On peut maintenant classer tout problème (avec l'encodage spécifié) dans une classe de complexité de temps.

Définition 8

$$\begin{aligned} \mathrm{DTIME}(t(n)) &= \{L | L \textit{ est décidé par une machine de Turing déterministe en temps } O(t(n))\} \\ \mathrm{NTIME}(t(n)) &= \{L | L \textit{ est décidé par une machine de Turing non-déterministe en temps } O(t(n))\} \end{aligned}$$

Définition 9

$$\begin{aligned} \mathcal{P} &= \cup_{k \in \mathbb{N}} \mathrm{DTIME}(n^k) \\ \mathcal{NP} &= \cup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k) \end{aligned}$$

$\mathcal{P}$ et $\mathcal{NP}$ sont donc l'ensemble de tous les langages pouvant être décidés par une machine de Turing déterministe, respectivement non-déterministe, en temps polynomial. Il est généralement reconnu que les problèmes qui se trouvent dans $\mathcal{P}$ possèdent une solution efficace en calcul séquentiel et que les problèmes qui se trouvent dans $\mathcal{NP} - \mathcal{P}$ ne possèdent pas de solution efficace. Puisque toute machine de Turing déterministe peut être considérée comme un cas particulier d'une machine de Turing non-déterministe, on a $\mathcal{P} \subseteq \mathcal{NP}$. Par contre, la question si cette inclusion est stricte, c'est-à-dire si $\mathcal{NP} \neq \mathcal{P}$, est une des grandes questions ouvertes de l'informatique théorique.

Espace de calcul

Tel que déjà mentionné, il n'y a pas que le temps de calcul, mais aussi l'espace utilisé qui est un facteur primordial afin de savoir si un problème peut être résolu en pratique.

Définition 10 *Soit M une machine de Turing déterministe possédant un ruban d'entrée à lecture seulement et un ruban de travail à lecture et écriture. L'espace de calcul de M sur un mot w est le nombre de cellules visitées sur le ruban de travail lors du calcul de M sur w.*

C'est afin de pouvoir parler d'une occupation d'espace sous-linéaire que nous avons séparé le ruban d'entrée du ruban de travail. En effet, une lecture complète du mot w de longueur n demande un passage sur les n premières cellules du ruban d'entrée et par conséquent, sans séparation du ruban d'entrée du ruban de travail, toute utilisation sous-linéaire d'espace équivaut à une lecture incomplète de l'entrée.
Dans le cas d'une machine de Turing non-uniforme, on a la définition similaire suivante :

Définition 11 *L'espace de calcul d'une machine de Turing non-uniforme M sur w est la somme du nombre de cellules visitées sur le ruban de travail et du logarithme de la longueur de l'oracle.*

L'utilité d'inclure dans la définition ci-dessus le logarithme de la longueur de l'oracle trouvera son explication dans l'énoncé du théorème 1.

Définition 12 *L'espace de calcul de M est la fonction*

$$\begin{aligned} f \quad &: \mathbb{N} \to \mathbb{N} \\ &n \mapsto \max_{|w|=n} \{\textit{espace de calcul de } M \textit{ sur } w\} \end{aligned}$$

On peut maintenant associer à chaque problème une classe de complexité d'espace comme on le faisait précédemment avec le temps.

Définition 13

$$\mathrm{DSPACE}(s(n)) = \{L | L \textit{ est décidé par une machine de Turing déterministe en espace } \mathcal{O}(s(n))\}$$

Définition 14

$$\mathcal{L} = \mathrm{DSPACE}(\log n)$$

$\mathcal{L}$ est la classe des problèmes qui peuvent être résolus par des machines de Turing déterministes en espace $\mathcal{O}(\log n)$.

Une fois qu'on connaît les demandes en temps et espace pour certains problèmes, il devient intéressant de se questionner sur la relation entre temps et espace de calcul, et en particulier si $\mathcal{L} = \mathcal{P}$. L'inclusion $\mathcal{L} \subseteq \mathcal{P}$ est connue. Par contre, on ne sait toujours pas si $\mathcal{P} \subseteq \mathcal{L}$; on ne sait même pas si $\mathcal{P} \subseteq \cup_{k \in \mathbb{N}} \mathrm{DSPACE}(\log^k n)$.

Réduction et complétude

Il est clair que si on peut trouver des solutions plus efficaces pour les problèmes les "plus difficiles" d'une classe de complexité, tous les problèmes de cette classe bénéficieront théoriquement de la nouvelle solution. Et c'est afin de pouvoir comparer le degré de difficulté des problèmes dans une même classe de complexité que les notions de réduction et de complétude ont été introduites.

Définition 15

- *Le langage A est réductible au langage A' en temps polynomial ($A \leq_P A'$) s'il existe une fonction f, calculable en temps polynomial, avec $x \in A \Leftrightarrow f(x) \in A'$.*

- *Un langage A est réductible au langage A' en espace logarithmique ($A \leq_L A'$) s'il existe une fonction f, calculable en espace logarithmique, satisfaisant la condition $x \in A \Leftrightarrow f(x) \in A'$.*

Il est aisé et fort utile de remarquer que les opérations de réduction, $\leq_L$ et $\leq_P$, sont toutes deux transitives.

Définition 16

- *Un langage A est $\mathcal{NP}$-complet si*

  1. $A \in \mathcal{NP}$

  2. *Pour tout langage $B \in \mathcal{NP}$, on a $B \leq_P A$.*

- *Un langage A est $\mathcal{P}$-complet si*

  1. $A \in \mathcal{P}$

  2. *Pour tout langage $B \in \mathcal{P}$, on a $B \leq_L A$.*

Ainsi l'existence d'un seul problème $\mathcal{P}$-complet qui se trouve dans $\mathcal{L}$ implique que tous les problèmes de $\mathcal{P}$ se trouveront dans $\mathcal{L}$ et par conséquent que $\mathcal{P} = \mathcal{L}$. En effet, supposons que A soit un problème $\mathcal{P}$-complet pour lequel il existe une solution en espace logarithmique. On peut alors trouver une solution en espace logarithmique pour tout problème dans $\mathcal{P}$ en calculant d'abord sa réduction vers A et ensuite la solution pour A. De la même façon la preuve de l'existence d'un problème $\mathcal{NP}$-complet qui se trouve dans $\mathcal{P}$, impliquerait que $\mathcal{P} = \mathcal{NP}$.

1.2 Programmes de branchement

Définition formelle

Un modèle de calcul alternatif est celui des programmes de branchement. Historiquement, il est issu de la nécessité de calculer des fonctions booléennes à l'aide de circuits [Lee59, Mas76].

Un programme de branchement est un graphe dans lequel chaque nœud permet de consulter la valeur d'une variable et implique un branchement en fonction de cette valeur. Deux nœuds particuliers, des puits, ne comportent pas de branchement, mais indiquent la valeur de la fonction calculée.

Définition 17 *Un programme de branchement est un graphe orienté acyclique composé*

- *d'un nœud de degré intérieur 0 (la source) et de degré extérieur 2,*
- *de nœuds intérieurs de degré extérieur 2 étiquetés par des variables booléennes,*
- *de deux nœuds de degré extérieur 0 (les puits) étiquetés par les constantes booléennes 1 et 0 (ou* OUI *et* NON*),*
- *d'arcs étiquetés par des constantes booléennes de sorte à ce que chaque nœud de degré extérieur 2 possède un arc sortant étiqueté par* OUI *et un arc sortant étiqueté par* NON.

Un programme de branchement calcule une fonction $f : \{0,1\}^k \to \{0,1\}$. *La valeur* $f(x)$ *est obtenue par la procédure suivante :*

- *se placer à la source du graphe ;*
- *répéter jusqu'à l'arrivée dans un des deux puits :*

  à partir du nœud courant, étiqueté x_i, *suivre l'arc dont l'étiquette correspond à la valeur de* x_i *;*

L'étiquette du puits atteint est la valeur de la fonction.

Comme dans le cadre des machines de Turing, il est important de définir la complexité de temps et la complexite d'espace, ici reliées au nombre de nœuds et à la longueur du chemin le plus long.

Définition 18 *La taille d'un programme de branchement est égale à son nombre de nœuds. La profondeur d'un programme de branchement est égale à la longueur de son chemin le plus long.*

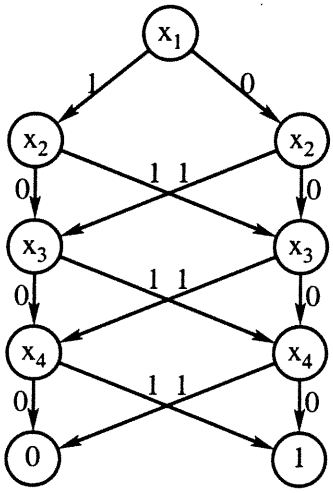


FIG. 1.3 – Programme de branchement pour la fonction parité de quatre bits

La figure 1.3 montre un programme de branchement qui calcule la fonction parité de quatre bits. La fonction parité est égale à 1 si et seulement si l'entrée comporte un nombre pair de 1. Elle peut être calculée en gardant en mémoire seulement la parité des bits lus précédemment. La partie gauche du programme de branchement correspond a une parité impaire, la partie droite à une parité paire. La taille de ce programme de branchement est 9 et sa profondeur est 4.

Définition 19 *Soit f_n une fonction booléenne, c'est-à-dire $f_n : \{0,1\}^n \to \{0,1\}$. Alors* $\mathrm{PB}(f_n)$ *est la taille du plus petit programme de branchement pour f_n.*

Comme dans le cadre des machines de Turing, de nombreuses variantes de programmes de branchement existent. Nous en utiliserons plusieurs dans le cadre de ce mémoire.

On peut généraliser la notion de programme de branchement afin de permettre plus de deux branchements à chaque nœud.

Définition 20 *Un programme de branchement n-généralisé est un graphe acyclique orienté composé*

- *d'un nœud de degré intérieur 0 (la source) et de degré extérieur n,*
- *de nœuds intérieurs de degré extérieur n étiquetés par des variables booléennes,*
- *de deux nœuds de degré extérieur 0 (les puits) étiquetés par les constantes booléennes 1 et 0 (ou* OUI *et* NON*),*
- *d'arcs étiquetés par des constantes dans $\{1, 2, \ldots, n\}$ de sorte à ce que chaque nœud de degré extérieur n possède un et un seul arc sortant étiqueté par i pour tout $1 \leq i \leq n$.*

Ce programme calcule une fonction $f : \{1 \ldots n\}^m \rightarrow \{0, 1\}$.

La procédure permettant de calculer la valeur de la fonction décrite par le programme de branchement suit le cas $n = 2$.

Il est souvent aussi fort utile de limiter l'accès d'un programme de branchement aux variables d'entrée. On peut ainsi restreindre le nombre de fois qu'un programme de branchement consulte les variables, ce qui mène à la notion de programme de branchement à degré de lecture k.

Définition 21 *Un programme de branchement à degré de lecture k (read-k-times-only branching program) est un programme de branchement dans lequel chaque variable est testée au plus k fois dans n'importe quel chemin de calcul.*

Un cas particulier des programmes de branchement à degré de lecture k est celui

des programmes de branchement à lecture unique.

Définition 22 *Un programme de branchement à lecture unique (read-once branching program ou FBDD – free binary decision diagram) est un programme de branchement dans lequel chaque variable est testée au plus une fois dans n'importe quel chemin de calcul.*

Bryant ([Bry85]) a introduit un type de programme de branchement dans lequel la consultation de l'entrée doit se faire selon un ordre prédéfini.

Définition 23 *Un ordonnancement π de $X_n = \{x_1, x_2, \ldots, x_n\}$ est une permutation de l'ensemble d'indices $I = \{1, 2, \ldots, n\}$.*

Définition 24 *Un π-OBDD (ordered binary decision diagram ou diagramme de décision binaire ordonné) est un programme de branchement où pour tout arc menant du nœud x_i au nœud x_j, on a $\pi(i) < \pi(j)$.*
Un OBDD est un π-OBDD pour un certain ordonnancement π.

Cette définition implique que chaque variable apparaît au plus une fois le long d'un chemin. Un OBDD est donc un cas particulier des programmes de branchement à lecture unique.
Les arbres de décision (*decision tree*) sont une autre variante des programmes de branchement.

Définition 25 *Un arbre de décision est un programme de branchement dont le graphe est un arbre.*

Vu la structure d'arbre, il est facile de voir que si une variable est testée plus d'une fois le long d'un chemin de calcul, le chemin comporte nécessairement une incohérence. En effet, supposons que la variable i est testée deux fois, aux nœuds n_s et n_t, le long d'un chemin de calcul. Sans perte de généralité, soit 0 l'affectation de i permettant de rejoindre le nœud n_t à partir du nœud n_s. Le chemin passant

par n_s et n_t et suivant, en partant du nœud n_t, l'arc étiqueté 1 constitue alors un chemin incohérent : la variable i ne peut pas prendre à la fois la valeur 0 et la valeur 1. Il est d'ailleurs possible d'enlever tous les chemins incohérents d'un arbre de décision T dans un temps $\mathcal{O}(|T|)$ [Weg00]. En exigeant qu'aucun chemin ne contienne une incohérence, les arbres de calcul deviennent alors, comme les OBDD, un cas particulier des programmes de branchement à lecture unique.

Relation entre machines de Turing et programmes de branchement

La profondeur d'un programme de branchement est reliée de manière évidente au temps de calcul dans le modèle d'une machine de Turing. En effet, suivre un arc dans un programme de branchement correspond intuitivement à une étape de calcul d'une machine de Turing et vice-versa.
La taille d'un programme de branchement peut être reliée à l'espace d'une machine de Turing non-uniforme grâce au théorème suivant [Cob66, Weg00].

Théorème 1 *Soit f une fonction booléenne pour laquelle il existe une borne inférieure à l'utilisation d'espace de toute machine de Turing M qui résoud f. Soit alors $S_f(n)$ la complexité d'espace de cette fonction booléenne f par rapport à une machine de Turing non-uniforme. De plus, soit $S_f(n) \geq \log n$ et $\mathrm{BP}(f_n) \geq n$. Alors $S_f(n) = \Theta(\log \mathrm{BP}(f_n))$*

Par conséquent, une fonction possède un programme de branchement de taille polynomiale si et seulement si elle peut être calculée par une machine de Turing non-uniforme en espace $\mathcal{O}(\log n)$. Ceci implique que si la fonction a besoin d'un programme de branchement de taille sur-polynomiale, le problème ne peut pas être traité en espace $\mathcal{O}(\log n)$ par une machine de Turing non-uniforme, et *a fortiori*

par une machine de Turing uniforme. Une manière de séparer $\mathcal{P}$ de $\mathcal{L}$ est donc de trouver une fonction dans $\mathcal{P}$ qui nécessite une taille de programme de branchement sur-polynomiale.

1.3 Calcul parallèle

Un ordinateur parallèle peut exécuter plusieurs instructions en même temps. Par ailleurs, le calcul parallèle est de plus en plus fréquent dans tous les domaines de l'informatique : *cluster*, calcul distribué, super-ordinateurs, les utilisations et réalisations abondent.

Tout comme on se pose la question si un problème peut être résolu en pratique de manière séquentielle, on peut aussi se demander si un problème se prête au calcul parallèle. Autrement dit, existe-t-il des problèmes qui sont fondamentalement séquentiels, qu'on ne peut pas décomposer de manière efficace afin de les traiter en parallèle ?

Afin de répondre à cette question, il importe de définir de nouveau ce que l'on entend par "se prêter" au calcul parallèle. Selon [GHR95], un problème est résoluble de manière hautement parallèle (*feasible highly parallel*) s'il existe un algorithme avec une complexité de temps en pire cas de $(\log n)^{\mathcal{O}(1)}$ et exigeant $n^{\mathcal{O}(1)}$ processeurs.

L'hypothèse du calcul parallèle (*Parallel Computation Thesis* [CS76]) relie l'espace séquentiel au temps parallèle. Ainsi, on peut profiter de la théorie bien développée du calcul séquentiel et appliquer ses résultats au calcul parallèle. La thèse du calcul parallèle indique que l'espace séquentiel est relié de manière polynomiale au temps parallèle.

En faisant de nouveau le lien avec les programmes de branchement, on voit qu'une

fonction exigeant un programme de branchement de taille exponentielle exigera un temps parallèle plus que polylogarithmique et n'est donc pas résoluble de manière hautement parallèle.

Si on pouvait trouver une fonction dans $\mathcal{P}$ qui nécessite une taille de programme de branchement exponentielle, on répondrait alors d'un coup à deux questions ouvertes très importantes de l'informatique théorique :

- Est-ce que $\mathcal{L} = \mathcal{P}$?
- Existe-t-il des problèmes pour lesquels on connaît une solution séquentielle efficace, mais qui ne se prêtent pas au calcul hautement parallèle ?

1.4 GEN

Dans la suite du mémoire, nous allons étudier la taille de divers types de programmes de branchement pour le problème de génération d'un groupoïde GEN, défini comme suit :

Définition 26 *Un groupoïde $G = (E, \times)$ est un ensemble $E = \{1, 2, \ldots, n\}$ muni d'une opération binaire interne, notée $\times$.*

Étant donné un groupoïde $G = (E, \times)$ et un ensemble $T \subseteq E$, on note $\langle T \rangle$ le plus petit groupoïde contenant T qui est fermé sous l'opération binaire $\times$.

Définition 27 *On nomme* GEN *le problème*

Donnée : *Un groupoïde $G = (E = \{1, 2, \ldots, n\}, \times)$ et un ensemble $T \subseteq E$.*

Question : *Est-ce que $n \in \langle T \rangle$?*

On peut restreindre le problème GEN en fixant T ou en imposant certaines propriétés à l'opération binaire. Voici quelques notations pour des cas particuliers :

- On note GEN_S (S pour "singleton") le problème GEN dans le cas où $T = \{1\}$.

- On note GEN_C le problème GEN dans le cas où l'opération binaire $\times$ est commutative.
- On note GEN_{SC} le problème GEN dans le cas où $T = \{1\}$ et où l'opération binaire $\times$ est commutative.

Une autre restriction possible est d'imposer que seulement une partie de la table de multiplication contienne des valeurs différentes de 1. On peut ainsi obtenir GEN_{1ligne} où seulement la première ligne du tableau de multiplication peut contenir des valeurs différentes de 1 et $T = \{1\}$.

Considérons le problème GEN_S pour l'opération binaire suivante :

$\times$	1	2	3
1	2	1	1
2	2	3	3
3	1	1	1

Dans ce cas $3 \in \langle\{1\}\rangle$ puisque $1 \times 1 = 2$ et $2 \times 2 = 3$.

En changeant la définition de l'opération binaire légèrement $(2 \times 2 = 2)$, on obtient

$\times$	1	2	3
1	2	1	1
2	2	2	3
3	1	1	1

Dans ce cas $3 \notin \langle\{1\}\rangle$ puisque $1 \times 1 = 2$ mais $1 \times 2 \neq 3$, $2 \times 1 \neq 3$ et $2 \times 2 \neq 3$.

Il existe plusieurs manières d'encoder les exemplaires de GEN. Les deux encodages les plus courants sont :

1. L'encodage de la table de multiplication en $n^2 \log_2 n$ bits et celui de l'ensemble de départ T en $|T| \log_2 n$ bits.
2. L'encodage de la table de multiplication en n^3 bits où (i, j, k) =VRAI si et seulement si $i \times j = k$ et celui de l'ensemble de départ T en n bits (l =VRAI si et seulement $l \in T$).

Peu importe l'encodage choisi, on n'encode pas l'ensemble de départ s'il est fixé, c'est-à-dire dans GEN_S et GEN_{SC}, seule la table de multiplication sera encodée.

Après toutes ces définitions, nous pouvons maintenant nous attarder au cœur de notre sujet : la complexité du problème GEN et la taille des programmes de branchement résolvant GEN.

Chapitre 2

Complexité du problème GEN

Dans le présent chapitre nous examinerons le degré de difficulté du problème GEN. Nous allons d'abord montrer que plus la taille d'un groupoïde est grande, plus faible est la probabilité que $n \notin \langle\{1\}\rangle$. Plus précisément, nous allons montrer que $\lim_{n\to\infty} P(n \notin \langle 1\rangle) = 0$, ce qui constitue un des résultats nouveaux de ce mémoire. Ceci pourrait à première vue sembler indiquer que le problème GEN n'est pas très difficile à résoudre : un algorithme probabiliste pourrait, dès que n est suffisamment grand, donner avec une très forte probabilité un résultat juste ; on pourrait par exemple penser à un algorithme qui répond tout simplement FAUX avec probabilité $\frac{1}{n}$. Par contre, nous prouverons, par la suite, que GEN et ses variantes GEN_S, GEN_C et GEN_{SC} sont tous des problèmes $\mathcal{P}$-complets. Une légère modification du problème GEN mène même à un nouveau problème $\mathcal{NP}$-complet dont la preuve de la $\mathcal{NP}$-complétude constitue un autre résultat original de ce mémoire.

2.1 Probabilité que GEN_S = vrai

Examinons tout d'abord la formule mathématique exacte qui décrit la probabilité pour un groupoïde aléatoire que GEN_S = VRAI.

Définition 28 *Soient $T(m,n)$ et $P(m,n)$, $1 \leq m \leq n$, respectivement le nombre et la fraction des n^{n^2} groupoïdes de taille $n \times n$ tels que $\langle\{1\}\rangle = \{1,2,\ldots,m\}$.*

Il est important de souligner le fait que l'ordre des éléments du groupoïde dans la table de multiplication n'est qu'une convention arbitraire. Par conséquent le nombre de groupoïdes de taille $n \times n$ tels que $\langle\{1\}\rangle = \{1,2,\ldots,m\}$ est égal au nombre de groupoïdes de taille $n \times n$ tels que $\langle\{1\}\rangle = \{1,c_1,c_2,\ldots,c_{m-1}\}$ où $c_i \in \{2,3,\ldots n\}$ pour tout i et $c_i \neq c_j$ pour $i \neq j$.

Théorème 2

$$\begin{aligned} T(1,n) &= n^{n^2-1} \\ T(m,n) &= n^{n^2-m^2}\left(m^{m^2} - \sum_{i=1}^{m-1}\binom{m-1}{i-1}T(i,m)\right) \end{aligned}$$

Preuve

Remarquons que pour que $\langle\{1\}\rangle = \{1\}$, il est nécessaire et suffisant que $1 \times 1 = 1$. La table de multiplication contient donc une case (celle représentant la valeur de la multiplication 1×1) fixée à 1 et $n^2 - 1$ cases qui peuvent chacune indépendamment prendre une valeur entre 1 et n. Par conséquent $T(1,n) = n^{n^2-1}$.

Pour que $\langle\{1\}\rangle = \{1,2,\ldots,m\}$, il faut que les premières m^2 cases de la table de multiplication ne contiennent que des nombres entre 1 et m. En effet, soit $1 \leq i,j \leq m$ et $k > m$. Supposons au contraire que $\{1,2,\ldots,m\} \subseteq \langle\{1\}\rangle$ et que $i \times j = k$. Puisque $i \in \langle\{1\}\rangle$ et $j \in \langle\{1\}\rangle$, on a donc aussi $k \in \langle\{1\}\rangle$. Par conséquent $\{1,2,\ldots,m,k\} \subseteq \langle\{1\}\rangle$ et $\langle\{1\}\rangle \subseteq \{1,2,\ldots,m\}$ n'est plus vrai ce qui

contredit notre hypothèse.

Le nombre de groupoïdes vérifiant cette première condition est $n^{n^2-m^2} \cdot m^{m^2}$ (m^2 cases peuvent prendre une valeur entre 1 et m, $n^2 - m^2$ cases une valeur entre 1 et n). De ce nombre, il faut enlever tous les groupoïdes de taille $m \times m$ dont la fermeture est plus petite, c'est-à-dire dont la fermeture contient seulement i éléments distincts, $1 \leq i \leq m-1$. Puisque chacune de ces fermetures doit contenir 1, il y a $\binom{m-1}{i-1}$ façons de choisir les i éléments qui la composent et par conséquent $\binom{m-1}{i-1}T(i,m)$ tels groupoïdes. Donc

$$T(m,n) = n^{n^2-m^2}\left(m^{m^2} - \sum_{i=1}^{m-1}\binom{m-1}{i-1}T(i,m)\right)$$

■

Corollaire 1

$$\begin{aligned} P(1,n) &= \frac{1}{n} \\ P(m,n) &= \left(\frac{m}{n}\right)^{m^2}\left(1 - \sum_{i=1}^{m-1}\binom{m-1}{i-1}P(i,m)\right) \end{aligned}$$

Preuve

Puisqu'il existe n^{n^2} groupoïdes de taille n, on a par définition $P(i,n) = \frac{T(i,n)}{n^{n^2}}$.

En posant $i = 1$ on obtient alors $P(1,n) = \frac{n^{n^2-1}}{n^{n^2}} = \frac{1}{n}$.

Pour $i > 1$, on obtient

$$\begin{aligned} P(m,n) &= \frac{T(m,n)}{n^{n^2}} \\ &= \frac{n^{n^2-m^2}\left(m^{m^2} - \sum_{i=1}^{m-1}\binom{m-1}{i-1}T(i,m)\right)}{n^{n^2}} \\ &= \left(\frac{m}{n}\right)^{m^2}\left(1 - \sum_{i=1}^{m-1}\binom{m-1}{i-1}P(i,m)\right) \end{aligned}$$

■

Théorème 3 $P(n \notin \langle\{1\}\rangle) = \sum_{m=1}^{n-1} \binom{n-2}{m-1} P(m,n)$

Preuve

$P(n \notin \langle 1\rangle)$ est la somme des probabilités d'obtenir une fermeture qui ne contient pas n. On a donc

$$\begin{aligned} P(n \notin \langle 1\rangle) &= P(\langle 1\rangle = \{1\}) + \\ &\quad P(\langle 1\rangle = \{1,2\}) + P(\langle 1\rangle = \{1,3\}) + \cdots + P(\langle 1\rangle = \{1,n-1\}) \\ &\quad P(\langle 1\rangle = \{1,2,3\}) + P(\langle 1\rangle = \{1,2,4\}) + P(\langle 1\rangle = \{1,2,5\}) \ldots \\ &\quad \vdots \\ &\quad P(\langle 1\rangle = \{1,2,3,\ldots,n-1\}) \\ &= \sum_{m=1}^{n-1} \binom{n-2}{m-1} P(m,n) \end{aligned}$$

■

En se servant du corollaire 1, on peut majorer $P(m,n)$ par $\left(\frac{m}{n}\right)^{m^2}$ et on obtient alors $P(n \notin \langle 1\rangle) \leq \sum_{m=1}^{n-1} \binom{n-2}{m-1} \left(\frac{m}{n}\right)^{m^2}$.

n	$P(n \notin \langle\{1\}\rangle)$
2	0.500
3	0.432
4	0.348
5	0.268081
6	0.207111
7	0.166167
8	0.13931
9	0.120687
10	0.10685
50	0.0200614

FIG. 2.1 – Quelques valeurs approximatives de $P(n \notin \langle\{1\}\rangle)$

Quand on regarde la figure 2.1 affichant $P(n \notin \langle\{1\}\rangle)$ en fonction de n, on remarque que P diminue et semble tendre vers 0 quand n tend vers l'infini. On va maintenant formellement prouver qu'on a en effet $\lim_{n\to\infty} P(n \notin \langle\{1\}\rangle) = 0$.

Théorème 4 *Pour* $n \geq 48$ *et* $2 \leq m \leq n-1$, $\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} \leq \frac{1}{n^2}$.

Preuve

- $m = 2$

Les inégalités suivantes sont vraies en autant que $n \geq 16$

$$\begin{aligned}\binom{n-2}{1}\left(\frac{2}{n}\right)^4 &= (n-2)\frac{16}{n^4} \\ &= \frac{n-2}{n}\frac{16}{n^3} \\ &\leq \frac{16}{n}\frac{1}{n^2} \\ &\leq \frac{1}{n^2}\end{aligned}$$

- $m = 3$

$$\begin{aligned}\binom{n-2}{2}\left(\frac{3}{n}\right)^9 &= \frac{(n-2)(n-3)}{2!}\left(\frac{3}{n}\right)^9 \\ &= \frac{(n-2)(n-3)}{2n^2}\frac{3^9}{n^7}\end{aligned}$$

Puisque $(n+6)(n-1) = n^2+5n-6 \geq 0$ pour $n \geq 1$, on a aussi $(n-2)(n-3) = n^2 - 5n + 6 \leq 2n^2$ pour $n \geq 1$. On peut donc écrire

$$\binom{n-2}{2}\left(\frac{3}{n}\right)^9 \leq \frac{3^9}{n^5}\frac{1}{n^2}$$

Il est facile de vérifier que pour $n \geq 8$, on a $n^5 \geq 32768 \geq 19683 = 3^9$. Ceci

permet de conclure que

$$\binom{n-2}{2}\left(\frac{3}{n}\right)^9 \leq \frac{1}{n^2}$$

- $4 \leq m \leq \frac{n}{2}$

$$\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} = \frac{(n-2)(n-3)\ldots(n-m)}{(m-1)!}\left(\frac{m}{n}\right)^{m^2}$$

Remarquons que $(n-2)(n-3)\ldots(n-m) \leq n^{m-1}$ et que $(m-1)! \geq m$. Par conséquent

$$\begin{aligned}\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} &\leq \frac{n^{m-1}}{m}\left(\frac{m}{n}\right)^{m^2} \\ &= m^m\left(\frac{m}{n}\right)^{m^2-m-1}\cdot\frac{1}{n^2} \\ &\leq m^m\left(\frac{1}{2}\right)^{m^2-m-1}\cdot\frac{1}{n^2} \\ &= \frac{1}{2^{m^2-m-1-m\log_2(m)}}\cdot\frac{1}{n^2}\end{aligned}$$

L'observation que $m^2 - m - 1 - m\log_2(m) \geq 0$ pour $m \geq 4$ permet alors de conclure que

$$\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} \leq \frac{1}{n^2}$$

- $\frac{n}{2} \leq m \leq n-1$

$$\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} = \frac{(n-2)(n-3)\ldots(m-1)m}{(n-m-1)!}\left(\frac{m}{n}\right)^{m^2}$$

Puisque $(n-2)(n-3)\ldots(m-1)m \leq n^{n-m-1} \leq n^{n-m}$ et $(n-m-1)! \geq 1$, on

peut écrire

$$\begin{aligned}\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} &\leq n^{n-m}\left(\frac{m}{n}\right)^{m^2} \\ &= n^{n-m}e^{m^2\ln\left(\frac{m}{n}\right)}\end{aligned}$$

Remarquons que pour $\frac{1}{2} \leq x \leq 1$, $\ln(x) - x + 1 \leq 0$. Par conséquent $\ln\left(\frac{m}{n}\right) \leq \frac{m-n}{n}$ pour $\frac{n}{2} \leq m \leq n$. Donc

$$\begin{aligned}\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} &\leq n^{n-m}e^{m^2\frac{m-n}{n}} \\ &\leq n^{n-m}e^{\left(\frac{n}{2}\right)^2\frac{m-n}{n}} \\ &= n^{n-m}e^{(m-n)\frac{n}{4}} \\ &= \frac{1}{e^{(n-m)\left(\frac{n}{4}-\ln(n)\right)}}\end{aligned}$$

Par ailleurs $\frac{n}{4} - \ln(n) \geq 2\ln(n)$ pour $n \geq 48$. D'où

$$\begin{aligned}\binom{n-2}{m-1}\left(\frac{m}{n}\right)^{m^2} &\leq \frac{1}{e^{(n-m)2\ln(n)}} \\ &\leq \frac{1}{e^2\ln(n)} \\ &= \frac{1}{n^2}\end{aligned}$$

■

Nous pouvez alors très facilement prouver le théorème suivant :

Théorème 5 $\lim_{n\to\infty} P(n \notin \langle 1\rangle) = 0$

Preuve

Pour $n \geq 48$, on a

$$\begin{aligned} P(n \notin \langle 1 \rangle) &\leq \sum_{m=1}^{n-1} \binom{n-2}{m-1} \left(\frac{m}{n}\right)^{m^2} \\ &\leq \frac{1}{n} + \sum_{m=2}^{n-1} \frac{1}{n^2} \\ &\leq \frac{1}{n} + \frac{1}{n} \\ &= \frac{2}{n} \end{aligned}$$

Par conséquent $\lim_{n\to\infty} P(n \notin \langle\{1\}\rangle) = 0$. ■

Nous venons donc de prouver que la probabilité que $n \notin \langle\{1\}\rangle$ tend vers zéro quand n tend vers l'infini plus rapidement que la fonction $\frac{2}{n}$. Ne laissons cependant pas ce résultat nous tromper sur la difficulté du problème GEN : nous allons maintenant prouver que GEN est $\mathcal{P}$-complet.

2.2 GEN et GEN_C sont $\mathcal{P}$-complets

Historiquement, un des premiers problèmes prouvés $\mathcal{P}$-complets a été le problème d'évaluation d'un circuit, appelé CVP (*circuit value problem*). Ce problème a été proposé par Ladner en 1975 [Lad75].

Définition 29 *Soit* $B_k = \{f | f : \{0,1\}^k \to \{0,1\}\}$ *l'ensemble de toutes les fonctions booléennes k-aires.*

Un circuit booléen est un graphe orienté acyclique dont le type de chaque nœud $\tau(v) \in \{I\} \cup B_0 \cup B_1 \cup B_2$. *Un nœud de type I possède un degré intérieur 0 et représente une entrée. Le circuit comporte une porte ayant degré extérieur 0. Cette porte est la sortie.*

Un circuit booléen calcule la valeur d'une fonction en propageant les valeurs booléennes calculées par les nœuds le long des arcs jusqu'à ce que la valeur de la porte de sortie soit connue.

Définition 30 *L'encodage $\overline{\alpha}$ d'un circuit α est une suite de quadruplets (v, g, l, r) représentant les portes de α. La porte numérotée v est de type g où $g \in \{I\} \cup B_0 \cup B_1 \cup B_2$ et ses entrées de gauche et de droite sont respectivement l et r (s'ils existent). Une porte de type I représente une entrée.*

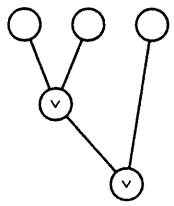


FIG. 2.2 – Représentation graphique d'un circuit

La figure 2.2 montre la représentation graphique du circuit encodé par

$$\langle (1, I, 0, 0), (2, I, 0, 0), (3, I, 0, 0), (4, \vee, 1, 2), (5, \vee, 4, 3) \rangle$$

Le problème CVP est alors défini comme suit :

Définition 31

Donnée : *L'encodage $\overline{\alpha}$ d'un circuit booléen et ses entrées $x_1, \ldots, x_n$*

Question : *Quelle est la valeur de la sortie du circuit sous $x_1, \ldots, x_n$?*

Une donnée CVP possible pour le circuit précédent est

$$\langle (1, I, 0, 0), (2, I, 0, 0), (3, I, 0, 0), (4, \vee, 1, 2), (5, \vee, 4, 3) \rangle$$
$$\langle 1, 0, 0 \rangle$$

La réponse est VRAI (le dernier nœud du circuit est bien VRAI lors de l'évaluation sur entrée $\langle 1, 0, 0 \rangle$).

Afin de se donner l'intuition que CVP est $\mathcal{P}$-complet, remarquons que, par définition, pour tout problème dans $\mathcal{P}$, il existe une machine de Turing M résolvant le problème en question et fonctionnant en temps polynomial. Le calcul de cette machine de Turing peut être représenté dans un tableau contenant une ligne par configuration. Ce tableau est de dimension $p(n) \times p(n)$, où p est un polynôme par rapport à la longueur de l'entrée. Comme déjà mentionné, la configuration d'une machine de Turing ne dépend que de la configuration qui la précède immédiatement. Plus précisément, la valeur d'une case $T(i, j)$ du tableau ne dépend que des valeurs des cases $T(i-1, j-1)$, $T(i-1, j)$ et $T(i+1, j)$. Ceci permet de construire un circuit pour le problème donné qui effectue étape par étape le calcul de la machine de Turing.

Théorème 6 GEN *est $\mathcal{P}$-complet selon $\leq_L$.*

Preuve [JL77, BM91]

Sachant que CVP est $\mathcal{P}$-complet, il suffit de montrer que GEN $\in \mathcal{P}$ et que CVP $\leq_L$ GEN afin de prouver que GEN est $\mathcal{P}$-complet.

1. GEN $\in \mathcal{P}$

   Intuitivement, on peut calculer une solution de GEN en effectuant d'abord toutes les multiplications possibles des éléments de l'ensemble de départ T. Si de nouveaux éléments sont générés, on effectue de nouveau toutes les multiplications possibles des éléments présents et ceci jusqu'à ce que n soit obtenu ou qu'on n'obtienne plus de nouveaux éléments auquel cas n ne peut pas être généré.

```
FC ← T
répéter
    Nouveaux = ∅
    pour toute possibilité de i,j ∈ FC
        si (x = i × j) ∉ FC
            Nouveaux = Nouveaux ∪ {x}
    FC = FC ∪ Nouveaux
tantque Nouveaux ≠ ∅ ET n ∉ Nouveaux
GEN = (n ∈ Nouveaux)
```

Lors du calcul on peut, en pire cas, obtenir n sous-ensembles différents des éléments de $\{1, 2, \ldots, n\}$ (dans ce cas, seul un nouvel élément est généré à chaque étape). Par ailleurs chaque suite de multiplications comporte moins de n^2 questions à tester. On peut donc effectivement calculer l'appartenance à GEN en temps polynomial.

2. CVP $\leq_L$ GEN

   Soit α un circuit. Sans perte de généralité, supposons que α ne comporte que des entrées et des portes ET, OU et NON.

   Remplaçons tout d'abord chaque fil du circuit par deux portes NON consécutives. Ainsi toute entrée d'une porte ET ou d'une porte OU n'apparaît qu'à cette porte comme entrée, c'est-à-dire si $\alpha_i = \text{ET}(j, k)$ ou $\alpha_i = \text{OU}(j, k)$ alors α_j et α_k n'apparaissent pas comme entrées à d'autres portes du circuit que α_i.

   Soit f le nombre de fils entrants dans des portes NON et soit p le nombre de portes du circuit ainsi modifié. On va maintenant créer un groupoïde de taille $f + p$. On crée un élément du groupoïde par fil entrant dans une porte NON.

$$E = \{e_{ji} \mid \alpha_i = \texttt{NOT}(j)\}$$

Ensuite, pour chaque porte α_i du circuit, on crée deux éléments du groupoïde i et $\bar{i}$. Notre but est que $i \in \langle T \rangle$ si et seulement si α_i évalue à vrai et que $\bar{i} \in \langle T \rangle$ si et seulement si α_i évalue à faux.

L'ensemble de départ T du groupoïde sera

$$\begin{aligned} T \;=\; & E \cup \{i \mid \alpha_i \text{ est une porte d'entrée de valeur OUI}\} \\ & \cup \{\bar{i} \mid \alpha_i \text{ est une porte d'entrée de valeur NON}\} \end{aligned}$$

Définissons le tableau de multiplication comme suit :

- si $\alpha_i = \texttt{AND}(j,k)$ alors $j \times k = k \times j = i$, $\bar{j} \times \bar{j} = \bar{i}$ et $\bar{k} \times \bar{k} = \bar{i}$.
- si $\alpha_i = \texttt{OR}(j,k)$ alors $\bar{j} \times \bar{k} = \bar{k} \times \bar{j} = \bar{i}$, $j \times j = i$ et $k \times k = i$
- si $\alpha_i = \texttt{NOT}(j)$ alors $e_{ji} \times \bar{j} = \bar{j} \times e_{ji} = i$ et $e_{ji} \times j = j \times e_{ji} = \bar{i}$
- si un produit n'a pas encore été défini précédemment, on le définit comme étant égal à g où g est un élément quelconque de T.

Si $\alpha_i = \text{ET}(j,k)$ ou $\alpha_i = \text{OU}(j,k)$ alors α_j et α_k n'apparaissent pas comme entrées à d'autres portes du circuit que α_i ; cette définition du tableau de multiplication est donc bien valide. Il est aussi facile de voir qu'en effet i appartient à $\langle T \rangle$ si et seulement si $\alpha_i = \text{OUI}$ et $\bar{i}$ appartient à $\langle T \rangle$ si et seulement si $\alpha_i = \text{NON}$. Par conséquent, le dernier élément du groupoïde sera généré si et seulement si la dernière porte du circuit évalue à VRAI, c'est-à-dire $(f+p) \in \langle T \rangle$ si et seulement si $\alpha_p = \text{VRAI}$. ■

Corollaire 2 GEN_C *est* $\mathcal{P}$*-complet selon* $\leq_L$.

Preuve

Il est aisé de remarquer que la loi de composition interne définie dans la preuve du théorème 6 est commutative. La même réduction que précédemment s'applique donc aussi pour GEN_C. ■

2.3 GEN_S et GEN_{SC} sont $\mathcal{P}$-complets

Nous allons maintenant prouver que même les restrictions de GEN à $T = \{1\}$ restent des problèmes $\mathcal{P}$-complets.

Théorème 7 GEN_S *est* $\mathcal{P}$*-complet selon* $\leq_L$.

Preuve

Comme on vient de prouver que GEN est $\mathcal{P}$-complet selon $\leq_L$, il suffit maintenant de prouver que $\text{GEN}_S \in \mathcal{P}$ et que $\text{GEN} \leq_L \text{GEN}_S$.

1. $\text{GEN}_S \in \mathcal{P}$

   La preuve est identique à celle pour le cas GEN $\in \mathcal{P}$. En effet, l'algorithme utilisé dans la preuve du théorème 6 résout, sans aucun changement, GEN_S.

2. $\text{GEN} \leq \text{GEN}_S$

   Soit une instance de GEN avec $T = \{t_1, t_2, \ldots, t_k\}$. On crée une instance de GEN_S de la manière suivante : GEN_S contiendra k nouveaux éléments, notés n_1, n_2, ..., n_k, suivi des éléments de GEN. La table de multiplication de GEN_S est définie par

   - $n_1 \times n_1 = n_2$, $n_2 \times n_2 = n_3$, ..., $n_{k-1} \times n_{k-1} = n_k$
   - $n_1 \times n_2 = n_2 \times n_1 = t_1$, $n_1 \times n_3 = n_3 \times n_1 = t_2$, ..., $n_1 \times n_{k-1} = n_{k-1} \times n_1 = t_k$.
   - Toutes les autres multiplications impliquant les nouveaux éléments n_1, n_2, ... , n_k sont égales à t_1.
   - La suite de la table de multiplication reste inchangée.

   Ainsi en partant de n_1, on peut d'abord générer l'ensemble $\{n_1, n_2, \ldots n_k\}$, et ensuite $\{t_1, t_2, \ldots, t_k\}$. Tous les éléments initialement présents dans l'instance de GEN sont donc générés. Puisque la partie de la table de multiplication qui concerne les "anciens" éléments reste inchangée, l'exemplaire de GEN_S est VRAI si et seulement si l'instance de GEN est VRAI. ■

Corollaire 3 GEN_{SC} *est P-complet selon* $\leq_L$.

Preuve

Remarquons tout d'abord que la partie de la table de multiplication, définie ci-dessus, impliquant les éléments $\{n_1, n_2, \ldots, n_k\}$ est commutative. On peut donc appliquer la même preuve que précédemment mais en réduisant GEN_C (qui est aussi $\mathcal{P}$-complet) à GEN_{SC}. ■

2.4 ExisteGEN est $\mathcal{NP}$-complet

On peut modifier légèrement le problème GEN et se demander quel ensemble de départ permet d'engendrer n. Cette modification mène à un nouveau problème $\mathcal{NP}$-complet.

Définition 32 *Soit le problème ExisteGEN :*

Donnée : *Une table de multiplication de dimension* $n \times n$ *et* m *couples* (x_1, x'_1), (x_2, x'_2), (x_3, x'_3), $\ldots$, (x_m, x'_m).

Question : *Existe-t-il* $(y_1, y_2, \ldots, y_m)$ *avec* $y_i \in \{x_i, x'_i\}$ *tel que* $n \in \langle\{y_1, y_2, \ldots, y_m\}\rangle$ *?*

Théorème 8 ExisteGEN *est* $\mathcal{NP}$*-complet.*

Preuve

1. ExisteGEN $\in \mathcal{NP}$

   Un choix d'éléments $(y_1, y_2, \ldots, y_m)$ et la suite des multiplications à effectuer afin d'engendrer n peut servir de certificat. Il suffit alors de vérifier qu'on a bien $y_i \in \{x_i, x'_i\}$ pour $1 \leq i \leq m$ et que la suite des multiplications engendre bien n.

2. 3SAT $\leq_P$ ExisteGEN

SAT, le problème de satisfaisabilité d'un ensemble de clauses, est parmi les problèmes $\mathcal{NP}$-complets les plus anciens et les plus connus : il a été prouvé $\mathcal{NP}$-complet par Cook en 1971 [Coo71].

3SAT est formellement défini comme suit :

Définition 33

Donnée : Une formule booléenne ϕ en forme normale conjonctive ayant trois litéraux par clause

Question : Est-ce que ϕ est satisfaisable ?

Étant donné une instance de 3SAT, on peut facilement construire un circuit qui résout 3SAT en reliant par OU toutes les variables d'une clause et en reliant par ET les portes représentant les clauses. Un exemple d'un circuit pour la formule booléenne $(x_1 \vee x_2 \vee \overline{x}_2) \wedge (x_3 \vee x_4 \vee x_5) \wedge (x_6 \vee x_7 \vee \overline{x}_8)$ est montré dans la figure 2.3.

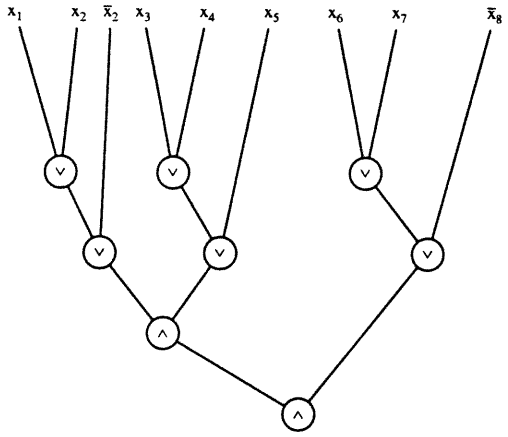


FIG. 2.3 – Circuit pour 3SAT

Nous modifions légèrement ce circuit en remplacant chaque entrée x_i, telle que son complément ne fait pas partie des entrées, par un circuit représentant $(x_i \vee \overline{x}_i)$. Cette modification double au plus le nombre d'entrées au circuit.

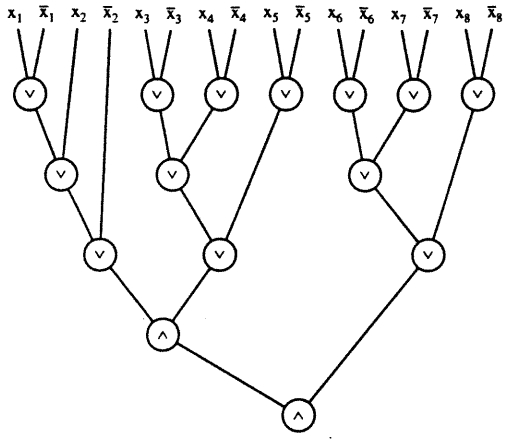


FIG. 2.4 – Circuit modifié pour 3SAT

On va maintenant transformer ce circuit en instance de ExisteGEN en numérotant toutes les portes et tous les fils du circuit : une porte sera notée dans la suite x_i, un fil reliant la sortie de la porte i à l'entrée de la porte j sera noté f_{ij}. Si notre circuit comporte p portes et f fils, on crée un groupoïde de $p + f$ éléments de manière à ce que le dernier élément corresponde à la dernière porte du circuit.

On définit la loi de composition interne du groupoïde de la manière suivante :

- $i \times j = k$ s'il existe x_k tel que $x_k = \text{ET}(x_i, x_j)$ et $i \times j = f_{11}$ sinon
- $i \times f_{ik} = k$ s'il existe x_k tel que $x_k = \text{OU}(x_i, x_j)$ ou $x_k = \text{OU}(x_j, x_i)$ et $i \times f_{ik} = f_{11}$ sinon
- $f_{ik} \times j = f_{11}$, $j \neq i$
- $f_{ik} \times f_{jl} = f_{11}$

Les choix pour l'ensemble de départ pour le circuit modifié avec $2n$ entrées seront les suivants : $(1, \overline{1})$, $(2, \overline{2})$, $(3, \overline{3})$, ..., $(n, \overline{n})$ et (f_{ij}, f_{ij}) pour tous les f_{ij} qui représentent un fil du circuit.

Si l'instance de 3SAT possède une solution, alors il existe une affectation de valeurs {VRAI, FAUX} aux variables qui rend l'évaluation vraie. Ceci revient à

choisir pour chaque entrée i entre x_i et $\overline{x}_i$. Puisque tous les fils font partie de l'ensemble de départ de ExisteGEN, la dernière valeur pourra être générée. Inversement, si la dernière valeur peut être obtenue dans ExisteGEN, alors le choix parmi $(1, \overline{1})$, $(2, \overline{2})$, etc. représente une affectation de valeurs aux variables de 3SAT qui rend toutes les clauses vraies. ■

Après avoir prouvé que les différentes instances du problème GEN, GEN_C, GEN_S et GEN_{SC} sont toutes $\mathcal{P}$-complètes, nous venons de voir qu'un petit changement à l'énoncé de GEN mène à un problème $\mathcal{NP}$-complet.

Au cours du prochain chapitre nous allons montrer des algorithmes permettant d'obtenir différents types de programmes de branchement pour GEN_S et GEN_{SC} et calculer, si possible, la taille de ces programmes de branchement. Remarquons immédiatement que ceci ne permet que d'obtenir des bornes supérieures sur la taille des programmes de branchement. C'est au cours du chapitre 4 que nous explorerons diverses méthodes permettant d'obtenir des bornes inférieures.

Chapitre 3

Réalisations concrètes de programmes de branchement

Nous allons maintenant montrer comment on peut construire divers types de programmes de branchement pour GEN_S et GEN_{SC}. Si possible, nous allons aussi calculer leur taille.

Un concept très utile afin de calculer la taille de nos programmes de branchement est celui de la fermeture courante, introduit par Kassardjian dans [Kas92].

Définition 34 *La fermeture courante d'un arc de programme de branchement est l'ensemble des éléments communs aux fermetures de* $\langle\{1\}\rangle$ *de tous les groupoïdes qui passent par cet arc.*

Toutes nos réalisations de programmes de branchement pour GEN_S et GEN_{SC} contiennent l'idée qu'il faut considérer seulement les éléments à l'intérieur de la fermeture courante. Ceci simplifie grandement la construction de programmes de branchement : on sait en particulier qu'à chaque fois que la réponse à la question d'un sommet est n, on doit créer l'arc le reliant au nœud OUI. On sait de plus que quand toutes les questions possibles pour la fermeture courante ont été posées sans qu'aucun nouvel élément n'ait été trouvé, on doit rejoindre le nœud NON. Ceci

revient d'ailleurs à la solution intuitive au problème GEN présentée au chapitre 1. On peut alors alléger les figures des programmes de branchement sans ambiguïté en omettant de dessiner les nœuds OUI et NON, en omettant d'attacher aux arcs des étiquettes quand ceux-ci se trouvent à l'intérieur de la fermeture courante et en omettant de dessiner les arcs menant aux nœuds OUI et NON. Les figures 3.1 et 3.2 représentent ainsi le même programme de branchement (pour GEN_S de dimension 3), le deuxième étant l'affichage simplifié.

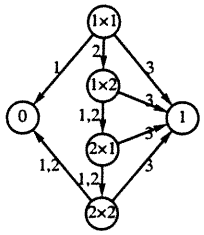


FIG. 3.1 – Programme pour GEN_S de dimension 3

FIG. 3.2 – Programme simplifié pour GEN_S de dimension 3

3.1 Programme de branchement pour GEN_S

L'idée la plus simple permettant d'obtenir un programme de branchement pour GEN_S est de créer un sous-programme séparé pour toutes les fermetures courantes possibles. Chaque sous-programme pose toutes les questions $i \times j$ possibles pour i, j dans la fermeture courante. Ce programme de branchement correspond à l'idée

intuitive initiale pour le problème GEN : on commence par poser la question 1×1, ensuite on pose les questions en fonction des valeurs déjà obtenues. On ne pose que des questions $i \times j$ où i et j se trouvent dans la fermeture courante.
L'algorithme suivant décrit la façon d'obtenir ce programme de branchement pour GEN_S.

```
// ----------------------------
// création des nœuds du programme
// ----------------------------
créer les nœuds 1 × 1, OUI et NON
pour toutes les fermetures courantes S possibles
   créer tous les nœuds i × j, i,j ∈ S
// ----------------------------
// création des arcs
// ----------------------------
   pour chaque nœud i × j
      pour x = 1 à n
         selon x
         cas x = n
            créer l'arc étiqueté par x reliant i × j à OUI
         cas x ∉ S
            créer l'arc étiqueté par x reliant i × j au premier
            nœud de la fermeture S ∪ {x}
         cas x ∈ S
            si i × j est le dernier nœud de la fermeture courante
               créer l'arc étiqueté par x reliant i × j à NON
            sinon
               créer l'arc étiqueté par x reliant i × j au
               prochain nœud de la fermeture courante
```

Théorème 9 *Il existe un programme de branchement pour* GEN_S *de dimension* n, $n \geq 2$, *ayant* $N(n) = 3 + 2^{n-4}(n+3)(n-2)$ *nœuds.*

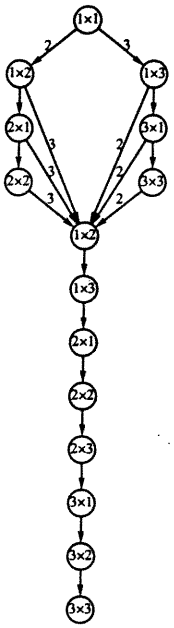


FIG. 3.3 – Programme pour GEN_S de dimension 4

Afin de prouver ce théorème, nous avons besoin du lemme suivant :

Lemme 1 *Pour $m \in \mathbb{N}$, $m \geq 2$, on a*

1. $\sum_{i=1}^{m} \binom{m}{i} i = m \cdot 2^{m-1}$
2. $\sum_{i=1}^{m} \binom{m}{i} i^2 = 2^{m-2} \cdot m(m+1)$

Preuve

On sait que $(y+x)^m = \sum_{i=0}^{m} \binom{m}{i} y^{m-i} x^i$ et par conséquent $(1+x)^m = \sum_{i=0}^{m} \binom{m}{i} x^i$.
Soit $f(x) = (1+x)^m$. Puisqu'il s'agit d'une somme finie, on obtient en dérivant f

$$\begin{aligned} f'(x) &= m(1+x)^{m-1} = \sum_{i=1}^{m} \binom{m}{i} i x^{i-1} \\ f'(1) &= m2^{m-1} = \sum_{i=1}^{m} \binom{m}{i} i \end{aligned}$$

On prouve ainsi la première partie du lemme 1. Afin de prouver la deuxième partie,

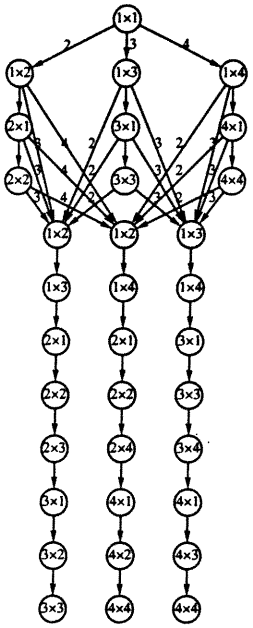


FIG. 3.4 – Programme pour GEN_S de dimension 5

dérivons de nouveau f :

$$\begin{aligned} f''(x) &= m(m-1)(1+x)^{m-2} = \sum_{i=2}^{m} \binom{m}{i} i(i-1)x^{i-2} \\ f''(1) &= m(m-1)2^{m-2} = \sum_{i=2}^{m} \binom{m}{i} i(i-1) \end{aligned}$$

Par conséquent

$$\begin{aligned} \sum_{i=2}^{m} \binom{m}{i} i^2 &= m(m-1)2^{m-2} + \sum_{i=2}^{m} \binom{m}{i} i \\ \sum_{i=1}^{m} \binom{m}{i} i^2 &= m(m-1)2^{m-2} + \sum_{i=1}^{m} \binom{m}{i} i \\ &= m(m-1)2^{m-2} + m2^{m-1} \\ &= 2^{m-2}\left(m(m+1)\right) \end{aligned}$$

■

Preuve du théorème 9

Remarquons tout d'abord qu'on peut facilement obtenir un programme de branchement pour GEN_S de dimension 2 possédant seulement les trois nœuds OUI, NON et 1×1. Un arc étiqueté 1 relie 1×1 et NON et un arc étiqueté 2 relie 1×1 et OUI. Par ailleurs $N(2) = 3 + 2^{2-4}(2+3)(2-2) = 3$.

Nous pouvons donc passer à la preuve du théorème pour $n \geq 3$. Par définition, toutes les fermetures courantes de GEN_S contiennent 1. Quand i et j sont dans la fermeture courante et $i \times j = n$, on peut rejoindre immédiatement le nœud OUI. Il y a par conséquent $n-2$ tailles de fermetures courantes qu'il faut considérer dans la construction d'un programme de branchement pour GEN_S de dimension n, $n \geq 3$. Pour les mêmes raisons on doit traiter $\binom{n-2}{i-1}$ fermetures courantes différentes de taille i. Et finalement, il existe i^2 questions possibles à l'intérieur d'une fermeture courante de taille i. Or, comme chaque programme de branchement commence par la question 1×1, il n'est plus nécessaire de reposer cette question à l'intérieur des sous-programmes. Le nombre de questions à considérer à l'intérieur d'une fermeture courante de taille i, $i \geq 2$, est donc $i^2 - 1$. Par ailleurs, tout programme de branchement pour GEN_S doit posséder les trois nœuds `oui`, `non` et 1×1.

Le nombre de nœuds $N(n)$ de ce programme de branchement pour GEN_S est alors :

$$\begin{aligned} N(n) &= 3 + \sum_{i=2}^{n-1} \binom{n-2}{i-1} (i^2 - 1) \\ &= 3 + \sum_{i=1}^{n-2} \binom{n-2}{i} (i^2 + 2i) \\ &= 3 + \sum_{i=0}^{n-2} \binom{n-2}{i} i^2 + 2 \sum_{i=0}^{n-2} \binom{n-2}{i} i \end{aligned}$$

On se servant du lemme 1 on obtient le nombre de nœuds du programme de

branchement pour GEN_S :

$$\begin{aligned} N(n) &= 3 + \sum_{i=0}^{n-2} \binom{n-2}{i} i^2 + 2 \sum_{i=0}^{n-2} \binom{n-2}{i} i \\ &= 3 + 2^{n-4}(n-2)(n-1) + 2^{n-2}(n-2) \\ &= 3 + 2^{n-4}\left(n^2 - 3n + 2 + 4n - 8\right) \\ &= 3 + 2^{n-4}\,(n+3)\,(n-2) \end{aligned}$$

■

3.2 Programme de branchement pour GEN_{SC}

Nous allons de nouveau créer un sous-programme par fermeture courante possible du groupoïde. La différence avec le cas plus général de GEN_{SC} est que la loi de multiplication de *GEN* étant commutative, on n'a plus besoin de poser la question $j \times i$ quand on a déjà précédemment posé la question $i \times j$ (pour $i \neq j$).
On obtient alors l'algorithme suivant :

```
// ----------------------------
// création des nœuds du programme
// ----------------------------
créer les nœuds 1 × 1, OUI et NON
pour toutes les fermetures courantes S possibles
   créer tous les nœuds i × j, i,j ∈ S,i ≤ j
// ----------------------------
// création des arcs
// ----------------------------
   pour chaque nœud i × j
      pour x = 1 à n
         selon x
```

cas $x = n$

créer l'arc étiqueté par x reliant $i \times j$ à OUI

cas $x \notin S$

créer l'arc étiqueté par x reliant $i \times j$ au premier nœud de la fermeture $S \cup \{x\}$

cas $x \in S$

si $i \times j$ est le dernier nœud de la fermeture courante

créer l'arc étiqueté par x reliant $i \times j$ à NON

sinon

créer l'arc étiqueté par x reliant $i \times j$ au prochain nœud de la fermeture courante

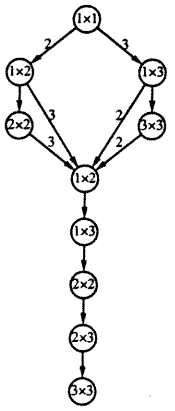


FIG. 3.5 – Programme pour GEN_{SC} de dimension 4

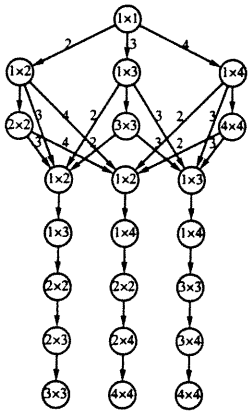


FIG. 3.6 – Programme pour GEN_{SC} de dimension 5

Théorème 10 *Il existe un programme de branchement pour* GEN_{SC} *de dimension* n, $n \geq 2$, *ayant* $3 + 2^{n-5}(n+5)(n-2)$ *nœuds.*

Preuve

Pour résoudre le problème GEN_{SC} de dimension 2, on a besoin du même programme de branchement que pour GEN_S de dimension 2 (voir la preuve du théorème 9). On a par ailleurs aussi que $N(2) = 3 + 2^{2-5}(2+5)(2-2) = 3$. La propriété à prouver est donc de nouveau vrai pour $n = 2$. Passons alors à la preuve pour $n \geq 3$. Le nombre de questions que l'on doit maintenant poser à l'intérieur d'une fermeture courante de taille i est $-1 + \sum_{j=1}^{i} j = \frac{i(i+1)}{2} - 1$ (la soustraction de 1 provient de nouveau du fait qu'on ne repose pas la question 1×1).

Le nombre de nœuds $N(n)$ d'un tel programme de branchement est alors

$$\begin{aligned} N(n) &= 3 + \sum_{i=2}^{n-1} \binom{n-2}{i-1} \left(\frac{i(i+1)}{2} - 1 \right) \\ &= 3 + \sum_{i=1}^{n-2} \binom{n-2}{i} \left(\frac{i^2+3i}{2} \right) \\ &= 3 + \frac{1}{2} \left(\sum_{i=0}^{n-2} \binom{n-2}{i} i^2 + 3 \sum_{i=0}^{n-2} \binom{n-2}{i} i \right) \end{aligned}$$

On peut de nouveau appliquer le lemme 1 et on obtient ainsi :

$$\begin{aligned} N(n) &= 3 + \frac{1}{2} \left(2^{n-4}(n-2)(n-1) + 3 \cdot 2^{n-3}(n-2) \right) \\ &= 3 + 2^{n-5}(n^2 - 3n + 2 + 6n - 12) \\ &= 3 + 2^{n-5}(n+5)(n-2) \end{aligned}$$

■

3.3 Programme de branchement à lecture unique pour $\mathbf{GEN}_S$

Nous allons maintenant montrer un algorithme générant un programme de branchement à lecture unique pour GEN_S en nous servant d'un raisonnement similaire à celui utilisé pour le programme de branchement général. En revisitant l'algorithme de la section 3.1 et en essayant de l'adapter pour obtenir un programme de branchement à lecture unique, il semble n'être plus suffisant de seulement considérer les fermetures courantes. Il semble maintenant nécessaire de tenir compte de la manière dont les fermetures courantes ont été obtenues afin de poser chaque question une seule fois le long d'un chemin de calcul. Autrement dit, la fermeture courante "$\{1, 2, 3\}$" sera traitée différemment de la fermeture courante "$\{1, 3, 2\}$" par notre algorithme. Par conséquent, nous créerons un sous-programme pour chaque permutation de toutes les fermetures courantes possibles.

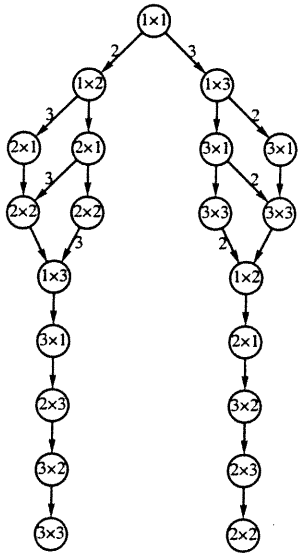


FIG. 3.7 – Programme à lecture unique pour GEN_S de dimension 4

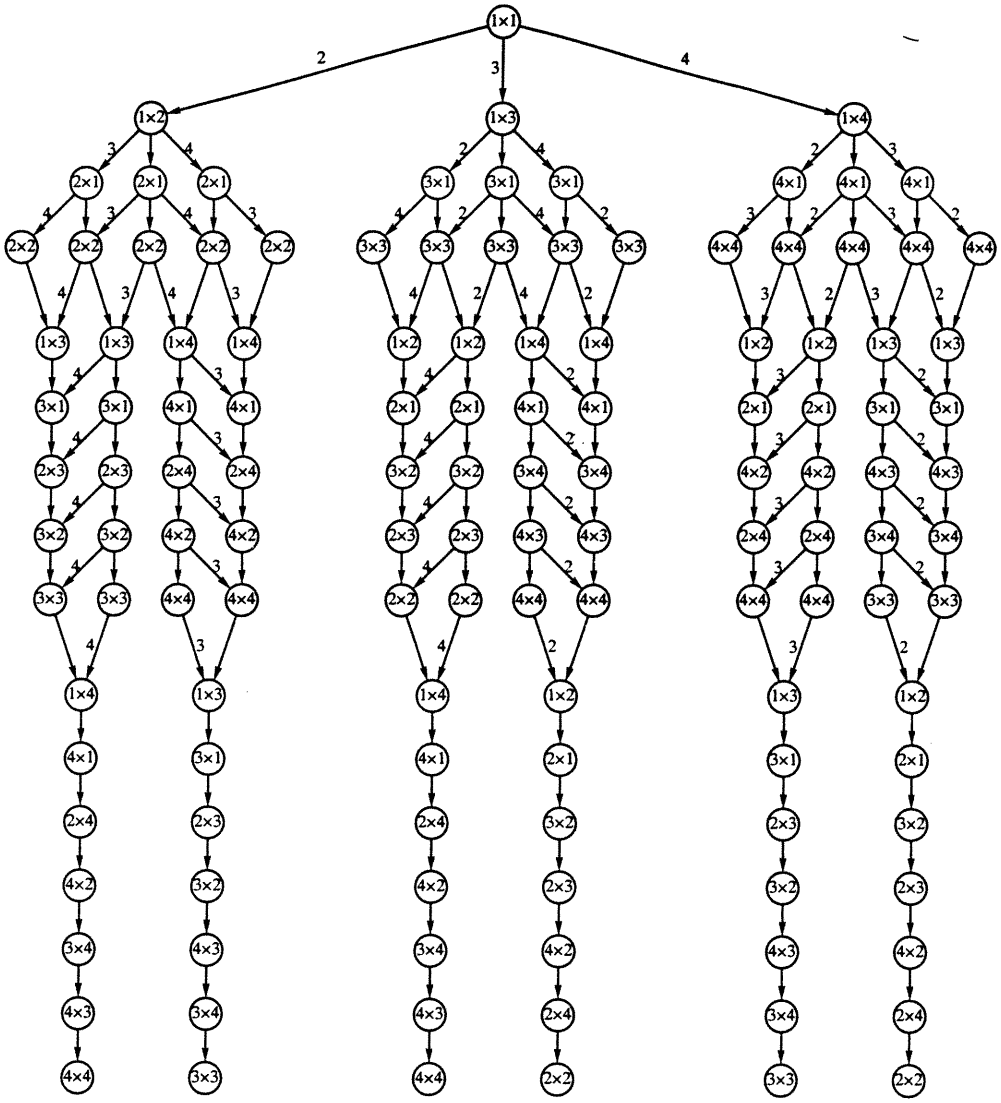


FIG. 3.8 – Programme à lecture unique pour GEN_S de dimension 5

L'algorithme permettant d'obtenir un programme de branchement à lecture unique pour GEN_S peut alors être décrit comme suit :

```
// ------------------------------
// création des nœuds du programme
// ------------------------------
créer les nœuds 1 × 1, OUI et NON
pour toutes les fermetures courantes S possibles
   pour toutes les permutations des éléments de S
      créer tous les nœuds i × j, i,j ∈ S qui n'ont pas encore été
      posés précédemment
// ------------------------------
// création des arcs
// ------------------------------
      pour chaque nœud i × j
         pour x = 1 à n
            selon x
            cas x = n
               créer l'arc étiqueté par x reliant i × j à OUI
            cas x ∉ S
               créer l'arc étiqueté par x reliant i×j au premier
               nœud de la fermeture S∪{x}, permutation actuelle
               augementée de x, qui n'a pas encore été posé
            cas x ∈ S
               si i × j est le dernier nœud de cette permutation
               de la fermeture courante
                  créer l'arc étiqueté par x reliant i × j à NON
               sinon
                  créer l'arc étiqueté par x reliant i × j au
                  prochain nœud de cette permutation de la
                  fermeture courante
```

Théorème 11 *Il existe un programme de branchement à lecture unique pour* GEN_S *de dimension* n *ayant* $3 + \sum_{i=1}^{n-2} \left((1 + i + i^2) \prod_{j=1}^{i} (n - j - 1) \right)$ *nœuds.*

n	nombre de nœuds
3	6
4	23
5	132
6	915
7	7178
8	62631
9	602808
10	6356870

FIG. 3.9 – Nombre de nœuds d'un programme de branchement à lecture unique pour GEN_S

Preuve

Pour GEN_S de taille n, il y a $n-2$ tailles de fermetures courantes à considérer (toutes les fermetures courantes contiennent 1 et aucune ne contient n). On peut alors décomposer le programme de branchement en $n-2$ niveaux; au niveau i, toutes les fermetures courantes sont au moins de taille i. Dans la figure 3.8, tous les nœuds de profondeurs 1, 2 et 3 se trouvent au niveau 2, tous les nœuds de profondeurs 4 à 8 se trouvent au niveau 3 et tous les nœuds de profondeurs 9 à 15 se trouvent au niveau 4. Visuellement, ces niveaux ont été espacés dans la figure 3.8 afin de faciliter la lecture. Remarquons que toutes les fermetures courantes d'un niveau ne sont pas de la même taille : au niveau 2, on peut par exemple avoir des fermetures courantes de tailles 2, 3 et 4, au niveau 3 des fermetures courantes de tailles 3, 4, ..., 8 et 9 (si l'ordre du groupoïde est assez élevé bien sûr) et ainsi de suite. Il est par ailleurs évident que la contribution du niveau i à la profondeur totale du programme de branchement est $2\cdot(i-1)+1$; cette profondeur équivaut au nombre de nouvelles questions que l'on peut poser à l'intérieur d'une fermeture courante de taille i.

Au niveau 2, on trouvera $(n-2)$ blocs de 3 questions avec une fermeture courante de taille 2, $(n-2)(n-3)$ blocs de 2 questions avec une fermeture courante de taille

3 et $(n-2)(n-3)(n-4)$ blocs d'une seule question avec une fermeture courante de taille 4. Par conséquent, le nombre de questions appartenant au niveau 2 est $(n-2)\cdot 3 + (n-2)(n-3)\cdot 2 + (n-2)(n-3)(n-4)$.

Au niveau 3, on trouvera $(n-2)(n-3)$ blocs de 5 questions avec une fermeture courante de taille 3, $(n-2)(n-3)(n-4)$ blocs de 5 questions avec une fermture courante de taille 4, $(n-2)(n-3)(n-4)(n-5)$ blocs de 5 questions avec une fermeture courante de taille 5, $\prod_{i=2}^{6}(n-i)$ blocs de 4 questions avec une fermeture courante de taille 6, $\prod_{i=2}^{7}(n-i)$ blocs de 3 questions avec une fermeture courante de taille 7, $\prod_{i=2}^{8}(n-i)$ blocs de 2 questions avec une fermeture courante de taille 8 et finalement $\prod_{i=2}^{9}(n-i)$ blocs d'une seule question avec une fermeture courante de taille 9 (toujours en supposant que l'ordre du groupoïde est assez élevé). Le nombre de nœuds $N(n)$ d'un tel programme est alors :

$$
\begin{aligned}
N(n) = \quad & 3 + (n-2)\cdot 3 + (n-2)(n-3)\cdot 2 + (n-2)(n-3)(n-4) \\
+ \quad & (n-2)(n-3)\cdot 5 + (n-2)(n-3)(n-4)\cdot 5 \\
& +(n-2)(n-3)(n-4)(n-5)\cdot 5 \\
& +(n-2)(n-3)(n-4)(n-5)(n-6)\cdot 4 \\
& +(n-2)(n-3)(n-4)(n-5)(n-6)(n-7)\cdot 3 \\
& \vdots \\
+ \quad & (n-2)(n-3)(n-4)\cdot 7 \\
& +(n-2)(n-3)(n-4)(n-5)\cdot 7 \\
& \vdots \\
& +(n-2)(n-3)\cdots(n-8)\cdot 7 \\
& +(n-2)(n-3)\cdots(n-9)\cdot 6 \\
& \vdots
\end{aligned}
$$

Un regroupement différent des termes (mettant en facteur $\prod_{j=1}^{i}(n-j-1)$ pour $1 \leq i \leq n-2$) permet de simplifier l'expression.

$$\begin{aligned} N(n) &= 3 \\ &\quad +(n-2)(3) \\ &\quad +(n-2)(n-3)(2+5) \\ &\quad +(n-2)(n-3)(1+5+7) \\ &\quad +(n-2)(n-3)(n-4)(5+7+9) \\ &\quad \vdots \end{aligned}$$

Considérons la suite f_n : 3, 2+5=7, 1+5+7=13, 5+7+9=21, 31, 43, 57, ... On remarque que $f_2 = 7 = 3 + (5-1) = f_1 + 4$, $f_3 = 13 = 7 + (7-1) = f_2 + 6$, $f_4 = f_3 + 8$ et ainsi de suite. Par conséquent f_n peut être décrite par la relation de récurrence $f_n = f_{n-1} + 2n$ ayant $f_1 = 3$ comme condition initiale.

$$\begin{aligned} f_n &= f_{n-1} + 2n \\ &= f_{n-2} + 2(n-1) + 2n \\ &= f_{n-3} + 2(n-2) + 2(n-1) + 2n \\ &\quad \vdots \\ &= 3 + 2 \cdot 2 + 2 \cdot 3 + 2 \cdot 4 + ... + 2(n-2) + 2(n-1) + 2n \\ &= 1 + 2\sum_{i=1}^{n} i \\ &= 1 + n(n+1) \\ &= n^2 + n + 1 \end{aligned}$$

On obtient alors $N(n) = 3 + \sum_{i=1}^{n-2}\left((1+i+i^2)\prod_{j=1}^{i}(n-j-1)\right)$ ■

3.4 Programme de branchement à lecture unique pour $\mathbf{GEN}_{SC}$

De nouveau, le programme de branchement pour GEN_{SC} peut être obtenu à partir de celui pour GEN_S en ne posant pas une question $i \times j$ quand la question $j \times i$ a déjà été posée.

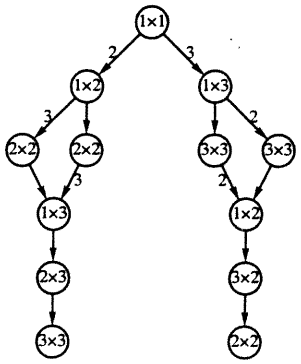


FIG. 3.10 – Programme à lecture unique pour GEN_{SC} de dimension 4

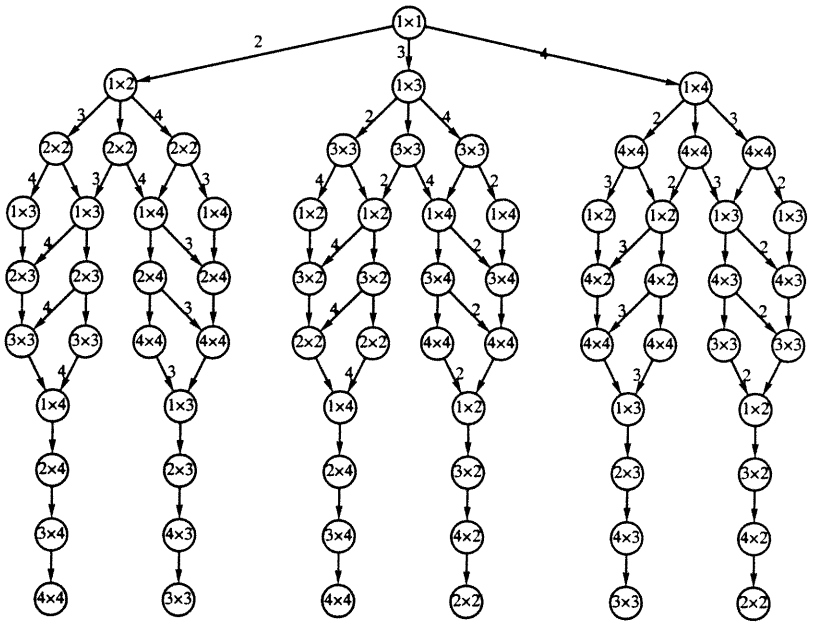


FIG. 3.11 – Programme à lecture unique pour GEN_{SC} de dimension 5

n	nombre de nœuds
3	5
4	15
5	75
6	491
7	3753
8	32295
9	308255
10	3233240

FIG. 3.12 – Nombre de nœuds d'un programme de branchement à lecture unique pour GEN_{SC}

Théorème 12 *Il existe un programme de branchement à lecture unique pour* GEN_{SC} *de dimension* n *ayant* $3 + \sum_{i=1}^{n-2} \left(\frac{1}{2}(2 + i + i^2) \prod_{j=1}^{i} (n - j - 1) \right)$ *nœuds.*

Preuve

Le même raisonnement que celui appliqué dans la preuve du théorème 11 nous montre que le nombre de nœuds $N(n)$ d'un tel programme est

$$
\begin{aligned}
N(n) = \quad & 3 \\
+ \quad & (n-2)\cdot 2 + (n-2)(n-3) \\
+ \quad & (n-2)(n-3)\cdot 3 + (n-2)(n-3)(n-4)\cdot 3 \\
& +(n-2)(n-3)(n-4)(n-5)\cdot 2 \\
& +(n-2)(n-3)(n-4)(n-5)(n-6) \\
+ \quad & (n-2)(n-3)(n-4)\cdot 4 \\
& +(n-2)(n-3)(n-4)(n-5)\cdot 4 \\
& +(n-2)(n-3)(n-4)(n-5)(n-6)\cdot 4 \\
& +(n-2)(n-3)\cdots(n-7)\cdot 3 \\
& \vdots
\end{aligned}
$$

En regroupant les termes, nous obtenons alors la formule suivante :

$$\begin{aligned} N(n) &= 3 \\ &\quad +(n-2)(2) \\ &\quad +(n-2)(n-3)(1+3) \\ &\quad +(n-2)(n-3)(3+4) \\ &\quad +(n-2)(n-3)(n-4)(2+4+5) \\ &\quad \vdots \end{aligned}$$

La suite 2, 4, 7, 11 est définie par la relation de récurrence $f_n = f_{n-1} + n$ avec la condition initiale $f_1 = 2$. On obtient alors

$$\begin{aligned} f_n &= f_{n-1} + n \\ &= f_{n-2} + (n-1) + n \\ &\quad \vdots \\ &= 2 + 2 + 3 + 4 + \ldots + (n-1) + n \\ &= 1 + \sum_{i=1}^{n} i \\ &= 1 + \frac{(n+1)n}{2} \end{aligned}$$

Le nombre de nœuds de ce programme de branchement à lecture unique pour GEN_{SC} de dimension n est alors

$$N(n) = 3 + \sum_{i=1}^{n-2} \left(\frac{1}{2}(2 + i + i^2) \prod_{j=1}^{i} (n-j-1) \right)$$

■

3.5 Arbre de décision pour GEN_S

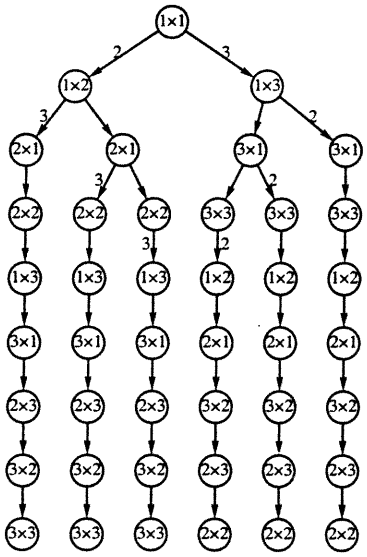


FIG. 3.13 – Arbre de décision pour GEN_S de dimension 4

Un algorithme permettant de construire un arbre de décision pour GEN_S peut facilement être obtenu à partir de l'algorithme générant un programme de branchement à lecture unique. De nouveau, on a besoin d'un sous-programme par permutation de toute fermeture courante possible. Mais cette fois-ci, deux chemins ne peuvent jamais se rejoindre et l'algorithme est alors le suivant :

```
créer les nœuds 1 × 1, OUI et NON
pour toutes les fermetures courantes S possibles
   pour toutes les permutations des éléments de S
      créer tous les nœuds i × j, i, j ∈ S qui n'ont pas encore été
      posés précédemment
      pour chaque nœud i × j
         pour x = 1 à n
            selon x
            cas x = n
               créer l'arc étiqueté par x reliant i × j à OUI
```

cas $x \notin S$
    créer tous les nœuds $i \times j$, $i, j \in S \cup \{x\}$ qui
    n'ont pas encore été posés précédemment
    créer l'arc étiqueté par x reliant $i \times j$ au premier
    nœud de l'ensemble créé dans l'étape précédente
cas $x \in S$
    si $i \times j$ est le dernier nœud de cette permutation
    de la fermeture courante
        créer l'arc étiqueté par x reliant $i \times j$ à NON
    sinon
        créer l'arc étiqueté par x reliant $i \times j$ au
        prochain nœud de cette permutation de la
        fermeture courante

Lemme 2 *Il existe un arbre de décision pour* GEN_S *de dimension* n *dont la hauteur est* $(n-1)^2$.

Preuve

L'algorithme décrit ci-haut pose chaque question $i \times j$, $1 \leq i, j \leq (n-1)$ une seule fois lors d'un chemin de calcul. Par conséquent, la hauteur de l'arbre est $(n-1) \cdot (n-1) = (n-1)^2$. ■

Lemme 3 *Tout arbre de décision de taille minimale pour* GEN_S *de dimension* n *est au plus de hauteur* $(n-1)^2$.

Preuve

Un arbre de décision de taille minimale ne peut pas poser une question plus d'une fois le long d'un chemin de calcul. Or, le problème GEN de dimension n possède n^2 variables. Par conséquent, la hauteur de tout arbre de décision minimal doit être inférieure ou égale à n^2.

Par ailleurs, un arbre de décision de taille minimale ne pose aucune question qui ne mène pas à un branchement. Par conséquent, les questions $i \times n$ et $n \times i$ pour

$1 \leq i \leq n$ ne feront pas partie d'un arbre minimal. Autrement dit, seules les questions $i \times j$ avec $1 \leq i, j \leq n-1$ seront posées. En remarquant qu'il existe $(n-1)^2$ telles questions, on peut conclure que la hauteur de tout arbre de décision de taille minimale pour GEN_S est inférieure ou égale à $(n-1)^2$. ■

Ce dernier lemme servira au chapitre 4 afin d'établir que la hauteur de tout arbre de décision minimal pour GEN_S est exactement $(n-1)^2$.

3.6 OBDD pour GEN_S

Un OBDD pour GEN_S ne peut plus être obtenu en se servant des mêmes raisonnements que précédemment puisqu'il est impossible de seulement poser des questions se trouvant à l'intérieur de la fermeture courante. Pour illustrer ceci, considérons un OBDD pour GEN_S de dimension 4. La section de gauche de la

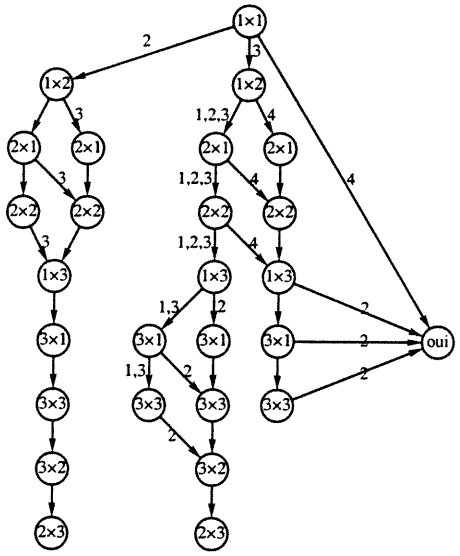


FIG. 3.14 – OBDD pour GEN_S de dimension 4

figure 3.14 se comporte comme un programme de branchement à lecture unique pour GEN_S. Dans la section de droite, nous avons été "forcés" de poser des ques-

tions comportant des éléments qui ne font pas partie de la fermeture courante. Par conséquent, il fallait garder "en mémoire" que, si on pouvait générer 2, alors on était capable de générer 4 et c'est pour cette raison que les questions 1×3, 3×1 et 3×3 mènent à OUI sur réponse 2. Il devient donc rapidement clair qu'un algorithme décrivant exactement comment on peut obtenir un OBDD pour GEN_S est loin d'être facile à concevoir et à exposer.

Dans ce chapitre, nous avons montré des bornes supérieures pour la taille de programmes de branchement et de programmes de branchement à lecture unique pour GEN. Ces bornes ont été obtenues en démontrant d'abord un algorithme permettant de générer un programme de branchement pour GEN et en calculant ensuite le nombre de nœuds générés par l'application de l'algorithme. Les algorithmes que nous avons exposés ne sont peut-être pas les meilleurs algorithmes possibles, mais ils ont l'avantage de suivre exactement le schéma de la solution intuitive de GEN ce qui nous donne des résultats satisfaisants et assez simples à décrire. Après avoir obtenu des intuitions sur les programmes de branchement pour GEN, tentons maintenant de prouver des bornes inférieures sur leur taille.

Chapitre 4

Bornes inférieures

Règle générale, des bornes inférieures pour la complexité d'un problème sont beaucoup plus difficiles à obtenir que des bornes supérieures. En effet, pour une borne supérieure, il suffit de montrer une façon (éventuellement parmi d'autres) de résoudre le problème et de calculer la taille de la mesure de complexité (comme par exemple le temps ou l'espace) nécessaire à cette solution particulière; afin d'obtenir une borne inférieure, il faut prouver qu'aucune solution utilisant moins que cette quantité de la mesure de complexité ne peut exister pour le problème en question.

Pour ce qui est de la taille de programmes de branchement, il existe fort peu de techniques qui permettent de trouver des bornes inférieures non triviales. En effet, aucune borne inférieure super-polynomiale n'est connue à ce jour pour un problème appartenant à la classe $\mathcal{NP}$. Par contre, pour les programmes de branchement restreints, comme par exemple les programmes de branchement à lecture unique et les OBDD, plusieurs techniques ont été développées au cours des quinze dernières années et des bornes inférieures exponentielles sont maintenant connues pour plusieurs problèmes (les premières bornes ont été obtenues par Žàk [Žàk84] et Wegener [Weg88]).

Nous allons, dans ce chapitre, énoncer et prouver formellement plusieurs bornes inférieures pour les tailles de programmes de branchement résolvant GEN et GEN_S. Nous montrerons en particulier les premières bornes inférieures exponentielles pour la taille d'un programme de branchement à lecture unique et la taille d'un OBDD capables de résoudre GEN.

4.1 Programme de branchement

Dans le souci de dresser un portrait complet des connaissances au sujet de la taille des programmes de branchement pour GEN, notons tout d'abord l'observation suivante de Kassardjian [Kas92] :

Observation *Tout programme de branchement résolvant* GEN *de dimension* n *doit être de taille au moins* $(n-1)^2$.

Supposons au contraire qu'on ait un programme de branchement résolvant GEN de dimension n ayant moins de $(n-1)^2$ nœuds. Alors, il existe forcément un couple $(i,j) \in \{1,2,\ldots,n-1\}^2$ tel que la question $i \times j$ n'est jamais posée dans le programme de branchement.

Considérons alors les groupoïdes définis par les tables de multiplications suivantes :

G1 :

	1	2			j		
1	**i**	1	1	1	1	1	1
	1	1	1	1	1	1	1
i	**j**	1	1	1	**1**	1	1
	1	1	1	1	1	1	1
	1	1	1	1	1	1	1

G2 :

	1	2			j		
1	**i**	1	1	1	1	1	1
	1	1	1	1	1	1	1
i	**j**	1	1	1	**n**	1	1
	1	1	1	1	1	1	1
	1	1	1	1	1	1	1

Soit $T = \{1\}$ dans les deux cas. Il est aisé de voir que dans G_1, $n \in \langle\{1\}\rangle$, mais que dans G_2, $n \notin \langle\{1\}\rangle$. Puisque les groupoïdes G_1 et G_2 sont identiques, sauf en

ce qui à trait à la définition du produit $i \times j$, le programme de branchement ne pourra pas les différencier et répondra soit OUI pour G_1 et G_2, soit NON pour G_1 et G_2. Ceci entre définitivement en contradiction avec l'hypothèse que le programme de branchement résolve GEN de dimension n.

En appliquant la méthode classique de Nechiporuk [Nec66] concernant les circuits booléens, Kassardjian arrivait aussi à obtenir la meilleure borne connue à ce jour sur la taille de programmes de banchement non restreints pour GEN_S

Théorème 13 *La taille de tout programme de branchement pour* GEN_S *est dans* $\Omega\left(\frac{n^3}{\log n}\right)$.

Pour les programmes de branchement non restreints résolvant GEN_S, l'écart entre les bornes inférieure et supérieure connues est donc particulièrement large : la meilleure borne inférieure connue est dans $\Omega\left(\frac{n^3}{\log n}\right)$, une borne supérieure est dans $\mathcal{O}\left(n^2 2^{n-4}\right)$. Cette situation mérite sans doute une recherche approfondie dans le futur car des résultats importants pourraient en découler.

4.2 Hauteur et taille de l'arbre de décision

Hauteur de l'arbre de décision

Un raisonnement similaire au cas précédent permet aussi d'obtenir la preuve que la hauteur de tout arbre de décision pour GEN_S est exactement égale à $(n-1)^2$.

Lemme 4 *La hauteur de tout arbre de décision résolvant* GEN_S *de dimension* n *doit être au moins* $(n-1)^2$.

Preuve

Considérons un arbre de décision minimal pour GEN_S, c'est-à-dire, un arbre de

décision pour GEN qui ne contient jamais une même question plusieurs fois le long d'un chemin de calcul.

Soit G un groupoïde avec $\{1, 2, \ldots, n-1\} \subseteq \langle\{1\}\rangle$ dont le parcours de l'arbre de décision contient moins de $(n-1)^2$ nœuds. Ainsi, il existe au moins un couple $(i, j) \in \{1, 2, \ldots, n\}^2$ tel que la question $i \times j$ n'est jamais posée dans l'arbre de décision sur entrée G. Or, comme il s'agit d'un arbre de décision minimal, après la dernière question, il y a un branchement vers OUI et NON. Il est maintenant essentiel de remarquer qu'il existe deux familles de groupoïdes qui suivent le même chemin que G dans l'arbre sauf en ce qui a trait à la dernière question : une famille dont les groupoïdes passent vers le nœud OUI et une autre dont les groupoïdes passent vers le nœud NON. Dans la famille de groupoïdes passant vers le nœud NON, il doit exister au moins un groupoïde ayant $i \times j = n$. Puisque $\{1, 2, \ldots, n-1\} \subseteq \langle\{1\}\rangle$ par hypothèse, on a donc bien pour ce groupoïde $\langle\{1\}\rangle = \{1, 2, \ldots, n\}$. On a alors aussi en particulier $n \in \langle\{1\}\rangle$. La réponse NON de l'arbre de décision est donc erronée pour ce groupoïde ce qui contredit l'hypothèse que l'arbre de décision résout GEN_S. Comme ce raisonnement est indépendant de i et j, la hauteur d'un arbre de décision doit être au moins $(n-1)^2$. ■

Théorème 14 *La hauteur d'un arbre de décision minimal pour* GEN_S *est exactement* $(n-1)^2$.

Preuve

Le lemme 4 indique que la hauteur d'un arbre de décision pour GEN_S doit être au moins $(n-1)^2$; d'un autre côté, le lemme 3 indique que la hauteur d'un arbre de décision pour GEN_S est inférieure ou égale à $(n-1)^2$. Par conséquent, la hauteur de tout arbre de décision pour GEN_S doit être exactement $(n-1)^2$. ■

Ce résultat impressionne essentiellement par le fait que la borne inférieure et la borne supérieure se rejoignent et permet ainsi de dire que tout arbre de décision

"raisonnable" pour GEN_S doit absolument être de hauteur $(n-1)^2$.

Taille de l'arbre de décision

Le théorème suivant est un cas particulier d'un théorème prouvé pour la première fois par Kassardjian [Kas92].

Théorème 15 *La taille d'un arbre de décision pour* GEN_S *de dimension* n *est supérieure ou égale à* 2^{n-2}.

Preuve

Dans un arbre de décision, il faut traiter chaque fermeture courante par un sous-programme séparé (sinon, on peut appliquer le même raisonnement que dans la preuve du lemme 4 et aboutir à une contradiction). On doit donc avoir au moins autant de sommets dans l'arbre de décision qu'il y a de fermetures courantes différentes. Or, le nombre de fermetures courantes possibles est égal au nombre de sous-ensembles de $\{1, 2, ..., n-1\}$ contenant 1, c'est-à-dire 2^{n-2}. ■

Passons maintenant aux nouvelles bornes inférieures exponentielles pour la taille de tout programme de branchement à lecture unique et de tout OBDD résolvant GEN.

4.3 Programme de branchement à lecture unique

Rappelons que depuis quinze ans, plusieurs bornes inférieures exponentielles sur la taille de programmes de branchement à lecture unique ont pu être établies. Deux excellents ouvrages énumérant les problèmes pour lesquels une telle borne est connue sont [Gál97] et [Weg00].

Une première tentative d'obtenir une borne inférieure exponentielle pour GEN a

été faite en 1992 par Kassardjian [Kas92], sans succès. Aujourd'hui, cette borne peut être obtenue grâce à l'application du concept de projection à lecture unique (*read-once projection*) introduit dans [SV85] et repris dans [BW95] et la connaissance de bornes inférieures exponentielles pour divers problèmes.

Définition 35 *La famille de fonctions (f_n) est une projection polynomiale de la famille (g_n), notée $f \leq_{proj} g$, si*

$$f_n(x_1, \ldots, x_n) = g_{p(n)}(y_1, \ldots, y_{p(n)})$$

où p est un polynôme et $y_i \in \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}, 0, 1\}$. La multiplicité de x_i est le nombre de j tels que $y_j \in \{x_i, \overline{x_i}\}$.
Une projection est à lecture unique (read-once projection), $f \leq_{rop} g$, si la multiplicité de chaque variable est bornée par 1.

Il existe des fonctions f et g telles que $f \leq_{proj} g$ et que la taille d'un programme de branchement pour g est linéaire alors qu'une borne inférieure exponentielle existe pour la taille de tout programme de branchement pour f. Par contre, en imposant une limite sévère sur la multiplicité de la projection, on obtient le résultat suivant :

Lemme 5 *Si $f \leq_{rop} g$ et la taille d'un OBDD (ou d'un programme de branchement à lecture unique) pour g est bornée inférieurement par une fonction q et la projection à lecture unique utilise un polynôme p, alors la taille d'un OBDD (respectivement programme de branchement à lecture unique) pour f est bornée inférieurement par $q \circ p$.*

Preuve

La preuve de ce lemme est identique pour les OBDD et les programmes de branchement à lecture unique. Afin d'alléger l'écriture, nous mentionnerons par la suite seulement les OBDD.

Si $f_n(x_1, \ldots, x_n) = g_{p(n)}(y_1, \ldots, y_{p(n)})$ et g possède un OBDD de taille q, voici comment on peut créer un OBDD de taille $q \circ p$ pour f. On part du programme de branchement pour $g_{p(n)}$ (qui est de taille $q \circ p$) et on remplace pour tout $1 \leq i \leq p(n)$ la i-ème variable par $y_i \in \{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}, 0, 1\}$. Si un nœud est étiqueté par $\overline{x_j}$, on le remplace par x_j et on inverse les arcs sortants.

Ceci donne de nouveau un OBDD. ■

Comme déjà mentionné, des bornes inférieures exponentielles sur la taille de programmes de branchement à lecture unique sont connues pour plusieurs fonctions. Afin de prouver notre borne pour GEN, nous allons nous servir de la fonction $ROW_n + COL_n$.

Théorème 16 ([Weg00]) *La taille de tout programme de branchement à lecture unique pour la fonction $ROW_n + COL_n$ testant si une matrice booléenne contient une ligne contenant seulement des 1 ou une colonne contenant seulement des 1 est bornée inférieurement par $\Omega(n^{-7/2}2^n)$.*

Théorème 17

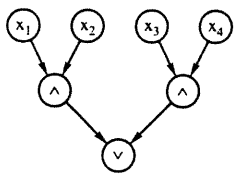
$$COL_n + ROW_n \leq_{\text{rop}} GEN$$

Preuve

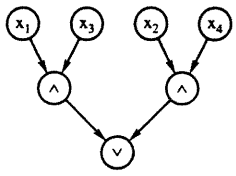
Soit $X = (x_1, x_2, \ldots, x_{n^2})$. Notons $f_n(X) = ROW_n + COL_n(X)$.

Considérons d'abord le cas $f_2(x)$ afin de développer une intuition pour le cas général.

Un circuit pour $ROW_2(X)$ est de la forme suivante :



De même un circuit pour $COL_2(X)$ est de la forme suivante :



Afin de créer un circuit pour $f_2(X)$, il suffit de rejoindre les sorties des circuits pour $ROW_2(X)$ et $COL_2(X)$ et de les rejoindre par une porte OU.

Le cas général, f_n, peut, de manière analogue, être décrit par le circuit suivant :

$$\begin{aligned} f(X) &= (x_1 \wedge x_2 \wedge x_3 \ldots x_n) \\ &\vee (x_{n+1} \wedge x_{n+2} \wedge x_{n+3} \ldots x_{2n}) \\ &\vdots \\ &\vee (x_1 \wedge x_{n+1} \wedge x_{2n+1} \ldots x_{n^2-n+1}) \\ &\vee (x_2 \wedge x_{n+2} \wedge x_{2n+2} \ldots x_{n^2-n+2}) \\ &\vdots \\ &\vee (x_n \wedge x_{2n} \wedge x_{3n} \ldots x_{n^2}) \end{aligned}$$

Ce circuit peut être transformé en entrée pour le problème *GEN* de plusieurs manières : on peut par exemple utiliser une réduction similaire à celle introduite dans la preuve du théorème 8. Nous allons toutefois utiliser ici une autre réduction fort simple : il suffit de créer un élément du groupoïde pour chaque porte du circuit ainsi qu'un nouvel élément 0 et de définir la loi de composition interne comme suit :

- si $x_k = \text{ET}(x_i, x_j)$, alors $i \times j = k$
- si $x_k = \text{OU}(x_i, x_j)$, alors $i \times i = k$ et $j \times j = k$
- dans tous les autres cas $i \times j = 0$.

Dans le cas $f_2(X)$, on obtient, en numérotant les portes du circuit de manière naturelle, le tableau de multiplication suivant :

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	5	8	0	0	0	0	0	0	0	0
2	0	0	0	0	9	0	0	0	0	0	0	0
3	0	0	0	0	6	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	7	0	0	0	0	0	0
6	0	0	0	0	0	0	7	0	0	0	0	0
7	0	0	0	0	0	0	0	11	0	0	0	0
8	0	0	0	0	0	0	0	0	10	0	0	0
9	0	0	0	0	0	0	0	0	0	10	0	0
10	0	0	0	0	0	0	0	0	0	0	11	0
11	0	0	0	0	0	0	0	0	0	0	0	0

Posons finalement que l'ensemble de départ $T = \{i | x_i = \text{VRAI}\}$. Cette définition de GEN implique qu'un élément i du groupoïde est dans $\langle T \rangle$ si et seulement si x_i est VRAI dans le circuit représentant $f_n(X)$. Par conséquent le dernier élément du groupoïde (représentant la dernière porte du circuit) sera dans $\langle T \rangle$ (et par conséquent GEN = VRAI) si et seulement si $f_n(X)$ est VRAI. On vient donc de prouver que $COL_n + ROW_n \leq_{\text{proj}} GEN$.

Déterminons maintenant la multiplicité des variables x_i ($1 \leq i \leq n^2$). On constate que la table de multiplication du groupoïde ne dépend que de la dimension n de la fonction f_n, mais pas de ses variables x_i ; seul l'ensemble de départ T dépend de la valeur des variables x_i. Or, T pourrait être encodé comme liste de $n^2 + 1$

booléens où la première variable est toujours VRAI et la i-ème variable ($i \geq 2$) est VRAI si et seulement si x_{i-1} est vrai. Par conséquent, chaque variable x_i de f_n apparaîtra une et une seule fois dans notre encodage de GEN et il s'agit donc d'une projection à lecture unique : $COL_n + ROW_n \leq_{\text{rop}} GEN$ ■

Théorème 18 *La taille d'un programme de branchement à lecture unique pour GEN est* $\in \Omega(n^{-7/10} \times 2^{n^{1/5}})$

Preuve

Un circuit pour ROW_n est de taille $2n^2 - 1$ (il s'agit d'un arbre binaire complet avec n^2 feuilles). De même la taille d'un circuit pour COL_n est aussi $2n^2 - 1$. Par conséquent, la taille d'un circuit pour $ROW_n + COL_n$ est $(2n^2-1)+(2n^2-1)+1 = 4n^2 - 1$. Ce circuit est transformé en entrée de GEN. La table de multiplication du groupoïde comporte alors exactement $(4n^2)^2 = 16n^4$ entrées. À ceci, il faut encore ajouter les $n^2 + 1$ variables de T. Par ailleurs, chaque entrée de la table de multiplication doit être encodée sur $2 \log n$ bits. Par conséquent la projection se fait en $\Omega(n^5)$.

Grâce au théorème 16 on sait que la taille d'un programme de branchement à lecture unique pour la fonction $ROW_n + COL_n$ est bornée inférieurement par $\Omega(n^{-7/2}2^n)$. Or, $(n^5)^{-7/10} \times 2^{(n^5)^{1/5}} = n^{-7/2}2^n$ et le lemme 5 permet alors de conclure que la taille d'un programme de branchement à lecture unique pour GEN est dans $\Omega(n^{-7/10} \times 2^{n^{1/5}})$. ■

4.4 OBDD pour GEN

Tout comme pour les progammes de branchement à lecture unique, on connaît aussi plusieurs problèmes qui nécessitent une taille de OBDD exponentielle. En particulier, la borne suivante est connue pour la fonction $ROW_n + COL_n$ [Weg00] :

Théorème 19 *La taille de tout OBDD pour* $ROW_n + COL_n$ *est dans* $\Omega\left(2^{\sqrt{n-1}}\right)$.

La connaissance de cette borne inférieure nous permet alors de déterminer une borne inférieure pour la taille de tout OBDD calculant GEN.

Théorème 20 *La taille d'un OBDD pour* GEN *est dans* $\Omega\left(2^{\sqrt{n^{\frac{1}{5}}-1}}\right)$.

Preuve

Nous savons grâce au théorème 17 que la fonction $ROW_n + COL_n \leq_{\text{rop}} \text{GEN}$. On peut alors appliquer le lemme 5 : sachant grâce au théorème 19 que la taille de tout OBDD pour $ROW_n + COL_n$ est bornée inférieurement par $\Omega\left(2^{\sqrt{n-1}}\right)$ il nous suffit d'observer que $2^{\sqrt{(n^5)^{\frac{1}{5}}-1}} = 2^{\sqrt{n-1}}$ afin de pouvoir conclure que la taille de tout OBDD pour GEN doit être dans $\Omega\left(2^{\sqrt{n^{\frac{1}{5}}-1}}\right)$. ■

Nous avons obtenu, dans ce chapitre, des bornes inférieures sur la taille de programmes de branchement résolvant GEN. Notons particulièrement les bornes inférieures exponentielles pour les programmes de branchement à lecture unique et les OBDD : elles constituent les meilleures bornes inférieures connues à ce jour pour ces types de programmes de branchement.

Conclusion

Nous avons établi dans ce mémoire qu'il est possible d'obtenir des bornes inférieures non triviales sur la taille de plusieurs types de programmes de branchement. En particulier, nous avons pu prouver que tout programme de branchement à lecture unique et tout OBDD pour GEN, ainsi que tout arbre de décision pour GEN_S doit forcément être de taille exponentielle. La validité de la borne inférieure pour le programme de branchement à lecture unique et l'OBDD a été démontrée grâce à l'application du concept de la projection à lecture unique et à la connaissance de bornes inférieures pour d'autres problèmes. C'est d'ailleurs la seule parmi les méthodes de preuves de bornes inférieures sur les programmes de branchement (voir [Weg00] pour une liste des techniques connues) que nous avons réussi à appliquer au problème GEN.

Maintenant que nous connaissons des bornes inférieures et supérieures pour les programmes de branchement à lecture unique et les OBDD résolvant GEN, il serait intéressant d'étudier de plus près l'écart entre ces deux bornes. Essayer de réduire cet écart pourrait donner des résultats très intéressants.

Dans cette optique, mentionnons qu'il est possible de relier les coefficients de Fourier d'une fonction à une borne inférieure sur la taille de son arbre de décision : pour tout $S \subseteq \{1, 2, \ldots, n\}$, la taille de l'arbre de décision pour une fonction booléenne f est supérieure ou égale à $2^{|S|} \sum_{T \supseteq S} |\hat{f}(T)|$ où $\hat{f}(T)$ sont les coefficients

de Fourier de f [Weg00]. La difficulté réside alors dans l'expression des coefficients de Fourier de la fonction f, dans notre cas de GEN. Nous n'avons pas réussi à trouver une expression algébrique générale des coefficients de Fourier de GEN mais nous avons pu calculer ces coefficients numériquement pour GEN_S de taille 4. Nous pouvons ainsi prouver que tout arbre de décision pour GEN_S doit posséder au moins 64 nœuds. Cette borne n'est par ailleurs pas nécessairement optimale : nous n'avons pas pu tester toutes les valeurs de S possibles dans notre logiciel pour des raisons de temps d'exécution et la borne de 64 nœuds est donc simplement la meilleure que nous avons pu obtenir.

Malheureusement, on ne connaît toujours pas de borne inférieure exponentielle sur la taille des programmes de branchement non restreints. Cette preuve est d'autant plus difficile à obtenir qu'aucune technique n'est connue à ce jour permettant de prouver ce type de bornes inférieures. La conjecture que $\mathcal{P} \neq \mathcal{L}$ est donc encore ouverte ainsi que la question fortement reliée de savoir s'il existe des problèmes qui possèdent une solution séquentielle efficace, mais qui ne possèdent pas de solution hautement parallèle.

Une direction que la recherche de bornes inférieures pour GEN_S pourrait prendre est celle de la complexité de la communication, direction qui dépasse le champ d'étude de ce mémoire. Il serait intéressant de montrer qu'il faut, pour des raisons d'entropie, de préférence poser des questions à l'intérieur de la fermeture courante. En effet, si on réussissait à prouver qu'il faut toujours poser des questions à l'intérieur de la fermeture courante, on aurait une preuve simple que tout programme de branchement pour GEN_S est de taille exponentielle [Kas92].

Bibliographie

[ABH+86] Miklós Ajtai, László Babai, Péter Hajnal, János Komlós, Pavel Pudlák, Vojtěch Rödl, Endre Szemerédi, and György Turán. Two Lower Bounds for Branching Programs. Dans *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 30–38, 1986.

[Ajt99] Miklós Ajtai. A Non-Linear Time Lower Bound for Boolean Branching Programs. Dans *40th Annual Symposium on Foundations of Computer Science, FOCS'99*, pages 60–70, New York, New York, 17–18 octobre 1999.

[All89] Eric W. Allender. P-Uniform Circuit Complexity. *Journal of the ACM*, 36(4) :912–928, octobre 1989.

[Bar86] David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. Dans *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 1–5, Berkeley, Californie, 28–30 mai 1986.

[BC82] Allan Borodin and Stephen A. Cook. A Time-Space Tradeoff for Sorting on a General Sequential Model of Computation. *SIAM Journal on Computing*, 11(2) :287–297, mai 1982.

[BGK+96] László Babai, Anna Gál, János Kollár, Lajos Rónyai, Tibor Szabó, and Avi Widgerson. Extremal Bipartite Graphs and Superpolynomial Lower Bounds for Monotone Span Programs. Dans *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, Philadelphia, Pennsylvanie, 22–24 mai 1996.

[BHST87] László Babai, Péter Hajnal, Endre Szemerédi, and György Turán. A Lower Bound for Read-Once-Only Branching Programs. *Journal of Computer and System Sciences*, 35(2) :153–162, octobre 1987.

[BM91] David A. Mix Barrington and Pierre McKenzie. Oracle Branching Programs and Logspace versus P. *Information and Computing*, 95(1) :96–115, novembre 1991.

[Bor77] Allan Borodin. On Relating Time and Space to Size and Depth. *SIAM Journal on Computing*, 6(4) :733–744, décembre 1977.

[Bry85] Randal E. Bryant. Symbolic Manipulation of Boolean Functions Using a Graphical Representation. Dans *Proceedings of the 22nd ACM/IEEE*

Design Automation Conference, pages 688–694, Los Alamitos, Californie, 1985. IEEE Computer Society Press.

[Bry92] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3) :293–318, septembre 1992.

[BST98] Paul Beame, Michael Saks, and Jayram S. Thathacher. Time-Space Tradeoffs for Branching Programs. Technical Report TR98-053, Electonic Colloquium on Computational Complexity, 9 septembre 1998.

[BW95] Beate Bollig and Ingo Wegener. Read-Once Projections and Formal Circuit Verification with Binary Decision Diagrams. Technical Report TR95-042, Electonic Colloquium on Computational Complexity, 1995.

[BW96] Beate Bollig and Ingo Wegener. Read-once projections and formal circuit verification with binary decision diagrams. Dans Claude Puech and Rüdiger Reischuk, editeurs, *13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 1996*, numéro 1046 dans Lecture Notes in Computer Science, pages 491–502. Springer-Verlag, 1996.

[BW98a] Beate Bollig and Ingo Wegener. A Very Simple Function that Requires Exponential Size Read-Once Branching Programs. *Information Processing Letters*, 66(2) :53–57, avril 1998.

[BW98b] Beate Bollig and Ingo Wegener. Completeness and Non-Completeness Results with Respect to Read-Once Projections. *Information and Computation*, 143(1) :24–33, mai 1998.

[Cob66] Alan Cobham. The Recognition Problem for the Set of Perfect Squares. Dans *Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory*, pages 78–88, Berkley, Californie, 26–28 octobre 1966.

[Coo70] Stephen A. Cook. Path Systems and Language Recognition. Dans *Conference Record of Second Annual ACM Symposium on the Theory of Computing*, pages 70–72, Northampton, Massachusetts, 4–6 mai 1970.

[Coo71] Stephen A. Cook. The Complexity of Theorem Proving Procedures. Dans *Conference Record of Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3–5 mai 1971.

[Coo73] Stephen A. Cook. An Observation on Time-Storage Trade Off. Dans *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 29–33, Austin, Texas, 30 avril–2 mai 1973.

[Coo79] Stephen A. Cook. Deterministic CFL's are Accepted Simultaneously in Polynomial Time and Log Squared Space. Dans *Conference Record*

of the Eleventh Annual ACM Symposium on Theory of Computing, pages 338–345, Atlanta, Georgia, 30 avril–2 mai 1979.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. Dans *17th Annual Symposium on Foundations of Computer Science*, pages 98–108, Houston, Texas, octobre 1976.

[Dun85] Paul E. Dunne. Lower Bounds on the Complexity of 1-time Only Branching Programs (Preliminary Version). Dans Lothar Budach, editeur, *Proceedings of the 5th Conference on Fundamentals of Computation Theory, Cottbus, September 1985*, numéro 199 dans Lecture Notes in Computer Science, pages 90–99. Springer-Verlag, 1985.

[Gál97] Anna Gál. A Simple Function that Requires Exponential Size Read-Once Branching Programs. *Information Processing Letters*, 62 :13–16, 1997.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation : P-Completeness Theory.* Oxford University Press, 1995.

[Gol77a] Leslie M. Goldschlager. *Synchronous Parallel Computation.* Thèse de doctorat, University of Toronto, 1977.

[Gol77b] Leslie M. Goldschlager. The Monotone and Planar Circuit Value Problems are Log Space Complete for P. *SIGACT News*, 9(2) :25–29, 1977.

[Gri91] Michelangelo Grigni. *Structure in Monotone Complexity.* Thèse de doctorat, Massachusetts Institute of Technology, juin 1991.

[GS92] Michelangelo Grigni and Michael Sipser. Monotone Complexity. Dans Michael S. Paterson, editeur, *Boolean Function Complexity*, volume 169, *London Mathematical Society Lecture Note Series*, pages 57–75. Cambridge University Press, 1992.

[GS95] Michelangelo Grigni and Michael Sipser. Monotone Separation of Logarithmic Space from Logarithmic Depth. *Journal of Computer and System Sciences*, 50(3) :433–437, juin 1995.

[GSS82] Leslie M. Goldschlager, Ralph A. Shaw, and John Staples. The Maximum Flow Problem is Log Space Complete for P. *Theoretical Computer Science*, 21(1) :105–111, octobre 1982.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[HW91] Rafi Heiman and Avi Widgerson. Randomized vs. Deterministic Decision Tree Complexity for Read-Once Boolean Functions. Dans *Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 172–179, Chicago, Illinois, 30 juin–3 juillet 1991.

[JL77] Neil D. Jones and William T. Laaser. Complete Problems for Deterministic Polynomial Time. *Theoretical Computer Science*, 3 :105–117, 1977.

[JLL76] Neil D. Jones, Y. Edmund Lien, and William T. Laaser. New Problems Complete for Nondeterministic Log Space. *Mathematical Systems Theory*, 10 :1–17, 1976.

[Jon75] Neil D. Jones. Space-Bounded Reducibility among Combinatorial Problems. *Journal of Computer and System Sciences*, 11 :68–85, 1975.

[Juk88] Stasys Jukna. Entropy of Contact Circuits and Lower Bound on their Complexity. *Theoretical Computer Science*, 57 :113–129, 1988.

[JŽ98] Stasys Jukna and Stanislav Žàk. On Branching Programs with Bounded Uncertainty (Extended Abstract). Dans Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editeurs, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98*, volume 1443, *Lecture Notes in Computer Science*, pages 259–270, Aalborg, Danemark, 13–17 juillet 1998. Springer-Verlag.

[Kas92] Vahé Kassardjian. Bornes sur la taille de programmes de branchement résolvant le problème d'appartenance à un groupoïde. Mémoire de maîtrise, Université de Montréal, avril 1992.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[KW87] Klaus Kriegel and Stephan Waack. Exponential Lower Bound for Real-Time Branching Programs. Dans Lothar Budach, Rais Gatic Bakharajev, and Oleg Borisovic Lipanov, editeurs, *Proceedings of the International Conference on Fundamentals of Computation Theory (FCT '87)*, volume 278, *Lecture Notes in Computer Science*, pages 263–267, Kazan, URSS, 22–26 juin 1987. Springer.

[KW88] Mauricio Karchmer and Avi Wigderson. Monotone Circuits for Connectivity Require Super-Logarithmic Depth. Dans *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 539–550, Chicago, Illinois, 1988.

[KW93] Mauricio Karchmer and Avi Widgerson. On Span Programs. Dans *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 102–111, San Diego, Californie, 18–21 mai 1993. IEEE Computer Society Press.

[Lad75] Richard E. Ladner. The Circuit Value Problem is Log Space Complete for P. *SIGACT News*, 7(1) :18–20, 1975.

[Lee59] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *The Bell System Technical Journal*, pages 985–999, juillet 1959.

[Mas76] William J. Masek. A fast Algorithm for the String Editing Problem and Decision Graph Complexity. Mémoire de maîtrise, Massachusetts Institute of Technology, 1976.

[Nec66] E. I. Nechiporuk. A Boolean Function. *Soviet Mathematics Doklady*, 7(4) :999–1000, 1966.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Par86] Ian Parberry. Parallel Speedup of Sequential Machines : A Defense of the Parallel Computation Thesis. *SIGACT News*, 18(1) :54–67, 1986.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among Complexity Measures. *Journal of the ACM*, 26(2) :361–381, avril 1979.

[Pip79] Nicholas Pippenger. On Simultaneous Resource Bounds. Dans *20th Annual Symposium on Foundations of Computer Science*, pages 307–311, San Juan, Puerto Rico, octobre 1979.

[Pon95] Stephen Ponzio. A Lower Bound for Integer Multiplication with Read-Once Branching Programs. Dans *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 130–139, Las Vegas, Nevada, 29 mai–1 juin 1995.

[Pon98] Stephen Ponzio. A Lower Bound for Integer Multiplication with Read-Once Branching Programs. *SIAM Journal on Computing*, 28(3) :798–815, 1998.

[PŽ83] Pavel Pudlak and Stanislav Žàk. Space Complexity of Computations. Technical report, Université de Prague, 1983.

[Raz85] Alexander A. Razborov. Lower Bounds on the Monotone Complexity of some Boolean Functions. *Soviet Mathematics- Doklady*, 31(2) :354–357, 1985.

[Rei85] John H. Reif. Depth-First Search is Inherently Sequential. *Information Processing Letters*, 20(5) :229–234, juin 1985.

[Ruz81] Walter L. Ruzzo. On Uniform Circuit Complexity. *Journal of Computer and System Sciences*, 22(3) :365–383, juin 1981.

[Sav70] Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4 :177–192, 1970.

[Sav72] John E. Savage. Computational Work and Time on Finite Machines. *Journal of the ACM*, 19(4) :660–674, octobre 1972.

[Sha49] Claude Elwood Shannon. The Synthesis of Two-Terminal Switching Circuits. *Bell System Technical Journal*, 28 :58–98, 1949.

[Sip97] Micheal Sipser. *Introduction to the Theory of Computation.* PWS, 1997.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time. Dans *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 1–9, Austin, Texas, 30 avril–2 mai 1973.

[SS93] Janos Simon and Mario Szegedy. A New Lower Bound Theorem for Read-Only-Once Branching Programs and its Applications. Dans Jin-Yi Cai, editeur, *Advances in Computational Complexity Theory*, volume 13, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 183–193. American Mathematical Society, Providence, Rhode Island, 1993.

[SV85] Sven Skyum and Leslie G. Valiant. A Complexity Theory Based on Boolean Algebra. *Journal of the ACM*, 32(2) :484–502, avril 1985.

[SŽ96] Petr Savický and Stanislav Žàk. A Large Lower Bound for 1-Branching Programs. Technical Report TR96-036, Electonic Colloquium on Computational Complexity, 11 juin 1996.

[Tur36] Alan Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42 :230–265, 1936.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner series in computer science, 1987.

[Weg88] Ingo Wegener. On the Complexity of Branching Programs and Decision Trees for Clique Functions. *Journal of the ACM*, 35(2) :461–471, avril 1988.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[Yao79] Andrew Chi-Chih Yao. Some Complexity Questions Related to Distributive Computing (Preliminary Report). Dans *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 219–213, Atlanta, Géorgie, 30 avril–2 mai 1979.

[Žàk84] Stanislav Žàk. An Exponential Lower Bound for One-Times-Only Branching Programs. Dans Michal Chytil and Václav Koubek, editeurs, *Mathematical Foundations of Computer Science 1984, Prague, Tchécoslovaquie*, volume 176, *Lecture Notes in Computer Science*, pages 562–566. Springer-Verlag, 1984.