Philippe Chassaing
Sylvain Conchon
Vincent Delecroix
Bruno Lévy
Loïc Paulevé
Nicolas Ray
Adrien Richard
Laurent Simon
Dmitry Sokolov

Informatique Mathématique
Une photographie en 2018

Emmanuel Jeandel et Laurent Vigneron (éd.)

CNRS Éditions

CNRS Éditions, 2018, ISBN : 978-2-271-11974-2.

Dépôt légal : mars 2018.

Sommaire

Les auteurs

Philippe Chassaing est Professeur à l'Université de Lorraine (Institut Élie Cartan). Son chapitre « Combinatoire et couplage probabiliste » parcourt quelques problèmes classiques de combinatoire et d'algorithmique, concernant les permutations, les fonctions de hachage, les arbres binaires, par exemple, pour lesquels un plongement dans un espace continu sous-jacent simplifie radicalement la solution.


Sylvain Conchon est Professeur à l'Université Paris-Sud et au Laboratoire de Recherche en Informatique (LRI). Laurent Simon est Professeur des Universités à Bordeaux INP et au LaBRI. Leur chapitre « Satisfaisabilité Propositionnelle (SAT) et Modulo Théories (SMT) » introduit les techniques modernes de décision pour la logique purement propositionnelle (solveurs CDCL : Conflict Driven Clause Learning), et les solveurs SMT qui combinent CDCL et petits moteurs de preuve pour théories.



Bruno Lévy est Directeur de Recherche Inria, Nicolas Ray est Chargé de Recherche Inria et Dmitry Sokolov est Maître de Conférences à l'Université de Lorraine.
Tous trois sont membres de l'équipe ALICE de l'Inria Nancy Grand-Est et du LORIA. Leur chapitre « Géométrie numérique » présente un ensemble d'algorithmes permettant de manipuler et d'optimiser des représentations de formes 3D.


Loïc Paulevé et Adrien Richard sont Chargés de Recherche CNRS respectivement au Laboratoire de Recherche en Informatique (LRI) à Orsay et au laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis (I3S). Leur chapitre « Analyse statique des réseaux booléens » présente une étude des points fixes et des trajectoires des réseaux booléens, et leurs applications en biologie des systèmes.

Vincent Delecroix est Chargé de Recherche CNRS à l'Université de Bordeaux et au LaBRI. Il travaille à la frontière entre combinatoire, géométrie et systèmes dynamiques. Il est fortement impliqué dans le développement du logiciel Sage.

Préface

Les Écoles des Jeunes Chercheurs/Chercheuses en Informatique Mathématique sont organisées chaque année depuis la création en 2006 du GDR de même nom, le GDR IM. Depuis 2013, les supports des cours donnés à cette école sont édités. Ces livres constituent autant de photographies instantanées de notre domaine, qui illustrent sa diversité, sa vitalité et sa perpétuelle évolution. Ils constituent probablement la meilleure introduction à l'Informatique Mathématique que l'on puisse trouver. Bien plus que de simples supports de cours, ces livres sont maintenant la «vitrine» de notre GDR.

Le présent ouvrage est le sixième de la série « Informatique Mathématique : une photographie en… » et correspond à l'édition 2018 de École organisée à Nancy du 26 au 30 mars.

Le but de cette école annuelle est de donner une formation complémentaire de haut niveau à de jeunes chercheurs (qui sont en général à plus ou moins deux ans de leur soutenance de thèse) : cela peut être pour eux une mise à niveau dans certains domaines, ou une véritable ouverture vers de nouvelles problématiques. Ceci est important : qui travaillera exactement sur son sujet de thèse dans 10 ans ? En leur montrant l'état de la recherche dans des domaines voisins de leur spécialité, l'école permet d'élargir la culture scientifique des doctorants et leur donne des outils qui leur permettront de mieux s'adapter à des environnements variés. Elle contribue ainsi à faciliter leur recrutement et leur mobilité. En leur donnant l'occasion de se rencontrer, de présenter leurs travaux et de confronter leurs idées, elle contribue aussi à créer une communauté de jeunes scientifiques autour des thèmes de l'informatique mathématique.

Nous remercions les auteurs d'avoir accepté avec enthousiasme de s'atteler à ce travail d'écriture, et pour l'ouvrage de grande qualité qui en résulte. Nous tenons aussi à remercier chaleureusement tous les membres du comité d'organisation de l'École Jeunes Chercheurs 2018.

À Lyon, Marseille et Nancy, le 15 novembre 2017,

Guillaume Theyssier et Jean-Michel Muller, co-directeurs du GDR IM
Emmanuel Jeandel et Laurent Vigneron, Coordinateurs de l'ouvrage

Remerciements

Les auteurs et le comité scientifique remercient très chaleureusement les relecteurs pour leur travail et leurs remarques sur les versions préliminaires de cet ouvrage, en particulier Irène Marcovici, Pierre Mercuriali, Renaud Vilmart et Paul Zimmermann. Les éditeurs remercient tout particulièrement Pascal Fontaine pour sa coordination du deuxième chapitre.

Les éditeurs remercient Valérie Berthé pour son aide à la mise en forme de cet ouvrage.

Chapitre 1

Combinatoire et couplage probabiliste

Philippe Chassaing

Dans ce cours on visera à approcher des quantités combinatoires, à étudier leur comportement asymptotique, à travers un couplage entre un objet combinatoire et une variable aléatoire continue déjà bien étudiée. Les couplages probabilistes sont, dans l'esprit, proches des méthodes bijectives spécifiquement combinatoires, et les utilisent parfois.

1.1 Introduction

Dans ce cours on visera à approcher des quantités combinatoires, à étudier leur comportement asymptotique, à travers un couplage entre un objet combinatoire et une variable aléatoire déjà bien étudiée. Un couplage signifiera souvent que l'objet combinatoire est une fonction de variables aléatoires, et construit à partir de variables aléatoires, bien connues. Les objets probabilistes de prédilection dans ce cours seront la loi uniforme sur l'intervalle $[0,1]$, la loi de Poisson et le processus de Poisson, et, plus tard, le mouvement brownien. Par exemple, il existe une construction bien connue d'une permutation aléatoire $\sigma \in \mathfrak{S}_n$ à partir d'un n-uplet $(U_1, U_2, \ldots, U_n)$ de variables aléatoires uniformes [1] indépendantes, selon laquelle la taille $X_n(\sigma)$ du cycle contenant 1 vérifie

$$X_n(\sigma) = \sum_{i=1}^{n} 1_{\{U_i \leqslant U_1\}}.$$

1. Dans la suite, sauf mention explicite du contraire, *variable uniforme* signifie *uniforme sur l'intervalle* $[0,1]$.

Après conditionnement par U_1, le calcul de l'espérance se ramène au calcul de la variance d'une loi binomiale de paramètres $n-1$ et U_1, ce qui donne :

$$\mathbb{E}\left[\left(\frac{X_n}{n}-U_1\right)^2\right]=\frac{1}{n}\mathbb{E}\left[1-U_1\right]+\frac{n-1}{n^2}\mathbb{E}\left[U_1\left(1-U_1\right)\right]\leqslant\frac{2}{3n}.$$

Il en découle la convergence en loi de X_n/n vers la loi uniforme, avec, comme bonus, une estimation de la vitesse de convergence. Comme souvent lors de ces couplages, on obtient la convergence en loi comme conséquence de la convergence dans L^2, de la convergence presque sûre, ou de la convergence en probabilité, voir section 1.5.

Dans une autre construction de σ, voir Section 1.2.2, le nombre $Y_n(\sigma)$ de descentes vérifie :

$$Y_n\left(\sigma\right)=\left\lfloor U_1+U_2+\ldots+U_n\right\rfloor. \tag{1.1}$$

On peut alors en déduire que :

$$2\sqrt{3}\,\frac{Y_n-\dfrac{n}{2}}{\sqrt{n}}\to\mathcal{N}\left(0,1\right),$$

ou encore, que le nombre eulérien $A_{n,k}$ de permutations de $\mathfrak{S}_n$ possédant k descentes satisfait, pour $k-n=\mathcal{O}(\sqrt{n})$:

$$A_{n,k}\simeq n!\,\sqrt{\frac{6}{\pi n}}\,e^{-\dfrac{3}{2n}(2k-n)^2}.$$

On peut aussi utiliser (1.1) pour borner $A_{n,k}$ quand $\sqrt{n}=o(k-n)$, en utilisant par exemple l'inégalité de Hoeffding (Théorème 1.5.8).

Plan du cours. On passera en revue des exemples de couplage entre des objets combinatoires et

- la loi uniforme,
- la loi exponentielle,
- le mouvement Brownien,

après avoir fait quelques rappels sur les lois de probabilité en question. Certains de ces couplages utilisent des bijections typiques de la combinatoire, la méthode de couplage [2] pouvant être vue comme une version probabiliste des méthodes bijectives spécifiquement combinatoires.

2. Une excellente référence sur le sujet est [121].

1.2 Loi uniforme et permutations

Dans cette section on donnera des exemples de couplages entre objets combinatoires et suites de variables aléatoires indépendantes suivant la loi uniforme.

Loi uniforme. À toute partie A de $\mathbb{R}^d$, borélienne, dont la mesure de Lebesgue[3] $\lambda(A)$ est finie et strictement positive, on associe une loi de probabilité, appelée *loi uniforme sur* A, de densité de probabilité f définie, pour $x \in \mathbb{R}^d$, par :

$$f(x) = \frac{1}{\lambda(A)} 1_A(x),$$

où 1_A est la *fonction indicatrice* de l'ensemble A. La densité f est donc nulle à l'extérieur de A mais égale à la constante $\frac{1}{\lambda(A)}$ sur A. Le cas particulier traité principalement dans cette page est le cas où $d = 1$ et où A est l'*intervalle* $[0, 1]$ de $\mathbb{R}$, auquel cas, pour $x \in \mathbb{R}$, la densité est donnée par :

$$f(x) = 1_{0 \leqslant x \leqslant 1}.$$

Invariance de la loi uniforme sur $[0, 1]$ Notons $\{x\}$ la partie fractionnaire du réel x, définie par $\{x\} = x - \lfloor x \rfloor$.

Proposition 1.2.1. *Si U suit la loi uniforme sur $[0, 1]$ alors, pour tout réel a, $\{a + U\}$ et $\{a - U\}$ ont même loi que U.*

Démonstration en secouant les mains. En notant $M(x)$ le point du cercle trigonométrique ayant pour affixe $e^{2i\pi x}$, on peut alors voir $M(U)$ comme un point tiré au hasard uniformément sur ce cercle. Les points $M(\{a + U\})$ et $M(\{a - U\})$ sont alors obtenus par rotation d'angle $2\pi a$ (resp. par symétrie par rapport à la droite d'angle directeur πa) qui sont des isométries laissant le cercle unité invariant. Il n'est donc pas étonnant que ces points suivent encore la loi uniforme sur le cercle unité[4]. ■

On peut en tirer immédiatement une conséquence qui sera utile lors de l'étude des nombres eulériens, Section 1.2.2 :

3. Par *mesure de Lebesgue* d'une partie de $\mathbb{R}$ (resp. $\mathbb{R}^2$, $\mathbb{R}^3$) comprendre sa longueur (resp. sa surface, son volume).

4. Cela traduit une propriété très particulière de la loi uniforme : elle est la mesure de Haar du groupe additif $\mathbb{R}/\mathbb{Z}$, *i.e.*, la seule mesure de probabilité de $\mathbb{R}/\mathbb{Z}$ qui soit invariante par les translations de $\mathbb{R}/\mathbb{Z}$.

Proposition 1.2.2. *Si la suite $V = (V_1, V_2, \ldots, V_n)$ est une suite de variables aléatoires indépendantes et uniformes sur $[0,1]$ et si $U_k = \{V_1 + V_2 + \cdots + V_k\}$, alors la suite $U = (U_1, U_2, \ldots, U_n)$ est une suite de variables aléatoires indépendantes et uniformes sur [0, 1].*

Démonstration. La loi conditionnelle de U_k, sachant que

$$(V_1, V_2, \ldots, V_{k-1}) = (a_1, a_2, \ldots, a_{k-1}),$$

est la loi de $\{a_1 + a_2 + \cdots + a_{k-1} + V_k\}$, qui se trouve être la loi uniforme sur [0, 1], comme on vient de le voir à la Proposition 1.2.1. Donc la loi conditionnelle de U_k sachant que $(V_1, V_2, \ldots, V_{k-1}) = (a_1, a_2, \ldots, a_{k-1})$ ne dépend absolument pas de $(a_1, a_2, \ldots, a_{k-1})$. Cela a deux conséquences :

- U_k suit la loi uniforme sur [0, 1] ;
- U_k est indépendante de la tribu engendrée par $(V_1, V_2, \ldots, V_{k-1})$ et, a fortiori, de la tribu engendrée par $(U_1, U_2, \ldots, U_{k-1})$, puisque la première tribu contient la deuxième.

Cela suffit pour conclure. ■

Il peut sembler surprenant que les variables $\{V_1 + V_2\}$ et $\{V_1 + V_2 + V_3\}$, par exemple, soient indépendantes, alors qu'elles dépendent toutes deux de manière cruciale des variables V_1 et V_2. C'est une conséquence particulière de la propriété d'invariance[4] de la loi uniforme.

1.2.1 Construction d'une permutation aléatoire uniforme à l'aide d'un échantillon de loi uniforme

Des mathématiciens comme Luc Devroye ou Richard P. Stanley ont popularisé l'utilisation de la loi uniforme sur [0, 1] pour l'étude des permutations aléatoires (tailles des cycles, nombres eulériens, analyse d'algorithmes de tri comme le tri rapide, par exemple) [62, 180, 63]. Soit $U = (U_1, U_2, \ldots, U_n)$ une suite de variables aléatoires i.i.d.[5] uniformes sur $[0,1]$. Pour tout entier k compris entre 1 et n, posons

$$\sigma(k, \omega) = \text{Card}\,\{i \text{ tel que } 1 \leqslant i \leqslant n, \text{ et tel que } U_i(\omega) \leqslant U_k(\omega)\}$$

Ainsi, $\sigma(k, \omega)$ s'interprète comme le *rang* de $U_k(\omega)$ dans l'échantillon, une fois celui-ci rangé dans l'ordre croissant.

Proposition 1.2.3. *L'application $k \mapsto \sigma(k, \omega)$ est une permutation aléatoire uniforme.*

5. i.i.d. = *indépendantes et de même loi de probabilité.*

Démonstration. Pour une permutation τ fixée, notons

$$A_\tau = \left\{ x \in \mathbb{R}^n \mid x_{\tau(1)} < x_{\tau(2)} < \cdots < x_{\tau(n)} \right\}$$

et posons

$$\tau.x = (x_{\tau(1)}, x_{\tau(2)}, \ldots, x_{\tau(n)}).$$

Alors

$$\{x \in A_\tau\} \Leftrightarrow \{\tau.x \in A_{\mathrm{Id}}\}.$$

Par ailleurs, de manière évidente, si $U(\omega) \in A_\tau$, alors

$$\{\forall k \text{ tel que } 1 \leqslant k \leqslant n, \quad \sigma(\tau(k), \omega) = k\}\,, \text{ ou encore } \{\sigma(., \omega) = \tau^{-1}\}.$$

Comme

$$\bigcup_{\tau \in \mathfrak{S}_n} A_\tau = \{x \in \mathbb{R}^n \mid \text{ les } x_i \text{ sont tous différents}\}$$

il en découle que

$$B = \mathbb{R}^n \setminus \left(\bigcup_{\tau \in \mathfrak{S}_n} A_\tau \right) = \bigcup_{1 \leqslant i < j \leqslant n} \left\{ x \in \mathbb{R}^n \mid x_i = x_j \right\}$$

Si $U(\omega) \in B$, il existe donc un couple $i < j$ tel que $U_i(\omega) = U_j(\omega)$, et, par suite, $\sigma(i, \omega) = \sigma(j, \omega)$. Ainsi $\sigma(., \omega)$ n'est pas une permutation. Finalement, comme B et les ensembles de type A_ρ forment une partition de $\mathbb{R}^n$, il suit que pour toute permutation τ,

$$\{U(\omega) \notin A_\tau\} \Rightarrow \{\sigma(., \omega) \neq \tau^{-1}\},$$

et par conséquent

$$\mathbb{P}\left(U \in A_\tau\right) = \mathbb{P}\left(\sigma = \tau^{-1}\right).$$

Lorsque les composantes d'un vecteur aléatoire, ici $U = (U_1, U_2, \ldots, U_n)$, sont indépendantes et possèdent des densités de probabilités, notées respectivement f_i, $1 \leqslant i \leqslant n$, on sait que le vecteur aléatoire U possède lui-même une densité f, définie par

$$f(x) = \prod_{i=1}^{n} f_i(x_i).$$

De même, une densité de probabilité du vecteur aléatoire $\tau.U$ est g, définie par :

$$g(x) = \prod_{i=1}^{n} f_{\tau(i)}(x_i).$$

Dans le cas, comme ici, où les composantes d'un vecteur aléatoire sont i.i.d., on peut choisir les densités de probabilités f_i toutes égales. Ainsi, les densités f et g des vecteurs aléatoires U et $\tau.U$ sont égales : les vecteurs aléatoires U et $\tau.U$ ont donc même loi. Par conséquent, pour toute permutation τ,

$$\mathbb{P}\left(U \in A_{\mathrm{Id}}\right) = \mathbb{P}\left(\tau.U \in A_{\mathrm{Id}}\right) = \mathbb{P}\left(U \in A_{\tau}\right) = \mathbb{P}\left(\sigma = \tau^{-1}\right),$$

donc toute permutation a la même probabilité $\mathbb{P}\left(U \in A_{\mathrm{Id}}\right)$. Il reste à montrer que σ est une permutation avec probabilité 1. Or :

$$\mathbb{P}\left(U \in B\right) = \mathbb{P}\left(\exists i < j \text{ tels que } U_i = U_j\right) \leqslant \sum_{1 \leqslant i < j \leqslant n} \mathbb{P}\left(U_i = U_j\right) = 0.$$

En effet l'hyperplan $\{x_i = x_j\}$ est de mesure de Lebesgue nulle, et on a vu que U possède une densité, donc :

$$\left\{\lambda(\{x_i = x_j\}) = 0\right\} \Rightarrow \left\{\mathbb{P}\left(U_i = U_j\right) = 0\right\}.$$

Finalement

$$\begin{aligned} n!\,\mathbb{P}\left(\sigma = \tau\right) &= n!\,\mathbb{P}\left(U \in A_{\tau^{-1}}\right) = n!\,\mathbb{P}\left(U \in A_{\mathrm{Id}}\right) \\ &= \sum_{\rho \in \mathfrak{S}_n} \mathbb{P}\left(U \in A_{\rho}\right) \\ &= \mathbb{P}\left(U \in B\right) + \sum_{\rho \in \mathfrak{S}_n} \mathbb{P}\left(U \in A_{\rho}\right) \\ &= 1, \end{aligned}$$

où la dernière égalité utilise le fait que B et les ensembles A_ρ forment une partition de $\mathbb{R}^n$. ■

La proposition ci-dessus reste vérifiée si la distribution de probabilité commune aux variables U_i possède une densité, quelle qu'elle soit, et non pas seulement pour la densité uniforme. Cependant la loi uniforme est particulièrement commode pour diverses applications.

1.2.2 Nombres de descentes d'une permutation aléatoire et nombres eulériens

Définition 1.2.4.

- *Pour $2 \leqslant j \leqslant n$, on dit que la permutation $\sigma \in \mathfrak{S}_n$ possède une* descente *en j si*

$$\sigma(j-1) > \sigma(j).$$

- *Le nombre de permutations de $\mathfrak{S}_n$ possédant exactement k descentes est appelé* nombre eulérien, *et il est noté $A(n,k)$.*

Soit $Y_n(\omega)$ le nombre de descentes d'une permutation $\sigma(\omega)$ tirée au hasard uniformément dans $\mathfrak{S}_n$. Bien sûr,

$$\begin{aligned}\mathbb{P}\left(Y_n = k\right) &= \frac{\text{nombre de cas favorables}}{\text{nombre de cas possibles}} \\ &= \frac{A(n,k)}{n!}.\end{aligned}$$

Pour une suite i.i.d. $(V_k)_{k\geqslant 1}$ de variables uniformes sur $[0,1]$, posons

$$S_n = V_1 + V_2 + \cdots + V_n.$$

On a alors [186, 76] :

Théorème 1.2.5 (S. Tanny, 1973)**.** *De manière équivalente,*

$$\mathbb{P}\left(Y_n = k\right) = \mathbb{P}\left(\lfloor S_n \rfloor = k\right) = \mathbb{P}\left(k \leqslant S_n < k+1\right),$$

ou bien

$$A(n,k) = n!\,\mathbb{P}\left(k \leqslant S_n < k+1\right).$$

Démonstration. On suppose la suite $U = (U_1, U_2, \ldots, U_n)$ construite à l'aide d'une suite $V = (V_1, V_2, \ldots, V_n)$ de variables aléatoires indépendantes et uniformes sur [0, 1], via la relation $U_k = \{V_1 + V_2 + \cdots + V_k\}$. On sait alors, grâce à la Proposition 1.2.2, que $U = (U_1, U_2, \ldots, U_n)$ est une suite de variables aléatoires indépendantes et uniformes sur [0, 1]. On construit alors une permutation aléatoire uniforme $\sigma(.,\omega)$ à l'aide de la suite U, comme indiqué à la Section 1.2.1 : il y a *descente* au rang i pour $\sigma(.,\omega)$ si $\sigma(i-1,\omega) > \sigma(i,\omega)$ ou, de manière équivalente, si $U_{i-1}(\omega) > U_i(\omega)$. Parallèlement, on dessine, sur le cercle trigonométrique, les points $M_k(\omega)$ ayant pour affixes $\mathrm{e}^{2\mathrm{i}\pi U_k(\omega)}$. On entreprend alors un voyage sur le cercle unité, consistant à parcourir les points $M_1(\omega)$, puis $M_2(\omega)$, puis etc., puis $M_n(\omega)$, dans cet ordre, en tournant toujours dans le sens trigonométrique, et en partant du point A d'affixe 1. Au pas numéro k, on parcourt ainsi un arc de longueur $2\pi V_k$. La longueur totale L du chemin ainsi parcouru est alors donnée par :

$$L = 2\pi\left(V_1 + V_2 + \cdots + V_n\right). \tag{1.2}$$

Par ailleurs, il y a *descente* au rang i pour $\sigma(.,\omega)$ si et seulement si l'étape du voyage ci-dessus allant du point $M_{i-1}(\omega)$ au point $M_i(\omega)$ traverse A.

Donc le nombre de descentes de $\sigma(.,\omega)$ est le nombre de traversées du point A, qui est aussi le nombre de tours *complets* du cercle unité effectués lors du voyage de A à $M_n(\omega)$. Au vu de (1.2), le nombre de tours complets s'écrit aussi :

$$\lfloor V_1(\omega) + V_2(\omega) + \cdots + V_n(\omega) \rfloor .$$

Ainsi, dans cette construction d'une permutation aléatoire uniforme σ, le nombre de descentes de $\sigma(.,\omega)$ est égal à $\lfloor S_n(\omega) \rfloor$. Le nombre de descentes d'une permutation aléatoire uniforme a donc même loi que $\lfloor S_n \rfloor$. ■

Il en découle immédiatement un théorème central limite pour Y_n :

Théorème 1.2.6.

$$\frac{Y_n - \frac{n}{2}}{\sqrt{n/12}} \to \mathcal{N}(0,1) .$$

Démonstration. Le théorème central limite ordinaire (Théorème 1.5.10) donne que

$$\frac{U_1 + U_2 + \cdots + U_n - \frac{n}{2}}{\sqrt{n/12}} \to \mathcal{N}(0,1) .$$

Or

$$\left| \frac{U_1 + U_2 + \cdots + U_n - \frac{n}{2}}{\sqrt{n/12}} - \frac{Y_n - \frac{n}{2}}{\sqrt{n/12}} \right| \leqslant \sqrt{\frac{12}{n}}.$$

En vertu du théorème de Slutsky (Théorème 1.5.11), les deux variables aléatoires ont donc la même loi limite. ■

1.2.3 Structure asymptotique des tailles des cycles

L'analyse de la décomposition des grandes structures discrètes en facteurs ou en composants, et notamment l'étude du comportement asymptotique de la taille de ces composants, a une riche histoire [12]. Par *taille*, on désigne, par exemple,

- le logarithme des facteurs premiers d'un entier naturel,
- le degré des facteurs irréductibles d'un polynôme sur un corps fini,
- le nombre de sommets de chaque arbre d'une forêt combinatoire ,
- ou encore le nombre de sommets des composantes connexes d'un graphe aléatoire.

Le problème du parking de Knuth fait aussi apparaître cette problématique, voir Exercice 1.4.7. L'exemple peut-être le plus célèbre est celui de la décomposition en cycle d'une permutation aléatoire, dont nous esquissons l'étude ici, à l'aide de la correspondance fondamentale de Foata. En combinatoire, la correspondance fondamentale de Foata est une correspondance entre

suites sans répétition et permutations, différente de la correspondance classique, où la suite sans répétition est la suite des images, par la permutation, des éléments 1, 2, 3, etc. dans cet ordre. La correspondance fondamentale de Foata facilite, par exemple, l'analyse combinatoire du nombre de cycles et de la taille des cycles d'une permutation.

Description

Il y a plusieurs manières d'encoder une permutation à l'aide d'une suite $\omega = (\omega_1, \omega_2, \ldots, \omega_n)$ de n nombres distincts tirés de $[\![1, n]\!]$:

1. de la manière la plus classique, à partir de ω, on obtient la permutation $T\omega = \sigma$ définie par $\sigma(i) = \omega_i$, $1 \leqslant i \leqslant n$; si ω=(2, 8, 10, 6, 1, 3, 12, 5, 4, 9, 7, 15, 11, 14, 13), alors

$$\begin{aligned}\sigma(1) &= 2\\ \sigma(2) &= 8\\ \sigma(3) &= 10\\ \sigma(4) &= 6\\ \ldots &= \ldots\end{aligned}$$

2. d'une manière plus liée à la structure des cycles, à partir de ω, on obtient la permutation $R\omega = \tau$ définie par $\tau(1) = \omega_1$, $\tau^2(1) = \omega_2$, $\ldots$, $\tau^k(1) = \omega_k$, tant que k est plus petit que la longueur ℓ_1 du cycle de τ contenant 1 (ℓ_1 est la taille de l'orbite de 1). Ainsi la position de 1 dans la suite ω signale la longueur de ce cycle : $\omega_{\ell_1} = 1$. Comment interpréter alors ω_{ℓ_1+1} ? Foata propose d'utiliser la suite restante $\omega' = (\omega_{\ell_1+1}, \omega_{\ell_1+2}, \ldots, \omega_n)$, si elle n'est pas vide, pour encoder les images du plus petit élément m_2 de cette suite

$$m_2 = \min\{\omega_k | \ell_1 + 1 \leqslant k \leqslant n\},$$

en posant $\tau(m_2) = \omega_{\ell_1+1}$, $\tau^2(m_2) = \omega_{\ell_1+2}, \ldots, \tau^k(m_2) = \omega_{\ell_1+k}$, là encore, tant que k est plus petit que la longueur ℓ_2 du cycle de τ contenant m_2 : la position de m_2 dans la suite ω signale d'ailleurs la longueur de ce cycle : $\omega_{\ell_1+\ell_2} = m_2$. On itère alors le procédé avec la suite $\omega'' = (\omega_{\ell_1+\ell_2+1}, \omega_{\ell_1+\ell_2+2}, \ldots, \omega_n)$, tant qu'elle n'est pas vide, pour encoder les images du plus petit élément de cette suite

$$m_3 = \min\{\omega_k | \ell_1 + \ell_2 + 1 \leqslant k \leqslant n\}.$$

Le nombre d'itérations de ce procédé est le nombre de cycles de la décomposition de τ en cycles disjoints. Cette seconde correspondance entre

suites sans répétition et permutations est précisément la correspondance fondamentale de Foata.

Exemple 1.2.7.

- *Toujours avec* ω=$(2,8,10,6,1,3,12,5,4,9,7,15,11,14,13)$, *on a*

$$\begin{aligned}
&m_1 = 1, \ell_1 = 5, \tau(1) = 2, \tau(2) = 8, \tau(8) = 10, \tau(10) = 6, \tau(6) = 1,\\
&m_2 = 3, \ell_2 = 1, \tau(3) = 3,\\
&m_3 = 4, \ell_3 = 3, \tau(4) = 12, \tau(12) = 5, \tau(5) = 4,\\
&m_4 = 7, \ell_4 = 2, \tau(7) = 9, \tau(9) = 7,\\
&m_5 = 11, \ell_5 = 2, \tau(11) = 15, \tau(15) = 11,\\
&m_6 = 13, \ell_6 = 2, \tau(13) = 14, \tau(14) = 13.
\end{aligned}$$

  - ⋆ *Ainsi la correspondance fondamentale de Foata associe à* ω *la permutation* τ *dont la décomposition en cycles disjoints est :*

$$\tau = 2,8,10,6,1_{\circlearrowleft}\ 3_{\circlearrowleft}\ 12,5,4_{\circlearrowleft}\ 9,7_{\circlearrowleft}\ 15,11_{\circlearrowleft}\ 14,13_{\circlearrowleft} = R(\omega).$$

  - ⋆ *Par ailleurs l'écriture traditionnelle de* τ *est :*

$$T^{-1} \circ R(\omega) = (2,8,3,12,4,1,9,10,7,6,15,5,14,13,11).$$

- *D'un autre côté, la décomposition en cycles disjoints de* σ *est :*

$$\sigma = 2,8,5,1_{\circlearrowleft}\ 10,9,4,6,3_{\circlearrowleft}\ 12,15,13,11,7_{\circlearrowleft}\ 14_{\circlearrowleft}.$$

  Ainsi la correspondance fondamentale de Foata associe à σ *la suite* $\widetilde{\omega}$ *suivante :*

$$\widetilde{\omega} = R^{-1} \circ T(\omega) = (2,8,5,1,10,9,4,6,3,12,15,13,11,7,14).$$

Cet encodage d'une permutation par une suite, attribué à Foata, est connu pour être bijectif. En effet :

Propriété 1.2.8. *Les cycles de la permutation* $R\omega$ *associée à la suite* ω *par la correspondance fondamentale de Foata sont décrits par les fragments de cette suite* ω *qui se terminent par les records vers le bas de la suite renversée* $\widetilde{\omega} = (\omega_n, \omega_{n-1}, \ldots, \omega_1)$. *En particulier le nombre de cycles dans la décomposition de la permutation* $R\omega$ *est égal au nombre de records vers le bas de la suite renversée* $\widetilde{\omega} = (\omega_n, \omega_{n-1}, \ldots, \omega_1)$.

Cela permet de retrouver les fins de cycles de la permutation encodée par la suite ω, et de vérifier ainsi que chaque suite possède un antécédent unique dans l'ensemble des permutations.

Exemple 1.2.9. *Dans l'exemple de la section précédente, les records vers le bas de la suite renversée apparaissent ci-dessous en gras et soulignés :*

$$\widetilde{\omega}=(\underline{\mathbf{13}}, 14, \underline{\mathbf{11}}, 15, \underline{\mathbf{7}}, 9, \underline{\mathbf{4}}, 5, 12, \underline{\mathbf{3}}, \underline{\mathbf{1}}, 6, 10, 8, 2),$$

ce qui, comparé à la décomposition en cycles disjoints de la permutation τ :

$$\tau=2, 8, 10, 6, 1_{\circlearrowleft}\, 3_{\circlearrowleft}\, 12, 5, 4_{\circlearrowleft}\, 9, 7_{\circlearrowleft}\, 15, 11_{\circlearrowleft}\, 14, 13_{\circlearrowleft},$$

indique comment inverser la correspondance fondamentale de Foata.

1.2.4 Stick-breaking process, processus de Poisson-Dirichlet et longueurs des cycles

Le comportement asymptotique de la taille des composants des grandes structures discrètes de type *logarithmique* [12] est décrit par un processus de Poisson-Dirichlet de paramètre θ, noté $PD(\theta)$: par exemple, pour la taille des cycles des permutations aléatoires [107], mais aussi pour les logarithmes des facteurs premiers d'un nombre entier [26], ou encore les degrés des facteurs irréductibles des polynômes aléatoires sur un corps fini [11], la limite est $PD(1)$, alors que pour les applications aléatoires de $[\![1, n]\!]$ dans lui-même la limite est $PD(1/2)$, cf. [3].

Stick-breaking process et processus de Poisson-Dirichlet

Le processus de Poisson-Dirichlet [106] de paramètre θ est une variable aléatoire $X = (X_k)_{k\geqslant 1}$ à valeurs dans le simplexe de dimension infinie :

$$S = \left\{ x = (x_k)_{k\geqslant 1} \mid x_i \geqslant 0, \forall i \geqslant 1 \text{ et } \sum_{i\geqslant 1} x_i = 1 \right\}.$$

La description la plus parlante du processus de Poisson-Dirichlet est donnée par l'algorithme suivant, qui produit le processus de Poisson-Dirichlet $X = (X_k)_{k\geqslant 1}$:

- casser un bâton de longueur 1, en deux morceaux de tailles aléatoires respectives $Y_1 = U_1$ et $R_1 = 1 - U_1$,
- puis casser à nouveau le morceau restant $R_1 = 1 - U_1$ en deux morceaux aléatoires $Y_2 = R_1 U_2$, et $R_2 = R_1(1 - U_2)$,
- puis casser à nouveau le morceau restant R_2 en deux morceaux aléatoires $Y_3 = R_2 U_3$, et $R_3 = R_2(1 - U_3)$, etc. de manière à produire une suite $Y = (Y_k)_{k\geqslant 1}$ à valeurs dans S,

- ordonner la suite $Y = (Y_k)_{k\geqslant 1}$ dans l'ordre décroissant pour obtenir une suite $X = (X_k)_{k\geqslant 1}$ à valeurs dans S, et, par ailleurs, décroissante.

Si les variables aléatoires réelles $(U_k)_{k\geqslant 1}$ sont indépendantes et suivent toutes la loi Beta de paramètre $(1, \theta)$, de densité :

$$f(x) = \theta (1-x)^{\theta-1} 1_{0<x<1},$$

alors, la suite $X = (X_k)_{k\geqslant 1}$ ainsi produite suit la loi de Poisson-Dirichlet de paramètre θ, et la suite Y suit la loi $GEM(\theta)$. Dans le cas particulier $\theta = 1$, qui décrit le comportement asymptotique de la décomposition des entiers en facteurs premiers, et des permutations en cycles, les variables aléatoires réelles $(U_k)_{k\geqslant 1}$ intervenant dans la définition du processus de Poisson-Dirichlet suivent la loi Beta de paramètre (1,1), qui n'est autre que la loi uniforme sur l'intervalle [0,1]. Notons que, pour $\theta = 1$, la loi de X_1 (premier terme de la suite X) est appelée *distribution de Dickman* et est omniprésente dans l'analyse probabiliste d'objets algébriques (taille du plus grand facteur premier d'un nombre entier tiré au hasard, degré du facteur irréductible de plus haut degré d'un polynôme tiré au hasard, taille du plus long cycle d'une permutation tirée au hasard, etc.) [12, 189].

Proposition 1.2.10. *La suite $Y(\omega)$ appartient à S si et seulement si*

$$\sum_{i\geqslant 1} U_i(\omega) = +\infty, \tag{1.3}$$

ce qui se produit avec probabilité 1 dans le cas du processus de Poisson-Dirichlet. Les $Y_i(\omega)$ sont donnés par la formule explicite

$$Y_i(\omega) = U_i(\omega) \prod_{1\leqslant k\leqslant i-1} (1 - U_k(\omega)), \tag{1.4}$$

et les restes $R_i(\omega)$ sont donnés par

$$R_i(\omega) = \prod_{1\leqslant k\leqslant i} (1 - U_k(\omega)). \tag{1.5}$$

Démonstration. L'équation (1.5) découle, par récurrence, de la description de l'algorithme, et entraîne immédiatement (1.4), puisque :

$$Y_n(\omega) = U_n(\omega) R_{n-1}(\omega) = R_{n-1}(\omega) - R_n(\omega). \tag{1.6}$$

La relation (1.3) est une condition nécessaire et suffisante pour que $\lim_n R_n = 0$, ce qui équivaut à $Y \in S$, puisque :

$$R_n + \sum_{k=1}^{n} Y_k = 1.$$

Finalement, avec probabilité 1, en vertu de la loi forte des grands nombres (Théorème 1.5.7) :

$$\lim_n n^{-1} (U_1 + U_2 + \ldots + U_n) = \mathbb{E}[U_1]$$

et

$$\mathbb{E}[U_1] = \frac{1}{1+\theta} > 0,$$

ce qui assure que (1.3) soit vérifié. ■

Lien avec les longueurs des cycles

Considérons une permutation au hasard sur n symboles, τ, ou encore la suite ω de n nombres tous différents qui lui est associée par la correspondance de Foata. Notons $X^{(n)}(\omega) = \left(X_k^{(n)}(\omega)\right)_{k\geqslant 1}$ la suite finie des longueurs des cycles de la décomposition de τ, rangées par ordre décroissant, longueurs toutes divisées par n, suite finie ensuite complétée par une suite infinie de zéros, de sorte que $X^{(n)}(\omega) \in S$. Par exemple, pour

$$\omega = (2, 8, 10, 6, 1, 3, 12, 5, 4, 9, 7, 15, 11, 14, 13)$$

et

$$\tau = 2, 8, 10, 6, 1_{\circlearrowleft}\ 3_{\circlearrowleft}\ 12, 5, 4_{\circlearrowleft}\ 9, 7_{\circlearrowleft}\ 15, 11_{\circlearrowleft}\ 14, 13_{\circlearrowleft},$$

on a

$$X^{(15)}(\omega) = \left(\frac{5}{15}, \frac{3}{15}, \frac{2}{15}, \frac{2}{15}, \frac{2}{15}, \frac{1}{15}, 0, 0, 0, 0, \ldots\right).$$

Alors

Théorème 1.2.11 ([107]). *La suite* $\left(X^{(n)}\right)_{n\geqslant 1}$ *converge en loi vers une distribution de Poisson-Dirichlet de paramètre 1.*

Étude des restes. Sachant le début $(\omega_1, \omega_2, \ldots, \omega_{n-k})$ d'une suite aléatoire uniforme

$$\omega = (\omega_1, \omega_2, \ldots, \omega_n)$$

de n nombres distincts tirés de $[\![1, n]\!]$, la fin de la suite, de longueur k, est uniformément distribuée parmi les $k!$ fins de suites possibles. Ainsi, dans la correspondance de Foata, les positions des nombres m_i (*i.e.* les positions des records successifs, qui dictent les longueurs des cycles) sont uniformément distribuées sur la place laissée par les cycles précédents. Cela fait apparaître

naturellement une version discrète du *stick-breaking process*, et dans cette version discrète, la loi des restes est simple à décrire. Posons $R_0^{(n)} = n$ et :

$$R_1^{(n)} = n - nY_1^{(n)}, \quad R_2^{(n)} = n - nY_1^{(n)} - nY_2^{(n)}, \quad \dots$$

ou bien :

$$nY_j^{(n)} = R_{j-1}^{(n)} - R_j^{(n)}, \qquad j \geqslant 1.$$

L'uniformité des suites résiduelles, remarquée en préambule, entraîne que la loi conditionnelle de $R_{\ell+1}^{(n)}$ sachant que $\left(R_j^{(n)}\right)_{1\leqslant j\leqslant \ell} = (r_1, r_2, \dots, r_{\ell-1}, k)$, est la loi uniforme sur $[\![0, k-1]\!]$: en effet, la position du minimum de la suite résiduelle, de longueur k, étant uniformément distribuée sur $[\![1, k]\!]$, il suit que le nombre de termes de la suite apparaissant après le minimum est uniformément distribué sur $[\![0, k-1]\!]$. Cela fait de la suite $R^n = \left(R_j^{(n)}\right)_{j\geqslant 0}$ une chaîne de Markov, partant de n, de probabilités de transition très simples

$$\mathbb{P}\left(R_{t+1}^{(n)} = j \,\middle|\, R_t^{(n)} = k\right) = \frac{1}{k}\, 1_{0\leqslant j<k}, \qquad k, j \in \mathbb{N}.$$

Or on peut engendrer une copie de cette chaîne de Markov à l'aide d'une suite $\left(U_j\right)_{j\geqslant 1}$ de variables aléatoires indépendantes uniformes sur $[0,1]$, via la relation de récurrence

$$\widehat{R}_j^{(n)} = \left\lfloor \widehat{R}_{j-1}^{(n)}(1-U_j) \right\rfloor, \; j \geqslant 1, \quad \text{et} \quad \widehat{R}_0^{(n)} = n,$$

ce qui permet de définir également une copie de Y^n, via :

$$n\widehat{Y}_j^{(n)} = \widehat{R}_{j-1}^{(n)} - \widehat{R}_j^{(n)}, \qquad j \geqslant 1. \tag{1.7}$$

Ainsi R^n et $\widehat{R}^n$ (resp. Y^n et $\widehat{Y}^n$) ont même loi. Posons

$$R_j = \prod_{k=1}^{j}(1-U_k), \; Y_j = R_{j-1} - R_j, \; j \geqslant 1, \;\text{ et } R_0 = 1. \tag{1.8}$$

On voit que

$$0 \leqslant nR_1 - \widehat{R}_1^{(n)} \leqslant 1,$$

et, par récurrence,

$$\left.\begin{array}{l} 0 \leqslant \widehat{R}_{j-1}^{(n)}\left(1-U_j\right) - \widehat{R}_j^{(n)} \leqslant 1 \\ 0 \leqslant nR_{j-1} - \widehat{R}_{j-1}^{(n)} \leqslant j-1 \end{array}\right\} \Rightarrow 0 \leqslant nR_j - \widehat{R}_j^{(n)} \leqslant (j-1)\left(1-U_j\right) + 1 \leqslant j.$$

Pour montrer la convergence en loi désirée, on se ramène aux suites finies de longueur quelconque, à l'aide du Théorème 1.5.12 : pour tout $j \geqslant 0$,

$$0 \leqslant R_j - \frac{\widehat{R}_j^{(n)}}{n} \leqslant \frac{j}{n},$$

donc, pour tout $k \geqslant 1$,

$$\frac{1}{n}\left(\widehat{R}_j^{(n)}\right)_{0\leqslant j\leqslant k} \xrightarrow{p.s.} \left(R_j\right)_{0\leqslant j\leqslant k}, \quad \left(\widehat{Y}_j^{(n)}\right)_{1\leqslant j\leqslant k} \xrightarrow{p.s.} \left(Y_j\right)_{1\leqslant j\leqslant k},$$

la deuxième convergence résultant de (1.7) et de (1.8). Comme la convergence presque sûre entraîne la convergence en loi,

$$\widehat{Y}^{(n)} \xrightarrow{\mathcal{L}} Y,$$

et, $Y^{(n)}$ ayant même loi que $\widehat{Y}^{(n)}$, on obtient le Théorème 1.2.11. ■

1.3 Loi exponentielle

Dans cette section on examinera quelques structures combinatoires liées de manière étroite à la loi exponentielle, comme, par exemple, l'arbre binaire de recherche, ou encore les surjections, à l'occasion du collectionneur de coupons.

Définition 1.3.1.

- *On dit qu'une variable aléatoire $X \geqslant 0$ suit la loi exponentielle de paramètre $\lambda \in]0, +\infty[$, et on note parfois $X \sim \mathcal{E}(\lambda)$, si X a pour densité*

$$f(x) = \lambda e^{-\lambda x} 1_{x>0}.$$

Notons que

$$\mathbb{E}[X] = \frac{1}{\lambda}, \quad Var(X) = \frac{1}{\lambda^2}, \quad \mathbb{P}(X \geqslant x) = e^{-\lambda x}, \quad x \geqslant 0.$$

- *On dit qu'une variable aléatoire réelle N suit la loi de Poisson de paramètre $\lambda \geqslant 0$ (et on note $N \sim \mathcal{P}(\lambda)$) si :*

$$\forall k \geqslant 0, \qquad \mathbb{P}(N = k) = \frac{\lambda^k}{k!} e^{-\lambda}.$$

Notons que

$$\mathbb{E}[N] = Var(N) = \lambda.$$

1.3.1 Propriétés

Rappelons que la loi exponentielle est la seule loi à densité amnésique, ou sans usure : si on interprète une variable exponentielle X d'espérance λ comme une durée de vie (positive), alors la durée de vie résiduelle, i. e. le temps qu'il reste à vivre au temps t, a même loi que X :

Propriété 1.3.2 (Propriété d'amnésie)**.** *Quel que soit $t > 0$:*

$$\mathbb{P}\left(X - t \geqslant u | X \geqslant t\right) = \mathbb{P}\left(X \geqslant u\right).$$

Une autre propriété importante porte sur l'infimum d'une famille de variables exponentielles :

Proposition 1.3.3. *Pour une famille $(X_i)_{i\in I}$ de variables exponentielles indépendantes de paramètres respectifs λ_i, on pose $Y = \inf_{j\in I}\left(X_j\right)$, et on note J la variable aléatoire telle que $X_J = Y$. Alors :*

1. *Y suit une loi exponentielle de paramètre $\lambda = \sum_i \lambda_i$ (avec la convention qu'une variable aléatoire Y suit une loi exponentielle de paramètre $+\infty$ si $\mathbb{P}\left(Y = 0\right) = 1$);*
2. *si $\lambda = \sum_i \lambda_i < +\infty$, alors J est bien défini, indépendant de Y, et pour tout i*
$$\mathbb{P}\left(J = i\right) = \frac{\lambda_i}{\lambda}.$$

Démonstration. Clairement, pour $u > 0$,

$$\mathbb{P}\left(Y \geqslant u\right) = \mathbb{P}\left(X_i \geqslant u, \forall i \in I\right) = \prod_{i\in I} \mathbb{P}\left(X_i \geqslant u\right) = e^{-u\lambda}.$$

On suppose maintenant que $\lambda < +\infty$, et on pose $Y_i = \inf_{j\in I\setminus\{i\}}\left(X_j\right)$. Si $X_i < Y_i$, alors J est bien défini et vaut i. Or Y_i suit la loi exponentielle de paramètre $\mu_i = \lambda - \lambda_i < +\infty$, et est indépendante de X_i. On va voir que

$$\mathbb{P}\left(X_i < Y_i\right) = \frac{\lambda_i}{\lambda}. \tag{1.9}$$

Ainsi, avec probabilité $1 = \sum_i \frac{\lambda_i}{\lambda}$, il existe un seul $i \in I$ tel que $X_i = Y$, donc J est bien défini avec probabilité 1, et sa loi est bien comme annoncé.

Calculons $\mathbb{P}(x < X_i < Y_i)$ pour $x \geqslant 0$ [6] :

$$\begin{aligned}
\mathbb{P}(x < Y,\, J = i) &= \mathbb{P}(x < X_i < Y_i) \\
&= \int_{\mathbb{R}^2} 1_{x<u<v} \lambda_i \mu_i e^{-\lambda_i u} e^{-\mu_i v}\, 1_{u>0} 1_{v>0}\, du\, dv \\
&= \int_{\mathbb{R}} 1_{x<u} \lambda_i e^{-\lambda_i u} \left(\int_u^{+\infty} \mu_i e^{-\mu_i v} dv \right) du \\
&= \int_{\mathbb{R}} 1_{x<u} \lambda_i e^{-\lambda_i u} e^{-\mu_i u} du \\
&= \frac{\lambda_i}{\lambda_i + \mu_i} e^{-(\lambda_i + \mu_i)x} = \frac{\lambda_i}{\lambda} e^{-\lambda x},
\end{aligned}$$

ce qui entraîne à la fois l'indépendance et les lois annoncées pour Y et J. ■

Les propriétés précédentes sont la clé de cette section car elles permettent d'établir un lien entre des objets discrets, les chaînes de Markov, et des objets continus, comme le processus de Poisson et les processus de Markov à temps continu, dont nous verrons un ou deux exemples, comme le processus de Yule, un peu plus loin. On donne maintenant deux définitions équivalentes du processus de Poisson sur $\mathbb{R}_+$. Pour cela, $\#A \in \mathbb{N} \cup \{+\infty\}$ désigne maintenant le nombre d'éléments d'un ensemble A.

Définition 1.3.4. *Un ensemble aléatoire Π de points de $\mathbb{R}_+$ est appelé processus de Poisson d'intensité λ si*

1. *pour tout intervalle $[a, b]$ de $\mathbb{R}_+$, $\#([a, b] \cap \Pi)$ suit la loi de Poisson de paramètre $\lambda(b - a)$;*
2. *et si pour toute suite $(I_k)_{1 \leqslant k \leqslant n}$ d'intervalles disjoints, la famille $(\#(I_k \cap \Pi))_{1 \leqslant k \leqslant n}$ est indépendante.*

On note habituellement $\#([0, t] \cap \Pi) = N_t$.

Définition 1.3.5. *Un ensemble aléatoire $\Pi = \{X_1 < X_2 < X_3 < \ldots\}$ de points de $\mathbb{R}_+$ est appelé processus de Poisson d'intensité λ si et seulement si $(X_1, X_2 - X_1, X_3 - X_2 \ldots)$ est une suite de variables exponentielles indépendantes de paramètre λ.*

Il est remarquable que ces deux définitions soient équivalentes, l'indépendance de la famille $(\#(I_k \cap \Pi))_{1 \leqslant k \leqslant n}$, en particulier, découlant de la propriété d'amnésie. Cependant le processus de Poisson possède d'autres propriétés intéressantes, dont la suivante, qui nous sera particulièrement utile :

6. Ce qui donne aussi (1.9), si on choisit $x = 0$.

Proposition 1.3.6 (Réunion et étiquetage). *Notons Π la réunion d'une famille indépendante de processus de Poisson $(\Pi_i)_{i\in I}$ d'intensités respectives $(\lambda_i)_{i\in I}$, avec $\lambda = \sum_{i\in I}\lambda_i < +\infty$. Alors Π est lui-même un processus de Poisson, d'intensité λ. De plus, avec probabilité 1, les Π_i sont tous disjoints. Après avoir écrit Π sous la forme $\{X_1 < X_2 < X_3 < \ldots\}$, comme dans la Définition 1.3.5, notons Y_j l'élément de I tel que $X_j \in \Pi_{Y_j}$. Alors la suite $(Y_j)_{j\geqslant 1}$ est une suite de variables i.i.d. satisfaisant*

$$\mathbb{P}\left(Y_j = i\right) = \frac{\lambda_i}{\lambda}, \quad \forall\,(i,j) \in I \times \mathbb{N}^\star.$$

Il en découle que réciproquement, si on étiquette les éléments d'un processus de Poisson Π avec des labels tirés au hasard dans I selon la loi de probabilité $(p_i)_{i\in I}$, et si on note Π_i le sous-ensemble de Π constitué des points étiquetés i, on obtient en la famille $(\Pi_i)_{i\in I}$ une famille de processus de Poisson indépendants d'intensités respectives $(\lambda p_i)_{i\in I}$.

1.3.2 Nombre de Stirling, collectionneur de coupons

Le nombre de Stirling de deuxième espèce $\left\{{m \atop n}\right\}$ compte le nombre de partitions d'un ensemble à m éléments en n parties non vides, de sorte que $n!\left\{{m \atop n}\right\}$ compte le nombre de surjections d'un ensemble à m éléments sur un ensemble à n éléments. Dans un problème d'allocation où on jette m items (boules) au hasard dans n emplacements (boîtes),

$$p_{m,n} = \frac{n!\left\{{m \atop n}\right\}}{n^m}$$

représente la probabilité qu'aucune des boîtes ne soit complètement vide. Dans le problème des anniversaires, $p_{m,365}$ représente la probabilité qu'on fête un anniversaire chaque jour de l'année, dans un groupe de m personnes.

Exercice 1.3.7. *Montrer que $p_{m,365} \geqslant 0,5$ dès que $m \geqslant 2287$.*

Dans le problème du collectionneur de vignettes (*coupon collector*), Petit Pierre recherche les vignettes (portraits) de chacun des n joueurs de l'équipe nationale de foot, basket, ou quidditch, par exemple, qui sont cachés dans l'emballage des tablettes de chocolat Célui : $p_{m,n}$ représente alors la probabilité que sa collection soit complète après l'achat de la m-ième tablette, et $p_{m,n} - p_{m-1,n}$ est la probabilité de compléter la collection exactement lors de l'achat de la m-ième tablette. Pour des applications, voir

[136, 72, 29]. Notons T_n le temps de complétion (ou le nombre de tablettes achetées). On a donc :

$$\mathbb{P}\left(T_n = m\right) = p_{m,n} - p_{m-1,n}, \quad \mathbb{P}\left(T_n \geqslant m\right) = p_{m,n}.$$

D'après [95], la propriété de réunion des processus de Poisson (Proposition 1.3.6) rend certains calculs de combinatoire plus simples : par exemple, soient $(\Pi_1, \Pi_2, \ldots, \Pi_n)$ n processus de Poisson indépendants de même paramètre $1/n$. Imaginons que les points de Π_k représentent les instants où Petit Pierre acquiert une vignette du joueur numéro k, des instants qui ne sont donc plus entiers, comme ils l'étaient auparavant : ils sont séparés par des intervalles de temps suivant une loi exponentielle de paramètre $1/n$, d'espérance n, alors qu'auparavant ces laps de temps représentaient des achats de tablettes et suivaient la loi géométrique de paramètre $1/n$, d'espérance n. Alors les points de $\Pi = \cup_{i=1}^{n} \Pi_i$ représentent les dates où Petit Pierre tente d'augmenter sa collection en achetant une tablette de chocolat. Par ailleurs, d'après la Propriété 1.3.6, les points de Π forment un processus de Poisson d'intensité 1, et sont donc espacés par des intervalles de largeur moyenne 1, indépendants et suivant la loi exponentielle. Ainsi la date T_n^c où Petit Pierre complète sa collection est la *position* du T_n-ième point de Π, et vérifie :

$$\mathbb{E}\left[T_n^c\right] = \mathbb{E}\left[T_n\right].$$

Notons Z_i la position du premier point du processus Π_i. Alors

$$T_n^c = \max_{1 \leqslant i \leqslant n} Z_i.$$

Comme les Z_i sont indépendants et vérifient

$$\mathbb{P}\left[Z_i \leqslant t\right] = 1 - e^{-t/n},$$

on a

$$\mathbb{P}\left(T_n^c \leqslant t\right) = \left(1 - e^{-t/n}\right)^n \tag{1.10}$$

puis, à l'aide de la formule du binôme appliquée à (1.10), en passant ensuite au complémentaire :

$$\begin{aligned}\mathbb{E}[T_n] &= \int_0^{+\infty} \mathbb{P}(T_n^c > t)\, dt \\ &= \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} \int_0^{+\infty} e^{-\frac{kt}{n}}\, dt \\ &= n \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} \frac{1}{k} \\ &= nH_n \left(= n \sum_{k=1}^{n} \frac{1}{k} \right),\end{aligned}$$

d'après une identité due à Euler, concernant le nombre harmonique H_n.

Exercice 1.3.8. *Montrer que dans le cas général, lorsque la vignette numéro i apparaît avec une probabilité v_i, $v_1 + v_2 + \ldots + v_n = 1$, le temps de complétion moyen est donné par :*

$$\sum_{I \subset [1,n], I \neq \emptyset} (-1)^{|I|+1} \frac{1}{\sum_{i \in I} v_i}.$$

Dans le cas uniforme, on peut même, à l'aide du couplage avec ces processus de Poisson, préciser les fluctuations de T_n autour de son espérance :

Proposition 1.3.9. *Si W_n est définie par $T_n = n \ln(n) + nW_n$, alors W_n converge en loi vers la loi de Gumbel, de fonction de répartition $e^{-e^{-t}}$.*

Démonstration. Tout d'abord, on voit que si W_n^c est définie par $T_n^c = n \ln(n) + nW_n^c$, alors W_n^c converge en loi vers la loi de Gumbel : en effet, pour $t \in \mathbb{R}$,

$$\begin{aligned}\lim_n \mathbb{P}(W_n^c \leqslant t) &= \lim_n \mathbb{P}(T_n^c \leqslant n \ln(n) + nt) \\ &= \lim_n \left(1 - \frac{e^{-t}}{n}\right)^n \\ &= e^{-e^{-t}},\end{aligned}$$

où la deuxième égalité découle de (1.10). Cela indique, d'une certaine manière, que le terme de second ordre du développement de T_n^c est un $\mathcal{O}(n)$.

On peut voir par ailleurs que

$$T_n^c - T_n = \mathcal{O}\left(\sqrt{n \ln(n)}\right).$$

En effet, d'après la Proposition 1.3.6, Π est un processus de Poisson d'intensité 1, pour lequel les positions des points (*i.e.* les dates d'achat des tablettes) et les numéros des vignettes obtenues lors de chaque achat sont *indépendants*, les numéros étant de plus, ici, uniformes. De la sorte, les intervalles de temps entre les achats forment une suite $X = (X_n)_{n\geqslant 1}$ de variables exponentielles d'espérances et de variances égales à 1, et X est indépendant de T_n. On peut donc écrire que :

$$T_n^c = \sum_{k=1}^{T_n} X_k,$$

et donc

$$\begin{aligned}
\mathbb{E}\left[(T_n^c - T_n)^2\right] &= \mathbb{E}\left[\left(-T_n + \sum_{k=1}^{T_n} X_k\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{k=1}^{T_n} (X_k - 1)\right)^2\right] \\
&= \sum_{\ell\geqslant 1} \mathbb{E}\left[\left(\sum_{k=1}^{\ell} (X_k - 1)\right)^2\right] \mathbb{P}\left(T_n = \ell\right) \\
&= \sum_{\ell\geqslant 1} \mathrm{Var}\left(\sum_{k=1}^{\ell} X_k\right) \mathbb{P}\left(T_n = \ell\right) \\
&= \sum_{\ell\geqslant 1} \ell\, \mathbb{P}\left(T_n = \ell\right) \\
&= \mathbb{E}\left[T_n\right] = nH_n \simeq n\ln(n).
\end{aligned}$$

Ainsi

$$\mathbb{E}\left[(W_n - W_n^c)^2\right] = \frac{1}{n^2}\mathbb{E}\left[(T_n - T_n^c)^2\right] \simeq \frac{\ln n}{n},$$

ou encore

$$W_n - W_n^c = \mathcal{O}\left(\sqrt{\frac{\ln n}{n}}\right).$$

En vertu du Théorème 1.5.11, on conclut que W_n et W_n^c ont même loi limite, la loi de Gumbel. ■

Cette construction permet aussi de retrouver une identité combinatoire classique concernant les nombres de Stirling.

Exercice 1.3.10. *Montrer que :*

$$\mathbb{P}\left(T_n^c \leqslant t\right) = \sum_{\ell \geqslant 1} \mathbb{P}\left(N_t = \ell\right) \mathbb{P}\left(T_n \leqslant \ell\right)$$

peut se réécrire de manière équivalente :

$$\sum_{\ell \geqslant n} \left\{ \begin{matrix} \ell \\ n \end{matrix} \right\} \frac{u^\ell}{\ell!} = \frac{\left(e^u - 1\right)^n}{n!}.$$

1.3.3 Arbre binaire de recherche et processus de Yule

Soit X un ensemble totalement ordonné. Un arbre binaire de recherche (dans la suite abrégé par ABR) étiqueté par des éléments de X est un arbre binaire dont chaque nœud interne est étiqueté par une clé appartenant à X, et tel que pour tout nœud interne, la clé du nœud soit supérieure à toutes les clés du sous-arbre gauche et inférieure à toutes les clés du sous-arbre droit partant de ce nœud. Les feuilles ne sont pas étiquetées. L'ABR associé à une suite finie d'éléments distincts de X est défini de manière récursive. Soient $x_1, x_2, \ldots, x_n$ n éléments distincts de X. L'ABR associé à $x = (x_1, x_2, \ldots, x_n)$ est l'arbre binaire défini par :

- Si $n = 0$, l'arbre ne comprend qu'une racine, non étiquetée.
- Sinon la racine est étiquetée par x_1.
- Le sous-arbre gauche est l'ABR correspondant à la sous-suite de x contenant les termes inférieurs à x_1.
- Le sous-arbre droit est l'ABR correspondant à la sous-suite de x contenant les termes supérieurs à x_1.

L'ABR associé à une suite de n éléments deux à deux distincts est donc un arbre binaire avec n nœuds internes et $n + 1$ feuilles, dont les sommets internes sont étiquetés, de manière à ce que l'étiquette d'un sommet soit plus grande que les étiquettes présentes dans le sous-arbre gauche et plus petite que les étiquettes présentes dans le sous-arbre droit.

Arbre binaire de recherche associé à une permutation uniforme

La forme de l'arbre associé à une suite finie ne dépend que de l'ordre entre les éléments, et pas de la valeur exacte de ceux-ci. Pour cette raison, il est souvent plus simple de remplacer la suite x par l'unique permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ dont les termes sont rangés dans le même ordre que ceux de x, permutation donnée par

$$\sigma_i = \#\left\{j \,:\, x_j \leqslant x_i\right\}. \tag{1.11}$$

L'arbre binaire de recherche associé à une permutation aléatoire uniforme de $\{1, \dots, n\}$ est noté ABR_n. Il s'agit d'un arbre binaire aléatoire à n sommets, mais il ne s'agit pas d'un arbre binaire aléatoire *uniforme*. Son comportement asymptotique est connu avec précision [152, 61, 65, 158, 128] : par exemple [158], sa hauteur $h(ABR_n)$ est proche de $\alpha \ln n + \beta \ln \ln n$ [7], avec α la solution de

$$\alpha \ln\left(\frac{2e}{\alpha}\right) = 1$$

plus grande que 1, et

$$\beta = \frac{3}{2\ln\left(\frac{\alpha}{2}\right)}.$$

Arbre de Yule

Soit λ un réel strictement positif. En probabilités, et en dynamique des populations, l'*arbre de Yule* $Y = (Y_t)_{t\geqslant 0}$ de paramètre λ est la représentation d'un processus de branchement à temps continu, *i.e.* d'un modèle pour l'évolution d'une population d'individus, ou de particules, à travers l'arbre généalogique de cette population. L'arbre de Yule $Y = (Y_t)_{t\geqslant 0}$ est défini de la manière suivante :

1. Au temps $t = 0$, il y a une particule.
2. Chaque particule meurt et donne naissance à deux particules à taux λ, i. e. au bout d'un laps de temps aléatoire de loi exponentielle de paramètre λ. Dans l'arbre généalogique de cette population de particules, chaque particule est représentée par un sommet au bout d'une arête de longueur égale à sa durée de vie.

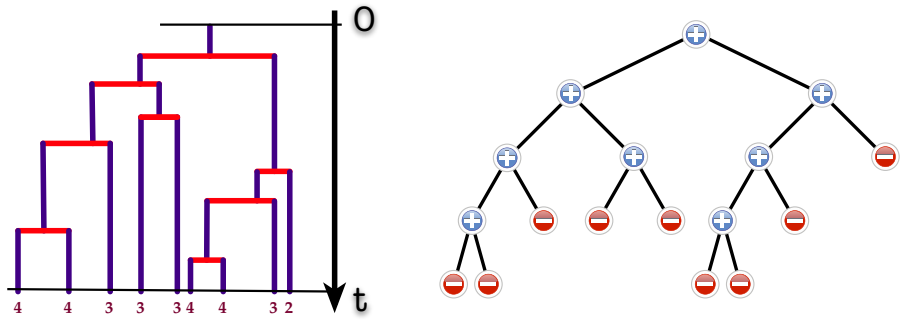


FIGURE 1.1 – *Arbre de Yule et arbre binaire associé. Les profondeurs des feuilles sont indiquées sous l'arbre de Yule, et le profil est ici* $L_9 = (0, 1, 4, 4, 0, 0, \dots)$.

7. $(\alpha, \beta) = (4.311\dots, 1.953\dots)$.

On appelle arbre binaire associé à $Y = (Y_t)_{t\geqslant 0}$ l'arbre combinatoire obtenu en oubliant la longueur des arêtes. La figure 1.1 présente un exemple d'arbre de Yule (tronqué au temps t) et son arbre binaire associé. L'arbre de Yule permet d'analyser aisément ABR_n, via le couplage suivant :

Proposition 1.3.11. *Soit $\tau_n = \inf\{t : |Y_t| = n\}$ le premier temps où l'arbre de Yule Y a n sommets. Alors*

1. *l'arbre binaire associé à $Y_{\tau_{n+1}}$ a la même loi que ABR_n ;*
2. *$Y_{\tau_{n+1}}$ est indépendant de τ_{n+1}.*

Démonstration. Pour toute permutation σ de $\{1,\dots,n\}$ et tout k entier entre 1 et $n+1$, on définit la permutation $f(\sigma,k)$ de $\{1,\dots,n+1\}$ par :

- Pour tout $i \in \{1,\dots,n\}$ tel que $\sigma(i) < k$, $f(\sigma,k)(i) = \sigma(i)$.
- Pour tout $i \in \{1,\dots,n\}$ tel que $\sigma(i) \geqslant k$, $f(\sigma,k)(i) = \sigma(i)+1$.
- $f(\sigma,k)(n+1) = k$.

Cette construction revient à choisir k comme image de $n+1$, tout en respectant l'ordre relatif des images des éléments précédents. Cela correspond à l'évolution de la permutation σ définie par (1.11), lors de l'ajout, à la liste $(x_1, x_2, \dots, x_n)$, d'un nouvel élément x_{n+1}.

Exemple 1.3.12. *Si $\sigma = (3,1,2,4)$, alors $f(\sigma,3) = (4,1,2,5,3)$.*

On considère le processus aléatoire $(\sigma_n)_{n\geqslant 0}$ défini par récurrence de la manière suivante :

- σ_0 est l'unique permutation sur l'ensemble vide.
- Pour tout $n \geqslant 0$, $\sigma_{n+1} = f(\sigma_n, X_n)$, où X_n est un entier uniforme sur $\{1,\dots,n+1\}$, indépendant de σ_n.

Lemme 1.3.13. *Pour tout $n \geqslant 0$, σ_n est une permutation uniforme parmi les permutations de $\{1,\dots,n\}$.*

Démonstration. Ce lemme se prouve par récurrence. Il est immédiat pour $n = 0$, et on suppose qu'il est vrai pour un certain $n \geqslant 0$. Soit $\sigma^\star$ une permutation de $\{1,\dots,n+1\}$. Il existe une unique permutation ρ de $\{1,\dots,n\}$ et un unique entier k entre 1 et $n+1$ tel que $\sigma^\star = f(\rho,k)$, donc :

$$\begin{aligned}\mathbb{P}\left(\sigma_{n+1} = \sigma^\star\right) &= \mathbb{P}(\sigma_n = \rho)\mathbb{P}(X_n = k)\\ &= \frac{1}{n!}\frac{1}{n+1} = \frac{1}{n+1!},\end{aligned}$$

ce qui prouve le lemme pour $n+1$. ■

On considère maintenant le processus $(ABR_n)_{n\geqslant 1}$, où ABR_n est l'ABR associé à σ_n, pour tout entier n. Cela donne la construction suivante pour la suite des ABR_n :

- Initialement, ABR_0 ne contient qu'une feuille.
- Pour passer de ABR_n à ABR_{n+1}, on choisit un entier aléatoire X_n uniforme entre 1 et $n+1$. La X_n-ième feuille dans le contour de l'arbre est remplacée par un nœud interne ayant deux feuilles comme enfants.

Ce couplage permet de démontrer la Proposition 1.3.11, par récurrence sur n, car $(Y_{\tau_{n+1}})_{n\geqslant 0}$ et $(ABR_n)_{n\geqslant 0}$ évoluent de la même manière : conditionnellement à l'arbre à n nœuds internes, une feuille est choisie au hasard uniformément, et est remplacée par un nœud interne ayant deux feuilles comme enfants. L'indépendance entre $Y_{\tau_{n+1}}$ et τ_{n+1} découle du fait que le choix de la feuille est indépendant de l'instant de ce choix, en vertu de la Proposition 1.3.3. ■

1.3.4 Étude de la hauteur de l'ABR via l'arbre de Yule

On note $L_{n,k}$ (resp. $\Lambda_{t,k}$) le nombre de feuilles à profondeur k dans ABR_n (resp. dans Y_t). Le processus $L_n = (L_{n,k})_{k\geqslant 0}$ est appelé le *profil* de ABR_n (voir figure 1.1) et le profil de Y_t est noté Λ_t. Le couplage entre Y et ABR_n décrit par la Proposition 1.3.11 permet une étude fine de L_n, cf. [154, 45, 46]. Nous allons illustrer cette technique de couplage en donnant une preuve simple du résultat classique de Devroye [61] :

Théorème 1.3.14.

$$\frac{h\left(ABR_n\right)}{\ln n} \xrightarrow{P} \alpha.$$

Soit $X = (X_0, X_1, X_2, \ldots)$ une suite i.i.d. de variables aléatoires de Bernoulli[8] de paramètre $1/2$, indépendante de Y. On considère X comme une branche infinie aléatoire de l'arbre binaire complet infini, de la manière suivante : en partant de la racine, si $X_i = 0$, alors, à la profondeur i, la branche infinie emprunte la branche menant au sous-arbre gauche (et elle emprunte la branche menant au sous-arbre droit si $X_i = 1$). Tout arbre binaire fini T peut être considéré comme un sous-arbre de l'arbre binaire complet infini. Ce faisant, chaque direction X est associée à une unique feuille de T, appelée la feuille de T dans la direction X.

Le profil Λ_t dépend étroitement de la structure géométrique de l'arbre dans son ensemble, mais la propriété d'amnésie, et ses conséquences, permettent de ramener l'évaluation de $\Lambda_{t,k}$ à l'étude d'une seule feuille, *tirée au hasard*.

8. *i.e.* $\mathbb{P}\left(X_i = 0\right) = \mathbb{P}\left(X_i = 1\right) = \frac{1}{2}$.

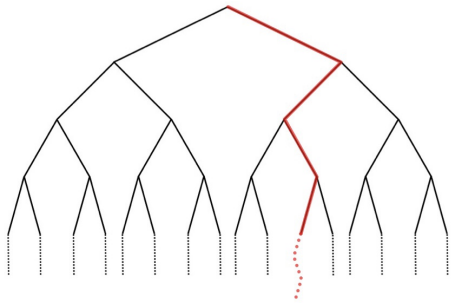


FIGURE 1.2 – *Le chemin* $(1010010\dots)$ *dans l'arbre binaire complet.*

Lemme 1.3.15. *Soit x la feuille de Y_t dans la direction X , et soit h_x sa profondeur dans Y_t. Alors*

$$\mathbb{E}\left[\Lambda_{t,k}\right] = 2^k \mathbb{P}\left(h_x = k\right).$$

Démonstration. On conditionne par Y_t. La probabilité que x soit une feuille donnée de profondeur k vaut 2^{-k}, puisque cela force le choix des k premiers termes de X. Il y a $\Lambda_{t,k}$ feuilles de profondeur k, donc

$$\mathbb{P}\left(h_x = k \,|\, Y_t\right) = 2^{-k}\Lambda_{t,k}.$$

Le lemme est obtenu en prenant l'espérance des deux termes de l'égalité précédente. ■

Dans un processus de Yule de paramètre 1, chaque individu meurt et donne naissance à deux sommets à taux 1. En conséquence, le long de la direction X, les instants où un sommet se divise forment un processus de Poisson d'intensité 1 sur $[0, t]$. Comme la hauteur de x dans Y_t est égale au nombre de divisions de sommets le long de X , on obtient :

$$\mathbb{P}\left(h_x = k\right) = \frac{e^{-t}t^k}{k!},$$

puis

$$\mathbb{E}\left[\Lambda_{t,k}\right] = \frac{e^{-t}t^k}{k!}\,2^k.$$

Si l'on prend $\gamma > 0$ et k de la forme $k_t = \lfloor \gamma t \rfloor$, on obtient, par la formule de Stirling :

$$\begin{aligned}
\ln\left(\mathbb{E}\left[\Lambda_{t,k_t}\right]\right) &= k_t \ln 2 - t + k_t \ln\left(t\right) - \ln\left(k_t!\right) \\
&= \gamma t \ln\left(2\right) - t + \gamma t \ln\left(t\right) - \gamma t \ln\left(\gamma t\right) + \gamma t + o(t) \\
&= t\left(\gamma \ln\left(\frac{2e}{\gamma}\right) - 1\right) + o\left(t\right).
\end{aligned}$$

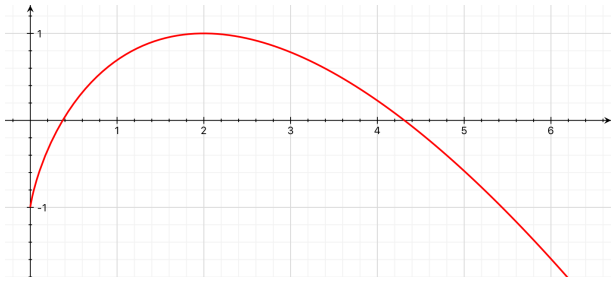


FIGURE 1.3 – *La fonction φ.*

La fonction $\varphi : x \to x \ln\left(\frac{2e}{x}\right) - 1$ est strictement concave, positive en 1, et elle tend vers -1 en 0 et vers $-\infty$ en $+\infty$. Il existe donc deux réels α^- et α tels que φ est positive sur $[\alpha^-; \alpha]$ et négative en dehors de cet intervalle. Ainsi le nombre moyen de feuilles à profondeur $\lfloor \gamma t \rfloor$ dans Y_t tend exponentiellement vite vers 0 lorsque $\gamma > \alpha$ et augmente exponentiellement lorsque $\alpha^- < \gamma < \alpha$. À l'aide de l'inégalité de Markov (voir section 1.5), cela permet de montrer que pour tout $\varepsilon > 0$, avec grande probabilité, il n'existe aucune feuille à profondeur plus grande que $(\alpha + \varepsilon)t$, et au moins une feuille à profondeur supérieure à $(\alpha - \varepsilon)t$, et donc :

$$\frac{h(Y_t)}{t} \xrightarrow{P} \alpha. \tag{1.12}$$

Le laps de temps $\tau_{n+1} - \tau_n$ est le temps qui s'écoule entre l'apparition de la n-ième feuille et l'apparition de la $(n+1)$-ième feuille dans $(Y_t)_{t \geqslant 0}$. Chacune des n feuilles présentes au temps τ_n se divise au bout d'un temps exponentiel de paramètre 1, indépendamment, et $\tau_{n+1} - \tau_n$ est le minimum de ces n variables exponentielles indépendantes, donc, en vertu de la proposition 1.3.3, $\tau_{n+1} - \tau_n$ suit une loi exponentielle de paramètre n, indépendamment de τ_n.

En conséquence, si $(E_i)_{i \geqslant 1}$ désigne une suite de variables aléatoires exponentielles indépendantes telle que pour chaque $i \geqslant 1$, E_i a pour paramètre i, pour espérance $1/i$ et pour variance $1/i^2$, alors τ_{n+1} a même loi que $\sum_{i=1}^{n} E_i$. En particulier,

$$\mathbb{E}[\tau_n] = H_n \simeq \ln n, \qquad \mathrm{Var}(\tau_n) \simeq \frac{\pi^2}{6}.$$

Par conséquent, en vertu de l'inégalité de Bienaymé-Tchebychev,

$$\frac{\tau_n}{\ln n} \xrightarrow{P} 1. \tag{1.13}$$

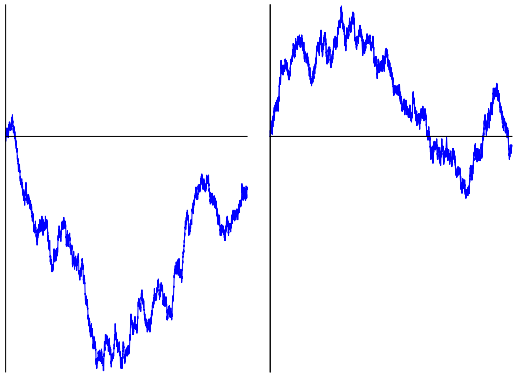


FIGURE 1.4 – *Deux trajectoires de mouvement Brownien (en fait, deux trajectoires de marche aléatoire simple symétrique sur 10000 pas).*

Avec (1.12), cela donne :

$$\frac{h\left(Y_{\tau_n}\right)}{\tau_n} \xrightarrow{P} \alpha, \tag{1.14}$$

puis en combinant le théorème de Slutsky avec (1.13) et (1.14), nous retrouvons le résultat de Devroye :

$$\frac{h\left(ABR_n\right)}{\ln n} \xrightarrow{P} \alpha,$$

comme indiqué.

1.4 Mouvement Brownien

Une approximation par le mouvement Brownien donne une interprétation éclairante des propriétés macroscopiques de nombreuses structures combinatoires de grande taille, comme le graphe aléatoire [2], les arbres simples [125], ou encore les schémas d'allocation [4]. Cette section a pour but d'illustrer le lien entre le mouvement Brownien et certains grands objets combinatoires à travers un couplage entre le problème du parking de Knuth [43] et le pont Brownien.

1.4.1 Définition et propriétés du mouvement Brownien

On peut voir le mouvement Brownien $B = (B(t))_{t\geqslant 0}$ comme une fonction tirée au hasard parmi les fonctions définies sur $\mathbb{R}_+$, continues, à

valeurs dans $\mathbb{R}$, ou bien comme une famille de variables aléatoires réelles, famille indicée par $\mathbb{R}_+$. Cet objet mathématique est apparu simultanément (*i.e.* entre 1900 et 1905) à quelques scientifiques, dont Bachelet et Einstein, à qui sa découverte est le plus souvent attribuée. On peut avoir une bonne idée de son comportement en simulant sur tableur une trajectoire longue de marche aléatoire simple symétrique [9], et en laissant le tableur ajuster l'échelle de la représentation graphique, voir figure 1.4. Que cela donne une bonne idée de l'allure des trajectoires du mouvement Brownien résulte du théorème de Donsker (1951), selon lequel une trajectoire de marche aléatoire sans biais sur n pas converge vers le mouvement Brownien, modulo une réduction d'échelle par un facteur $1/n$ (resp. $1/\sqrt{n}$) le long de l'axe horizontal (resp. vertical). Les trajectoires de marches aléatoires sont des chemins, au sens combinatoire, et il existe de nombreuses bijections entre graphes, arbres, cartes d'une part, et chemins combinatoires, d'autre part, ce qui explique l'apparition fréquente du mouvement Brownien pour expliquer le comportement des grandes structures combinatoires.

Pour une définition plus formelle, rappelons d'abord que :

Définition 1.4.1. *On dit qu'une variable aléatoire réelle X_t suit la loi normale centrée (que X_t est gaussienne centrée) de variance $t > 0$ (et on note $X_t \sim \mathcal{N}(0,t)$) si X_t a pour densité de probabilité f, définie sur $\mathbb{R}$ par*

$$f(x) = \frac{1}{\sqrt{2\pi t}} e^{-\frac{x^2}{2t}}.$$

Plus généralement, une variable aléatoire réelle X_t suit la loi normale centrée de variance $t \geqslant 0$ si X_t a même loi que $\sqrt{t}X_1$.

Définition 1.4.2. *Une fonction continue $B = (B_t)_{t\geqslant 0}$ tirée au hasard est un* mouvement Brownien standard *si*

- $\mathbb{P}(B_0 = 0) = 1$,
- *pour tout $k \geqslant 1$, et tout k-uplet $(t_1 < t_2 < \ldots < t_k)$ de réels strictement positifs, $\left(B_{t_\ell} - B_{t_{\ell-1}}\right)_{1\leqslant \ell \leqslant k}$ est une famille de variables indépendantes (avec la convention $t_0 = 0$),*
- *pour tout couple (t,s) de réels positifs, $B_{t+s} - B_t \sim \mathcal{N}(0,s)$.*

On peut voir le pont Brownien $b = (b_t)_{0\leqslant t\leqslant 1}$ comme le mouvement Brownien conditionné à revenir en 0 au temps 1 (*i.e.*, conditionné à $B_1 = 0$), et on peut voir l'excursion Brownienne normalisée comme le pont Brownien conditionné à rester strictement positif sur tout l'intervalle $]0,1[$. Ces deux

9. qui fait à chaque pas un saut de ± 1 avec équiprobabilité.

processus ont des définitions alternatives, plus rigoureuses, données ci-dessous. Pour tout $t > 0$ notons g_t (resp. d_t) le dernier zéro de $t \to B_t$ à gauche de t (resp. le premier zéro à droite de t), en gardant à l'esprit que $\mathbb{P}(B_t \neq 0) = 1$. On a alors, avec probabilité 1 :

$$g_t < t < d_t, \quad B_{g_t} = B_{d_t} = 0, \quad \forall u \in]g_t, d_t[, B_u \neq 0,$$

et on peut poser :

Définition 1.4.3.

- *Le* pont Brownien $b = (b(t))_{t \geqslant 0}$ *est défini par :*

$$\forall t \in [0,1], \quad b(t) = B_t - tB_1.$$

- *L'excursion Brownienne normalisée enjambant t est définie par $e_t = (e_t(u))_{0 \leqslant u \leqslant 1}$, où :*

$$\forall t \in [0,1], \quad e_t(u) = \frac{1}{\sqrt{d_t - g_t}} \left| B_{g_t + u(d_t - g_t)} \right|.$$

Sa loi est la même quel que soit $t > 0$. Ainsi, un processus ayant même loi que e_t est appelé excursion Brownienne normalisée *et est souvent noté $e = (e(u))_{0 \leqslant u \leqslant 1}$.*

Ces deux processus sont liés par la *transformation de Verwaat* [23] :

Théorème 1.4.4. *Soit un couple (U, e) de variables aléatoires indépendantes, U uniforme sur $[0,1]$, e une excursion Brownienne normalisée. Prolongeons*[10] *e en une fonction $\widetilde{e}$ définie et continue sur $\mathbb{R}$, périodique de période 1, et posons*

$$\forall s \in [0,1], \qquad b(s) = \widetilde{e}(U + s) - \widetilde{e}(U).$$

Alors b est un pont Brownien.

On voit que b atteint son minimum en $1 - U$, i. e. en un point uniforme de $[0,1]$. Ce minimum est $-\widetilde{e}(U)$ d'où le corollaire suivant :

Théorème 1.4.5. *Le processus $\widetilde{b} - \min b$ est, à un décalage de l'argument près, la périodisée d'une excursion Brownienne normalisée.*

10. Une fonction continue f, définie sur un intervalle $[a,b]$, telle que $f(b) = f(a)$, se prolonge de manière unique en une fonction continue sur $\mathbb{R}$, de période $b - a$, la *périodisée* $\widetilde{f}$ de f.

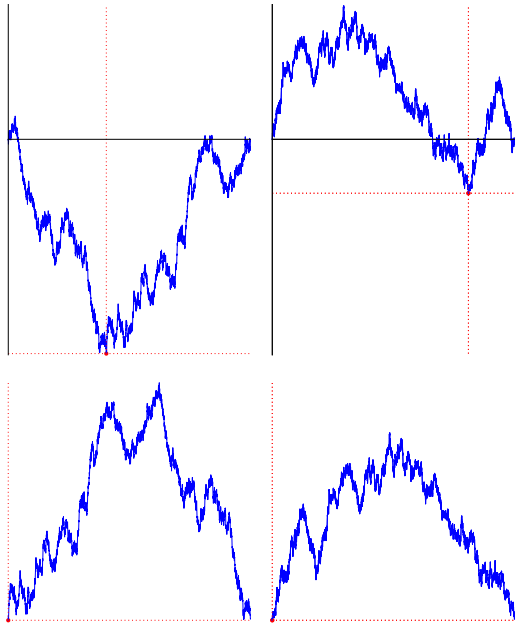


FIGURE 1.5 – *Trajectoires de ponts, en haut, et d'excursions, en bas, liées par la transformation de Verwaat.*

1.4.2 Parking, ou tables de hachage

Si on dispose de m tiroirs et de n types d'objets ($n \leqslant m$), l'idéal pour ranger puis retrouver rapidement chaque objet serait de ranger chaque type d'objet dans un tiroir consacré à ce seul type, tiroir dont le numéro est calculable simplement à partir du type au moyen d'une fonction dite "de hachage". L'utilisation du système de rangement est alors rapide : étant donné le nom x du type, on calcule la valeur numérique "hachée" $h(x)$, puis, dans le tiroir $h(x)$, on retrouve du premier coup les objets cherchés. Si n est petit devant m aucun type ne sera rangé dans le même tiroir mais d'après le paradoxe des anniversaires, dès que $m = \mathcal{O}\left(n^2\right)$ il existe une bonne probabilité de collisions lors du rangement. Pour qu'une telle méthode soit efficace, il faudrait donc en gros prévoir un nombre de tiroirs nettement supérieur au *carré* du nombre de types ! Une solution au problème des collisions est d'utiliser des fonctions de hachage *de secours*. D'après [42],

> "En termes informatiques, on dispose d'une table $T[1\,..\,m]$ dont chaque entrée peut contenir une donnée et des informations auxiliaires. Les données sont issues d'un univers $\mathcal{U}$ (par exemple les chaînes alphabétiques de longueur au plus 20). On dispose d'une fonction h dite "fonction de

> hachage" : celle-ci [11] envoie le domaine $\mathcal{U}$, en général très grand (ici de cardinalité $2 \cdot 10^{29}$), sur un intervalle beaucoup plus petit $[1\,..\,m]$ (le nombre de cases ou tiroirs en général entre 10^2 et 10^6). On range alors chaque donnée x à l'adresse $h(x)$ dans la table. Pour régler les collisions qui sont statistiquement inévitables, lorsque la case de $h(x)$ est déjà prise, on choisit de ranger x à la première case disponible parmi $1 + h(x)$, $2 + h(x)$, etc. De plus, dans la variante circulaire du procédé, on recommencera à partir de la première case (case numéro 1) suite à un échec sur la case m."

D'après la légende, c'est ce problème qui a éveillé l'intérêt de D.E. Knuth pour l'analyse d'algorithmes ; on peut trouver un développement approfondi sur le *linear probing* dans [110, Ch. 6.4]. Une configuration de hachage à n objets et m places est donc déterminée par une suite d'éléments de $\{1, \dots, m\}$ ayant longueur n, *i.e.*, une application de $\{1, \dots, n\}$ dans $\{1, \dots, m\}$ tirée au hasard, avec équiprobabilité des m^n configurations. Un des paramètres fondamentaux d'une configuration est le *déplacement total* $D_{m,n}$ défini comme la somme (sur x) des distances (circulaires) entre l'endroit où x est placé et la valeur de $h(x)$. Cette variable aléatoire $D_{m,n}$ décrit le coût de construction de la table de hachage, lequel coût se situe entre 0 et

$$0 + 1 + \cdots + (n-1) = \binom{n}{2} \simeq \frac{n^2}{2}.$$

En réalité, comme l'a montré Knuth [110, Theorem K] il y a quelque cinquante ans :

$$\mathbb{E}\left[D_{m,n}\right] = n + \frac{n}{2}\left(\frac{n-1}{m} + \frac{(n-1)(n-2)}{m^2} + \dots\right),$$

ce qui permet d'obtenir le comportement asymptotique, non trivial d'ailleurs, cf. [110, 42] : on a, pour $0 < \alpha < 1$,

$$\mathbb{E}\left[D_{m,\alpha m}\right] \sim \alpha\, \frac{2-\alpha}{2-2\alpha}\, m, \quad \text{et} \quad \mathbb{E}\left[D_{m,m-1}\right] \sim \sqrt{\frac{\pi}{8}}\, m^{3/2}. \tag{1.15}$$

On voit donc apparaître une transition de phase entre $\alpha < 1$ et $\alpha = 1$, qui demande exploration. Une première avancée est le résultat de Flajolet, Viola et Poblete :

Théorème 1.4.6 ([73])**.**

$$m^{-3/2}\, D_{m,m-1} \xrightarrow{\mathcal{L}} \int_0^1 e\,(s)\, ds. \tag{1.16}$$

11. Par exemple, $h(x)$ interprète x comme un nombre en base 26, puis le réduit modulo m.

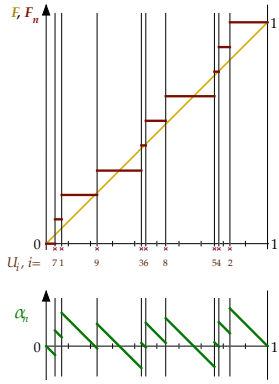


FIGURE 1.6 – *Fonction de répartition empirique et erreur empirique.*

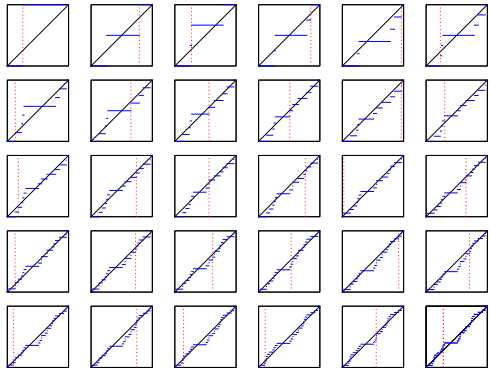


FIGURE 1.7 – *Simulation du théorème fondamental de la statistique.*

Cela suggère une convergence de la structure globale vers une excursion Brownienne, convergence dont (1.16) serait une conséquence. La prochaine section décrit un couplage naturel entre le parking de Knuth et des variables uniformes sur $[0,1]$, qui fait apparaître le lien avec le pont et l'excursion. Cela permet de donner une démonstration non calculatoire du Théorème 1.4.6 et de décrire précisément la transition de phase.

1.4.3 Nombre de visites et fonction de répartition empirique

Dans cette section, on suppose que $m = n+1$, et on se donne un n-uplet $U = (U_1, U_2, \ldots, U_n)$ de variables aléatoires uniformes sur $[0,1]$, de *fonction de répartition empirique* associée définie, pour $0 \leqslant t \leqslant 1$, par :

$$F_n(t) = \frac{1}{n}\sum_{k=1}^{n} 1_{t \geqslant U_k} = \frac{\text{nombre de } U_i \text{ inférieurs à } t}{n}.$$

FIGURE 1.8 – *Parking dans une rue circulaire en sens unique, ou* linear probing. *Cas* $(m, n) = (10, 9)$, *déplacement total égal à 9.*

L'*erreur empirique* α_n (resp. le *processus empirique* $\beta_n(t)$) sont définis, pour $0 \leqslant t \leqslant 1$, par :

$$\alpha_n(t) = F_n(t) - t, \quad \beta_n = \sqrt{n}\,\alpha_n.$$

En 1933, Glivenko et Cantelli ont établi qu'avec probabilité 1, la fonction aléatoire F_n converge uniformément, sur $[0, 1]$, vers $F(t) = t$, résultat qualifié depuis de « théorème fondamental de la statistique ». Des résultats tout aussi centraux pour la statistique mathématique, dûs à Donsker, et à quelques autres, précisent la rapidité de la convergence de α_n, voir section 1.5. Or, à travers un couplage très naturel du problème du parking de Knuth avec la suite U, voir [43, 44], ces théorèmes centraux en statistique entraînent le théorème 1.4.6, et donnent une foule d'informations supplémentaires sur un problème fondamental en analyse d'algorithmes.

Une configuration de parking aléatoire est définie à l'aide de U de la manière suivante : la k-ième voiture (dans l'ordre chronologique) essaye d'abord de se garer sur la place numérotée

$$p_k = \lceil (n+1)\,U_k \rceil\,,$$

puis éventuellement sur les places $p_k + 1$, $p_k + 2$, etc. jusqu'à trouver une place libre. Soit d_ℓ le nombre de voitures dont le premier essai est à la place ℓ. Comme la fonction de répartition empirique F_n d'une part, et d_ℓ d'autre part, comptent le nombre de U_i appartenant à des sous-intervalles de $[0, 1]$, on peut exprimer d_ℓ (et, éventuellement, plus tard, $D_{m,n}$) en fonction de F_n :

$$d_\ell = n\left(F_n\left(\frac{\ell}{n+1}\right) - F_n\left(\frac{\ell-1}{n+1}\right)\right). \tag{1.17}$$

Finalement, soit v_ℓ le nombre de visites à la place ℓ (*i.e.*, le nombre de voitures ayant essayé de se garer en ℓ avec ou sans succès). On voit que :

$$D_{m,n} = -n + \sum_{\ell=1}^{n+1} v_\ell.$$

En effet, une et une seule des visites de chacune des n voitures ne compte pas comme un déplacement, par exemple sa première visite, à la place p_k.

Il sera commode, dans la suite, de prolonger nos fonctions à $\mathbb{Z}$ de manière périodique, via :

$$\forall \ell \in \mathbb{Z}, \qquad d_{\ell+n+1} = d_\ell,\ v_{\ell+n+1} = v_\ell,$$

et, afin que (1.17) reste en vigueur pour $k \in \mathbb{Z}$ et que α_n soit périodique de période 1, il faut convenir que

$$\forall x \in \mathbb{R}, \qquad F_n(x+1) = 1 + F_n(x). \tag{1.18}$$

Remarquons que si on pose :

$$e_m(t) = \frac{1}{\sqrt{m}} v_{\lceil mt \rceil},$$

on obtient que

$$\begin{aligned}\int_0^1 e_m(u)\,du &= m^{-3/2}\left(D_{m,m-1} + m - 1\right)\\ &= m^{-3/2} D_{m,m-1} + \mathcal{O}\left(m^{-1/2}\right),\end{aligned}$$

et une forme de convergence, encore à préciser, de e_m vers e, pourrait fournir une deuxième démonstration, probabiliste, de (1.16). C'est le but de ce qui suit : établir le lien entre (v_ℓ) et (d_ℓ), guère surprenant, donc le lien entre (v_ℓ) et F_n ou α_n, à travers la relation (1.17), puis utiliser successivement le théorème de Donsker (Théorème 1.5.14), qui relie β_n au pont brownien :

$$\beta_n \xrightarrow{\mathcal{L}} b, \tag{1.19}$$

et la transformation de Verwaat, qui relie le pont à l'excursion, pour montrer la convergence de e_m vers e, et en déduire le théorème 1.4.6. Or toutes ces étapes sont des résultats classiques de probabilité ou de statistique, sauf le raisonnement combinatoire élémentaire liant (v_ℓ) à (d_ℓ), qui est donné dans les quelques lignes qui suivent.

Soit L le numéro de la place qui reste libre à la fin, numéro caractérisé par $v_L = d_L = 0$. Tout d'abord, on a

$$v_{\ell+1} = v_\ell + d_{\ell+1} - 1_{\ell \neq L},$$

car parmi les v_ℓ voitures visitant ℓ, s'il en existe, seule une d'entre elles, celle qui se gare en ℓ, ne visite pas $\ell + 1$. Puis, en sommant de $\ell = L$ à

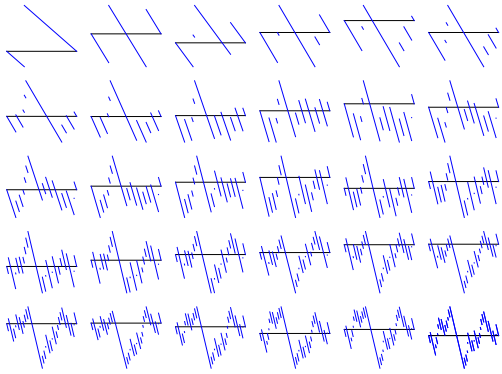


FIGURE 1.9 – *Simulation du théorème de Donsker.*

$\ell = L + k - 1$ et en utilisant 1.18,

$$\begin{aligned} v_{k+L} &= v_L - k + 1 + \sum_{\ell=1}^{k} d_{\ell+L} \\ &= -k + 1 + n\left(F_n\left(\frac{k+L}{n+1}\right) - F_n\left(\frac{L}{n+1}\right)\right) \end{aligned}$$

et :

$$v_{k+L} - \frac{n+1-k}{n+1} = n\left(\alpha_n\left(\frac{k+L}{n+1}\right) - \alpha_n\left(\frac{L}{n+1}\right)\right). \tag{1.20}$$

Or, pour $1 \leqslant k \leqslant n$, $v_{k+L} \geqslant 1$, donc

$$\alpha_n\left(\frac{k+L}{n+1}\right) > \alpha_n\left(\frac{L}{n+1}\right).$$

Ainsi la seule place vide L est caractérisée comme réalisant le minimum de α_n ou de β_n. Par ailleurs l'invariance par rotation du parking suggère fortement que L suit la loi uniforme sur $[1\,..\,m]$, ce qui évoque la transformation de Verwaat (Théorème 1.4.4). Finalement, puisqu'ici $m = n - 1$, on peut réécrire (1.20) comme suit :

$$\begin{aligned} \frac{1}{\sqrt{m}}\, v_{k+L} + \mathcal{O}\left(m^{-1/2}\right) &\simeq \sqrt{n}\left(\alpha_n\left(\frac{k+L}{n+1}\right) - \min \alpha_n\right). \\ &\simeq \beta_n\left(\frac{k+L}{n+1}\right) - \min \beta_n. \end{aligned} \tag{1.21}$$

Au vu du Théorème 1.4.5, de (1.19) et de (1.21), cela termine la "démonstration" de la convergence de e_m vers e, modulo quelques détails techniques peu ardus.

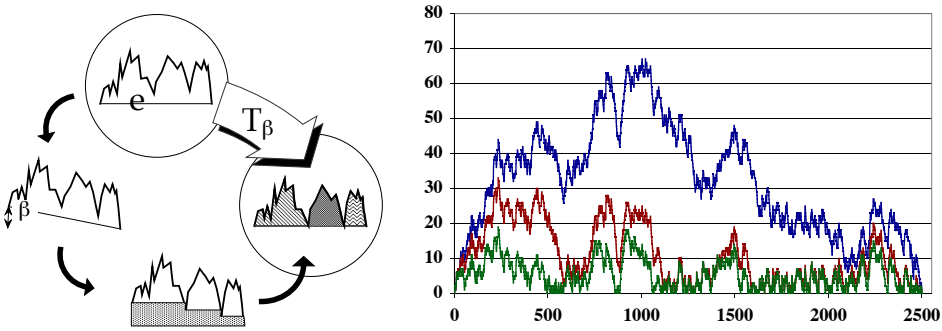


FIGURE *1.10 – Transition de phase : l'effet de T_β sur e, et, à droite, e_m pour différentes valeurs de β.*

Revenons sur la transition de phase révélée par les estimations (1.15) de Knuth : le déplacement d'une voiture est de l'ordre de la moitié de la taille d'un bloc de voitures garées consécutivement, et ces blocs sont d'une largeur maximale $\mathcal{O}(\ln m)$ presque tout du long [153], donc la deuxième estimée de (1.15) devrait être $\mathcal{O}(m \ln m)$. Ce que (1.15) suggère est un mécanisme curieux d'agrégation des blocs, au moment de la saturation du parking : les dernières $\mathcal{O}(\sqrt{m})$ voitures à se garer rencontrent des blocs de taille $\mathcal{O}(m)$. La convergence de e_m vers e, dans l'exercice qui suit, permet de confirmer et de préciser cette intuition.

Exercice 1.4.7 (Transition de phase)**.** *Montrer que pour $m - n \simeq \beta\sqrt{m}$,*

$$e_m \xrightarrow{\mathcal{L}} T_\beta e,$$

où $T_\beta e$ est le processus défini, pour $0 \leqslant t \leqslant 1$, par

$$T_\beta e(t) = e(t) - \beta t - \inf\{e(s) - \beta s \,|\, 0 \leqslant s \leqslant t\}.$$

Vérifier que $\lim_{\beta\to+\infty} T_\beta e(t) = 0$. Conjecturer

$$\lim_{\beta\to+\infty} \lim_m m^{-3/2}\mathbb{E}\left[D_{m,m-\beta\sqrt{m}}\right].$$

Quel est le rapport entre les blocs de places occupées et les zéros de $T_\beta e$? Que dire alors de l'ordre de grandeur de la taille des blocs dans le parking quand le nombre de places vides est $\beta\sqrt{m}$ (voir figure 1.10) ?

1.5 Annexe : préliminaires probabilistes

On rappelle ici quelques propriétés et théorèmes utiles pour ce cours, tous tirés de [103], sauf mention du contraire.

Théorème 1.5.1 (Inégalité de Markov). *Soit X une variable aléatoire positive ou nulle. Pour tout réel strictement positif α,*

$$\mathbb{P}\left(X \geqslant \alpha\right) \leqslant \frac{\mathbb{E}\left[X\right]}{\alpha}.$$

Théorème 1.5.2 (Inégalité de Bienaymé-Tchebychev). *Soit X une variable aléatoire d'espérance et de variance finies. Pour tout réel strictement positif α,*

$$\mathbb{P}\left(|X - \mathbb{E}[X]| \geqslant \alpha\right) \leqslant \frac{Var(X)}{\alpha^2}.$$

Définition 1.5.3 (Convergence en probabilité). *Soit $(X_n)_n$ une suite de variables aléatoires réelles définies sur un même espace de probabilité $(\Omega, \mathcal{A}, P)$. On dit que X_n converge vers X* en probabilité *si*

$$\forall \varepsilon > 0, \qquad \lim_{n\to\infty} \mathbb{P}\left(|X_n - X| \geqslant \varepsilon\right) = 0.$$

On note alors

$$X_n \xrightarrow{P} X.$$

Propriété 1.5.4. *En particulier, si $\sqrt{Var\left(X_n\right)} = o\left(\mathbb{E}\left[X_n\right]\right)$, ou bien, équivalemment, $\mathbb{E}\left[X_n\right]^2 \simeq \mathbb{E}\left[X_n^2\right]$, alors*

$$\frac{X_n}{\mathbb{E}\left[X_n\right]} \xrightarrow{P} 1.$$

Si de plus $\lim_n \mathbb{E}\left[X_n\right] = \ell$,

$$X_n \xrightarrow{P} \ell.$$

Définition 1.5.5 (Convergence dans L^p). *Soit $(X_n)_n$ une suite de variables aléatoires réelles définies sur un même espace de probabilité $(\Omega, \mathcal{A}, P)$. Pour $p \geqslant 1$, on dit que X_n converge vers X* dans L^p *si*

$$\lim_n \mathbb{E}\left[|X_n - X|^p\right] = 0.$$

On note alors

$$X_n \xrightarrow{L^p} X.$$

Définition 1.5.6 (Convergence presque sûre)**.** *Soit $(X_n)_n$ une suite de variables aléatoires réelles définies sur un même espace de probabilité $(\Omega, \mathcal{A}, P)$. On dit que X_n converge* presque sûrement *vers X si*

$$\mathbb{R}\left(\omega \in \Omega \,\middle|\, \lim_n X_n(\omega) = X(\omega)\right) = 1.$$

On note alors

$$X_n \xrightarrow{p.s.} X.$$

Théorème 1.5.7 (Loi forte des grands nombres, Andreï Kolmogorov, 1929)**.** *Si $(X_n)_{n>0}$ est une suite de variables aléatoires i.i.d., on a équivalence entre :*

(i) $\mathbb{E}(|X_1|) < +\infty$,

(ii) la suite $\frac{X_1+\cdots+X_n}{n}$ *converge presque sûrement.*

De plus, si l'une de ces deux conditions est remplie, alors la suite $\frac{X_1+\cdots+X_n}{n}$ *converge presque sûrement vers la constante* $\mathbb{E}(X_1)$.

L'inégalité de Hœffding et le théorème central limite sont deux manières d'estimer la rapidité de convergence dans la loi forte des grands nombres, voir ci-dessous.

Théorème 1.5.8 (Inégalité de Hœffding)**.** *Soit une suite $(X_k)_{1\leqslant k\leqslant n}$ de variables aléatoires réelles indépendantes vérifiant, pour $a < b$,*

$$\forall k, \qquad \mathbb{P}(a \leqslant X_k \leqslant b) = 1.$$

On pose

$$S_n = X_1 + X_2 + \cdots + X_n.$$

Alors, pour tout $t > 0$,

$$\mathbb{P}\left(|S_n - \mathbb{E}[S_n]| \geqslant t\right) \leqslant 2\exp\left(-\tfrac{2t^2}{n(b-a)^2}\right).$$

Définition 1.5.9 (Convergence en loi)**.** *Soit X une variable aléatoire et soit $(X_n)_n$ une suite de variables aléatoires définies sur des espaces de probabilité quelconques, à valeurs dans un même espace métrique (E, d). On dit que X_n converge vers X* en loi *si pour toute fonction continue bornée $\phi : E \to \mathbb{R}$, on a :*

$$\lim_n \mathbb{E}\left[\phi(X_n)\right] = \mathbb{E}\left[\phi(X)\right].$$

On note alors

$$X_n \xrightarrow{\mathcal{L}} X.$$

Théorème 1.5.10 (Théorème central limite, Pierre-Simon de Laplace, 1809)**.** *Si $(X_n)_{n>0}$ est une suite de variables aléatoires i.i.d. d'espérance commune m, de variance commune $\sigma^2 \in]0,+\infty[$, alors*

$$\frac{X_1 + \cdots + X_n - nm}{\sigma\sqrt{n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

Notons que la convergence presque sûre, comme la convergence dans L^p, entraîne la convergence en probabilité. Ces 3 convergences, à leur tour, entraînent la convergence en loi, ce qui est crucial pour obtenir la convergence en loi par couplage. Dans le même esprit, certains couplages permettent d'obtenir des résultats de convergence à travers le théorème suivant :

Théorème 1.5.11 (Théorème de Slutsky)**.** *Si X_n converge en loi vers X, et si Y_n converge en probabilité (dans L^1, dans L^2) vers une constante c, alors le couple (X_n, Y_n) converge en loi vers le couple (X, c). On en déduit en particulier que $X_n + Y_n$ converge en loi vers $X + c$, et que $Y_n X_n$ converge en loi vers cX.*

Pour la convergence vers le processus de Poisson-Dirichlet, nous avons besoin d'un critère de convergence en loi pour les suites aléatoires de nombres réels :

Théorème 1.5.12. *Soit $\xi = (\xi^1, \xi^2, \xi^3, \ldots)$, $\xi_n = (\xi_n^1, \xi_n^2, \xi_n^3, \ldots)$ des suites aléatoires de nombres réels. Alors $\xi_n \xrightarrow{\mathcal{L}} \xi$ si et seulement si, pour tout $k \geqslant 1$,*

$$\left(\xi_n^1, \xi_n^2, \ldots, \xi_n^k\right) \xrightarrow{\mathcal{L}} \left(\xi^1, \xi^2, \ldots, \xi^k\right).$$

Les théorèmes suivants, fondamentaux en statistique mathématique, sont précieux pour la section 1.4.3. Une référence encyclopédique est [175]. On rappelle que b désigne le port brownien, et que :

$$F_n(t) = \frac{1}{n}\sum_{k=1}^{n} 1_{t \geqslant U_k}, \quad \beta_n(t) = \sqrt{n}\,(F_n(t) - t), \quad \alpha_n = \beta_n/\sqrt{n}.$$

Théorème 1.5.13 (Glivenko-Cantelli, 1933)**.** *Avec probabilité 1, F_n converge* uniformément *vers t sur [0,1], ou, de manière équivalente, α_n converge uniformément vers* 0*.*

Ce théorème, appelé « théorème fondamental de la statistique », confirme qu'un échantillon assez grand permet de reconstituer la loi du phénomène étudié, ici la loi uniforme, dont la fonction de répartition est t. Les théorèmes suivants estiment de différentes manières la rapidité de convergence dans le théorème 1.5.13.

Théorème 1.5.14 (Donsker, 1952)**.**

$$\beta_n \xrightarrow{\mathcal{L}} b.$$

Ainsi, dans la convergence de F_n vers F, l'erreur est un $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$, ce que l'inégalité DKW exprime aussi, de manière différente :

Théorème 1.5.15 (Inégalité DKW, Dvoretzky-Kiefer-Wolfowitz, 1956)**.**

$$\Pr\left(\sup_t |\beta_n(t)| \geqslant x\right) \leqslant 2\ \exp(-2x^2).$$

Le dernier théorème nous dit, en quelque sorte, que le terme de troisième ordre du développement asymptotique de F_n est un $\mathcal{O}\left(\frac{\ln n}{n}\right)$:

Théorème 1.5.16 (Komlós, Major & Tusnády, 1975)**.** *Il existe une suite $(b_n)_{n\geqslant 1}$ de ponts browniens telle que :*

$$F_n(t) = F(t) + \frac{b_n(t)}{\sqrt{n}} + \frac{r_n(t)}{n},$$

i.e. $r_n(t) = \sqrt{n}\,(\beta_n(t) - b_n(t))$, avec de plus :

$$\Pr\left(\sup_{0\leqslant t\leqslant 1} |r_n(t)| \geqslant A\log n + x\right) \leqslant Me^{-\mu x}.$$

où A, M et μ sont des constantes universelles.

Remerciements. Merci à Vincent Delecroix pour les figures produites à l'aide de SAGE, à Emmanuel Jeandel, Irène Marcovici et Paul Zimmermann pour leur relecture patiente et méticuleuse, à Brigitte Chauvin pour ses encouragements.

Chapitre 2

Satisfaisabilité Propositionnelle (SAT) et Modulo Théories (SMT)

Sylvain Conchon
Laurent Simon

Le cours est en deux parties, la première étant axée sur les solveurs SAT (satisfaisabilité propositionnelle), la seconde sur les solveurs SMT (satisfaisabilité modulo théories).

Nous présenterons d'abord les éléments clés de ce qui compose les "Solveurs SAT Modernes" pour la résolution du problème de la satisfaisabilité propositionnelle. Après une introduction historique sur la quête de la résolution pratique du problème NP-Complet canonique "SAT", nous expliquerons les caractéristiques algorithmiques et les structures de données des solveurs CDCL (Conflict Driven Clause Learning), les points essentiels devant être intégrés à ce solveurs.

À l'instar des solveurs SAT, les solveurs SMT sont de plus en plus utilisés, dans divers domaines. Bénéficiant des progrès extraordinaires réalisés ces dernières années par les solveurs SAT sur lesquels ils sont basés, les démonstrateurs SMT permettent de décider de la satisfaisabilité de formules logiques contenant des symboles de théories particulières. Nous tenterons dans cette seconde partie de faire découvrir les principaux concepts sous-jacents à la conception et l'implémentation de tels solveurs, plus particulièrement les théories utilisées en pratique et leurs procédures de décision ainsi que les techniques de combinaison de théories.

2.1 Introduction

À la croisée des mathématiques discrètes et de l'informatique théorique se trouve l'un des problèmes les plus importants de l'informatique : le problème SAT, pour satisfaisabilité (ou satisfiabilité). Intuitivement, ce problème capture la difficulté de toute une famille de problèmes *difficiles* que l'on rencontre dans un très grand nombre de problèmes théoriques ou pratiques. Le problème SAT tient une place à part dans cette famille de problèmes difficiles. Cela est tout d'abord lié à l'histoire, puisque SAT a été le premier problème *difficile*, au sens de la théorie de la complexité. Mais ce qui le rend encore aujourd'hui essentiel à un très grand nombre d'applications, ce sont les progrès actuels observés dans sa résolution pratique. Comme nous allons le voir, malgré une impossibilité théorique forte, les solveurs actuels arrivent à résoudre des problèmes de taille gigantesque.

Le problème SAT s'exprime en logique propositionnelle. Extrêmement simple, cette logique colle parfaitement à la force de calcul brute des machines actuelles. Son apparente simplicité permet en effet l'élaboration d'algorithmes compacts et efficaces – en pratique – pour attaquer toute une classe de problèmes importants, mais intraitables – en théorie –. Il est frappant de constater combien cette logique est ancienne, puisqu'elle a été formalisée par Aristote lui-même. C'est donc de manière assez paradoxale que l'on retrouve cette logique au cœur de démonstrateurs automatiques permettant la vérification de microprocesseurs, de programmes, ou encore de problèmes de cryptographie, de théorèmes mathématiques et bioinformatiques de première importance. Cela explique certainement pourquoi Edmund Clarke, l'un des Turing Awards 2007, ait ainsi déclaré « la résolution pratique du problème SAT est une technologie clé pour l'informatique du 21ème siècle » [25].

Dans la section suivante, nous proposons tout d'abord d'introduire le type de problèmes typiquement accessibles à ce formalisme, puis nous présentons l'historique des progrès observés dans la résolution pratique du test de satisfaisabilité (SAT), en nous focalisant sur les techniques actuelles utilisées en pratique.

Le problème SMT, pour satisfaisabilité modulo théories, est une généralisation du problème SAT à des logiques plus riches, étendues avec des symboles de fonction et de prédicat particuliers, dont le sens est fixé par des théories prédéfinies. Ainsi, on peut s'intéresser au problème de la satisfaisabilité modulo la théorie de l'arithmétique linéaire sur les entiers (LIA), la théorie des tableaux (Array) ou la théorie des vecteurs de bits (BV), etc. Plus généralement, la théorie sous-jacente à un problème SMT

est souvent une union de théories plus élémentaires (ainsi, on s'intéressera aux problèmes SMT LIA+BV, ou LIA+Array, etc.).

Nous verrons dans la deuxième partie de ce cours que la résolution d'un problème SMT s'appuie sur la coopération efficace entre un solveur SAT et des petits moteurs de preuve pour les théories, appelés *procédures de décision*. Après avoir présenté deux exemples de théories (la théorie de l'égalité et la théorie des inéquations), nous verrons comment interfacer un solveur SAT avec une procédure de décision. Nous terminerons cette partie par une présentation des techniques pour combiner les procédures de décision de différentes théories.

2.2 Problème de satisfaisabilité en logique propositionnelle

Dans cette section, nous présentons la logique propositionnelle depuis sa définition syntaxique jusqu'à l'algorithmique la plus récente permettant de résoudre en pratique des instances de taille industrielle. Nous présentons également l'intérêt théorique du problème SAT et présentons les avancées de manière chronologique.

2.2.1 Logique propositionnelle et SAT : définitions

Formalisée il y a plus de 2000 ans, cette logique remonte à la naissance du mot « raisonnement » lui-même, le mot « syllogisme ». En Grec ancien, un syllogisme portait trois sens : calcul, hypothèse et raisonnement. Ce *raisonnement-calcul* se caractérise par l'articulation entre des propositions (les *prémisses*) et une *conclusion*, qui déjà, pour Aristote, ne pouvaient prendre comme valeurs que vrai ou faux. Parmi les syllogismes proposés, le *modus ponens* est certainement l'un des plus simples et des plus connus. Il exprime simplement que « *si* a *est vrai* **et** *si* a *implique* b **alors** b *est vrai* ».

Une logique se définit d'abord de manière syntaxique, la sémantique étant définie par la suite sur ce langage. Intuitivement, les machines ne travailleront qu'au niveau syntaxique, manipulant des expressions vides de sens. Ainsi, le langage des formules propositionnelles peut se définir formellement à partir d'un ensemble fini de variables propositionnelles (ou symboles propositionnels) $\mathcal{V}$, de deux constantes *vrai* et *faux* ainsi que d'un ensemble de *connecteurs logiques* : $\neg$ (négation), $\vee$ (disjonction), $\wedge$ (conjonction), $\rightarrow$ (implication), $\leftrightarrow$ (équivalence), $\oplus$ (exclusion). On note $Var(f)$ l'ensemble des variables propositionnelles de $\mathcal{V}$ apparaissant dans la formule f, et on nomme les formules de base ainsi : un **littéral** (ou

x	y	$x \vee y$	$x \wedge y$	$x \rightarrow y$	$x \leftrightarrow y$	$x \oplus y$	$\neg x$
faux	*faux*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
faux	*vrai*	*vrai*	*faux*	*vrai*	*faux*	*vrai*	*vrai*
vrai	*faux*	*vrai*	*faux*	*faux*	*faux*	*vrai*	*faux*
vrai	*vrai*	*vrai*	*vrai*	*vrai*	*vrai*	*faux*	*faux*

TABLE 2.1 – *Valuation sur les fonctions usuelles définies sur les connecteurs* $\vee, \wedge, \rightarrow, \leftrightarrow, \oplus$ *et* $\neg$.

formule atomique) est une formule de la forme x (littéral positif) ou $\neg x$ (littéral négatif) avec $x \in \mathcal{V}$. Les littéraux x et $\neg x$ sont dits **littéraux complémentaires**. Une **clause** est une disjonction finie de littéraux (par exemple $C = x_1 \vee \neg x_2 \vee x_3$ est une clause). Un **produit** est une conjonction finie de littéraux (par exemple $P = x_1 \wedge x_2 \wedge \neg x_3$ est un produit).

La sémantique classique de la logique propositionnelle repose sur la notion *d'interprétation*. On se donne un ensemble $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$, constitué de deux *valeurs de vérité*, et l'on associe chaque variable propositionnelle à une de ces valeurs. Une **interprétation** I d'un ensemble de variables $V \subseteq \mathcal{V}$ est une application ayant pour domaine V et pour co-domaine $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$. Lorsque $V \neq \mathcal{V}$, on dit que l'interprétation est *partielle* (au contraire d'une interprétation *totale*).

Plus simplement, une interprétation totale donne à chaque variable une valeur de vérité unique qui permettra d'évaluer la formule propositionnelle. Souvent, on peut calculer la valeur de la formule sans avoir à donner à chaque variable une valeur (par exemple $x \vee y$ est vrai dès que x est vrai, sans avoir à donner de valeur à y). On parle donc dès lors d'interprétation partielle.

Par exemple, la formule $(x \vee \neg y) \wedge (\neg x \vee \neg z)$ est évaluée à vrai selon l'interprétation partielle $x = \mathtt{V}, z = \mathtt{F}$ et fausse suivant l'interprétation partielle $x = \mathtt{F}, y = \mathtt{V}, z = \mathtt{V}$. Plus formellement, pour évaluer une formule étant donnée une interprétation des variables, on peut récursivement définir l'interprétation de la formule en utilisant la table de vérité classique utilisée sur les connecteurs usuels de la logique (table 2.1).

SAT : à la recherche de modèles

Une fois que la sémantique est fixée, on peut se demander si, étant donnée une formule f, il existe une interprétation I qui la **satisfait** (on aurait $[\![f]\!]_I = \mathtt{V}$). Si c'est le cas, on dit que I est un **modèle** de f, ce que l'on note $I \models f$. Inversement, si $[\![f]\!]_I = \mathtt{F}$, on dit que I **falsifie** f et que I est un

contre-modèle de f, ce qui se note $I \not\models f$. Le problème qui nous intéresse ici est donc de savoir si une formule donnée admet un modèle (problème de **satisfaisabilité**, SAT). Intuitivement, on voit bien que si aucun indice ne nous aide, on pourrait avoir à essayer l'un après l'autre les $2^{Var(f)}$ modèles, ce qui est absolument impossible en pratique (cet ordre de grandeur est uniquement indicatif : calculer la complexité la plus faible d'un algorithme résolvant le problème SAT est cependant assez étudié, voir [201] pour un état de l'art, les résultats donnés ayant un majorant de l'ordre de $1.48^{Var(f)}$). À titre d'exemple, si l'on devait énumérer tous les modèles possibles sur une formule à 260 variables, alors les 2^{260} modèles à tester dépasseraient l'une des estimations communes du nombre de particules de l'univers. Quand on sait, de plus, que les solveurs modernes traitent des formules avec plusieurs millions de variables en quelques minutes, on comprend combien le problème de la résolution pratique de SAT est fascinant. Le nombre de modèles potentiels peut rapidement dépasser l'entendement.

Lorsqu'une formule n'admet aucun modèle, on dit qu'elle est **insatisfaisable** (on dit aussi que f est une **contradiction** ou est **incohérente** ou UNSAT), ce qui est alors noté $\models \neg f$. Au contraire, si toute interprétation sur $Var(f)$ est un modèle de f, alors on dit que f est une **tautologie**, ce qui est noté $\models f$ (on parle aussi de formule **valide**). Par exemple : $(x \rightarrow (y \rightarrow x))$ est une tautologie ; $(x \leftrightarrow (\neg x))$ est une formule insatisfaisable ; et $(x \vee (y \oplus z))$ est une formule satisfaisable (toute interprétation satisfaisant x est un modèle), mais non tautologique (l'interprétation affectant x à **F** et y, z à **V** est un contre-modèle).

La règle de résolution Avant de pouvoir utiliser en pratique une règle syntaxique (adaptée à l'automatisme des machines) permettant de raisonner sur une formule donnée, il faut pouvoir manipuler la formule et l'écrire sous une forme canonique, adapté à la règle de *résolution* que l'on va introduire dans la suite.

Une formule g est une **conséquence sémantique** d'une formule f (noté $f \models g$), si tout modèle de f est un modèle de g. De plus, pour vérifier si g est une conséquence logique de f, il « suffit » de vérifier que la formule $f \wedge (\neg g)$ est bien UNSAT. Lorsque la conséquence est vérifiée dans les deux directions (*i.e.* si $f \models g$ et $g \models f$), alors les deux formules f et g ont exactement les mêmes modèles, et on dit qu'elles sont **équivalentes**. Cette notion permet certaines transformations syntaxiques des formules, dès lors que celles-ci préservent l'équivalence sémantique. On pourra par exemple utiliser l'*idempotence* du connecteur $\vee$ pour réécrire $(f \vee f)$ en f. De nombreuses lois classiques de la logique permettent de réécrire une formule, comme l'élimination de la double négation ($f \equiv \neg(\neg f)$) ou encore

les *lois de De Morgan*. Citons tout de même deux dernières règles à titre d'illustration. La distributivité de $\vee$ sur $\wedge$: $(f \vee (g \wedge h)) \equiv (f \wedge g) \vee (f \wedge h)$; et l'élimination de $\oplus$: $(f \oplus g) \equiv (\neg f \wedge g) \vee (f \wedge \neg g)$.

Les formes normales Le problème SAT, s'il est défini sur n'importe quelles formules en logique propositionnelle, s'exprime généralement sur des formules écrites sous **forme normale conjonctive** ($\mathcal{FNC}$) (conjonction de clauses). Une formule f est dite sous **forme normale disjonctive** ($\mathcal{FND}$) si et seulement si elle correspond à une disjonction de produits. On notera que le problème SAT est trivial sur une formule sous $\mathcal{FND}$.

En profitant des lois usuelles des connecteurs logiques, (comme les *lois de De Morgan*, la distributivité, l'associativité, ...), on peut maintenant écrire toute formule f en une formule équivalente, dans laquelle notamment tous les connecteurs $\rightarrow$, $\leftrightarrow$ et $\oplus$ ont été éliminés, ainsi que les doubles négations. Il ne reste alors plus qu'une formule bâtie sur la disjonction, la conjonction et la négation. Si on prend soin de « pousser » les négations jusqu'aux variables, on obtient une formule sous *forme normale négative* ($\mathcal{FNN}$). On peut en dernier lieu utiliser les propriétés de distributivité des opérateurs pour écrire la formule de départ sous $\mathcal{FNC}$ ou $\mathcal{FND}$, au choix.

Par exemple, si on doit travailler sur la formule $\neg(f \oplus g)$, on va d'abord utiliser la règle vue pour la réécrire en $\neg((\neg f \wedge g) \vee (f \wedge \neg g))$ pour ensuite pousser la négation la plus extérieure jusqu'aux variables : $(f \vee \neg g) \wedge (\neg f \vee g)$. On obtient dès lors une $\mathcal{FNN}$.

Malheureusement en pratique, ces substitutions ne suffisent pas pour transformer n'importe quelle formule sous $\mathcal{FNC}$, car la taille de la formule normalisée peut croître de manière exponentielle si l'on se restreint aux méthodes de réécritures ci-dessus.

Tseitin [193] a introduit un mécanisme permettant de normaliser n'importe quelle formule sous $\mathcal{FNC}$ en préservant non pas l'équivalence, mais l'**équi-satisfiabilité**. Intuitivement, il s'agit d'introduire un ensemble de nouvelles variables dont la valeur de vérité représente la satisfaisabilité, ou non, de sous-formules. Par exemple, lorsqu'il faut encoder un *circuit*, une variable x est associée à chaque *porte* $f(x_1, \ldots, x_n)$ telle que x soit sémantiquement équivalente à la valeur f de la porte : $x \leftrightarrow f(x_1, \ldots, x_n)$. En répétant ce processus au niveau de chaque sous formule élémentaire, la formule globale peut s'écrire ensuite simplement comme la conjonction de tous les encodages, ce qui évite l'explosion due à la distribution des connecteurs logiques les uns sur les autres. Par exemple, si la sous-formule f s'écrit comme la disjonction $g \vee h$, on introduit deux nouvelles variables x_g et x_h représentant la satisfaisabilité des deux sous-formules g et h, respectivement. La satisfaisabilité de f peut alors s'écrire comme

la $\mathcal{FNC}$ suivante $(\neg f \vee x_g \vee x_h) \wedge (f \vee \neg x_g) \wedge (f \vee \neg x_h)$ (ce qui encode l'équivalence logique introduite). En pratique, l'introduction de nouvelles variables est essentielle pour contenir l'explosion combinatoire (syntaxique) que représenterait l'écriture de toute formule sous $\mathcal{FNC}$.

La règle de résolution

La résolution [166] est l'une des règles de déduction de base en logique propositionnelle : soient deux clauses c_1, c_2 et x une variable propositionnelle, la règle de résolution est la règle d'inférence suivante :

$$(x \vee c_1),\ (\neg x \vee c_2) \vdash_{\mathcal{R}}\ c_1 \vee c_2$$

La clause $c_1 \vee c_2$ est appelée **clause résolvante** sur x des clauses $(x \vee c_1)$ et $(\neg x \vee c_2)$. Cette règle est très proche du raisonnement commun. Elle peut être vue comme une application directe de la règle de *coupure* (si on a $f \rightarrow g$ et $g \rightarrow h$, alors on a $f \rightarrow h$), dans le cas particulier où f est une clause, g un littéral et h un produit. Bien entendu, cette simple règle de déduction est correcte : on a bien $(c_1 \vee x) \wedge (c_2 \vee \neg x) \models c_1 \vee c_2$. Elle est aussi complète pour la réfutation : si f est insatisfaisable, alors on a la garantie qu'il existe bien une suite finie de résolutions permettant de déduire la clause vide.

Une preuve par résolution dans Σ d'une clause c' fondamentale est une suite de clauses $P = c_1, \ldots, c_k$ telle que $c_k = c'$ et telle que l'on ait, pour tout $i \leqslant k$: ou bien $c_i \in \Sigma$; ou bien il existe c_m et c_n ($m < i$ et $n < i$) dans P tels que c_i soit la résolvante de c_m et c_n. On note $\Sigma \vdash_{\mathcal{R}} c'$ le fait qu'il existe une telle preuve. Sauf mention contraire, nous notons simplement $\Sigma \vdash c'$ pour $\Sigma \vdash_{\mathcal{R}} c'$.

2.2.2 Difficulté théorique de SAT, et problèmes pratiques

On ne peut parler du problème SAT sans aborder quelques notions de complexité. Il s'agit en effet du premier problème démontré comme NP-complet [51]. Cette notion est primordiale à bien des égards, notamment parce qu'elle permet de classer les problèmes selon une complexité indépendante des algorithmes qui les résolvent en pratique. Pour une introduction à ce sujet, le lecteur intéressé pourra consulter les incontournables [82] et [113].

Pour simplifier, nous introduisons donc la notion de classe de complexité à l'aide de la notion de fonction calculable. Intuitivement, une fonction f est dite *calculable* simplement s'il existe un algorithme qui permet de calculer son résultat (nous imposons juste à ce niveau que le programme finisse un jour son calcul, quel que soit le résultat de la fonction). On peut

alors mesurer le temps de calcul de f par le nombre d'étapes élémentaires de l'algorithme.

La classe P est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial. En terme de fonction calculable, ces problèmes sont caractérisés par la réponse à la question « Que vaut $f(x)$? », avec f calculable en temps polynomial par rapport à la taille de la donnée x. L'ensemble des problèmes appartenant à cette classe sont parfois appelés *faciles*. En théorie, le passage à l'échelle des exemples traités est toujours possible, moyennant des machines plus puissantes, ce qui n'est plus le cas pour les problèmes appartenant aux classes définies ci-dessous (sous l'hypothèse $P \neq NP$).

La classe NP est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial de manière non déterministe. Cette classe correspond à l'ensemble des problèmes de décision s'exprimant logiquement comme $\exists x.f(x)$, ou « existe-t-il un x tel que l'on ait $f(x)$ évalué à vrai ? », avec f une fonction calculable en temps polynomial. Intuitivement, on voit que le problème consiste à *deviner* la bonne solution x (c'est la partie non déterministe). En pratique, les meilleurs algorithmes connus de recherche de ces solutions sont de coût exponentiel dans le pire des cas. Pourtant, rien ne prouve que l'on ne trouvera jamais d'algorithmes polynomiaux pour résoudre des problèmes NP-complets, même si la conjecture $P \neq NP$ est très forte. Quoi qu'il en soit, ces problèmes ont une forte caractéristique pratique : on peut **vérifier une solution du problème en temps polynomial** (il suffit de calculer $f(x)$).

Étant donné un problème de décision, nous pouvons aussi définir son problème complémentaire, comme $\forall x.f(x)$, ou « a-t-on $f(x)$ pour tout x ? », où, encore une fois, f est une fonction calculable en temps polynomial par rapport à la taille de x. La classe $CoNP$ est l'ensemble de ces problèmes, dont le problème complémentaire appartient à NP. De nouveau, une très forte conjecture, non encore prouvée ou infirmée, découle de cette définition : $NP \neq CoNP$.

Enfin, grâce à la notion de complétude, nous avons en main les armes pour classer les problèmes entre eux. Intuitivement, les problèmes NP-Complets sont les plus difficiles des problèmes de NP : si l'on savait en résoudre, ne serait-ce qu'un seul, en temps polynomial alors on saurait les résoudre tous en temps polynomial (la complétude est basée sur la notion de *réduction polynomiale*, non introduite ici par souci de place).

SAT a donc été prouvé comme NP-complet par le fameux théorème de Cook-Levin. Cela le place au centre de la hiérarchie polynomiale, qui classe les problèmes par difficulté croissante.

Intérêt de SAT, d'un point de vue pratique

Attaquer de front le problème SAT est longtemps resté extrêmement difficile en pratique, voire un problème à fuir à tout prix. Pour étudier la performance pratique des solveurs SAT, il a tout d'abord été proposé de générer des problèmes aléatoires, au début des années 1990 [132]. Leur utilisation dans le but de progresser pour la résolution de problèmes réels n'a cependant duré qu'un temps, jusqu'à ce que soit mis à jour certains phénomènes insoupçonnés de ces instances [133]. À la fin des années 1990, en effet, la séparation entre solveurs dédiés aux formules aléatoires et ceux dédiés aux formules industrielles (ou issus du monde réel) était consommée [115, 116].

Puis, en 2001, conjointement avec l'apparition de ce que l'on appelle les « solveurs modernes », est apparue la notion de *Bounded Model Checking* [24, 155] qui permet, en déroulant un certain nombre d'étapes, de transformer en SAT des problèmes de logiques temporelles, ou encore d'atteignabilité d'automates. Lorsque ces automates représentent le fonctionnement d'un microprocesseur, un état particulier, *faute*, est généralement représenté, et l'existence d'un chemin entre l'état initial et la faute représente un exemple de mauvais fonctionnement. Cette longueur de chemin doit être bornée (*Bounded*) pour garantir la transformation en logique propositionnelle. Il est aussi possible de tester l'équivalence de deux circuits (le circuit de référence et le circuit optimisé, effectivement implanté dans un CPU) grâce à SAT, en construisant une formule dont la satisfiabilité implique l'équivalence des deux circuits. On va trouver aussi des moteurs SAT dans l'état de l'art des méthodes de résolution de problèmes importants de bioinformatique [47], de cryptographie ou encore au cœur des méthodes les plus performantes en planification (une plongée dans les nombreux problèmes soumis aux compétitions SAT [114] permet de voir l'étendue des problèmes pouvant être attaqués par SAT). Le nombre de problèmes pour lesquels SAT est devenu essentiel ne cesse de croître. Ainsi, plus récemment, on peut aussi utiliser les SAT solveurs comme des oracles NP de manière efficace, et concevoir des applications demandant des milliers d'appels SAT.

2.2.3 Résoudre SAT en pratique

Depuis plus de vingt-cinq ans, et la première compétition DIMACS en 1993 [64], une grande partie de la communauté SAT se mobilise pour tenter de résoudre, en pratique, ce problème en théorie difficile. Ces progrès théoriques et pratiques, motivés par des compétitions annuelles [114], sont certainement l'une des évolutions majeures de l'intelligence artificielle

(IA) de ces dernières années (Moshe Vardi, Professeur à la Rice University, USA, propose d'ailleurs d'utiliser le terme de « Deep Solving » pour rendre compte de cette évolution majeure).

Dans ce cours, nous nous limiterons aux algorithmes complets permettant de prouver à la fois SAT et UNSAT. Nous omettons ainsi tout un pan de recherche autour des méthodes incomplètes et méthodes de recherches locales, parfois très efficaces en pratique.

DP60 : Davis et Putnam Dans sa première version [56], l'algorithme de Davis et Putnam fonctionnait par « oublis » successifs des variables de la formule. Prenons la décomposition de Shannon, on sait que toute formule f peut se réécrire en $f' \equiv (x \wedge f|_{\{x\}}) \vee (\neg x \wedge f|_{\{\neg x\}})$ pour tout $x \in Var(f)$, la formule $f|_{\{x\}}$ (resp. $f|_{\{\neg x\}}$) étant f dans laquelle toutes les occurrences de x ont été remplacées par $vrai$ (resp. $faux$). La formule obtenue f' n'a plus aucune occurrence de x, mais préserve la satisfaisabilité de f. La difficulté pratique de DP vient de la réécriture de f' sous $\mathcal{FNC}$, celle-ci faisant intervenir la distribution des deux ensembles de clauses obtenus l'un sur l'autre.

Prenons la formule sous $\mathcal{FNC}$ suivante (on utilise ici la notation plus compacte $\overline{x}$ pour $\neg x$) :

$x_1 \vee x_4$
$\overline{x_1} \vee x_4 \vee x_{14}$
$\overline{x_1} \vee \overline{x_3} \vee \overline{x_8}$
$x_1 \vee x_8 \vee x_{12}$
$x_1 \vee x_5 \vee \overline{x_9}$
$x_2 \vee x_{11}$
$\overline{x_3} \vee \overline{x_7} \vee x_{13}$
$\overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$x_8 \vee \overline{x_7} \vee \overline{x_9}$

Si on veut éliminer x_1, on peut factoriser les clauses contenant x_1 et celles contenant $\overline{x_1}$. On obtient :

$$x_1 \vee \begin{pmatrix} x_4 \\ x_8 \vee x_{12} \\ x_5 \vee \overline{x_9} \end{pmatrix}$$
$$\overline{x_1} \vee \begin{pmatrix} x_4 \vee x_{14} \\ \overline{x_3} \vee \overline{x_8} \end{pmatrix}$$
$x_2 \vee x_{11}$
$\overline{x_3} \vee \overline{x_7} \vee x_{13}$
$\overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$x_8 \vee \overline{x_7} \vee \overline{x_9}$

Ce qui se traduit, en utilisant la règle de résolution sur x_1 :

$$\left(\begin{array}{l} x_4 \\ x_8 \vee x_{12} \\ x_5 \vee \overline{x_9} \end{array}\right) \vee \left(\begin{array}{l} x_4 \vee x_{14} \\ \overline{x_3} \vee \overline{x_8} \end{array}\right)$$
$x_2 \vee x_{11}$
$\overline{x_3} \vee \overline{x_7} \vee x_{13}$
$\overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$x_8 \vee \overline{x_7} \vee \overline{x_9}$

La variable x_1 a bien été éliminée, mais la formule n'est plus sous $\mathcal{FNC}$. Il faut maintenant distribuer les deux ensembles l'un sur l'autre pour obtenir la nouvelle formule suivante :

$x_4 \vee x_{14}$
$x_4 \vee \overline{x_3} \vee \overline{x_8}$
$x_8 \vee x_{12} \vee x_4 \vee x_{14}$
$x_5 \vee \overline{x_9} \vee x_4 \vee x_{14}$
$x_5 \vee \overline{x_9} \vee \overline{x_3} \vee \overline{x_8}$
$x_2 \vee x_{11}$
$\overline{x_3} \vee \overline{x_7} \vee x_{13}$
$\overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$x_8 \vee \overline{x_7} \vee \overline{x_9}$

Ici, on a bien une $\mathcal{FNC}$ qui est satisfaisable si et seulement si la formule initiale était satisfaisable. De plus, cette formule ne contient plus x_1.

Pour prouver la satisfaisabilité (ou plutôt la non satisfaisabilité) il suffit d'éliminer toutes les variables de la formule. Si l'on n'obtient pas la clause vide (clause vide de tout littéral, obtenue en faisant par exemple une résolution entre la clause x et la clause $\neg x$), alors la formule est satisfaisable. En général, la succession d'élimination de variable va cependant entraîner une explosion combinatoire du nombre de clauses à gérer, ce qui rend cette approche inutilisable en pratique, sauf cas particuliers. En effet, DP, revisité dans [60, 163, 178], a fait l'objet de beaucoup de travaux, de par ses liens avec des classes polynomiales estimées proches d'instances du monde réel (basée sur la *largeur induite* [195]), mais aussi, plus récemment, pour traiter des problèmes au dessus de SAT, notamment des problèmes de *Compilations de Bases de Connaissances* ou QBF. En effet, si l'on met de côté les clauses produites tout au long du calcul, il est possible, lorsque toutes les variables ont été éliminées, d'énumérer tous les modèles de la formule dans un temps proportionnel à leur nombre [60]. On voit que DP répond de fait à un problème plus difficile que le *simple* problème de satisfaisabilité. À part pour quelques problèmes spécifiques de faible largeur induite [60], DP60 n'a jamais vraiment pu rivaliser avec la version de 1962, basée sur un parcours systématique des modèles potentiels de la formule, avec retours arrières en cas d'échecs. La largeur induite est une mesure structurelle des

Algorithme 1 : Procédure DPLL62.

Initialisation : $\mathcal{F}$ une formule propositionnelle;
Procédure DPLL($\mathcal{F}$)
si *il existe dans* $\mathcal{F}$ *un* ***littéral pur*** ℓ **alors** Renvoyer DPLL($\mathcal{F}_\ell$);
si *il existe dans* $\mathcal{F}$ *une* ***clause unitaire*** ℓ **alors** Renvoyer DPLL($\mathcal{F}_\ell$);
si $\mathcal{F}$ *contient au moins une* ***clause vide*** **alors** Renvoyer faux;
si $\mathcal{F}$ ***est vide*** **alors** Renvoyer vrai;
$v \leftarrow$ une variable de $\mathcal{F}$ **(Heuristique de choix pour la division)**
si DPLL($\mathcal{F}_v$) **alors**
  Renvoyer vrai
sinon
  Renvoyer DPLL($\mathcal{F}_{\neg v}$)

problèmes, permettant de borner l'accroissement de la taille de la formule lors de l'élimination successive des variables. On en retrouvera toutefois une version limitée – mais indispensable aux approches *modernes* –, dans les prétraitements des formules, section 2.2.3.

DPLL62 : l'anticipation doit limiter les retours arrières Plutôt que de réécrire la formule sous $\mathcal{FNC}$ après chaque élimination de variable, et de tenter de lutter contre l'explosion combinatoire en mémoire comme dans DP60, il a été proposé dans DPLL62 [55] de remplacer la disjonction dans f' (voir ci-dessus) par une alternative de choix, une *division* de l'espace de recherche, en vérifiant d'abord la satisfaisabilité de $f|_{\{x\}}$ puis, si le résultat est négatif, en vérifiant la satisfaisabilité de $f|_{\{\neg x\}}$. Cette notation (parfois notée plus simplement f_x quand l'ensemble de littéraux est un singleton) représente la simplification de la formule f par l'interprétation partielle indicée. Cette interprétation partielle est notée par l'ensemble des littéraux vrais (plutôt que la valeur de chaque variable). L'algorithme DPLL s'écrit ainsi simplement sous forme de retours arrières chronologiques, comme illustré par l'algorithme 1. Une liste non exhaustive de quelques solveurs DPLL62 marquants est par exemple SATZ [119], KCNFS [66], SATO [203] et GRASP [177].

On notera que la règle du *littéral pur* (une variable n'apparaissant que positivement ou négativement dans la formule) n'est pas toujours incorporée dans les solveurs, car souvent jugée non rentable, expérimentalement. Par contre, la détection de *clauses unitaires* est un des éléments fondamentaux de tout solveur SAT (y compris pour les solveurs SAT « modernes », dont c'est l'un des éléments clés. Une clause unitaire est une clause de

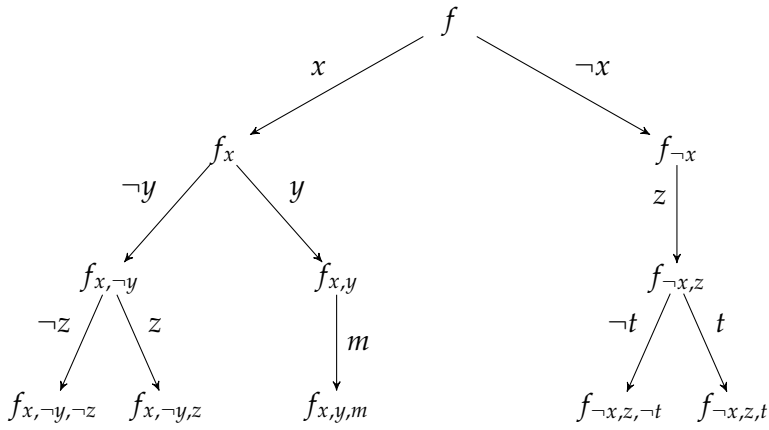


FIGURE 2.1 – *Exemple d'arbre illustrant le parcours arborescent d'une recherche de modèle par un algorithme de type DPLL.*

longueur 1. Cependant, il faut bien noter que cette détection se fait par rapport à l'interprétation partielle courante. Si le solveur a par exemple choisi successivement pour v les littéraux x puis $\neg y$ puis z, une clause initiale $\neg x \vee t \vee y \vee \neg z$ deviendrait soudainement unitaire. Sur une machine récente, et sur un problème industriel typique, on peut mesurer plus d'un million de ces détections par seconde).

Même si l'algorithme est récursif, il faut voir sa trace comme l'exploration d'un arbre dont les feuilles sont des contradictions ou une affectation totale des variables sans contradiction. La figure 2.1 représente la recherche effectuée typiquement par un algorithme de type DPLL. On voit par exemple que le littéral m apparaît dans une clause unitaire lorsque f est simplifiée par l'interprétation partielle $x = \mathtt{V}, y = \mathtt{V}$.

L'autre élément primordial est la **fonction heuristique de choix**, sur laquelle une grande partie de l'efficacité de ces solveurs repose. De nombreuses heuristiques ont été proposées pour tenter de limiter la taille de l'arbre de recherche. L'heuristique la plus connue, appelée MOMS (pour Maximum number of Occurences in Minimum Size clauses) fut introduite dans [55, 84]. Cette heuristique privilégie la variable ayant le plus grand nombre d'occurrences dans les clauses les plus courtes. Dans bien des cas, cependant, elle risque de ne simplifier qu'une seule des deux branches de la recherche. Pour équilibrer les deux sous-arbres, l'heuristique de Jeroslow & Wang [101] estime la variable x à l'aide de la formule $\alpha \times m(x) \times m(\neg x) + m(x) + m(\neg x) + 1$ où $m(x)$ est une mesure de représentativité de x et α une constante (l'heuristique BOHM [39], souvent

utilisée, est un raffinement de MOMS avec cette même idée d'équilibrage des sous-arbres).

Il existe des heuristiques demandant encore plus de calcul. Ainsi, [119] propose de privilégier la variable permettant un grand nombre de propagations unitaires, en cascades (recherchant ainsi le nombre maximum d'*avalanches* [13]), après son affectation, dans les deux sous-arbres. Il s'agit là d'une heuristique à forte anticipation (*lookahead*). Poussée plus loin, [93] a proposé une heuristique de double anticipation qui a donné de bons résultats sur les instances aléatoires.

CDCL : les solveurs « modernes »

En privilégiant le calcul heuristique, les solveurs DPLL, décrits ci-dessus, doivent constamment (après chaque affectation et chaque libération de littéral) mettre à jour de nombreux compteurs : à chaque nœud de l'arbre, on doit en effet être capable d'estimer la présence de tous les littéraux dans la formule simplifiée par l'interprétation partielle courante (des clauses ont été supprimées et des clauses ont été réduites). En cherchant à alléger ces mécanismes, il a été proposé une structure de données permettant une détection paresseuse des clauses unitaires, toujours par rapport à l'affectation courante des variables. Cette structure, appelée *Watched Literals* dans [135], a révolutionné l'application de SAT aux instances industrielles, en autorisant le traitement de problèmes avec plusieurs centaines de milliers de variables. Cette structure de données doit cependant céder une contrepartie de première importance : il n'existe plus aucun moyen de connaître, durant la recherche, le nombre de clauses satisfaites, ou le nombre de clauses binaires (ou même unitaires) relativement à l'affectation courante. Pour pouvoir proposer une heuristique dans ces conditions, les solveurs ont troqué de la visibilité contre de la mémoire. Fonctionnant à l'aveugle, l'intégralité du solveur est passée d'un solveur par anticipation (DPLL62) à un solveur basé sur l'apprentissage, grâce à une analyse précise de son passé, et des conflits rencontrés. Aujourd'hui ces mécanismes sont particulièrement bien cernés [69], et peuvent se résumer à moins de 1000 lignes de code [98] où toutes les techniques utilisées sont profondément interdépendantes. Heuristiques, redémarrages ultra-rapides, sauts arrières non chronologiques, propagations unitaires, gestion des clauses utiles à la suite de la recherche et bien entendu structures de données sont entièrement dédiés à l'apprentissage, et offrent souvent l'image d'un algorithme de plus en plus éloigné de la recherche arborescente binaire classique.

Comme on le voit en lisant la description de l'algorithme 2, le solveur essaye d'atteindre rapidement un conflit (une clause vide), puis apprend

Algorithme 2 : Formulation itérative de DPLL : une formulation CDCL (Conflict-Driven Clause Learning), pleinement tournée vers l'apprentissage.

$\mathcal{I} = \emptyset$, $profondeur = 0$;
tant que *Vrai* **faire**
  Effectuer la Propagation Unitaire (PU) sur $(\Sigma, \mathcal{I})$
  si *un conflit est apparu* **alors**
    si $profondeur = 0$ **alors** Renvoyer UNSAT ;
    C = la clause conflit déduite
    ℓ = l'unique littéral de C affecté à la profondeur du conflit
    $profondeur = max\{profondeur(x) : x \in C/\{l\}\}$
    $\mathcal{I}$ = $\mathcal{I}$ moins tous les littéraux assignés à une profondeur supérieure à $profondeur$
    $(\Sigma, \mathcal{I}) = (\Sigma \cup \{C\}, \mathcal{I}.l)$
  sinon
    si $\mathcal{I}$ *est total* **alors** Renvoyer SAT;
    Choisir un littéral ℓ apparaissant dans $\Sigma|\mathcal{I}$
    $\mathcal{I} = \mathcal{I}.\ell$
    $profondeur = profondeur + 1$

une clause, appelée clause assertive (ou clause FUIP, décrite plus loin), permettant de calculer par ailleurs le niveau de retour arrière nécessaire.

Dans cet algorithme, la Propagation Unitaire est primordiale. Elle consiste à assigner tous les littéraux apparaissant dans des clauses unitaires à vrai. Cette propagation a souvent lieu en cascade, une décision entraînant parfois plusieurs centaines de propagations unitaires (ainsi, décider que x est vrai sur les clauses $\neg x \vee y$, $\neg x \vee \neg y \vee z$ va propager $y = \mathtt{V}, z = \mathtt{V}$). A la fin de la propagation unitaire, soit la clause vide a été trouvée (un conflit est apparu) soit on a la certitude que plus aucune clause unitaire demeure dans la formule simplifiée.

L'heuristique privilégiera les variables vues dans les analyses de conflits récentes (en pratique, toutes les variables vues verront leur score incrémenté d'une valeur, qui elle-même croît exponentiellement après chaque conflit, en suivant une suite géométrique, généralement de raison 1.05). L'analyse de conflit et la notion de FUIP (pour *First Unique Implication Point*) est primordiale (elle a été montrée optimale dans la taille des sauts arrières [14] et dans la qualité des clauses apprises [15]).

Sur l'exemple de la figure 2.2, la base de clause initiale contient les clauses suivantes, devenues unitaires par rapport à l'affectation courante

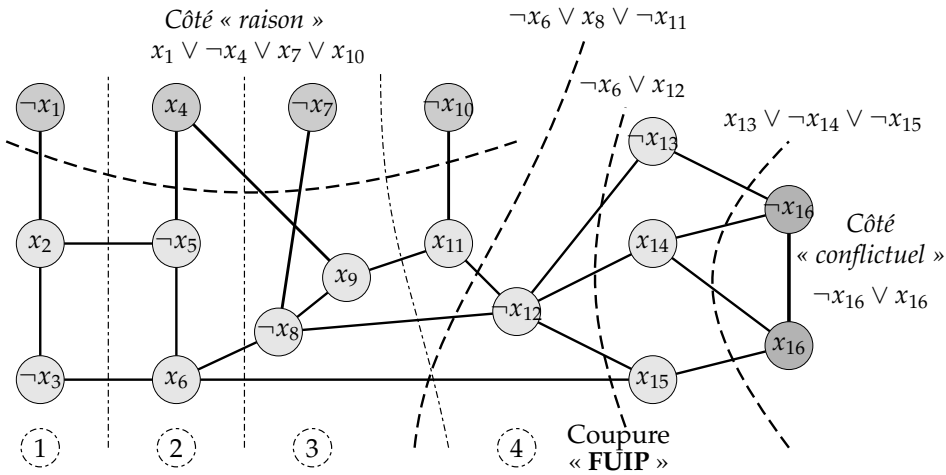


FIGURE 2.2 – *Exemple de graphe d'implication associé aux différents schémas d'apprentissage possibles.*

(entre parenthèses apparaît le niveau de décision auquel elles sont détectées comme étant unitaires) : $x_1 \vee x_2$ et $\neg x_2 \vee \neg x_3$ (niveau 1), $\neg x_4 \vee \neg x_2 \vee \neg x_5$ et $x_5 \vee x_3 \vee x_6$ (niveau 2), $x_7 \vee \neg x_6 \vee \neg x_8$ et $x_8 \vee \neg x_4 \vee x_9$ (niveau 3), ainsi que $x_{10} \vee \neg x_9 \vee x_{11}$, $\neg x_1 \vee x_8 \vee \neg x_{12}$, $x_{12} \vee \neg x_{13}$, $x_{12} \vee x_{14}$, $\neg x_6 \vee x_{12} \vee x_{15}$, $x_{13} \vee \neg x_{14} \vee \neg x_{16}$, et $\neg x_{14} \vee x_{15} \vee x_{16}$ pour le niveau 4. Sur l'exemple donné, on suppose que les différentes variables de division (ou décision) sont, dans l'ordre, les littéraux $\neg x_1$, x_4, $\neg x_7$ et enfin $\neg x_{10}$. Le conflit apparaît lors de la propagation unitaire du quatrième niveau de décision. On représente sur la figure les différentes coupes permettant d'expliquer le conflit, séparant le côté « raison » (les décisions) du côté « conflictuel » (la contradiction). Plusieurs coupures sont possibles, mais une seule contient un littéral FUIP, c'est la *clause assertive* : elle ne contient qu'un seul littéral affecté au dernier niveau de décision (il existe toujours un FUIP, dans la mesure où l'on peut remonter depuis le conflit jusque la dernière variable de décision). On remarquera que dans la clause assertive $\neg x_6 \vee x_{12}$, le troisième niveau de décision n'apparaît plus. Il suffit donc de faire un retour arrière directement au niveau 2, où l'on pourra propager le littéral x_{12}, grâce à la clause assertive nouvellement apprise, devenue unitaire. Le graphe du conflit est un graphe dirigé sans cycle. L'analyse du conflit revient à remonter, en largeur d'abord, depuis le conflit jusqu'à ce que la coupure du graphe calculée ne contienne plus qu'un littéral du dernier niveau de décision. La mise à jour des valeurs heuristiques est effectuée lors de cette phase d'analyse,

en augmentant d'une valeur b toutes les variables vues pendant l'analyse (toutes les variables *à droite* de la coupure). En pratique, la valeur de b est multipliée par une constante, pour se focaliser sur les variables vues dans les derniers conflits (b est en général multiplié par 1.05 à chaque conflit).

De manière théorique, il est intéressant de noter que le parcours en largeur du graphe des conflits revient en fait à effectuer des résolutions entre les clauses responsables de la propagation unitaire des variables du graphe, depuis la clause conflictuelle jusqu'à l'obtention de la clause assertive. Cela a permis à [151] de montrer que les CDCL ont la puissance des systèmes de preuve basés sur la résolution générale. Si, de plus, on ajoute que les solveurs DPLL sont eux basés sur la résolution régulière, on a là des outils théoriques montrant que la puissance des CDCL dépasse celle des DPLL. En effet, il existe dès lors des problèmes difficiles pour les DPLL et faciles pour les CDCL (l'inverse n'est pas vrai).

On ne peut terminer cette courte introduction aux solveurs modernes sans évoquer quelques-uns de leurs composants essentiels, comme les restarts rapides [98, 122] (qui ne sont pas au sens propre des redémarrages, mais plutôt un réordonnancement des dépendances entre variables, puisque les valeurs heuristiques sont conservées : le solveur demeure dans le même espace de recherche), la sauvegarde de phase [150], qui permet de privilégier la dernière polarité des variables lors du branchement, ainsi que la gestion agressive de la base de clauses apprises [15]. Enfin, il faut bien noter, pour conclure sur les solveurs modernes, que beaucoup de choses restent à comprendre dans leurs mécanismes, et que l'on peut encore s'attendre à de nouveaux progrès importants à ce sujet dans les prochaines années. De par leur vélocité et malgré l'apparente simplicité des fonctions heuristiques, les solveurs modernes peuvent s'apparenter à des systèmes complexes, dont le comportement est particulièrement difficile à prévoir et que la communauté cherche, paradoxalement, à mieux comprendre. Ils offrent ainsi d'incroyables performances sur les instances du monde réel, mais comment formaliser les propriétés de telles instances ?

Prétraitement des formules

De manière à combler, en partie, la lacune des solveurs modernes, incapables par exemple de détecter une variable pure, ou des équivalences de littéraux, on leur adjoint quasi systématiquement des techniques de prétraitement. Lors de la compétition SAT 2005, le seul système de prétraitement SATELITE [68] avait ainsi pu résoudre un grand nombre des problèmes industriels proposés. Ces prétraitements sont généralement des applications successives de règles d'oubli de variables (algorithme de DP

limité pour garantir une diminution de la taille de la formule), de résolution hyper-binaire [18], ainsi que de raisonnement sur les clauses bloquées [100]. Même s'ils présentent une *garantie* de rapidité dans le temps maximal passé dans la simplification, leur intégration dans les solveurs CDCL pose problème, dans la mesure où le moindre calcul quadratique dans le nombre de variables serait voué irrémédiablement à l'échec, du fait de la taille parfois gigantesque de certains problèmes. L'élimination des symétries a aussi fait l'objet de nombreux travaux (voir [170], chapitre 6), mais ce n'est que très récemment que les performances dans la détection des symétries ont permis d'espérer embarquer ces procédures dans les solveurs modernes, nécessitant de pouvoir manipuler des instances avec des millions de clauses [104].

Limitations et challenges pour SAT

SAT est un problème qui demande un usage intensif de la mémoire, sujet aux limites des « *memory bound functions* ». Ces fonctions ne permettent pas de suivre les progrès de la loi de Moore sur la rapidité des processeurs, et limitent grandement les possibilités de parallélisation efficace, notamment lors de l'utilisation de plusieurs cœurs. Intuitivement, ces fonctions ont un temps de calcul dominé par le temps passé à lire et/ou écrire en mémoire. Ces algorithmes parcourent de grandes portions de mémoire de manière imprévisible, empêchant tout mécanisme de cache d'être réellement efficace, et limitant la décomposition du problème en plusieurs endroits distincts de la mémoire. Dans ce cadre, et malgré d'importants progrès (voir [170, 91]), la difficulté reste importante et la parallélisation efficace des CDCL représente certainement l'un des plus importants défis pour la communauté SAT toute entière même si de nombreux travaux cherchent à améliorer significativement les performances des SAT solveurs lorsque des milliers de CPUs sont à disposition. Beaucoup reste encore à faire.

2.3 Satisfiabilité Modulo Théories (SMT)

Dans cette partie, nous présentons une extension du problème SAT où les variables booléennes sont remplacées par des contraintes élémentaires (égalités, différences, etc.) entre des termes d'une (union de) théorie(s) du premier ordre (comme l'arithmétique, la théorie de l'égalité, la théorie des tableaux, etc.). Ce problème étendu est appelé SMT (Satisfiabilité Modulo Théories). Pour traiter de telles formules logiques, nous allons devoir modifier l'algorithme CDCL présenté dans la partie précédente afin de le faire coopérer avec des petits moteurs de preuve qui savent prendre en charge ces contraintes.

Dans la suite, nous supposons que le lecteur connaît les notions élémentaires de la logique du premier ordre : syntaxe (signature, termes, formules), modèles (domaines, interprétations), validité logique, théories. Nous rappelons brièvement ces notions dans la section suivante.

2.3.1 Logique du premier ordre : définitions et notations

Nous rappelons ici les définitions et notions élémentaires de la logique du premier ordre.

Signatures et termes. Une *signature* Σ est un ensemble fini de symboles de fonctions et de prédicats. Chaque symbole a une *arité* qui représente le nombre d'arguments auxquels il doit être appliqué. Les *constantes* sont des symboles de fonction d'arité 0. Toute signature Σ est supposée contenir le symbole $=$ de prédicat d'égalité. Traditionnellement, on utilise des lettres en minuscules f, g, h, etc. pour désigner des symboles de fonction, et des lettres en majuscules $P, Q, R, \ldots$ pour les symboles de prédicats.

Soit $\mathcal{V}$ un ensemble de *variables*, distinctes des symboles de Σ. On note $T(\Sigma, \mathcal{V})$ l'ensemble des *termes* associés à Σ, *i.e.* le plus petit ensemble contenant $\mathcal{V}$ et tel que $f(t_1, \ldots, t_n) \in T(\Sigma, \mathcal{V})$ si $t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$ et $f \in \Sigma$. L'ensemble $T(\Sigma, \emptyset)$ est celui des *termes sans variable* (en anglais, *ground terms*).

Formules. Une *formule atomique* est de la forme $P(t_1, \ldots, t_n)$, où $t_1, \ldots, t_n$ sont des termes de $T(\Sigma, \mathcal{V})$ et P est un symbole de prédicat de Σ. Les *littéraux* sont des formules atomiques (ou leur négation). Les *formules* sont construites inductivement à partir des formules atomiques et de connecteurs booléens ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, etc.) ainsi que des quantificateurs $\forall$ et $\exists$. Une *formule sans variable* (en anglais *ground formula*) ne contient que des *termes*

sans variable. Une variable est *libre* dans une formule si elle n'est liée par aucun quantificateur. Une *formule close* (en anglais *sentence*) est une formule sans variable libre.

Modèles. Un *modèle* $\mathcal{M}$ pour une signature Σ et un ensemble de variables $\mathcal{V}$ est défini par : (1) un domaine $\mathcal{D}_\mathcal{M}$, (2) une interprétation $f^\mathcal{M}$ pour chaque symbole de fonction $f \in \Sigma$, (3) un sous-ensemble $P^\mathcal{M}$ de $\mathcal{D}_\mathcal{M}^n$ pour chaque prédicat $P \in \Sigma$ d'arité n, et enfin (4) un dictionnaire, traditionnellement nommé $\mathcal{M}$ comme le modèle, qui associe à chaque variable $x \in \mathcal{V}$ une valeur $\mathcal{M}(x)$ du domaine $\mathcal{D}_\mathcal{M}$. La *cardinalité* d'un modèle $\mathcal{M}$ est la cardinalité de $\mathcal{D}_\mathcal{M}$.

Sémantique. Étant donnés une signature Σ, un ensemble de variables $\mathcal{V}$ et un modèle $\mathcal{M}$ pour Σ et $\mathcal{V}$, on définit la sémantique des termes par les deux équations suivantes :

$$\begin{array}{lcl} \mathcal{M}[x] & = & \mathcal{M}(x) \\ \mathcal{M}[f(t_1, \ldots, t_n)] & = & f^\mathcal{M}(\mathcal{M}[t_1], \ldots, \mathcal{M}[t_n]) \end{array}$$

Étant donnée une formule ϕ dont les termes sont dans $T(\Sigma, \mathcal{V})$, on définit la valeur de vérité de ϕ par rapport à $\mathcal{M}$ à l'aide d'une relation binaire $\models$ définie de la manière suivante :

$$\begin{array}{lcl} \mathcal{M} \models t_1 = t_2 & = & \mathcal{M}[t_1] = \mathcal{M}[t_2] \\ \mathcal{M} \models P(t_1, \ldots, t_n) & = & (\mathcal{M}[t_1], \ldots, \mathcal{M}[t_n]) \in P^\mathcal{M} \\ \mathcal{M} \models \neg\Phi & = & \mathcal{M} \not\models \Phi \\ \mathcal{M} \models \Phi_1 \wedge \Phi_2 & = & \mathcal{M} \models \Phi_1 \text{ et } \mathcal{M} \models \Phi_2 \\ \mathcal{M} \models \Phi_1 \vee \Phi_2 & = & \mathcal{M} \models \Phi_1 \text{ ou } \mathcal{M} \models \Phi_2 \\ \mathcal{M} \models \forall x.\Phi & = & \mathcal{M}\{x \mapsto v\} \models \Phi \text{ pour tout } v \in \mathcal{D}_\mathcal{M} \\ \mathcal{M} \models \exists x.\Phi & = & \mathcal{M}\{x \mapsto v\} \models \Phi \text{ pour un certain } v \in \mathcal{D}_\mathcal{M} \end{array}$$

On dira qu'une formule Φ est *satisfiable* s'il existe un modèle $\mathcal{M}$ tel que $\mathcal{M} \models \Phi$, sinon Φ est *insatisfiable*. Deux formules Φ_1 et Φ_2 sont *equisatisfiables* si Φ_1 est satisfiable quand Φ_2 est satisfiable, et réciproquement. Une formule Φ est *valide* si $\neg\Phi$ est *insatisfiable*.

Théories. Une *théorie du premier ordre* T sur une signature Σ est un ensemble de formules closes. Une théorie est *cohérente* (en anglais *consistent*) si elle a (au moins) un modèle.

2.3.2 De petits moteurs de preuve

La technique SMT repose sur des *petits moteurs* de preuve qu'on appelle *procédures de décision*. Ces briques logicielles sont des (semi-)algorithmes pour décider la satisfiabilité de formules pour des théories élémentaires comme l'égalité entre termes non-interprétés, l'arithmétique linéaire sur les entiers ou les rationnels, l'arithmétique de Presburger, des structures de données (listes, tableaux, vecteurs de bits), etc.

Voici des exemples de formules pour trois théories : (1) la théorie de l'égalité avec un symbole non-interprété f, (2) la théorie de l'arithmétique linéaire (sur les rationnels) et (3) la théorie des tableaux.

$$(1) \quad f(f(f(x))) = x \wedge f(f(f(f(f(x))))) = x \wedge f(x) \neq x$$

$$(2) \quad x + y = 19 \wedge x - y = 7 \wedge x \neq 13$$

$$(3) \quad a[i \leftarrow x] = b \wedge a = b \wedge b[i] = y \wedge b[i \leftarrow x][j] = y \wedge i = j$$

En pratique, les formules intéressantes à prouver sont souvent formées d'un mélange de symboles appartenant à plusieurs théories. Les exemples suivants mélangent (1) la théorie des tableaux avec celle de l'arithmétique linéaire sur les entiers et (2) les mêmes théories plus celle de l'égalité avec un symbole non interprété f.

$$(1) \quad v[i \leftarrow v[j]][i] \neq v[i] \wedge i + j \leqslant 2j \wedge j + 4i \leqslant 5i$$

$$(2) \quad x + 2 = y \wedge f(a[x \leftarrow 3][y - 2]) \neq f(y - x + 1)$$

Comme on le voit dans ces exemples, ces procédures de décision sont utilisées pour décider la satisfiabilité de *conjonctions* de contraintes (ou prédicats) élémentaires (égalités, inéquations, relations d'ordre, etc.). Nous allons voir dans les deux sous-sections suivantes des procédures de décision pour deux théories : la théorie de l'égalité avec symboles non interprétés et la théorie des inéquations. Nous verrons ensuite des algorithmes pour *combiner* ces petits moteurs de preuve afin de décider des conjonctions de contraintes mélangeant plusieurs théories.

Incrémentalité, backtrackabilité, explications et implications. Comme nous le verrons dans la section 2.3.3, les procédures de décision utilisées par un démonstrateur SMT doivent posséder quatre propriétés : (1) l'*incrémentalité*, (2) la *backtrackabilité*, (3) la *production d'explications* et (4) la détection de *contraintes impliquées*. Les deux premières ne sont pas indispensables, mais elles améliorent l'efficacité du démonstrateur.

1. **Incrémentalité**. La procédure de décision doit pouvoir être appelée successivement sur une suite d'ensembles de contraintes $\mathcal{C}_0 \subset \mathcal{C}_1 \subset \ldots \subset \mathcal{C}_k$ de telle sorte que le traitement d'un ensemble $\mathcal{C}_i$ ne nécessite pas de tout refaire depuis le début, mais réutilise le traitement effectué pour $\mathcal{C}_{i-1}$.

2. **Backtrackabilité.** Les structures de données utilisées par la procédure de décision doivent permettre de revenir efficacement à un état antérieur de l'algorithme sans avoir à tout ré-initialiser.

3. **Explications.** Quand l'ensemble $\mathcal{C}$ de contraintes est insatisfiable, la procédure de décision doit produire un sous-ensemble $\mathcal{U}$ de $\mathcal{C}$ incohérent. Cet ensemble $\mathcal{U}$, appelé en anglais *unsat core*, représente la raison pour laquelle $\mathcal{C}$ est insatisfiable. $\mathcal{U}$ doit être le plus petit possible, c'est-à-dire qu'il doit contenir le moins possible de contraintes redondantes ou inutiles. Par exemple, si $\mathcal{C} = \{x < y, y < z, z < 10, x < z, x > 15, a > b\}$, l'ensemble $\mathcal{U}$ devrait seulement être égal à $\{z < 10, x < z, x > 15\}$.

4. **Contraintes impliquées.** Les procédures de décision doivent enfin permettre de détecter de nouveaux faits (ou contraintes) impliqués par un ensemble de contraintes $\mathcal{C}$. En particulier, comme nous le verrons en section 2.3.3, les procédures devront pouvoir détecter des égalités entre variables.

Théorie de l'égalité avec symboles non interprétés

Cette théorie, appelée également *théorie libre de l'égalité*, donne un sens au prédicat d'égalité $=$ en présence de symboles de fonctions d'arité quelconque. Le sens de chaque symbole de fonction n'est pas défini. Dit autrement, il s'agit là de la théorie *syntaxique* de l'égalité.

Les termes de cette théorie appartiennent à deux catégories syntaxiques :

(1) les variables x, y, z, etc.

(2) les applications de fonctions $f(x), g(x, y)$, etc.

Les contraintes élémentaires sont soit des égalités ou des différences entre les termes comme par exemple $x = f(y, z)$, $g(x) \neq h(y)$ ou $x = y$.

La théorie est définie à partir de trois axiomes et d'un schéma d'axiomes.

(Réflexivité) $\forall x.\ x = x$

(Symétrie) $\forall xy.\ x = y \Rightarrow y = x$

(Transitivité) $\forall xyz.\ x = y \wedge y = z \Rightarrow x = z$

(Congruence) Pour tout symbole de fonction f d'arité n

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.$$
$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Les trois premiers axiomes expriment que le prédicat d'égalité est une relation d'équivalence (relation réflexive, symétrique et transitive). Le schéma d'axiomes ajoute le fait que c'est une congruence, c'est-à-dire que pour tout symbole de fonction f, l'égalité se propage aux applications de f pour des arguments deux à deux égaux.

Procédure de décision. La procédure de décision pour cette théorie est appelée *fermeture par congruence*.

Étant donnée une conjonction de contraintes élémentaires $\mathcal{C}$ de la forme $\bigwedge_i t_i \bowtie u_i$, où $\bowtie$ est une contrainte d'égalité $=$ ou de différence $\neq$, l'algorithme consiste tout d'abord à séparer l'ensemble $\mathcal{C}$ en deux ensembles $\mathcal{E}$ et $\mathcal{D}$, contenant respectivement les contraintes d'égalités et de différences.

$$\overbrace{\bigwedge_i t_i = u_i}^{\mathcal{E}} \quad \wedge \quad \overbrace{\bigwedge_j t_j \neq u_j}^{\mathcal{D}}$$

Ensuite, un graphe dirigé acyclique (DAG) est construit pour représenter tous les termes (et sous-termes) apparaissant dans l'ensemble $\mathcal{E} \cup \mathcal{D}$. La boucle principale de la fermeture par congruence est donnée par l'algorithme 3. Elle consiste à maintenir une structure *union-find* pour manipuler des classes d'équivalence entre les nœuds du graphe : deux nœuds n et m étant dans la même classe quand les termes t_1 et u_1 qu'ils représentent respectivement sont égaux soit (1) parce que $t_1 = u_1 \in \mathcal{E}$, soit (2) parce que l'égalité $t_1 = u_1$ est une conséquence de $\mathcal{E}$ et des axiomes de la théorie de l'égalité.

On rappelle qu'une structure *union-find* permet de maintenir une partition d'un ensemble fini à l'aide de deux fonctions :

- `find(`t`)` : renvoie le représentant d'un élément `t`
- `union(`t_1`,`t_2`)` : fusionne les classes d'équivalences de t_1 et t_2

Algorithme 3 : Fermeture par congruence.

```
1 for every nodes n, m ∈ G labeled with the same symbol do
2     if find(n) ≠ find(m) and
3        find (n_i) = find (m_i) for every children of n and m then
4         merge the classes of n and m by union (n,m)
```

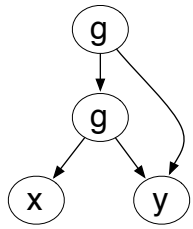
La figure 2.3 illustre le fonctionnement de cet algorithme pour décider la satisfiabilité de la conjonction $g(x,y) = x \wedge g(g(x,y),y) \neq x$. Le DAG construit initialement avec tous les termes et sous-termes du problème est donné en (1).

La structure *union-find* est ensuite initialisée (schéma (2)) avec les nœuds du graphe en mettant dans la même classe les nœuds représentant des termes égaux selon $\mathcal{E}$. Les classes d'équivalences sont matérialisées par des traits en pointillés. Ainsi, le nœud central étiqueté par g (représentant le terme $g(x,y)$) est relié au nœud x car $g(x,y) = x \in \mathcal{E}$.

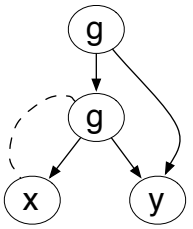
L'algorithme cherche alors à appliquer l'axiome de congruence. Pour cela, il recherche deux nœuds n et m non encore égaux (donc non reliés par des traits en pointillés) et étiquetés par le même symbole de fonction dont les fils sont égaux deux à deux. Les deux nœuds étiquetés par le symbole `g` (encadrés en rouge dans le schéma (3)) remplissent ces critères et leurs classes d'équivalences sont fusionnées. Cette union de deux classes implique, par transitivité, que les nœuds représentant les termes $g(g(x,y),x)$ et x sont maintenant dans la même classe (schéma (4)).

Une fois la boucle principale terminée, l'algorithme se termine en s'assurant que toutes les différences $t_k \neq t_l \in \mathcal{D}$ sont effectivement vraies dans le graphe, *i.e.* que les nœuds représentant respectivement les termes t_k et t_l ne sont pas reliés par un trait en pointillé. Dans notre exemple, la différence $g(g(x,y),x) \neq x$ n'est pas satisfaite puisque les nœuds représentant ces deux termes sont dans la même classe d'équivalence. On en déduit donc que cette formule n'est pas satisfiable.

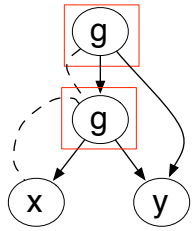
Exercice 1 En utilisant la procédure de décision de *fermeture par congruence*, déterminer la satisfiabilité de la formule suivante : $f(f(f(x))) = x \wedge f(f(f(f(f(x))))) = x \wedge f(x) \neq x$.



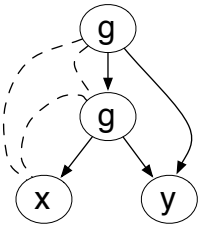
(1) DAG initial



(2) $g(x,y) = x$



(3) $g(x,y) = x \wedge y = y \Rightarrow g(g(x,y),y) = g(x,y)$



(4) $g(g(x,y),y) = g(x,y) \wedge g(x,y) = x \Rightarrow g(g(x,y),y) = x$

FIGURE 2.3 – *Exemple de fermeture par congruence.*

Incrémentalité, backtrackabilité, explications et implications. Une procédure incrémentale avec génération d'explications est présentée dans l'article de Nieuwenhuis et Oliveras *Proof-producing Congruence Closure* [139].

Théorie des inéquations

Cette théorie, appelée en anglais *Difference Logic*, est un fragment de l'arithmétique linéaire (sur les entiers ou les rationnels) où les contraintes élémentaires sont restreintes à la forme $x - y \leqslant c$, avec x et y deux variables et c une constante numérique. Bien que restrictive, d'autres contraintes peuvent être encodées dans ce fragment. Une solution à un ensemble de telles contraintes est représentée par un dictionnaire σ qui à chaque variable x associe un entier ou un rationnel $\sigma(x)$.

(1) Encodage des contraintes strictes :
- sur $\mathbb{Z}$, $x - y < c$ est remplacée par $x - y \leqslant c - 1$
- sur $\mathbb{Q}$, $x - y < c$ est remplacée par $x - y \leqslant c - \delta$, où δ est une constante suffisamment petite (en pratique, δ est simplement une constante *symbolique*)

(2) Encodage des contraintes de la forme $x \leqslant c$:
Pour cela, on remarque que pour toute solution σ d'un ensemble de contraintes, on peut construire une nouvelle solution σ' en décalant $\sigma(x)$ par une même constante k, pour tout x.
Ainsi, $x \leqslant c$ est encodée par $x - y_{zero} \leqslant c$, où y_{zero} est une nouvelle variable, et pour toute solution σ, on construit une solution σ' telle que $\sigma'(y_{zero}) = 0$, *i.e.* on décale de $k = -\sigma(y_{zero})$.

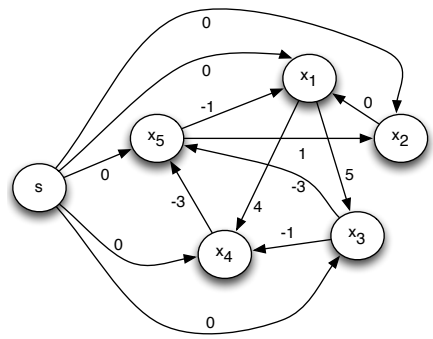
Procédure de décision. Étant donné un ensemble de variables $\{x_1, \ldots, x_n\}$ et une conjonction $\mathcal{C}$ de contraintes élémentaires de la forme $\bigwedge_i x_i - y_i \leqslant c_i$, la procédure de décision pour cette théorie consiste à construire un graphe pondéré $\mathcal{G}(V, E)$ tel que :

- $V = \{s, x_1, \ldots, x_n\}$, chaque nœud correspond à une variable du problème plus une nouvelle variable s
- $E = \{y_i \xrightarrow{c_i} x_i \mid x_i - y_i \leqslant c_i \in \mathcal{C}\} \cup \{s \xrightarrow{0} x_i \mid 1 \leqslant i \leqslant n\}$, chaque arête correspond à une contrainte du problème plus une arête de poids nul entre s et chaque variable du problème.

L'exemple de la figure 2.4 illustre la construction du graphe $\mathcal{G}(V, E)$ pour l'ensemble $\mathcal{C}$ de contraintes donné à gauche.

$$
\begin{aligned}
x_1 - x_2 &\leqslant 0 \\
x_1 - x_5 &\leqslant -1 \\
x_2 - x_5 &\leqslant 1 \\
x_3 - x_1 &\leqslant 5 \\
x_4 - x_1 &\leqslant 4 \\
x_4 - x_3 &\leqslant -1 \\
x_5 - x_3 &\leqslant -3 \\
x_5 - x_4 &\leqslant -3
\end{aligned}
$$

(1) Ensemble $\mathcal{C}$ d'inéquations



(2) Graphe $\mathcal{G}(V, E)$

FIGURE 2.4 – *Initialisation du graphe des inéquations.*

Dans la suite, on appelle *chemin* entre deux nœuds a et b, une séquence d'arêtes de la forme $a \xrightarrow{c_1} v_1 \xrightarrow{c_2} \cdots \xrightarrow{c_{n-1}} v_{n-1} \xrightarrow{c_n} b$ et le *poids* de ce chemin est la somme $c_1 + \cdots + c_n$. Parmi tous les chemins entre a et b, on distingue le *plus court chemin*, *i.e.* le chemin ayant le poids le plus petit.

La procédure de décision pour cette théorie repose sur le théorème suivant.

Théorème 1. *Étant donnée une conjonction d'inéquations $\mathcal{C}$ de la forme*

$$\bigwedge_i x_i - y_i \leqslant c_i$$

et $\mathcal{G}(V, E)$ le graphe correspondant :

1. *Si $\mathcal{G}$ a un cycle* négatif, *i.e. un chemin de la forme $v_1 \xrightarrow{c_1} v_2 \xrightarrow{c_2} \ldots \xrightarrow{c_{n-1}} v_n \xrightarrow{c_n} v_1$ tel que $c_1 + c_2 + \cdots + c_{n-1} + c_n < 0$, alors $\mathcal{C}$ est insatisfiable.*
2. *Sinon, une solution est $x_1 = \delta(s, x_1), \ldots, x_n = \delta(s, x_n)$, où $\delta(s, x_i)$ est le* chemin le plus court *entre s et x_i.*

Preuve. (1) Tout cycle négatif $v_1 \xrightarrow{c_1} v_2 \xrightarrow{c_2} \ldots \xrightarrow{c_{n-1}} v_n \xrightarrow{c_n} v_1$ correspond à un ensemble de contraintes :

$$\begin{aligned} v_2 - v_1 &\leqslant c_1 \\ v_3 - v_2 &\leqslant c_2 \\ &\ldots \\ v_1 - v_n &\leqslant c_n \end{aligned}$$

Si on additionne ces inéquations, on obtient que $0 \leqslant c_1 + c_2 + \cdots + c_n$ ce qui contredit l'hypothèse du cycle négatif $c_1 + c_2 + \cdots + c_n < 0$.
(2) Si $\mathcal{G}(V, E)$ n'a pas de cycle négatif alors pour toute arête $y_i \xrightarrow{c} x_i$ on a $\delta(s, x_i) \leqslant \delta(s, y_i) + c$, ou de manière équivalente $\delta(s, x_i) - \delta(s, y_i) \leqslant c$. Ainsi, en posant $x_i = \delta(s, x_i)$ et $y_i = \delta(s, y_i)$ on satisfait la contrainte $x_i - y_i \leqslant c$.

□

La détection de cycles négatifs peut être réalisée à l'aide d'un algorithme de plus court chemin. Un algorithme bien connu est celui de Bellman-Ford (voir l'algorithme 4) qui calcule les plus courts chemins à partir d'un sommet source dans un graphe orienté pondéré.

Cet algorithme maintient une *borne supérieure* $d[x]$, pour chaque nœud x, qui représente le poids du plus petit chemin de s à x. Le cœur de l'algorithme consiste à mettre à jour $d[x]$ en utilisant une technique de *relaxation* qui, pour chaque arête $y \xrightarrow{c} x$, teste si on peut améliorer le plus court chemin $d[x]$ vers x trouvé jusque-là en passant par y. Les chemins sont eux stockés dans un tableau π tel que $\pi[x]$ contient le prédécesseur de x. La figure 2.5 donne le résultat de cet algorithme sur l'exemple précédent.

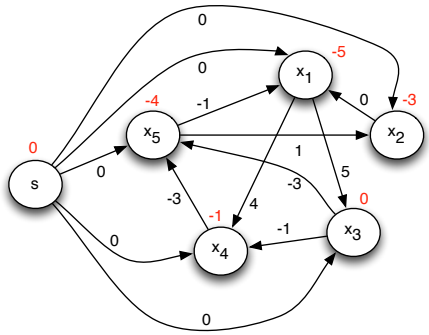
Preuve de correction. Supposons que $\mathcal{G}(V, E)$ contienne un cycle négatif $v_0 \xrightarrow{c_0} v_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{k-1}} v_k$ avec $v_0 = v_k$. Supposons que l'algorithme de

Algorithme 4 : Algorithme de Bellman-Ford.

```
1  for each x_i ∈ V do
2      d[x_i] := ∞;
3  d[s] ← 0;
4  for i = 1 to |V| − 1 do
5      for each y_i --c--> x_i ∈ E do
6          if d[x_i] > d[y_i] + c then
7              d[x_i] ← d[y_i] + c;
8              π[x_i] ← y_i
9  for each y_i --c--> x_i ∈ E do
10     if d[x_i] > d[y_i] + c then
11         return Negative Cycle Detected (use π to reconstruct the cycle)
```

Bellman-Ford ne trouve pas ce cycle. Ainsi, $d[v_i] \leqslant d[v_{i-1}] + c_{i-1}$ pour tout $i = 1, 2, \ldots, k$. Par sommation de ces inéquations, on obtient $\sum_{i=1}^{k} d[v_i] \leqslant \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} c_{i-1}$, c'est-à-dire $\sum_{i=1}^{k} d[v_i] - \sum_{i=1}^{k} d[v_{i-1}] \leqslant \sum_{i=1}^{k} c_{i-1}$. Mais, puisque $v_0 = v_k$, on a $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$. Donc, $0 \leqslant \sum_{i=1}^{k} c_{i-1}$, ce qui est impossible puisque ce cycle est supposé être négatif.

□



$$x_1 = -5$$
$$x_2 = -3$$
$$x_3 = 0$$
$$x_4 = -1$$
$$x_5 = -4$$

FIGURE 2.5 – *Plus courts chemins et solution de l'ensemble de contraintes de la figure 2.4.*

Exercice 2 En utilisant l'algorithme de Bellman-Ford, déterminer la satisfiabilité de $x \leqslant 1, x - y \leqslant 2, y - z \leqslant 3, z - x \leqslant -6$.

Incrémentalité, backtrackabilité, explications et implications. Ces propriétés pour cette procédure de décision sont discutées et étudiées dans l'article de Nieuwenhuis et Oliveras *DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic* [138].

2.3.3 Interfacer SAT et procédures de décision

Dans cette section, on s'intéresse au problème SMT en se restreignant à des formules d'une *unique* théorie `T` disposant d'une procédure de décision `Tsolve`. Nous verrons dans la section suivante comment traiter le problème plus général avec plusieurs théories.

Combinaison hors ligne

Étant donnée une formule ϕ, la technique la plus simple pour résoudre le problème SMT est de convertir ϕ en une forme normale disjonctive (DNF) $\bigvee_{i \in I} \mathcal{C}_i$ puis d'appeler $\texttt{Tsolve}(\mathcal{C}_i)$ sur chaque conjonction de contraintes $\mathcal{C}_i$, jusqu'à ce qu'une de ces contraintes soit `T`-satisfiable.

Plutôt que de faire cette conversion en DNF, nous allons utiliser un solveur SAT pour traiter efficacement la structure booléenne de ϕ. L'algorithme 5 implémente un solveur SMT dit *hors ligne* (*offline* en anglais).

Algorithme 5 : Algorithme SMT hors ligne.

```
1  function SmtOffline(φ) begin
2      f ← T2B(φ);
3      while True do
4          (res, M) ← CDCL(f);
5          if res = UNSAT then
6              return UNSAT
7          else
8              (res, uc) ← Tsolve(B2T(M));
9              if res = SAT then
10                 return SAT
11             else
12                 f ← f ∧ ¬T2B(uc)
```

L'algorithme commence par calculer une abstraction booléenne de ϕ (supposée être en forme normale conjonctive – CNF) à l'aide de la fonction `T2B`. Cette transformation consiste simplement à remplacer chaque littéral

de ϕ par une variable booléenne fraîche (une implémentation efficace de `T2B` cherchera à réaliser un partage maximal de ces variables). La formule f ainsi obtenue est alors passée à un solveur SAT `CDCL`. L'appel à $\texttt{CDCL}(f)$ renvoie un couple (res, M) où res indique si f est satisfiable et, si c'est le cas, M contient un modèle booléen pour cette formule. Ce modèle est ensuite passé à la procédure de décision `Tsolve` afin de s'assurer qu'il est cohérent modulo la théorie `T`. Pour cela, il faut d'abord retrouver les littéraux de départ à l'aide d'une fonction `B2T`. L'appel à $\texttt{Tsolve}(\texttt{B2T}(M))$ renvoie un couple (res, uc) où res indique si le modèle est satisfiable modulo `T`. Si $res = SAT$ alors la formule ϕ est donc satisfiable modulo `T`. Dans le cas contraire, il faut « bloquer » ce modèle booléen pour que le SAT solveur cherche un autre modèle. Pour cela, on utilise l'explication (*unsat core*) uc renvoyée par la procédure de décision afin de recommencer la recherche de modèle sur la formule $f \wedge \neg\texttt{T2B}(uc)$.

L'exemple de la figure 2.6 illustre le fonctionnement du solveur SMT hors ligne sur la formule suivante :

$$\begin{gathered}\neg(a = b)\\ (x = a \vee x = b)\\ (y = a \vee y = b)\\ (z = a \vee z = b)\\ \neg(x = y)\end{gathered}$$

Le principal avantage d'un solveur SMT hors ligne est qu'il permet de réutiliser un solveur SAT « sur l'étagère », sans rien à avoir à modifier. Le solveur SAT communique avec le solveur SMT uniquement à travers les variables booléennes manipulées par les fonctions `T2B` et `B2T`. Cette technique permet donc de profiter des avancées les plus récentes sur les solveurs SAT. Malheureusement, cette modularité peut aussi s'avérer être un désavantage. En effet, le solveur SAT n'étant pas guidé par la théorie pour rechercher un modèle, il fait des choix de variables ou des déductions logiques (BCP [1]) « à l'aveugle », pouvant l'amener à explorer des parties de l'espace de recherche inutiles où il se perd complètement.

CDCL(T)

Pour remédier aux problèmes des solveurs SMT hors ligne, on va intégrer directement le raisonnement modulo théorie dans l'algorithme CDCL [1]. Cette procédure SMT, appelée CDCL(T), est donnée dans l'algorithme 6.

1. Voir partie sur SAT.

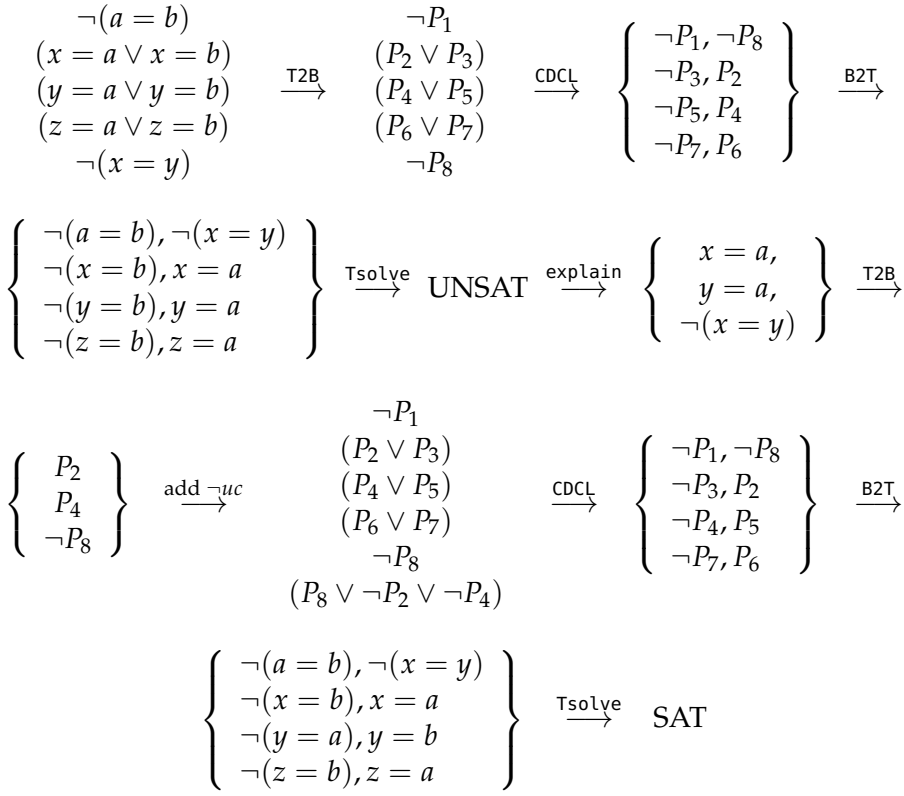


FIGURE 2.6 – *Fonctionnement du SMT hors ligne.*

L'extension de l'algorithme CDCL au raisonnement modulo théorie nécessite peu de modifications. Tout d'abord, sans que cela soit explicite dans l'algorithme, on suppose que toutes les structures de données du solveur SAT sont adaptées pour manipuler directement des littéraux avec des symboles de fonctions et de prédicats.

Le premier mécanisme à étendre est celui de la déduction booléenne (BCP). Celle-ci est réalisée par une fonction `theory_and_boolean_propagation`(ϕ,μ) (ligne 5). Cette fonction réalise d'abord la phase de déduction BCP, puis, si aucun conflit n'est détecté, elle appelle la procédure de décision de la théorie `T` pour vérifier la cohérence du modèle μ modulo `T`. Cette phase de déduction peut également être complétée par la propagation de littéraux déduits par la procédure de décision de la théorie. Ces littéraux pourront à leur tour permettre de déduire de nouvelles clauses unitaires pour continuer la boucle de déduction.

Le deuxième mécanisme à modifier est celui de l'analyse de conflits. La fonction `theory_and_boolean_conflict_analysis` (ligne 13) utilise les

Algorithme 6 : Algorithme CDCL(T).

```
1  function cdclT(φ) begin
2      μ ← [];
3      dl ← 0;
4      while True do
5          res ← theory_and_boolean_propagation(φ,μ);
6          if res = SAT then
7              if all_variables_are_assigned(φ,μ) then
8                  return SAT
9              l ← pick_a_branching_literal(φ,μ);
10             dl ← dl + 1;
11             push(μ,l@dl)
12         else
13             (lvl, cls) =
                theory_and_boolean_conflict_analysis(φ,μ);
14             if lvl < 0 then
15                 return UNSAT
16             backtrack(φ, μ, lvl);
17             learn(cls);
18             dl ← lvl
```

explications renvoyées par la procédure de décision pour trouver le niveau et la clause conflit modulo la théorie.

Comme la procédure de décision est maintenant partie prenante des choix et rebroussements du solveur SAT, il est fondamental qu'elle possède les propriétés d'incrémentalité et de backtrackabilité décrites dans la section 2.3.2.

L'exemple décrit dans la figure 2.7 illustre le fonctionnement de l'algorithme CDCL(T) sur un problème SMT. Le tableau à double colonnes décrit les différentes étapes de l'algorithme. Les quatre premières lignes indiquent des choix de littéraux de branchement (correspondant aux lignes 9 - 11 dans CDCL(T)). La ligne suivante correspond à une propagation modulo théorie où le littéral $(x_1 - x_2) > 3$ est détecté par la procédure de décision. Cette déduction permet de déclencher de nouvelles propagations booléennes (BCP). Cela amène à trouver un conflit entre les trois littéraux $(x_1 - x_3 > 6)$, $(x_3 = x_5 + 4)$ et $(x_1 - x_5 \leqslant 1)$, ce qui conduit à calculer la clause conflit $x_1 - x_3 \leqslant 6 \vee (x_3 \neq x_5 + 4) \vee (x_2 - x_3 > 2)$. Cet enchaîne-

$$(x_2 - x_3 \leqslant 2) \vee A_1$$
$$\neg A_2 \vee (x_1 - x_5 \leqslant 1)$$
$$(x_1 - x_2 \leqslant 3) \vee A_2$$
$$(x_3 + x_4 < 5) \vee (x_1 - x_3 > 6) \vee \neg A_1$$
$$(x_1 - x_2 \leqslant 3) \vee A_1$$
$$(x_2 - x_4 \leqslant 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1$$
$$(x_3 = x_5 + 4) \vee A_1 \vee A_2$$

@1	$(x_1 - x_3 > 6)$
@2	$(x_3 = x_5 + 4)$
@3	$(x_2 - x_4 \leqslant 6)$
@4	$(x_2 - x_3 \leqslant 2)$
T-propagate	$(x_1 - x_3 > 6) \wedge (x_2 - x_3 \leqslant 2) \Rightarrow (x_1 - x_2 > 3)$
BCP	A_1
BCP	A_2
BCP	$(x_1 - x_5 \leqslant 1)$
T-conflict	$(x_1 - x_3 > 6) \wedge (x_3 = x_5 + 4) \wedge (x_1 - x_5 \leqslant 1)$
backtrack	$x_1 - x_3 \leqslant 6 \vee (x_3 \neq x_5 + 4) \vee (x_2 - x_3 > 2)$
keep@1	$(x_1 - x_3 > 6)$
keep@2	$(x_3 = x_5 + 4)$
BCP	$(x_2 - x_3 > 2)$
...	

FIGURE 2.7 – *Fonctionnement de CDCL(T).*

ment illustre l'intérêt de combiner les raisonnements booléen et théorie au sein de l'algorithme CDCL. L'algorithme poursuit alors en rebroussant au niveau de décision @2 et en ajoutant par BCP le littéral $x_2 - x_3 > 2$ (voir dans la partie SAT le mécanisme de calcul d'une clause conflit en appliquant le principe de résolution).

2.3.4 Combinaison de procédures de décision

Dans la section précédente, nous avons décrit l'algorithme CDCL(T) qui combine un solveur SAT avec une *unique* procédure de décision pour une théorie T. En pratique, les formules à vérifier impliquent souvent plusieurs théories. Dans cette section, on s'intéresse aux techniques pour combiner plusieurs procédures de décision.

L'algorithme de Nelson-Oppen

Un des principaux problèmes sous-jacents à l'implémentation d'un solveur SMT est le problème de la construction d'une procédure de décision pour l'union de théories. Ce problème peut être défini de la manière suivante :

Si $\mathcal{T}_1$ et $\mathcal{T}_2$ sont deux théories *cohérentes* [2] :

1. Est-ce que l'union $\mathcal{T}_1 \cup \mathcal{T}_2$ est cohérente ?
2. Peut-on construire une procédure de décision $\mathcal{T}_1 \cup \mathcal{T}_2$ à partir de procédures de décision pour $\mathcal{T}_1$ et $\mathcal{T}_2$?

Bien qu'il n'y ait pas de réponse dans le cas général, des résultats ont été donnés pour certaines classes de théories. Par exemple, quand $\mathcal{T}_1$ et $\mathcal{T}_2$ n'ont pas de symboles communs (*i.e.* quand leurs signatures sont disjointes), le théorème suivant de Tinelli et Harandi [191] (corollaire 3.3 page 9) apporte une réponse à cette question.

Théorème 2. *Soient deux théories cohérentes et disjointes $\mathcal{T}_1$ et $\mathcal{T}_2$, si $\mathcal{T}_1$ et $\mathcal{T}_2$ admettent toutes les deux un modèle de cardinalité infinie* [2]*, alors l'union $\mathcal{T}_1 \cup \mathcal{T}_2$ est cohérente.*

Preuve. Par le théorème ascendant de Löwenhein-Skolem [94, 174] (qui dit pour résumer que si une théorie possède un modèle infini, elle possède un modèle de n'importe quelle cardinalité infinie), on peut choisir pour $\mathcal{T}_1$ et $\mathcal{T}_2$, respectivement, deux modèles $\mathcal{A}_1$ et $\mathcal{A}_2$ de même cardinalité infinie. Supposons par l'absurde que l'union $\mathcal{T}_1 \cup \mathcal{T}_2$ ne soit pas cohérente. Par le théorème de *cohérence jointe*, il doit exister une formule Φ dans l'intersection de ces deux théories telle que $\mathcal{T}_1 \models \Phi$ et $\mathcal{T}_2 \models \neg\Phi$. Puisque $\mathcal{T}_1$ et $\mathcal{T}_2$ sont disjointes, Φ ne peut contenir que des égalités $x = y$ ou différences $x \neq y$ entre variables. Un résultat bien connu en théorie des modèles est que deux modèles de la théorie vide (théorie sans symbole de fonction ou de prédicat autre que $=$) de même cardinalité sont *isomorphes*. Par conséquent, les réduits des modèles $\mathcal{A}_1$ et $\mathcal{A}_2$ à la théorie vide sont isomorphes et ils sont donc soit tous les deux des modèles de Φ, ou aucun des deux ne l'est. Ceci contredit notre hypothèse. □

Une fois que l'on sait que l'union de théories est cohérente pour une certaine classe de théories, il reste à trouver un moyen de combiner les

2. Voir définition en section 2.3.1

procédures de décision pour ces théories. Cependant, cela n'est pas possible en général puisqu'il existe des théories indécidables qui sont l'union de théories décidables (voir [28]).

Même dans le cas simple de théories *disjointes* ci-dessus, la combinaison n'est pas une question triviale. Voyons cela sur un exemple.

Prenons deux théories $\mathcal{T}_1$ et $\mathcal{T}_2$, où $\mathcal{T}_1$ est la théorie de l'arithmétique linéaire et $\mathcal{T}_2$ la théorie de l'égalité avec symboles non interprétés. Soit Φ la formule suivante :

$$(\Phi) \quad f(x) - x = 0 \wedge f(2x - f(x)) \neq x.$$

Une première idée pour décider la satisfiabilité de cette formule consiste à appliquer l'algorithme suivant :

1. décomposer Φ en deux formules logiques *pures*[3] Φ_1 et Φ_2,
2. conclure que Φ est satisfiable si et seulement si Φ_1 et Φ_2 sont prouvées satisfiables en utilisant les procédures de décision de $\mathcal{T}_1$ et $\mathcal{T}_2$, respectivement.

La première étape de l'algorithme, dite de « purification », consiste à appliquer récursivement la méthode suivante : si a est un sous-terme de Φ qui contient uniquement des symboles d'une théorie, alors remplacer a par une variable fraîche z et ajouter l'équation $z = a$ à Φ_i. Après ce mécanisme d'abstraction par variables, Φ est composée de deux formules *pures* Φ_1 et Φ_2 :

$$\begin{array}{ll} (\Phi_1) & z_1 = f(x) \wedge z_2 = f(x) \wedge f(z_3) \neq x \\ (\Phi_2) & z_1 - x = 0 \wedge 2x - z_2 = z_3. \end{array}$$

Il est facile de voir que Φ_1 et Φ_2 sont satisfiables dans $\mathcal{T}_1$ et $\mathcal{T}_2$, respectivement. Cependant, la conjonction $\Phi_1 \wedge \Phi_2$ ne l'est pas. En effet, de $z_1 = f(x)$ et $z_2 = f(x)$, on déduit que $z_1 = z_2$. Maintenant, $z_1 - x = 0$ implique $z_1 = x$, et à partir de $2x - z_2 = z_3$ on déduit la classe d'équivalence $z_1 = z_2 = z_3 = x$. Au final, par congruence, on déduit que $f(x) \neq z_2$ ce qui contredit $f(x) = z_2$. Par conséquent, cette méthode naïve de combinaison ne définit pas une procédure de décision.

Cet exemple illustre une propriété importante de la combinaison de procédures de décision : la non-satisfiabilité d'une telle conjonction $\Phi_1 \wedge \Phi_2$ est localisée dans une formule close, appelée *interpolant*, qui ne contient que des symboles communs aux deux théories. Dans l'exemple précédent, les signatures des deux théories étant disjointes, cet interpolant ne peut

3. Une formule est *pure* si elle ne contient que des symboles d'une seule signature

être constitué que d'égalités (ou différences) entre variables. Un interpolant pour la conjonction $\Phi_1 \wedge \Phi_2$ ci-dessus est par exemple $x = z_1 \wedge x = z_3$.

Ce résultat découle du théorème de Robinson et Craig [174, 94].

Théorème 3 (Cohérence jointe)**.** *Étant données deux théories* cohérentes $\mathcal{T}_1$ *et* $\mathcal{T}_2$*, l'union* $\mathcal{T}_1 \cup \mathcal{T}_2$ *est incohérente si et seulement s'il existe une formule close* Φ *ne contenant que des symboles communs à* $\mathcal{T}_1$ *et* $\mathcal{T}_2$ *telle que* $\mathcal{T}_1 \models \Phi$ *et* $\mathcal{T}_2 \models \neg\Phi$*.*

La plupart des algorithmes pour combiner des procédures de décision implémentent des techniques *efficaces* pour calculer de tels interpolants. C'est le cas en particulier de l'algorithme inventé par G. Nelson et D. Oppen [137].

Cet algorithme s'applique à des théories (de signatures) *disjointes*, c'est-à-dire ne partageant aucun symbole (de fonction ou de prédicat) autre que celui d'égalité. Les théories doivent également avoir la propriété d'être *stables à l'infini* (*stably infinite* en anglais), c'est-à-dire que toute formule satisfiable dans ces théories doit l'être au moins pour un modèle de cardinalité infinie. De nombreuses théories ont cette propriété. Par exemple, l'arithmétique linéaire ou non-linéaire, la théorie de l'égalité avec symboles non interprétés ou la théorie des tableaux. Par contre, certaines théories comme la théorie des vecteurs de bits ou celle des types énumérés ne l'ont pas.

Le cadre théorique sur lequel s'applique la méthode de Nelson-Oppen étant fixé, l'algorithme 7 décrit son fonctionnement pour des théories *convexes*. Une théorie est convexe si pour toute formule ϕ, lorsque ϕ implique une disjonction d'égalités entre variables, alors ϕ implique nécessairement une de ces égalités. Plus formellement,

$$\begin{array}{c}\text{Si } \mathcal{T} \models \forall \vec{x}.(\phi \Rightarrow x_1 = y_1 \vee \ldots \vee x_n = y_n) \text{ alors} \\ \text{il existe } 1 \leqslant i \leqslant n \text{ tel que } \mathcal{T} \models \forall \vec{x}.(\phi \Rightarrow x_i = y_i).\end{array}$$

Les théories de l'arithmétique linéaire sur les réels et l'égalité avec symboles non interprétés ont cette propriété. Par contre, l'arithmétique linéaire sur les entiers ne la possède pas. Par exemple, dans cette théorie, pour toute variable x, il est vrai que $1 \leqslant x \leqslant 2 \Rightarrow x = 1 \vee x = 2$, mais aucune des deux égalités n'est impliquée.

La propriété de convexité pour une théorie `T` permet de définir une interface pour sa procédure de décision `Tsolve` : pour toute conjonction de contraintes ϕ, `Tsolve`(ϕ) renvoie un couple (res, eq) où res indique si ϕ est satisfiable modulo `T` et, si c'est le cas, eq contient l'ensemble des égalités

Algorithme 7 : Algorithme de Nelson-Oppen pour deux théories convexes.

```
1  function NO(φ) begin
2      (φ1, φ2) ← purification(φ);
3      while True do
4          (res1, eq1) ← Tsolve1(φ1);
5          (res2, eq2) ← Tsolve2(φ2);
6          if res1 = UNSAT or res2 = UNSAT then
7              return UNSAT
8          if eq1 = eq2 = ∅ then
9              return SAT
10         φ1 ← φ1 ∧ eq1 ∧ eq2;
11         φ2 ← φ2 ∧ eq1 ∧ eq2;
```

$x = y$ impliquées par ϕ, entre des variables contenues dans ϕ, et telles que $x = y \notin \phi$.

L'algorithme de Nelson-Oppen commence par purifier la formule ϕ en entrée. Il entre ensuite dans une boucle pour faire coopérer les deux procédures de décision par échange d'égalités. Les deux formules ϕ_1 et ϕ_2 obtenues par purification sont envoyées aux procédures de décision $\texttt{Tsolve}_1$ et $\texttt{Tsolve}_2$, respectivement. Si l'une des deux formules est insatisfiable, alors $NO(\phi)$ renvoie UNSAT. Dans le cas contraire, si les procédures de décision ne renvoient aucune nouvelle égalité impliquée, l'algorithme s'arrête en concluant que ϕ est satisfiable. Autrement, les deux formules ϕ_1 et ϕ_2 sont renforcées avec la conjonction $eq_1 \wedge eq_2$ et la boucle recommence.

L'exemple suivant illustre le fonctionnement de l'algorithme de Nelson-Oppen pour deux théories convexes (l'arithmétique linéaire et la théorie de l'égalité). La formule Φ sur laquelle est appliqué l'algorithme est la suivante :

$$f(x) - x = 0 \wedge f(2x - f(x)) \neq x.$$

Le tableau de la figure 2.8 donne les étapes de la boucle principale de l'algorithme après que la phase de purification de Φ ait produit les deux formules ϕ_1 et ϕ_2 suivantes :

$$\begin{aligned} \phi_1 &\equiv z_1 = f(x) \wedge z_2 = f(x) \wedge f(z_3) \neq x \\ \phi_2 &\equiv z_1 - x = 0 \wedge 2x - z_2 = z_3. \end{aligned}$$

(1)	ϕ_1	$z_1 = f(x) \wedge z_2 = f(x) \wedge f(z_3) \neq x$
	ϕ_2	$z_1 - x = 0 \wedge 2x - z_2 = z_3$
	(res_1, eq_1)	SAT, $z_1 = z_2$
	(res_2, eq_2)	SAT, $z_1 = x$
(2)	ϕ_1	$z_1 = f(x) \wedge z_2 = f(x) \wedge f(z_3) \neq x \wedge \mathbf{z_1 = z_2} \wedge \mathbf{z_1 = x}$
	ϕ_2	$z_1 - x = 0 \wedge 2x - z_2 = z_3 \wedge \mathbf{z_1 = z_2} \wedge \mathbf{z_1 = x}$
	(res_1, eq_1)	SAT, $\emptyset$
	(res_2, eq_2)	SAT, $x = z_3$
(3)	ϕ_1	$z_1 = f(x) \wedge z_2 = f(x) \wedge f(z_3) \neq x \wedge z_1 = z_2 \wedge z_1 = x \wedge \mathbf{x = z_3}$
	ϕ_2	$z_1 - x = 0 \wedge 2x - z_2 = z_3 \wedge z_1 = z_2 \wedge z_1 = x \wedge \mathbf{x = z_3}$
	(res_1, eq_1)	UNSAT, –
	(res_2, eq_2)	SAT, –

FIGURE 2.8 – *Fonctionnement de l'algorithme de Nelson-Oppen.*

Le tour de boucle (1) donne les valeurs de ϕ_1 et ϕ_2 en entrée de boucle, ainsi que les couples (res_1, eq_1) et (res_2, eq_2) renvoyés par les procédures de décision. Les deux formules étant satisfiables dans leurs théories respectives, elles sont renforcées par la conjonction de nouvelles égalités impliquées $z_1 = z_2 \wedge z_1 = x$. Dans le tour de boucle (2), la nouvelle formule ϕ_1 est toujours satisfiable, mais aucune nouvelle égalité n'est impliquée. La formule ϕ_2 est elle aussi satisfiable, mais une nouvelle égalité $x = z_3$ est trouvée par la procédure de décision de l'arithmétique. En ajoutant cette égalité à ϕ_1, le troisième tour de boucle termine l'algorithme après avoir détecté que ϕ_1 est maintenant insatisfiable.

Correction de l'algorithme. La terminaison de l'algorithme et sa sûreté (si UNSAT est renvoyé, c'est bien que ϕ est UNSAT) sont immédiates puisqu'il ne peut y avoir qu'un nombre fini de nouvelles égalités entre variables et que chaque égalité ajoutée est une conséquence logique de ϕ_1 ou ϕ_2.

La preuve de complétude de l'algorithme consiste à montrer que si $\texttt{Tsolve}_1(\phi_1)$ et $\texttt{Tsolve}_2(\phi_2)$ renvoient tous les deux SAT, alors $\phi_1 \wedge \phi_2$

est bien satisfiable pour l'union des deux théories. Soient $\mathcal{M}_1$ et $\mathcal{M}_2$, les modèles de ϕ_1 et ϕ_2, respectivement. Pour que ϕ soit satisfiable dans l'union des théories, il suffit de combiner $\mathcal{M}_1$ et $\mathcal{M}_2$ en un modèle $\mathcal{M}$ qui soit non seulement un modèle de l'union des théories, mais aussi un modèle de $\phi_1 \wedge \phi_2$. Cette construction s'appelle une *amalgamation*.

Pour montrer que cette amalgamation est possible, il suffit de montrer (voir [94, 174]) qu'il existe une partition $\mathcal{D}$ de l'ensemble des variables partagées entre ϕ_1 et ϕ_2 qui soit compatible avec ϕ_1 et ϕ_2.

La construction de $\mathcal{D}$ s'appuie sur l'invariant de boucle suivant : ϕ_1 et ϕ_2 sont de la forme $\phi_1 \equiv \phi_1' \wedge \psi$ et $\phi_2 \equiv \phi_2' \wedge \psi$, où ψ est une conjonction d'égalités entre variables et ϕ_1' (resp. ϕ_2') ne contient aucune égalité entre variables (la preuve est laissée en exercice). Ainsi, si l'algorithme renvoie SAT, alors puisque $eq_1 = eq_2 = \emptyset$, on en déduit que *pour tout couple* de variables (x, y), ϕ_1 implique (modulo $\mathcal{T}_1$) $x = y$ si et seulement si ϕ_2 implique (modulo $\mathcal{T}_2$) $x = y$. On définit alors $\mathcal{D}$ de sorte que deux variables x et y partagées sont dans la même classe d'équivalence *si et seulement si* $\models \psi \Rightarrow x = y$. Supposons maintenant que ϕ_1 (resp. ϕ_2) ne soit pas compatible avec $\mathcal{D}$. Cela veut dire qu'il existe deux variables partagées x et y telles que $\mathcal{D}(x) \neq \mathcal{D}(y)$ et $\phi_1 \Rightarrow x = y$. Par construction de $\mathcal{D}$ et l'invariant de boucle de l'algorithme, ce cas est impossible. □

L'algorithme de Nelson-Oppen a deux points faibles. Le premier est qu'il nécessite d'instrumenter les procédures de décision pour déduire les égalités impliquées par une formule. Le deuxième est que le traitement des théories *non-convexes* nécessite d'une part d'étendre le mécanisme de déduction pour générer des disjonctions d'égalités, et d'autre part, de traiter ces disjonctions au sein même de l'algorithme de la combinaison de théories, alors que ces disjonctions seraient mieux traitées par un solveur SAT. Les deux sections suivantes présentent des variantes de cet algorithme pour éviter ces deux écueils.

L'analyse par cas à la demande

Cette technique nécessite d'étendre l'interface de la théorie avec une fonction `splitting_on_demand` pour ajouter à la formule d'entrée des clauses d'analyse par cas [20]. L'algorithme 8 montre la modification à apporter à `cdclT()` : il suffit d'appeler cette nouvelle fonction après avoir propagé les déductions booléennes et modulo théories. Le solveur SAT pourra ainsi être utilisé pour réaliser des analyses par cas.

Il est important de remarquer que les clauses d'analyse par cas peuvent contenir des termes ou littéraux non présents dans la formule de départ.

L'ajout de nouveaux éléments peut évidemment compromettre la terminaison de l'algorithme mais également sa sûreté. Pour garantir que la fonction `cdclT` s'arrête, on impose que l'ensemble des nouveaux littéraux des clauses d'analyse par cas soit fini. Pour garantir sa sûreté, il faut contraindre la procédure de décision à ne produire que des clauses qui préservent l'équisatisfiabilité à chaque tour de boucle. Ces restrictions ne semblent pas être une limitation pour les théories utilisées en pratique.

Algorithme 8 : CDCL(T) avec lemmes d'analyse par cas.

```
1  function cdclT(φ) begin
2      μ ← [];
3      dl ← 0;
4      while True do
5          res ← theory_and_boolean_propagation(φ,μ);
6          if res = SAT then
7              if all_variables_are_assigned(φ,μ) then
8                  return SAT
9              splitting_on_demand(φ,μ);
10             ...
11         else
12             ...
```

La combinaison retardée (DTC)

Cette approche, appelée en anglais *Delayed Theory Combination* [37, 33], consiste à faire coopérer directement les procédures de décision avec le solveur SAT, ce dernier se chargeant de l'échange des égalités (entre variables d'interface) entre les procédures.

L'algorithme 9 peut être vu comme une intégration de l'algorithme de Nelson-Oppen pour combiner deux théories $\mathcal{T}_1$ et $\mathcal{T}_2$ dans l'algorithme CDCL. Après avoir purifié la formule en entrée ϕ en deux formules ϕ_1 et ϕ_2, l'algorithme DTC initialise une variable α contenant l'ensemble des atomes sur lesquels effectuer des décisions, ainsi que toutes les égalités entre les variables partagées par ϕ_1 et ϕ_2. La variable μ contient la pile des décisions et des littéraux impliqués.

La boucle `repeat` implémente, en séquence, la déduction des contraintes booléennes et modulo les théories $\mathcal{T}_1$ et $\mathcal{T}_2$. La propagation des contraintes booléennes est classique. Pour déduire de nouveaux

Algorithme 9 : Delayed Theory Combination.

```
1  function dtc(φ) begin
2      μ ← [];
3      (φ1, φ2) ← purification(φ);
4      α ← atoms(φ1 ∧ φ2, interface_equalities(φ1, φ2));
5      while True do
6          repeat
7              res ← boolean_propagation(φ1 ∧ φ2,μ);
8              if res = UNSAT then
9                  (lvl, cls) = boolean_conflict_analysis(φ1 ∧ φ2,μ);
10                 if lvl < 0 then return UNSAT;
11                 backjump_and_learn(φ, μ, cls,lvl);
12             else
13                 μ1^e = extract_theory1_and_interface_equalities(μ);
14                 res1 ← Tsolve1(φ1, μ1^e);
15                 if res1 = UNSAT then
16                     (lvl, cls1) = theory1_conflict_analysis(φ1,μ1^e);
17                     if lvl < 0 then return UNSAT;
18                     backjump_and_learn(φ, μ, cls1, lvl);
19                 else
20                     μ2^e = extract_theory2_and_interface_equalities(μ);
21                     res2 ← Tsolve2(φ2, μ2^e);
22                     if res2 = UNSAT then
23                         (lvl, cls2) = theory2_conflict_analysis(φ2, μ2^e);
24                         if lvl < 0 then return UNSAT;
25                         backjump_and_learn(φ, μ, cls2, lvl);
26                     else
27                         if all_variables_are_assigned(α) then return SAT;
28                         theory1_splitting_on_demand(φ1, μ1^e);
29                         theory2_splitting_on_demand(φ2, μ2^e);
30         until fixpoint on μ;
31         ℓ ← pick_a_branching_literal(α);
32         decide_on(μ, ℓ)
```

littéraux impliqués modulo la théorie $\mathcal{T}_1$, on doit d'abord extraire les faits de cette théorie, ainsi que les contraintes d'égalités entre variables partagées, qui ont été ajoutées dans la pile μ. L'ensemble obtenu μ_1^e est passé en argument à la procédure de décision de la théorie $\mathcal{T}_1$ (avec la formule ϕ_1). En cas de conflit, on extrait une clause de conflit modulo $\mathcal{T}_1$ et on rebrousse dans l'arbre de décision. La propagation des contraintes modulo $\mathcal{T}_2$ est identique. Si tous les littéraux déduits sont satisfiables et que tous les atomes contenus dans α ont une valeur de vérité, on arrête l'algorithme en renvoyant SAT. Sinon, on ajoute à la fin de la boucle des lemmes d'analyse par cas, comme vu précédemment. Lorsqu'on atteint le point fixe de cette propagation de contraintes, le solveur SAT ajoute un nouveau littéral de décision dans μ. Ce littéral est choisi parmi l'ensemble α, qui rappelons-le, contient toutes les égalités entre variables partagées. Ainsi, le solveur SAT construit bien une partition entre ces variables, d'une manière similaire à l'algorithme de Nelson-Oppen.

La combinaison dirigée par les modèles

Un des inconvénients de l'algorithme DTC est que le solveur SAT doit gérer un nombre quadratique d'égalités entre variables partagées. Il choisit également en aveugle parmi ces égalités.

Pour remédier à ces deux problèmes, la combinaison dirigée par les modèles (*model based theory combination* [58]) étend l'algorithme DTC afin de permettre aux procédures de décision de déduire *simplement* des égalités. Mais contrairement à l'algorithme de Nelson-Oppen, les égalités déduites par une procédure de décision sont celles impliquées par un modèle *candidat* de cette procédure.

Par exemple, si la conjonction de contraintes suivante est donnée à la théorie de l'arithmétique linéaire sur les entiers :

$$0 \leqslant x \leqslant 1 \wedge 0 \leqslant y \leqslant 1 \wedge z = y - 1$$

la procédure de décision peut construire le modèle $[x \mapsto 0; y \mapsto 0; z \mapsto -1]$ et déduire l'égalité $x = y$.

Bien sûr, ces égalités peuvent s'avérer être en contradiction avec les autres théories et nécessiter de rebrousser. Dans l'exemple précédent, il est tout à fait possible que le modèle pour cet ensemble de contraintes soit celui où x et y prennent des valeurs différentes.

Ainsi, pour permettre la gestion (avec rebroussement) de ces égalités dirigées par les modèles, il suffit de les générer lors de l'appel à la fonction `theory`i`_splitting_on_demand`. Ainsi, une égalité impliquée $x = y$ sera

donnée sous la forme d'une clause binaire $x = y \vee x \neq y$, en donnant une priorité de décision au littéral $x = y$.

2.3.5 Pour aller plus loin

On renvoie le lecteur intéressé par une formalisation rigoureuse et complète de l'algorithme CDCL(T) à l'article de Nieuwenhuis et al. [140]. L'algorithme de Nelson-Oppen, ainsi que les preuves de correction, sont également donnés dans plusieurs articles, voir par exemple [191, 50].

Il existe aussi d'autres schémas de combinaison de procédures de décision implantés dans les solveurs SMT. Parmi eux, l'algorithme de R. Shostak [176, 167] inventé au début des années 1980. Cette méthode permet de combiner la théorie de l'égalité avec symboles non interprétés avec une autre théorie, dite de Shostak, disposant d'une interface réduite à deux fonctions :

— un *canonizer*, pour mettre les termes de la théorie en forme normale ;
— un *solver*, pour résoudre des équations pures de la théorie et renvoyer la solution sous la forme d'une substitution (des variables de l'équation vers des termes).

Ces deux fonctions sont alors utilisées pour étendre l'algorithme de clôture par congruence que nous avons vu en section 2.3.2. La combinaison de plusieurs (canonizers et solvers de) théories de Shostak est discutée dans [173, 112].

Un autre schéma de combinaison, appelé MCSAT, a été introduit récemment par L. de Moura et D. Jovanovic [59]. Cette méthode de combinaison met au même plan le solveur SAT et toutes les procédures de décision. L'idée principale est que l'interface de chaque solveur doit implanter une phase de construction de modèles (affectation de valeurs aux variables de son domaine, propagation de contraintes) et une phase de résolution (pour la résolution des conflits modulo théorie). Ainsi, ces procédures coopèrent en intervenant *directement* dans une structure de données partagée appelée *trail*.

On renvoie également le lecteur aux articles [89, 67, 102] pour une présentation approfondie du traitement de l'arithmétique linéaire.

Enfin, nous recommandons la lecture de trois livres qui couvrent une très grande partie de ce domaine de recherche :

— *Handbook of Satisfiability* [25]
— *The Calculus of Computation* [34]
— *Decision Procedures* [111]

2.4 Conclusion

Les solveurs SAT et SMT sont devenus incontournables en informatique, au point qu'on peut réellement parler de la *révolution SAT et SMT*. Ces outils sont aujourd'hui utilisés dans tant de domaines qu'une liste exhaustive serait très longue. Pour n'en donner que quelques-uns, citons l'intelligence artificielle, la conception de micro-processeurs, la vérification de logiciels, la synthèse de programmes, etc.

Cette révolution est avant tout liée à l'augmentation incroyable des performances de ces démonstrateurs qui sont maintenant capables de traiter des problèmes de taille industrielle.

Dans ce cours, nous avons présenté les principales techniques et algorithmes des démonstrateurs SAT et SMT modernes. Le lecteur qui souhaite approfondir ce sujet pourra trouver des compléments d'information dans les références bibliographiques contenues dans ces notes de cours.

Chapitre 3

Géométrie numérique

Bruno Lévy
Nicolas Ray
Dmitry Sokolov

3.1 Introduction

Dans ce chapitre, nous nous intéressons à des algorithmes de *géométrie numérique*, à savoir des algorithmes pour construire, manipuler, optimiser des représentations 3D des objets. Nous nous intéressons particulièrement aux algorithmes qui construisent et manipulent des *maillages*, à savoir qui représentent les objets 3D sous la forme d'une collection de sommets, d'arêtes et de polygones (et éventuellement polyèdres pour les maillages volumiques). Cette représentation est très largement répandue et utilisée par de nombreux domaines d'applications, tels que la simulation numérique, ou encore le graphisme, voir figure 3.1.

Nous avons choisi un ensemble de trois algorithmes particuliers, que nous avons sélectionnés à la fois en raison de leur intérêt pratique, mais surtout parce qu'ils ont une structure cachée qui a une certaine esthétique. D'un point de vue plus pragmatique, cette structure peut être utilisée pour prouver certaines propriétés et ainsi développer des algorithmes très efficaces. Les explications de ce cours ont en priorité l'objectif de donner une bonne intuition du raisonnement qui permet d'analyser la structure

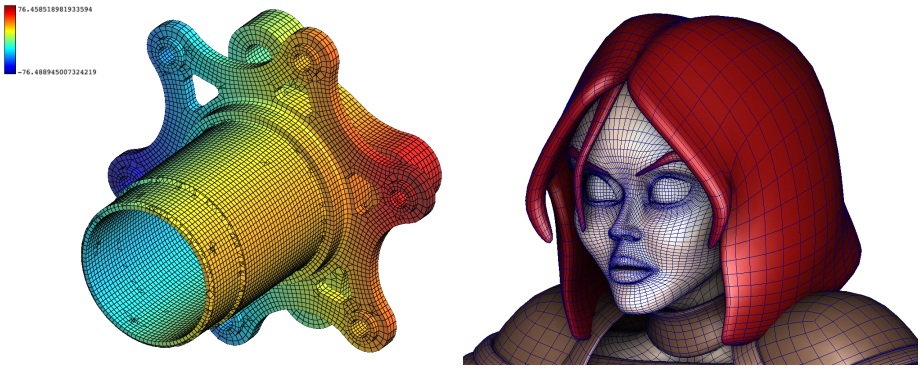


FIGURE 3.1 – *Deux exemples d'utilisation de maillages, pour la modélisation 3D et la simulation numérique de type éléments finis (à gauche) et pour le graphisme par ordinateur (à droite). Crédit : Grabcad et CGSociety.*

mathématique du problème [1], illustré dans des cas particuliers par des calculs élémentaires développés ici.

Ceci permet ensuite de voir comment exploiter la structure du problème afin de développer des algorithmes et des implantations efficaces.

Nous encourageons le lecteur à refaire les petits calculs et démonstrations élémentaires de ce cours [2], et à procéder éventuellement de même en lisant les versions générales et approfondies des articles et ouvrages référencés en 3.5, pour « recréer lui-même » le raisonnement et ainsi s'approprier complètement sa structure [3].

Plan du cours.

Nous nous intéresserons tout d'abord à la **résolution de systèmes d'équations linéaires** (3.2), utile pour une grande classe d'algorithmes en géométrie numérique qui consistent à résoudre un ensemble d'équations dont les inconnues sont les coordonnées des sommets du maillage. Nous étudierons en particulier l'algorithme du *gradient conjugué*, qui en exploitant une orthogonalité dans un certain espace, permet d'obtenir d'excellentes performances dans de nombreux cas.

Nous nous intéresserons ensuite au problème de la **paramétrisation de surfaces triangulées** (3.3), un problème important en graphisme par ordinateur (pour le plaquage de textures). Le *théorème de Tutte* permet de caractériser une classe de paramétrisations valides, et peut se traduire

1. « *Teach principles, not formulas !* » – Richard Feynman.
2. et à nous indiquer toute erreur/coquille éventuelle !
3. « *What I cannot create, I don't understand.* » – Richard Feynman.

directement en un algorithme effectif pour construire une paramétrisation. Nous étudierons une preuve récente, qui exploite de manière élégante une relation entre la topologie de la surface et la topologie des champs de vecteurs définis sur cette surface (Poincaré–Hopf).

Enfin, nous aborderons le sujet du **transport optimal** (3.4), qui permet de mesurer des distances entre des fonctions (ou des objets plus généraux). Ce thème très général a connu un regain d'intérêt ces dernières années, avec une communauté de recherche très active en France (sous l'impulsion de personnalités mathématiques comme Yann Brenier et Cédric Villani), et l'essor de nouveaux domaines applicatifs, tels que l'apprentissage machine. Dans un contexte particulier (semi-discret), nous verrons comment la structure mathématique du problème (dualité de Kantorovich) fait directement apparaître des objets classiques en géométrie algorithmiques (diagrammes de Voronoi généralisés), et ainsi se traduit directement en un algorithme de calcul efficace.

3.2 Résolution de systèmes d'équations linéaires

Introduction

La résolution de systèmes linéaires est la clé de voûte d'un grand nombre d'algorithmes d'optimisation numérique rencontrés dans nombre de contextes applicatifs. Ainsi, de tels systèmes linéaires jouent un rôle central dans les problèmes d'optimisation de type moindres carrés, dans les méthodes de simulation numérique fondées sur les éléments finis, dans des algorithmes d'optimisation de fonctions non-linéaires [4], ...

Le gradient conjugué est un algorithme de résolution de systèmes linéaires. Il est très utilisé, car très efficace et relativement simple à implanter. En revanche, il est assez difficile d'avoir une bonne intuition de son fonctionnement, ce que nous proposons d'étudier ici. Le gradient conjugué peut très bien être utilisé comme une « boîte noire », mais il est plus satisfaisant intellectuellement d'en comprendre le fonctionnement. De plus, d'un point de vue pratique, ceci permet de l'utiliser de manière plus efficace, en tenant compte du conditionnement de la matrice, en réglant le nombre d'itérations en fonction du type de problème, ou encore en le reprogrammant de manière à réaliser les calculs en place (à savoir, sans stocker explicitement de matrices en mémoire).

Notre approche va consister à adopter le point de vue de quelqu'un qui chercherait à réinventer l'algorithme. Nous allons tout d'abord préciser

4. qui résolvent une série de problèmes linéaires.

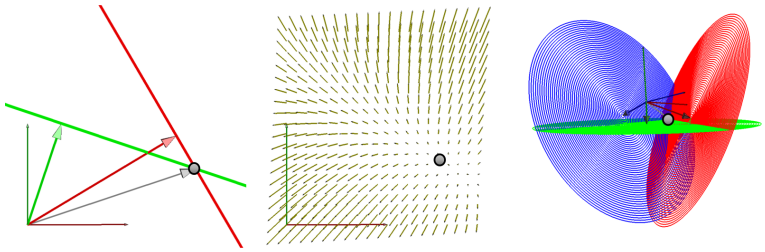


FIGURE 3.2 – *Interprétations géométriques de la résolution de $Ax - b = 0$. Le produit scalaire de x avec chacune des lignes de A est fixé par les coordonnées de b. Chacune de ces contraintes impose que x soit situé sur un hyperplan (droite en 2D, plan en 3D), ce qui définit x comme étant l'intersection de ces hyperplans : droites verte et rouge dans notre exemple 2D (à gauche) et les trois plans en 3D (à droite). Au milieu, $v = Ax - b$ est vu comme un champ de vecteurs : la solution est le point où le vecteur s'annule. Ce champ étant intégrable, il pourra être vu comme le gradient d'une fonction à minimiser… sous certaines conditions.*

quel est le problème résolu par la méthode, puis nous présenterons deux méthodes de résolution (descente de gradient et « conjugaison ») qui pourront finalement être combinées pour obtenir le gradient conjugué. Nous verrons aussi comment se comportent ces algorithmes dans des conditions proches de leur utilisation habituelle *i.e.*, avec des matrices creuses.

3.2.1 Comprendre le problème

L'algorithme du gradient conjugué résout le problème suivant :

> Étant donnée une matrice A de dimension $n \times n$, symétrique définie positive et un vecteur $b \in \mathbb{R}^n$, trouver le vecteur $x \in \mathbb{R}^n$ tel que $Ax - b = 0$.

Soit a_{ij} le terme de A situé sur la ligne i et la colonne j. Nous rappelons que :

— la matrice A est **symétrique** $\iff a_{ij} = a_{ji}$;
— la matrice A est **définie positive** $\iff x^\top Ax \geqslant 0, \forall x \in \mathbb{R}^n$.

On comprend assez bien que l'on souhaite résoudre un système linéaire — ce qui géométriquement correspond à calculer l'intersection des hyperplans qui correspondent à chaque équation (voir figure 3.2). Il est moins évident de comprendre les conditions sur la matrice A. On va donc chercher à comprendre les raisons pour lesquelles on se restreint aux matrices définies positives. Pour ce faire,

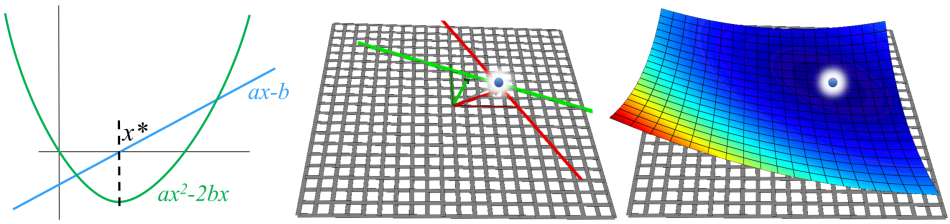


FIGURE 3.3 – *En 1D, la solution x^* de $ax - b = 0$ est aussi le minimum de $ax^2 - 2bx$ (à gauche). En 2D, la solution du système linéaire $Ax - b = 0$ est l'intersection des droites correspondant à chaque équation (au milieu), et est aussi le minimum de la forme quadratique associée $x^\top Ax - 2b^\top x$ (à droite).*

— nous verrons que si A est symétrique définie positive, alors la résolution d'un système d'équations linéaires $Ax - b = 0$ et la minimisation de la forme quadratique associée $x^\top Ax - 2b^\top x$ sont deux problèmes équivalents. Pour ce faire, nous verrons que l'on peut minimiser $f(x) = x^\top Ax - 2b^\top x$ en résolvant $(A + A^\top)x - 2b = 0$ (ce qui implique que A soit symétrique pour avoir l'équivalence avec $Ax - b = 0$) et que $f(x)$ admette un minimum (A strictement définie positive) ;
— nous présenterons alors quelques propriétés utiles de la matrice A (dues au fait qu'elle soit strictement définie positive) ;
— puis nous introduirons une nouvelle matrice M dont l'application linéaire associée joue un rôle central pour la compréhension de l'algorithme du gradient conjugué.

Equivalence $\min(x^\top Ax - 2b^\top x) \iff Ax - b = 0$?

Le minimum d'une forme quadratique (s'il existe) est le point qui annule le gradient ; cette section développe ce lien. Le point de vue minimisation conduit à des méthodes de résolution numérique dites itératives comme la descente de gradient (qui définit une suite infinie de points qui converge vers la solution). Le point de vue « résolution d'un système linéaire » amène plutôt à des méthodes de résolution directes/exactes qui vont donner la solution en n itérations, comme le pivot de Gauss par exemple. Comme nous le verrons par la suite, le gradient conjugué est à la fois un solveur itératif car chaque étape le rapproche de la solution, et un solveur direct du fait de choisir des directions conjuguées qui garantissent de trouver la solution en n itérations.

Nous allons voir que minimiser une forme quadratique revient à

résoudre un système linéaire (figure 3.3). La fonction $f(x) = x^\top Ax - 2b^\top x$ est l'écriture matricielle de la forme quadratique $f(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij}x_i x_j - \sum_{i=1}^n 2b_i x_i$.

En $1D$, A et b deviennent des réels (a et b), et le problème est de minimiser $ax^2 - 2bx$. Cette fonction étant dérivable partout, si elle admet un minimum, c'est forcément quelque part où sa dérivée est nulle. Ainsi, on peut écrire $x = \text{argmin}_x(ax^2 - 2bx) \Rightarrow ax - b = 0$. Ceci étant, ce point $x^* = b/a$ peut ne pas exister (si $a = 0$) ou correspondre au maximum de f (si $a < 0$)! Dans l'écriture matricielle de ce cas 1D, la contrainte $a > 0$ devient que A doit être définie positive!

Dans le cas général, le raisonnement est similaire : $f(x)$ étant dérivable partout, si un minimum x^* existe, ∇f le gradient de f s'annule en x^* *i.e.*, $\forall l, \frac{\partial}{\partial x_l} f(x^*) = 0$. Calculons donc le gradient de f :

$$\begin{aligned}
\frac{\partial}{\partial x_l} f(x) &= \frac{\partial}{\partial x_l}(x^\top Ax - 2b^\top x) && (3.1)\\
&= \frac{\partial}{\partial x_l}\Big(\sum_{i=1}^n \sum_{j=1}^n a_{ij}x_i x_j - \sum_{i=1}^n 2b_i x_i\Big) && (3.2)\\
&= -2b_l + \sum_{i=1}^n \sum_{j=1}^n \begin{cases} 2a_{ll}x_l & \text{si } i = l, j = l \\ a_{il}x_i & \text{si } i \neq l, j = l \\ a_{lj}x_j & \text{si } i = l, j \neq l \\ 0 & \text{si } i \neq l, j \neq l \end{cases} && (3.3)\\
&= -2b_l + \sum_{i=1}^n a_{il}x_i + \sum_{j=1}^n a_{lj}x_j && (3.4)\\
&= -2b_l + \sum_{i=1}^n (a_{li} + a_{il})x_i. && (3.5)
\end{aligned}$$

Dans ce calcul, on remarque que les éléments non nuls de la somme sont « localisés » sur une ligne et une colonne de la matrice A, ce qui permet de remplacer la double somme par deux simples sommes. Puis, la matrice étant carrée, ces sommes portent sur le même nombre d'éléments, ce qui permet de les fusionner.

Finalement, annuler tous les $\frac{\partial}{\partial x_l} f(x)$ revient à résoudre le système $(A + A^\top)x - 2b = 0$. Ainsi, si A est symétrique, $2Ax - 2b$ est le gradient de la forme quadratique $f(x) = x^\top Ax - 2b^\top x$. Résoudre $Ax - b = 0$ revient donc à annuler le gradient de $f(x)$. Si, de plus, A est strictement définie positive, la solution de $Ax - b = 0$ sera le minimum de $f(x)$.

En résumé : Pour résoudre notre problème, il suffit de minimiser $x^\top Ax - 2b^\top x$. Si A est symétrique et définie positive alors la solution existe et est unique.

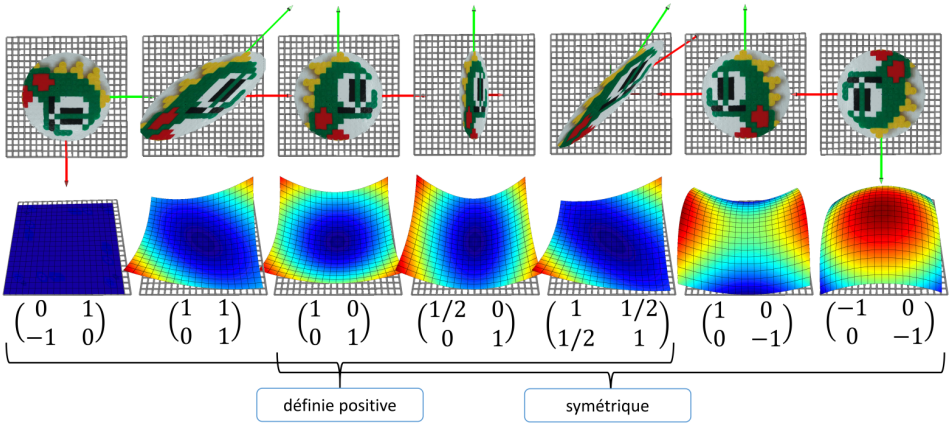


FIGURE 3.4 – *Quelques exemples de matrices 2D pour illustrer la symétrie et la positivité : une rotation, un cisaillement, l'identité, un étirement, un étirement non aligné sur les axes, une symétrie axiale, et une inversion. La première ligne montre comment la matrice déforme une image, la seconde montre la forme quadratique associée, et la troisième donne son expression — Attention : la symétrie est généralement supposée pour une matrice définie positive.*

Implications du fait que A soit symétrique définie positive

On a vu que la symétrie de A est nécessaire pour que $Ax - b$ soit le gradient d'une forme quadratique. D'autre part, le fait que A soit définie positive assure que le point qui annule le gradient est bien un minimum de f. La restriction à ce type de matrices permet d'exploiter des propriétés intéressantes de leurs valeurs/vecteurs propres. Voici celles qui nous intéressent dans le cadre de ce cours :

Implication pour les valeurs propres : A est supposée définie positive, donc $x^\top Ax > 0, \forall x \neq 0$. En particulier, pour le vecteur propre v de valeur propre λ (*i.e.*, $Av = \lambda v$), on a $v^\top \lambda v > 0$, ce qui implique $\lambda > 0$ (et par là même, que $\lambda \in \mathbb{R}$). Ainsi, les valeurs propres de A sont réelles et positives.

Implication pour les vecteur propres Soient v_1, v_2 deux vecteurs propres (distincts) $Av_1 = \lambda_1 v_1$ et $Av_2 = \lambda_2 v_2$. Par symétrie de A on a :

$$v_1^\top Av_2 = v_2^\top Av_1 \tag{3.6}$$
$$v_1^\top \lambda_2 v_2 = v_2^\top \lambda_1 v_1 \tag{3.7}$$
$$(\lambda_2 - \lambda_1)(v_1^\top v_2) = 0. \tag{3.8}$$

Donc, si v_1 et v_2 ont des valeurs propres différentes, ils sont orthogonaux.

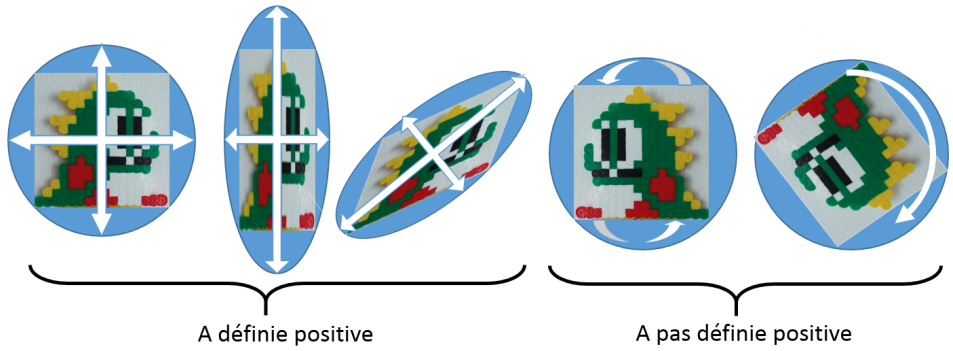


FIGURE 3.5 – *Applications linéaires 2D : voyons comment les matrices transforment l'image. De gauche à droite : la matrice identité ne change pas l'image, de manière plus générale, une matrice diagonale étire (ou compresse) l'image dans les directions des axes, de façon encore plus générale, toutes les matrices symétriques définies positives étirent (ou compressent) dans des directions orthogonales, par contre, une matrice antisymétrique va retourner des axes i.e., étirer avec un coefficient négatif, et une matrice de rotation va « étirer » avec un coefficient complexe.*

Même s'il existe une infinité de vecteurs propres (due à des valeurs propres multiples), on peut démontrer que A admet une base de vecteurs propres orthogonaux.

Interprétation géométrique On peut décomposer x dans la base des vecteurs propres normalisés v_i de A par simple projection puisqu'ils sont orthogonaux. Ainsi $Ax = \sum_{i=1}^{n}(x^\top v_i)\lambda_i v_i$. Ce qui veut dire que l'application linéaire définie par A est simplement un étirement de l'espace dans les directions de chaque vecteur propre, et avec un facteur donné par la valeur propre associée (figure 3.5).

En résumé : Le fait que A soit symétrique définie positive implique que ses valeurs propres soient réelles et positives, et qu'il existe un ensemble de vecteur propres qui définit une base orthogonale.

Une application linéaire bien utile : $M = \sqrt{A}$

On définit M comme étant la matrice symétrique ayant les mêmes vecteurs propres que A mais dont les valeurs propres sont les racines carrées de celles de A. Cette matrice est aussi à coefficients réels, symétrique et strictement définie positive. La notation $M = \sqrt{A}$ vient du fait que $M^\top M = MM = A$ comme on peut le voir :

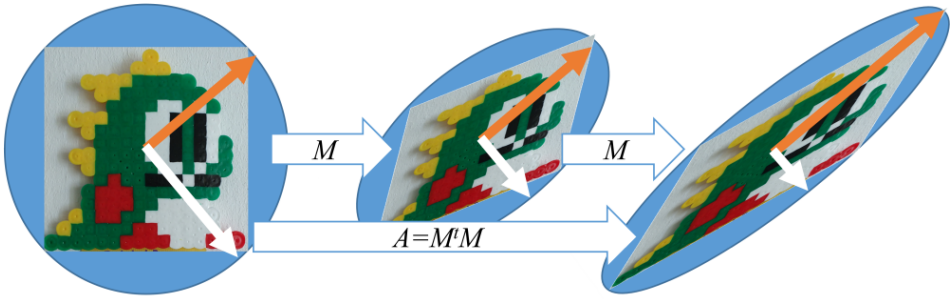


FIGURE 3.6 – *L'application linéaire associée à A est l'application de deux fois celle associée à M. Les vecteurs propres (flèches oranges et blanches) de A et M sont les mêmes : seules les valeurs propres sont différentes (mises au carré).*

— soit analytiquement :

$$M^\top Mx = MMx = \sum_{i=1}^{n}\sum_{j=1}^{n}\sqrt{\lambda_i}(\sqrt{\lambda_j}x^\top v_j)v_i = \sum_{i=1}^{n}\lambda_i v_i(x^\top v_i) = Ax$$

(parce que les v_i forment une base orthonormée)

— soit géométriquement : dans la direction de chaque vecteur propre, on étire deux fois avec un coefficient de $\sqrt{\lambda_i}$, ce qui revient à étirer d'un coup de λ_i (figure 3.6).

La A-orthogonalité L'intérêt de définir cette matrice M est que l'on peut très facilement calculer des produits scalaires dans l'espace transformé par M. En effet, si l'on veut calculer le produit scalaire entre les images de deux vecteurs u et v par l'application M, on doit évaluer $(Mu)^\top Mv = u^\top M^\top Mv = u^\top Av$, ce qui se fait sans expliciter M, mais par un produit avec la matrice A, qui est une donnée du problème.

Remarque : En analyse tensorielle, A définit le tenseur métrique de l'application linéaire associée à M : c'est une application bilinéaire qui permet de mesurer le produit scalaire (donc des longueurs et les angles) dans l'espace transformé par M.

Implications pratiques

Considérons l'application linéaire définie par M.

On peut calculer très efficacement $(Mu)^\top Mv = u^\top Av$ le produit scalaire des images Mu et Mv de deux vecteurs quelconques u et v. On peut donc très facilement savoir si Mu et Mv sont orthogonaux (il suffit que $u^\top Av = 0$), et calculer la norme de l'image d'un vecteur : $\|Mu\| = \sqrt{u^\top Au}$.

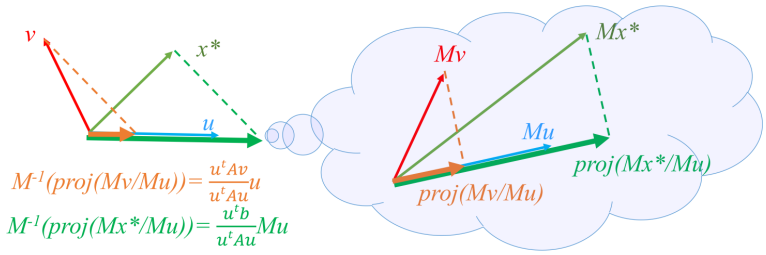


FIGURE 3.7 – *Soient u et v deux vecteurs quelconques et x^* la solution de $Ax - b = 0$. On peut facilement calculer les produits scalaires $Mu.Mv$ et $Mu.Mx^*$. Cela permet de projeter v et x^* sur u d'une manière qui serait orthogonale dans M.*

On peut même calculer efficacement $(Mu)^\top Mx^* = u^\top b$, le produit scalaire des images Mu et Mx^* d'un vecteur quelconque u et de la solution de notre problème $x^* = A^{-1}b$.

En pratique, on va utiliser ces propriétés pour projeter des vecteurs v et la solution x^* sur d'autres vecteurs u de sorte à ce que cette projection, regardée dans l'espace transformé par l'application linéaire associée à M, soit orthogonale (figure 3.7).

En résumé : On sait calculer des produits scalaires entre les images de vecteurs transformés par l'application linéaire $M = \sqrt{A}$. En particulier, on sait calculer les produits scalaires $(Mu)^\top Mv$ et $(Mu)^\top Mx^*$.

(Re)-formulation du problème

L'énoncé initial peut être reformulé en adoptant le point de vue « minimisation » et/ou en utilisant l'espace transformé par M. Ce qui donne les trois algorithmes que nous détaillerons par la suite.

- **Descente de gradient** (section 3.2.2) On peut minimiser la forme quadratique $x^\top Ax - 2bx$, comme toute autre fonction convexe, par une descente de gradient. Avoir une forme quadratique permet même de calculer un pas de descente optimal.
- **Conjugaison** (section 3.2.3) Cette méthode raisonne dans l'espace transformé par M, et projette directement la solution sur une base orthogonale. Les vecteurs dont les images par M forment une base orthogonale sont dits conjugués deux à deux (par rapport au produit scalaire défini par A). C'est pour cela que l'on appelle cette méthode « conjugaisons ». *Attention : cette méthode n'a qu'un intérêt pédagogique, il ne faut pas l'utiliser.*

— **Gradient conjugué** (section 3.2.4) Le gradient conjugué est une descente de gradient dont les directions de descente sont modifiées de sorte à être conjuguées les unes par rapport aux autres.

Les trois algorithmes ne diffèrent que par le calcul de la direction du pas suivant, et par le critère de fin de boucle. Sinon, ils utilisent exactement la même structure :

Algorithme général :
Initialisation $x^{(0)} \leftarrow 0$
Tant que le système n'a pas convergé, on itère sur k :
— calcul de la direction du pas suivant $d^{(k)}$
— calcul de la taille du pas suivant $\alpha^{(k)}$
— mise à jour de la solution $x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} d^{(k)}$

En résumé : On va utiliser chaque astuce indépendamment (dualité avec problème de minimisation et le calcul de produit scalaire dans l'espace transformé par M). Puis, en les combinant, on obtiendra l'algorithme du gradient conjugué.

3.2.2 Descente de gradient

Le gradient d'une fonction donne la direction dans laquelle la fonction varie le plus rapidement. La descente de gradient consiste, à partir d'une position initiale $x^{(0)} = 0$, à se déplacer dans le sens du gradient de sorte à diminuer le plus rapidement possible la valeur de la fonction. Ainsi, on construit une suite $x^{(k)}$ qui converge vers la solution x^*. Pour ce faire, $x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} \nabla f(x^{(k)})$. Dans le cas de notre forme quadratique, $\nabla f(x^{(k)}) = 2Ax^{(k)} - 2b$.

Algorithme :
Initialisation $x^{(0)} \leftarrow 0$
On itère sur k :

$$\begin{aligned} r^{(k)} &\leftarrow b - Ax^{(k)} \\ \alpha^{(k)} &\leftarrow \frac{r^{(k)\top} r^{(k)}}{r^{(k)\top} A r^{(k)}} \\ x^{(k+1)} &\leftarrow x^{(k)} + \alpha^{(k)} r^{(k)} \end{aligned}$$

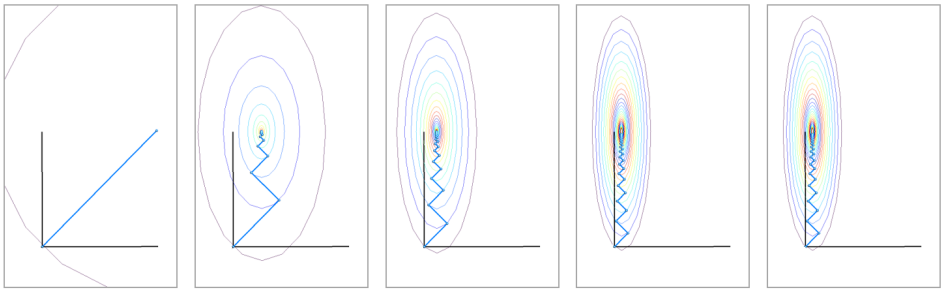


FIGURE 3.8 – *Descente de gradient à pas optimal avec une matrice A diagonale, dont a_{00} prend des valeurs de 1 (gauche) à 5 (droite), $a_{11} = 1$ et $a_{10} = a_{01} = 0$. Les segments bleus relient les $x^{(k)}$ aux $x^{(k+1)}$, et les ellipses sont les iso-valeurs de $f(x) = x^\top Ax - 2b^\top x$.*

Calcul de $\alpha^{(k)}$: de combien avancer ?

Comme on se déplace dans le sens opposé au gradient, si l'on avance d'un tout petit peu, on est sûr de diminuer f. Ceci étant, on minimise une forme quadratique qui, restreinte à la droite de recherche, reste une forme quadratique (une parabole). Il est donc possible de trouver directement le pas optimal (celui qui minimisera $f(x^{(k+1)})$ le long de la direction $r^{(k)}$) en annulant la dérivée $df(x^{(k+1)})/d\alpha^{(k)}$, avec $x^{(k+1)} = x^{(k)} + \alpha^{(k)} r^{(k)}$.

On cherche $\alpha^{(k)}$ tel que $x^{(k)} + \alpha^{(k)} r^{(k)}$ minimise f le long de la droite d'origine $x^{(k)}$ et de vecteur directeur $r^{(k)}$. Pour ce faire, on va annuler la dérivée de f le long de cette droite : on veut que le produit scalaire du gradient par la direction de recherche s'annule en $x^{(k+1)}$.

Ici, la direction de recherche $d^{(k)}$ est égale à $r^{(k)}$, mais l'on va garder des notations différentes pour en préserver la sémantique, ce qui sera utile par la suite pour unifier les différentes approches.

$$\begin{aligned}
d^{(k)\top} r^{(k+1)} &= 0 \\
d^{(k)\top} (b - Ax^{(k+1)}) &= 0 \\
d^{(k)\top} (b - A(x^{(k)} + \alpha^{(k)} r^{(k)})) &= 0 \\
\alpha^{(k)} &= \frac{d^{(k)\top} (Ax^{(k)} - b)}{d^{(k)\top} Ar^{(k)}} \\
\alpha^{(k)} &= \frac{d^{(k)\top} r^{(k)}}{d^{(k)\top} Ar^{(k)}}
\end{aligned}$$

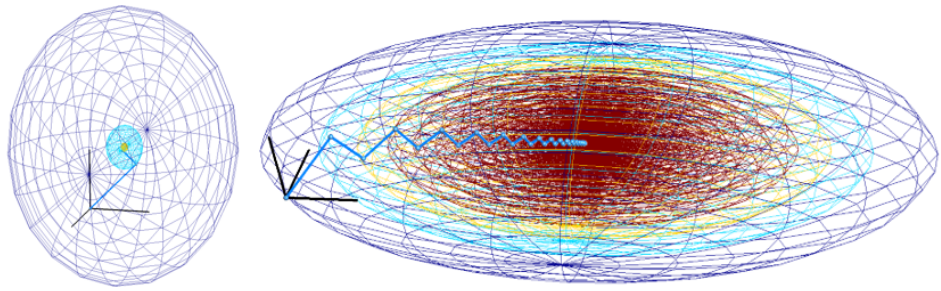


FIGURE 3.9 – *En 3D, résolution par descente de gradient avec une matrice A diagonale. On observe une convergence extrêmement rapide lorsque A est proche de l'identité (gauche), et une convergence particulièrement lente lorsque ses éléments diagonaux sont* $0.5, 1$ *et* 2 *(droite).*

Convergence : quand s'arrêter ?

Il n'y a pas de solution miraculeuse. On peut s'arrêter lorsque la norme du gradient devient trop petite, ou que la différence $f(x^{(k)}) - f(x^{(k+1)})$ devient trop petite.

Les figures 3.8 et 3.9 montrent dans les cas $2D$ et $3D$ comment évolue la solution. Quand toutes les valeurs propres de A sont égales, l'algorithme converge en une itération, mais lorsqu'elles sont très différentes la vitesse de convergence s'effondre. Le ratio maximal entre les valeurs propres s'appelle le conditionnement de la matrice, et reflète bien la difficulté du problème.

En résumé : La descente de gradient est très simple, et a une convergence très rapide lors des premières itérations. Par contre, son comportement est très sensible au conditionnement de la matrice par la suite.

3.2.3 Approche par conjugaison

On va oublier un instant la formulation par minimisation d'une forme quadratique, et on va chercher une méthode de projection qui résolve $Ax = b$. Plus précisément, on va faire des projections orthogonales dans l'espace transformé par M en utilisant les produits scalaires que l'on sait calculer sans connaître M. Plus précisément, on va utiliser le calcul des projections illustrées par la figure 3.7.

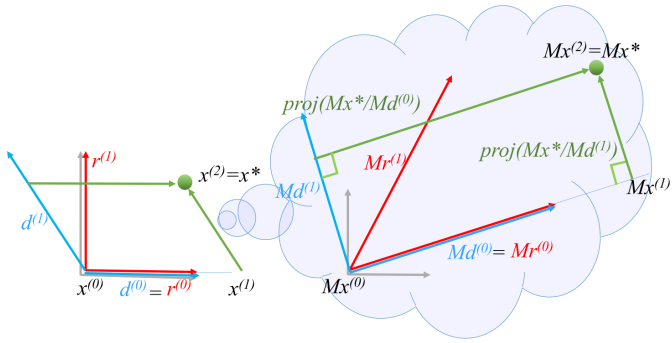


FIGURE 3.10 – *Algorithme de conjugaisons 2D. On définit une base de vecteurs* $r^{(k)}$*, puis on définit* $d^{(0)} \leftarrow r^{(0)}$ *et* $d^{(1)} \leftarrow M^{-1}(Mr^{(1)} - proj(Mr^{(1)}/Md^{(0)}))$*. On trouve ensuite les coordonnées de* x^* *dans la base des* $d^{(k)}$ *avec* $x^* \leftarrow \Sigma_k M^{-1} proj(Mx^*/Md^{(k)})$*.*

Algorithme naïf avec directions conjuguées

Dans l'espace M, on sait projeter la solution sur des vecteurs. Ainsi, dans M, il suffirait de projeter la solution Mx^* sur une base orthogonale $Md^{(k)}$ pour avoir la solution, comme illustré par la figure 3.10. On va construire les $d^{(k)}$ tels que les $Md^{(k)}$ forment une base orthogonale. Pour ce faire, on utilise l'algorithme de Gram-Schmidt qui construit les $Md^{(k)}$ les uns après les autres, et qui n'a besoin que de calculer des produits scalaires. On exploite alors le fait de pouvoir calculer les produits scalaires dans M sans connaître M pour construire les $d^{(k)}$ et non pas les $Md^{(k)}$. Ensuite, on décompose Mx^* dans les $Md^{(k)}$ et, par linéarité de M, on en déduit x^* en fonction des $d^{(k)}$.

— **Construction d'une base de vecteurs** $Md^{(k)}$.
Prenons une base de vecteurs $r^{(k)}$ linéairement indépendants : par exemple $r_i^{(k)} = \begin{cases} 1 & \text{si } i = k \\ 0 & \text{si } i \neq k \end{cases}$

Comme M est strictement définie positive, les $Mr^{(k)}$ sont linéairement indépendants.
On initialise $d^{(0)} \leftarrow r^{(0)}$.

Pour assurer l'orthogonalité des images des $d^{(k)}$ dans M, on utilise l'algorithme de Gram-Schmidt : on va construire chaque $Md^{(k+1)}$ à partir de $Mr^{(k+1)}$ en lui retirant à chaque étape sa projection sur l'espace vectoriel engendré par les $Md^{(k)}$. Ce calcul est réalisé comme dans la figure 3.7 :

On rappelle que la projection d'un vecteur v sur un axe représenté par le vecteur u est un vecteur de direction u et de norme $v\cos(\angle(u,v))$. On peut le calculer avec deux produits scalaires : $proj(v,u) = \frac{u^\top v}{u^\top u}u$. Ici, on utilise cette équation avec $v = Mr^{(k+1)}$ et $u = Md^{(i)}$ pour projeter la nouvelle direction sur les précédentes et ainsi décomposer $Mr^{(k+1)}$ sur les $Md^{(i)}$.

$$Md^{(k+1)} = Mr^{(k+1)} - \sum_{i=0}^{k} \frac{(Mr^{(k+1)})^\top Md^{(i)}}{(Md^{(i)})^\top Md^{(i)}} Md^{(i)} \quad (3.9)$$

$$= Mr^{(k+1)} - \sum_{i=0}^{k} \frac{r^{(k+1)\top} Ad^{(i)}}{d^{(i)\top} Ad^{(i)}} Md^{(i)} \quad (3.10)$$

Cette relation est intéressante, mais nos inconnues sont les $d^{(k)}$, et non pas les $Md^{(k)}$. Du coup, on multiplie de part et d'autre de l'égalité par M^{-1} pour construire les $d^{(k)}$ (voir figure 3.7).

$$d^{(k+1)} \leftarrow r^{(k+1)} - \sum_{i=0}^{k} \frac{r^{(k+1)\top} Ad^{(i)}}{d^{(i)\top} Ad^{(i)}} d^{(i)}. \quad (3.11)$$

C'est cette formule qui va être utilisée dans l'algorithme car elle ne fait plus apparaître la matrice M que l'on ne connaît pas, mais seulement A et les $d^{(i)}$ précédents.

— **On projette Mx^* dans cette base**
Comme précédemment, on utilise la formule de projection n'utilisant que deux produits scalaires :

$$Mx^* = \sum_{k=0}^{n} \frac{(Mx^*)^\top Md^{(k)}}{(Md^{(k)})^\top Md^{(k)}} Md^{(k)} \quad (3.12)$$

$$= \sum_{k=0}^{n} \frac{x^* Ad^{(k)}}{d^{(k)\top} Ad^{(k)}} Md^{(k)} \quad (3.13)$$

$$= \sum_{k=0}^{n} \frac{b^\top d^{(k)}}{d^{(k)\top} Ad^{(k)}} Md^{(k)}. \quad (3.14)$$

Encore une fois, on ne peut pas réaliser ce calcul puisque l'on ne connaît pas M, mais on peut trouver x^* directement en multipliant la formule de part et d'autre par M^{-1}, puis en la simplifiant :

$$x^* \leftarrow \sum_{k=0}^{n} \frac{b^\top d^{(k)}}{d^{(k)\top} Ad^{(k)}} d^{(k)} \quad (3.15)$$

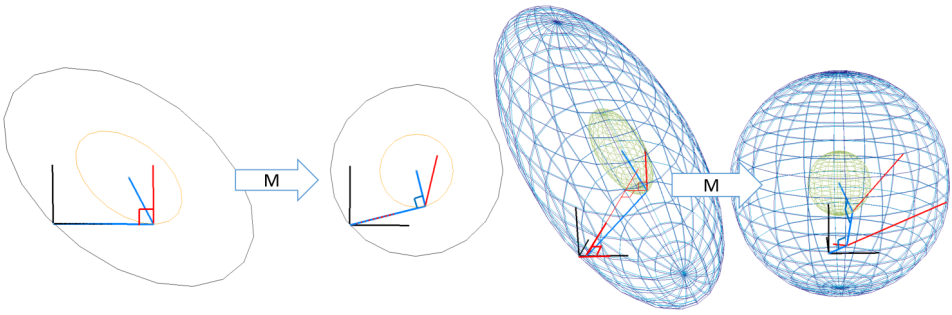


FIGURE 3.11 – *Les directions $r^{(k)}$ (rouge) sont orthogonales, et les directions $Md^{(k)}$ (bleu) le sont dans l'espace déformé par M.*

Changeons l'ordre : Il n'est pas nécessaire de calculer tous les $r^{(k)}$ avant de commencer à projeter Mx^* sur les $Mr^{(k)}$. Pour rendre l'algorithme de conjugaison plus proche de celui du gradient conjugué, nous allons regrouper les deux boucles en une seule qui calculera les $r^{(k)}$ au fur et à mesure :

Algorithme par conjugaison :

— $x^{(0)} \leftarrow 0, d^{(0)} \leftarrow r^{(0)}$

— Pour k de 0 à n :

  — $x^{(k+1)} \leftarrow x^{(k)} + \frac{b^\top d^{(k)}}{d^{(k)\top} A d^{(k)}} d^{(k)}$

  — $d^{(k+1)} \leftarrow r^{(k+1)} - \sum_{i=0}^{k} \frac{r^{(k+1)\top} A d^{(i)}}{d^{(i)\top} A d^{(i)}} d^{(i)}$

Attention : *Il n'est pas possible de calculer $d^{(n)}$ car $r^{(n)}$ n'existe pas… donc il suffit de quitter l'algorithme dès que l'on a la solution $x^* = x^{(n)}$.*

En résumé : On peut résoudre $Ax = b$ en construisant une base orthogonale dans M et en projetant l'image de la solution dessus… juste en faisant des produits scalaires dans M.

Propriétés importantes

— De par la construction avec l'algorithme de Gram-Schmidt, on va pouvoir déduire que des vecteurs sont orthogonaux aux $r^{(i)}$ ($\forall i < k$) s'ils sont orthogonaux aux $d^{(i)}$ et vice et versa. De plus, cela fonctionne aussi avec les $Mr^{(i)}$ et les $Md^{(i)}$.

  — $\boxed{D^{(k)} = \mathbf{span}\{r^{(k)}\} = \mathbf{span}\{d^{(k)}\}}$ On peut le prouver par récurrence car $d^{(k)}$ est une combinaison linéaire de $r^{(k)}$ et $d^{(i)}$ avec $i \leqslant k$.

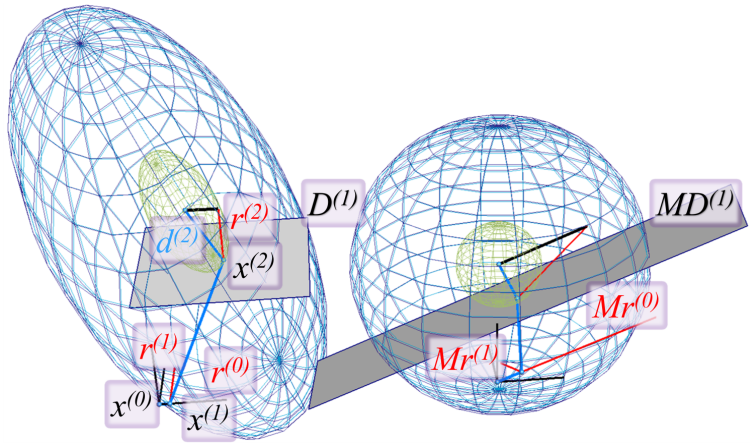


FIGURE 3.12 – *Le span $D^{(1)}$ de $r^{(0)}$ et $r^{(1)}$ (et donc de $d^{(0)}$ et $d^{(1)}$) est orthogonal à $r^{(2)}$. De même, on voit que $MD^{(1)}$, son image par M, est orthogonale à $Md^{(2)}$. On remarque aussi que $x^{(2)} - (x^{(1)} + r^{(1)}) \in D^{(2)}$ i.e., le segment $x^{(2)}, x^{(1)} + r^{(1)}$ (en noir) est parallèle à $D^{(2)}$.*

- — $\boxed{r^{(i)}, d^{(i)}, x^{(i+1)} \in D^{(k)}, \forall i \leqslant k}$ Ils sont tous calculés comme combinaisons linéaires des $r^{(i)}$ avec $i \leqslant k$.
- — Les directions de descente originales $Mr^{(k)}$ et celles modifiées par Gram-Schmidt $Md^{(k)}$ ont les même produits scalaires avec $Md^{(k)}$ (première propriété). Symétriquement, si les $r^{(k)}$ sont orthogonaux, on obtient une relation similaire (seconde propriété) dans l'espace non transformé par M. Cette dernière propriété est nécessaire pour rendre le gradient conjugué efficace.
  - — $\boxed{r^{(k)\top} A d^{(k)} = d^{(k)\top} A d^{(k)}}$ Pour construire $Md^{(k)}$ (avec Gram-Schmidt), on a retiré à $Mr^{(k)}$ un vecteur de $MD^{(k-1)}$, qui ne change donc pas dans le produit scalaire avec $Md^{(k)}$.
  - — $\boxed{r^{(k)\top} r^{(i)} = 0, \forall i \neq k \Rightarrow r^{(k)\top} d^{(k)} = r^{(k)\top} r^{(k)}}$ Pour construire $d^{(k)}$ (avec Gram-Schmidt), on a retiré à $r^{(k)}$ un vecteur de $D^{(k-1)}$, qui ne change donc pas dans le produit scalaire avec $r^{(k)}$ si $r^{(k)} \perp D^{(k-1)}$.

Analyse de performance. Cet algorithme est en $O(n^3)$ en temps d'exécution et en espace de stockage à cause de la somme des projections sur les directions précédentes dans Gram-Schmidt, ce qui est regrettable.

En résumé : Cet algorithme ne doit pas être implémenté car il est très inefficace. Il permet cependant d'énoncer des propriétés indépendantes du choix des $r^{(k)}$ et qui seront très utiles pour le gradient conjugué.

3.2.4 Le Gradient conjugué

Le gradient conjugué diffère de l'algorithme précédent par sa façon de construire les $r^{(k)}$. Au lieu de prendre la base naturelle, on définit à chaque itération $r^{(k)} = b - Ax^{(k)}$. Ce choix a deux avantages : il annule la plupart des termes de l'équation 3.11 et permet à l'algorithme de converger numériquement bien avant d'atteindre la n^{ieme} itération.

On peut réécrire l'algorithme comme suit :

Adaptation directe	Adaptation optimisée
$x^{(0)} \leftarrow 0$	$x^{(0)} \leftarrow 0$
$r^{(0)} \leftarrow b - Ax^{(0)}$	$r^{(0)} \leftarrow b - Ax^{(0)}$
$d^{(0)} \leftarrow r^{(0)}$	$d^{(0)} \leftarrow r^{(0)}$
Pour k de 0 à $n-1$:	Pour k de 0 à $n-1$:
$\alpha^{(k)} \leftarrow \frac{b^\top d^{(k)}}{d^{(k)\top} A d^{(k)}}$	$\alpha^{(k)} \leftarrow \frac{r^{(k)\top} r^{(k)}}{d^{(k)\top} A d^{(k)}}$
$x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} d^{(k)}$	$x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} d^{(k)}$
$r^{(k+1)} \leftarrow b - Ax^{(k+1)}$	$r^{(k+1)} \leftarrow r^{(k)} - \alpha^{(k)} A d^{(k)}$
$\beta_i^{k+1} \leftarrow \frac{r^{(k+1)\top} A d^{(i)}}{d^{(i)\top} A d^{(i)}}, \forall i$	$\beta^{(k+1)} \leftarrow -\frac{r^{(k+1)\top} r^{(k+1)}}{r^{(k)\top} r^{(k)}}$
$d^{(k+1)} \leftarrow r^{(k+1)} - \sum_{i=0}^{k} \beta_i^{(k+1)} d^{(k)}$	$d^{(k+1)} \leftarrow r^{(k+1)} - \beta^{(k+1)} d^{(k)}$

Pour obtenir la version optimisée, il faut tout d'abord énoncer quelques propriétés importantes, à partir desquelles on pourra simplifier les calculs.

Propriétés importantes

Définition récursive des $r^{(k)}$. $\boxed{r^{(k+1)} = r^{(k)} + \alpha^{(k)} A d^{(k)}}$

Par définition $r^{(k+1)} = b - Ax^{(k+1)}$. Or $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$, donc $r^{(k+1)} = b - A(x_i + \alpha^{(k)} d^{(k)}) = r^{(k)} + \alpha^{(k)} A d^{(k)}$.

Le pas optimal peut être calculé par descente ou par la conjugaison. $\boxed{\alpha^{(k)} = \frac{d^{(k)\top} r^{(k)}}{d^{(k)\top} A r^{(k)}} = \frac{b^\top d^{(k)}}{d^{(k)\top} A d^{(k)}}}$

On calcule le pas $\alpha^{(k)}$ de sorte à ce que le gradient au nouveau point soit orthogonal à la direction de recherche. Pour la descente de gradient on cherche $d^{(k)} \perp r^{(k+1)}$, et pour la méthode par conjugaison, on cherche $Md^{(k)} \perp Mr^{(k+1)}$. Comme placer $x^{(k+1)}$ au minimum de $|Ax - b|^2$ est équivalent à placer $Mx^{(k+1)}$ au minimum de $|Ax - b|^2$ transformé par M, on obtient le même $\alpha^{(k)}$ dans les deux cas.

Nous avons déjà établi que $d^{(k)}Ar^{(k)} = d^{(k)\top}Ad^{(k)}$ (égalité des dénominateurs). Sachant que $x^{(k)} \in D^{(k-1)}$, on retrouve l'égalité des numérateurs $r^{(k)\top}d^{(k)} = (b - Ax^{(k)}t)d^{(k)} = b^\top d^{(k)}$.

Orthogonalité des $r^{(k)}$. $\boxed{r^{(k)}r^{(i)} = 0, \forall k < i}$

Les $r^{(k)}$ sont orthogonaux. Nous allons démontrer que $r^{(k)}r^{(i)} = 0$, $\forall i < k$ par récurrence sur les k.

— Pour $k = 1$, la propriété est vraie par construction $r^{(0)}r^{(1)} = 0$, lors du calcul de $\alpha^{(1)}$.
— Supposons que $r^{(i)}r^{(k)} = 0, \forall i < k$.
— Démontrons que $r^{(i)}r^{(k+1)} = 0, \forall i < k + 1$.
Notons que $r^{(i)}r^{(k+1)} = r^{(i)}r^{(k)} - \alpha^{(k)}r^{(i)}Ad^{(k)}$.

  — **Cas** $i < k$: Le premier terme est nul par l'hypothèse de la récurrence. Le second terme est nul car $r^{(i)}Ad^{(k)} = d^{(i)}Ad^{(k)}$ et que les $d^{(i)}$ sont orthogonaux.

  — **Cas** $i = k$: On a $r^{(k)}r^{(k+1)} = 0$ de part la construction de $\alpha^{(k)}$ (dans le cas de la descente de gradient).

En résumé : Les $r^{(k)}$ sont orthogonaux deux à deux.

Simplifications

Les simplifications suivantes utilisent la nouvelle définition des $r^{(k)}$ (sinon on les aurait faites dans la méthode de conjugaison sans le gradient).

Les deux premières simplifications sont simples et n'affectent que légèrement les performances : la première remplace $b^\top d^{(k)}$ par $r^{(k)\top}r^{(k)}$ (il faut prendre la définition des $\alpha^{(k)}$ obtenus pour la descente de gradient, et utiliser la propriété $d^{(k)\top}r^{(k)} = r^{(k)\top}r^{(k)}$), tandis que la seconde calcule $r^{(k+1)}$ par sa définition récursive.

La dernière simplification est fondamentale car elle fait passer chaque itération en $O(n)$ au lieu de $O(n^2)$ (avec des matrices creuses) : elle utilise l'orthogonalité des $r^{(k)}$ pour simplifier l'écriture des $\beta_j^{(k+1)}$.

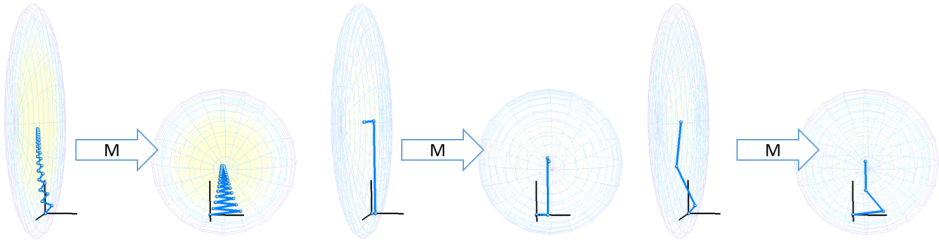


FIGURE 3.13 – *Évolution des $x^{(k)}$ avec une descente de gradient, la méthode de projection de l'erreur dans M, et le gradient conjugué.*

$$
\begin{aligned}
\text{Par définition : } \beta_i^{(k+1)} &= \frac{r^{(k+1)\top} A d^{(i)}}{d^{(i)\top} A d^{(i)}} \\
\text{D'après la définition récursive des } r^{(k)} : Ad^{(i)} &= \frac{r^{(i)} - r^{(i+1)}}{\alpha^{(i)}} \\
\text{Donc : } \beta_i^{(k+1)} &= \frac{r^{(k+1)\top}\left(r^{(i)} - r^{(i+1)}\right)}{\alpha^{(i)} d^{(i)\top} A d^{(i)}} \\
&= \frac{r^{(k+1)\top} r^{(i)} - r^{(k+1)\top} r^{(i+1)}}{\alpha^{(i)} d^{(i)\top} A d^{(i)}} \\
\text{Par orthogonalité des } r^{(k)}\text{, si } i \neq k\text{, on a } \beta_i^{(k+1)} &= \frac{-r^{(k+1)\top} r^{(i+1)}}{\alpha^{(i)} d^{(i)\top} A d^{(i)}} \\
\text{En utilisant } \alpha^{(i)} &= \frac{r^{(i)\top} r^{(i)}}{d^{(i)\top} A d^{(i)}} \\
\text{On a } \beta_i^{(k+1)} &= \frac{-r^{(k+1)\top} r^{(i+1)}}{r^{(i)\top} r^{(i)}}
\end{aligned}
$$

Par orthogonalité des $r^{(k)}$, $\beta_i^{(k+1)} = 0$ si $k \neq i$, ce qui permet de simplifier les notations en omettant le i : on utilise $\beta^{(k+1)}$ au lieu de $\beta_k^{(k+1)}$ car tous les autres $\beta_i^{(k+1)}$ sont nuls.

En résumé : Grâce à l'orthogonalité des $r^{(k)}$ il suffit de calculer un β par itération.

Performances

En l'absence d'erreurs d'arrondi, le gradient conjugué converge en n itérations, et chaque itération ne requiert qu'une multiplication d'une

matrice par un vecteur et cinq multiplications d'un vecteur par un autre. En pratique, le gradient conjugué est utilisé sur des grandes matrices creuses comme un solveur itératif qui est moins sensible au conditionnement de la matrice que la descente de gradient.

Il est intéressant d'observer le comportement du gradient conjugué sur ce type de matrices. Nous allons voir deux cas simples : un Laplacien 1D et un Laplacien 2D discrétisé sur une grille régulière. Bien que simples, ces choix montrent l'impact d'un facteur prédominant pour le comportement des solveurs : la disposition des zéros dans la matrice, qui correspond aux voisinages géométriques dans l'espace discrétisé.

Les matrices creuses définissent des graphes qui relient les indices i et j des coefficients a_{ij} non nuls. En modélisation par éléments finis (FEM) et en *geometry processing*, ces graphes sont très proches de la structure du maillage sur lequel on travaille : en effet, les équations aux dérivées partielles vont par définition lier des éléments géométriquement proches. Les deux Laplaciens sont des exemples 1D et 2D qui permettent de voir comment se comportent les algorithmes que nous venons de présenter avec ce type de données.

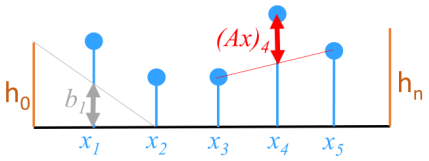
Laplacien 1D. Nous représentons une fonction f par le vecteur x tel que $f(i) = x_i$. Ce que l'on appelle Laplacien 1D ici est un système linéaire qui donne en chaque point de la fonction sa dérivée seconde, estimée par différences finies. Le problème est de trouver une fonction (un vecteur x) dont cette dérivée seconde est nulle, et qui respecte des contraintes au bord, comme illustré dans la figure 3.14.

Ce problème est très intéressant car il est relativement simple à analyser. De plus, on peut visualiser le comportement des différents algorithmes sur une seule image (figure 3.15) : il suffit d'afficher les $x_i^{(k)}$ en fonction de la position géométrique i et de l'itération k. De plus, ce problème est extrêmement complexe à résoudre car le conditionnement du système est très mauvais : pour $n = 5$ il est de $13.6 = 1.9/0.14$ et pour $n = 20$, il vaut $200 = 2/.01$. Géométriquement, cela correspond à vouloir résoudre un problème dont les iso-valeurs de $x^\top Ax - 2b^\top x$ sont des ellipses dont les axes ont des rapports de taille de 13.6 et 200 !

Pour chaque algorithme, on observe les comportements suivants (figure 3.15) :

— La descente de gradient minimise rapidement la dérivée seconde près des contraintes, mais peine à propager les contraintes vers l'intérieur du domaine.

— La méthode par projection propage la contrainte de gauche au fur



$$A = \begin{pmatrix} 1 & -1/2 & 0 & 0 & 0 \\ -1/2 & 1 & -1/2 & 0 & 0 \\ 0 & -1/2 & 1 & -1/2 & 0 \\ 0 & 0 & -1/2 & 1 & -1/2 \\ 0 & 0 & 0 & -1/2 & 1 \end{pmatrix} ,b= \begin{pmatrix} h_0/2 \\ 0 \\ 0 \\ h_n/2 \end{pmatrix}$$

FIGURE 3.14 – *Problème de Laplacien 1D. On a une fonction définie en n points (ici, $n = 5$) et cherche à minimiser la dérivée seconde estimée par différences finies i.e., $\Sigma_i x_i - (x_{i-1} + x_{i+1})/2$. Au bord, on fixe des valeurs h_0 et h_n, ce qui donne le problème $Ax - b = 0$ avec A et b définis dans la figure.*

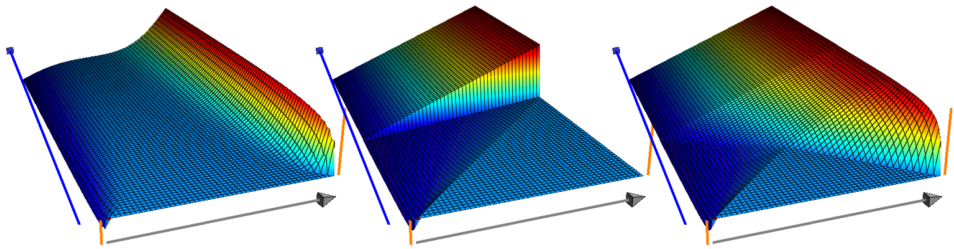


FIGURE 3.15 – *Évolution de la solution du Laplacien 1D avec $n = 20$: l'axe gris représente la fonction échantillonnée par x, et l'axe bleu est celui des itérations, et les segments oranges sont les contraintes du bord. Les trois solveurs sont (de gauche à droite) une descente de gradient, une conjugaison et un gradient conjugué.*

et à mesure, mais ne tient compte de celle de droite qu'à la dernière itération. Ce comportement est certes dû au choix des $r^{(k)}$, mais illustre bien la nécessité de faire l'ensemble des itérations.

— Le gradient conjugué se comporte comme s'il ne voyait qu'une région autour des contraintes (définies au bord, en orange), sur laquelle il résout parfaitement le problème. Cette région grandit à chaque itération pour recouvrir tout le domaine à partir de la moitié des itérations, mais il faut attendre les n itérations pour que chacune des deux contraintes ait eu le temps d'influencer l'ensemble du domaine.

Laplacien 2D. Le Laplacien 2D fonctionne sur le même principe que le Laplacien 1D. On a discrétisé une fonction f sur une grille 2d et à chaque nœud de la grille on associe une coordonnée de x. Le Laplacien est estimé par différence finie, donc la matrice A est définie par : $a_{ii} = 1$, $a_{ij} = -1/4$ si les nœuds i et j sont voisins, et $a_{ij} = 0$ sinon.

Dans le cas 2D, on ne peut plus représenter la simulation sur une seule

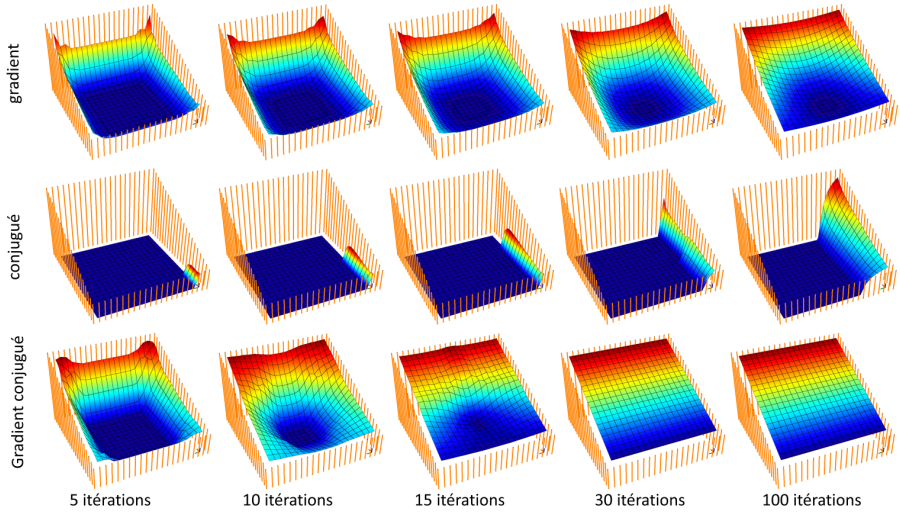


FIGURE 3.16 – *Évolution de la solution du Laplacien 2D sur une grille de* 20×20. *La solution x est affichée par la hauteur de la fonction, et les contraintes de bords sont données par les segments oranges.*

image, alors nous affichons l'état à certaines itérations. Grâce à la figure 3.16, on peut analyser le comportement des différents algorithmes :

- Descente de gradient : Comme avec le Laplacien 1D, on observe que les premières itérations lissent bien la fonction, mais que le solveur peine à propager les contraintes vers l'intérieur du domaine. Ceci étant, en 100 itérations, la convergence sur cette grille de 20×20 semble similaire à celle observée sur la grille 1D de $n = 20$. La difficulté ne semble pas être due aux 400 variables, mais plutôt à la distance aux contraintes (qui est à peu près la même en 1D et en 2D).
- Conjugaison : le solveur règle les x_i les uns après les autres, et n'a pas convergé car il n'y a pas eu les 400 itérations nécessaires.
- Gradient conjugué : il converge en moins de 30 itérations, ce qui est donc loin des 400 itérations nécessaires (méthode directe) pour assurer la convergence. Ce comportement s'explique par le temps nécessaire à chaque contrainte pour influencer l'ensemble du domaine.

Conclusion. La difficulté à résoudre des problèmes dans lesquels A est une matrice creuse est fortement liée à la distance maximale (par rapport au graphe d'adjacence défini par A) à laquelle une contrainte peut influencer

la solution. De fait, pour un nombre similaire de variables, les problèmes 1D paraissent plus difficiles que les problèmes 2D.

Récapitulatif

On a présenté le gradient conjugué comme l'unification de deux méthodes (gradient et conjugaison). On a aussi vu ce qui rend cette méthode si efficace, et on en a observé le comportement sur des problèmes classiques dans lesquels A est une matrice creuse. Récapitulons le contenu :

- **Objectif :** Résoudre $Ax = b$ pour une matrice A définie positive.
- **Descente de gradient :** Résoudre $Ax - b = 0$ équivaut à minimiser $x^\top Ax - 2b$
  - On peut descendre le gradient avec un pas optimal.
  - Méthode itérative, extrêmement dépendante du conditionnement de A.
- **Conjugué :** Il existe M telle que $M^\top M = A$.
  - On sait modifier une famille de vecteurs $r^{(k)}$ pour construire une autre famille $d^{(k)}$, dont les images par M sont orthogonales, à savoir $Md^{(k)}.Md^{(i)} = 0$;
  - On sait décomposer x^* sur les $d^{(k)}$ en projetant Mx^* sur les $Md^{(k)}$;
  - Méthode directe : requiert n itérations, faisant chacune $O(n)$ multiplications matrice/vecteur.
- Le **Gradient Conjugué** (GC) peut être obtenu en modifiant l'une ou l'autre des méthodes :
  - Le GC est une descente de gradient dont on corrige la direction de descente pour ne pas retourner dans une direction déjà optimisée ;
  - Le GC est un algorithme de conjugaison dont les $r^{(k)}$ sont soigneusement choisis (le gradient) pour se rapprocher au plus vite de la solution ;
  - Le GC est donc une méthode directe et itérative **à la fois**, qui converge généralement en peu d'itérations ;
  - C'est un algorithme efficace car à l'étape k le gradient est déjà orthogonal (dans M) à tous les $d^{(i)}$, sauf $d^{(k-1)}$. Cela permet de faire un nombre constant de multiplications matrice/vecteur par itération.

3.3 Paramétrisation de surfaces triangulées

Dans cette section, nous introduisons la paramétrisation de maillages, un problème courant en géométrie numérique.

Intuitivement, une paramétrisation d'une surface 3D est un « dépliage » de cette surface sur un espace 2D. Cette représentation en 2D a de nombreuses applications, la plus évidente étant le plaquage de textures. Dans cette section, nous présentons les notions de bases en topologie, qui permettent de caractériser la classe de surfaces admettant un tel « dépliage » §3.3.2. Nous considérerons ensuite le cas particulier d'un disque topologique dont le bord a été mis en correspondance avec un polygone convexe 2D. Dans ce cas particulier, le théorème du plongement barycentrique de Tutte donne une condition suffisante pour qu'une paramétrisation soit valide §3.3.3. D'autre part, il est relativement facile de déduire de ce théorème un algorithme permettant de construire une telle paramétrisation §3.3.4. Nous décrirons ici le principal argument de l'élégante preuve du théorème de Tutte élaborée par Gortler, Gotsman et Thurston. De manière remarquable, leur preuve n'utilise que des notions de base en topologie ainsi qu'un simple argument de comptage.

Introduction

Avant de présenter une définition formelle, nous donnons tout d'abord une idée intuitive de la paramétrisation d'une surface. Étant donnée une surface triangulée, figure 3.17 (à gauche), une paramétrisation « déplie » le maillage en 2D, figure 3.17 (à droite). Un tel « dépliage » est une représentation intéressante, qui peut être considérée en quelque sorte comme une « carte » de l'objet. Cette « carte » peut être utilisée par diverses applications.

En restant toujours à un niveau intuitif, si $\mathbf{p}_i = (x_i, y_i, z_i)$ dénote le sommet du maillage d'indice i (avec $1 \leqslant i \leqslant n_v$ où n_v dénote le nombre de sommets), une paramétrisation d'un maillage est un ensemble de deux coordonnées supplémentaires $\mathbf{u}_i = (u_i, v_i)$ associées à chaque sommet du maillage.

De plus, pour définir une paramétrisation, ces coordonnées doivent être telles qu'il n'existe aucune intersection entre les triangles en 2D. Nous allons à présent aborder une méthode permettant de calculer des coordonnées (u_i, v_i) ayant cette propriété. Avant d'entrer dans le vif du sujet, nous présenterons des applications de la paramétrisation de maillages. Nous supposons pour le moment que les coordonnées (u_i, v_i) sont données, nous verrons plus loin comment les calculer.

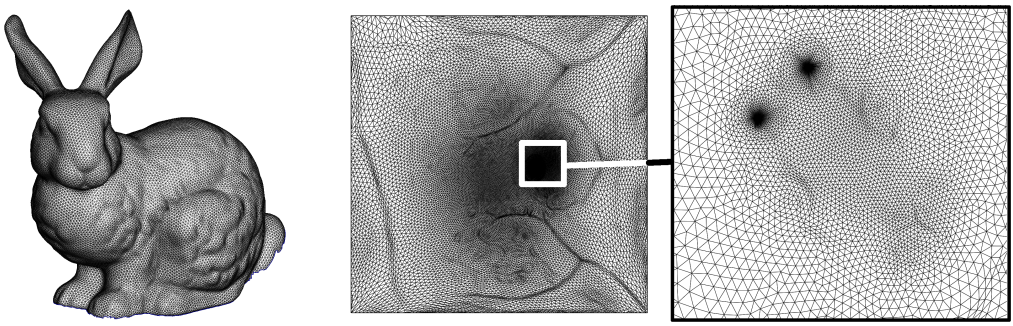


FIGURE 3.17 – *À gauche : une surface triangulée en 3D (crédit : Université de Stanford). À droite : une paramétrisation 2D de la même surface triangulée, et un zoom dans la région de la tête.*

3.3.1 Applications de la paramétrisation de maillages

Le plaquage de textures, largement utilisé dans les industries cinématographique et du jeu vidéo, est l'une des principales applications de la paramétrisation de maillages. L'idée, évoquée sur la figure 3.18, consiste en quelque sorte à « emballer » le maillage dans une image. Le plaquage de textures utilise une bijection entre le maillage 3D et le domaine 2D. Ce dernier est alors appelé *espace paramétrique* ou encore *espace de texture* dans ce contexte particulier. Une telle bijection peut être définie à partir de l'interpolation linéaire des coordonnées (u_i, v_i) associées à chaque sommet de la triangulation.

Considérons un point 2D $\mathbf{u} = (u, v)$ de l'espace paramétrique et cherchons à quel point 3D il correspond. Si le point $\mathbf{u}$ est un sommet $\mathbf{u}_i = (u_i, v_i)$, alors il correspond au point 3D $\mathbf{p}_i = (x_i, y_i, z_i)$. Maintenant si le point $\mathbf{u}$ est un point 2D arbitraire, alors son homologue 3D peut être défini comme le résultat de la procédure suivante :

1. déterminer le triangle 2D de sommets $\mathbf{u}_i = (u_i, v_i), \mathbf{u}_j = (u_j, v_j), \mathbf{u}_k = (u_k, v_k)$ qui contient le point $\mathbf{u} = (u, v)$;
2. calculer les *coordonnées barycentriques* $\lambda_1(\mathbf{u}), \lambda_2(\mathbf{u}), \lambda_3(\mathbf{u})$ du point $\mathbf{u}$ par rapport au triangle $[\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k]$;
3. enfin, $\mathbf{u}$ est mis en correspondance avec $\lambda_1(\mathbf{u})\mathbf{p}_i + \lambda_2(\mathbf{u})\mathbf{p}_j + \lambda_3(\mathbf{u})\mathbf{p}_k$.

Les coordonnées barycentriques d'un point $\mathbf{u}$ par rapport au triangle $[\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k]$ sont les uniques [5] nombres réels $\lambda_1(\mathbf{u})$,$\lambda_2(\mathbf{u})$,$\lambda_3(\mathbf{u})$ tels que :

5. si le triangle n'est pas dégénéré.

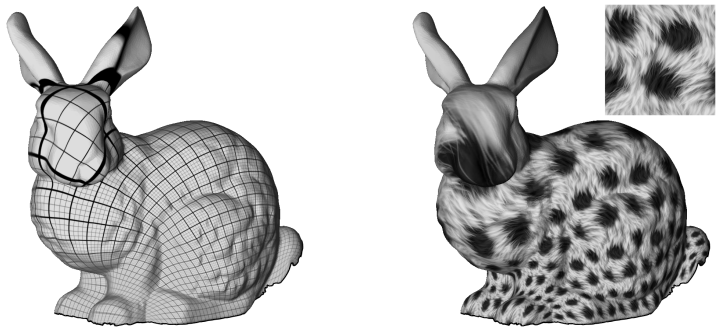


FIGURE 3.18 – *Une paramétrisation d'un maillage peut être utilisée pour « emballer » un objet 3D dans une image. L'image utilisée est une grille régulière (à gauche) et une texture de « fourrure » (à droite). Crédits : le modèle 3D provient de l'Université de Stanford, et la texture de fourrure de* `www.myfreetextures.com` *(sous licence Creative Commons).*

$$\begin{cases} \lambda_1(\mathbf{u})\mathbf{u}_i + \lambda_2(\mathbf{u})\mathbf{u}_j + \lambda_3(\mathbf{u})\mathbf{u}_k &= \mathbf{u} \\ \lambda_1(\mathbf{u}) + \lambda_2(\mathbf{u}) + \lambda_3(\mathbf{u}) &= 1 \end{cases}$$

Ainsi, les coordonnées barycentriques sont solutions du système linéaire suivant :

$$\begin{pmatrix} u_i & u_j & u_k \\ v_i & v_j & v_k \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1(\mathbf{u}) \\ \lambda_2(\mathbf{u}) \\ \lambda_3(\mathbf{u}) \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

qui se résout, par exemple, en inversant la matrice grâce à la formule de Cramer :

$$\begin{pmatrix} \lambda_1(\mathbf{u}) \\ \lambda_2(\mathbf{u}) \\ \lambda_3(\mathbf{u}) \end{pmatrix} = \frac{1}{2A(\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k)} \begin{pmatrix} v_j - v_k & u_k - u_j & \begin{vmatrix} u_i & u_j \\ v_i & v_j \end{vmatrix} \\ v_k - v_i & u_i - u_k & \begin{vmatrix} u_j & u_k \\ v_j & v_k \end{vmatrix} \\ v_i - v_j & u_j - u_i & \begin{vmatrix} u_k & u_i \\ v_k & v_i \end{vmatrix} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

où $A(\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k)$ dénote l'aire du triangle 2D $[\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_k]$.

Réciproquement, pour un point 3D arbitraire $\mathbf{p} = (x, y, z)$ sur le maillage, contenu par le triangle 3D $t = [\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k]$, nous pouvons calculer les coordonnées barycentriques $\lambda_1(\mathbf{p}), \lambda_2(\mathbf{p}), \lambda_3(\mathbf{p})$ de $\mathbf{p}$ par rapport à t en utilisant la même formule appliquée aux coordonnées 2D dans une

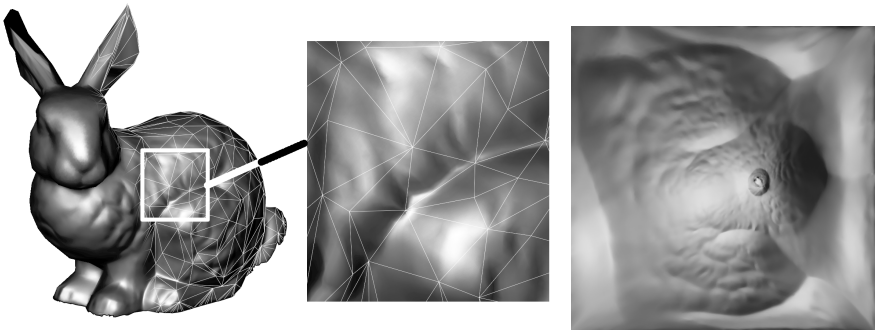


FIGURE 3.19 – *Une application de la paramétrisation de maillages : le plaquage de normales. Un maillage grossier (à gauche, à comparer avec celui de la figure 3.17) est muni d'une texture qui encode les détails géométriques fins du maillage original à haute résolution (à droite).*

base du plan support de t. Il est possible de vérifier que cette procédure définit l'inverse de l'application $(u, v) \to (x, y, z)$ définie précédemment.

En utilisant cette interpolation linéaire, si les $\mathbf{u}_i$ sont tels que les triangles n'ont pas d'intersection en 2D, alors nous définissons ainsi une correspondance bijective entre tous les points 3D de la surface et tous les points 2D du domaine paramétrique. Cette correspondance peut être utilisée par de nombreuses applications. Par exemple, le *plaquage de textures* consiste à dessiner une image dans l'espace paramétrique et à la transporter sur la surface 3D par l'application interpolée linéairement. La figure 3.18 montre un exemple.

Cette technique est largement utilisée à la fois dans l'industrie cinématographique et dans les jeux vidéos, exploitant les cartes graphiques (GPU). Ces dernières contiennent des circuits et du micro-logiciel optimisé afin de calculer de manière très efficace l'interpolation linéaire et les coordonnées barycentriques associées, permettant une visualisation en temps réel, utilisée aussi bien par les jeux vidéos que par la visualisation scientifique. Le plaquage de textures peut également être utilisé pour associer un attribut de haute résolution à un maillage grossier, comme illustré en figure 3.19. En encodant les détails géométriques de fine échelle dans une image (ici le vecteur normal), il est possible en quelque sorte de « remplacer des triangles par des pixels », et ainsi de réduire à la fois l'espace mémoire requis et les temps d'affichage de manière très significative.

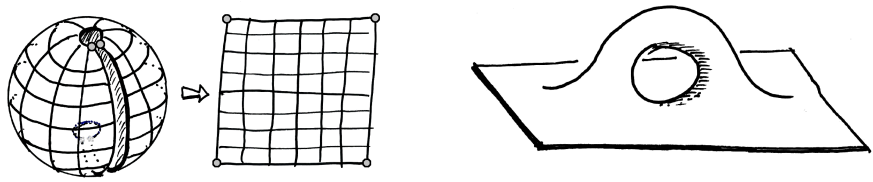


FIGURE *3.20 – Certaines surfaces ne peuvent pas être dépliées sans être découpées.*

3.3.2 Notions de topologie

Comme une paramétrisation est une fonction bijective entre une surface et un sous-ensemble de $\mathbb{R}^2$, de manière évidente, toutes les surfaces ne peuvent pas admettre de paramétrisation. Par exemple, une sphère ne peut pas être dépliée sur un plan sans la découper ou sans y percer un trou, voir figure 3.20 (à gauche). Une condition nécessaire est que la surface ait un bord. Toutefois, la présence d'un bord n'est pas une condition suffisante. Considérons la surface de la figure 3.20 (à droite) : à cause de la « poignée » (ou « anse » comme l'anse d'une tasse à café), il est impossible de la déplier en 2D sans générer d'intersection. Ce dernier exemple nous montre la nécessité de définir un critère permettant de déterminer si un maillage donné présente une telle anse. Le nombre d'anses d'une surface se nomme le *genre* de la surface (ou *genus* en anglais).

Une sphère, voir figure 3.21 (A,B), est de genre 0, un tore, voir figure 3.21 (C), est de genre 1, un double tore, voir figure 3.21 (E), est de genre 2. Le genre (noté g à partir de maintenant) est une propriété globale de la surface. À première vue, le caractère global de g semble rendre son calcul difficile, mais le genre peut être déduit de la caractéristique d'Euler–Poincaré $\mathcal{X}$, une autre quantité très facile à calculer, définie par $\mathcal{X} = n_v - n_e + n_f$, où n_v dénote le nombre de sommets, n_e le nombre d'arêtes et n_f le nombre de faces. La caractéristique d'Euler–Poincaré et le genre sont tous deux des *invariants topologiques*, qui ne dépendent ni de la forme de la surface, ni de la manière dont cette dernière est décomposée en polygones. Nous allons tout d'abord voir comment le genre et $\mathcal{X}$ sont liés dans le cas d'une surface connexe sans bord, et verrons ensuite comment ce résultat se généralise à des surfaces avec plusieurs composantes connexes et plusieurs bords.

Par exemple, considérons la sphère de la figure 3.21 (A), décomposée en 8 sommets, 12 arêtes et 6 faces à la manière d'un cube. Sa caractéristique d'Euler–Poincaré vaut $\mathcal{X} = 8 - 12 + 6 = 2$. Considérons maintenant une décomposition de la même sphère à la manière d'un tétraèdre, comme sur la figure 3.21 (B), nous obtenons alors $\mathcal{X} = 4 - 6 + 4 = 2$ à nouveau.

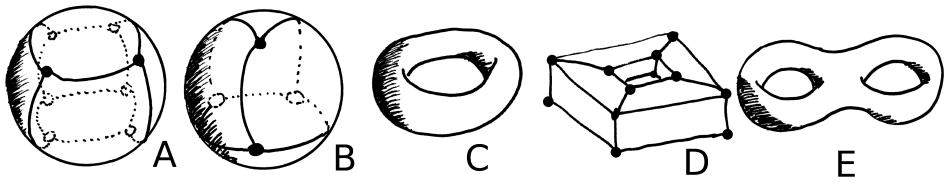


FIGURE 3.21 – *Déterminer le genre (nombre d'anses) à l'aide de la caractéristique d'Euler–Poincaré.*

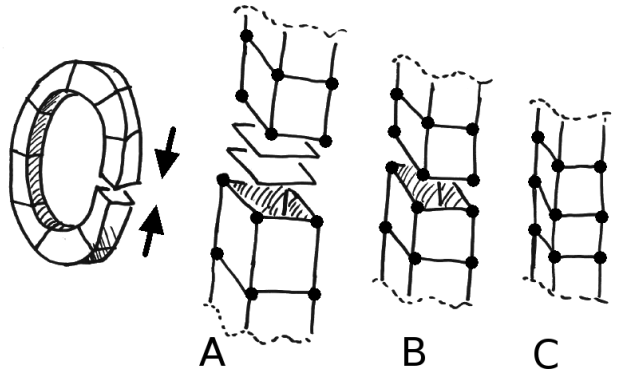


FIGURE 3.22 – *Augmenter le genre par « chirurgie topologique » : un cas particulier. A : détruire deux faces (Δn_f = -2) ; B : fusionner les deux bords ($\Delta n_v = -4$; $\Delta n_e = -4$) ; C : le résultat ($\Delta\mathcal{X} = \Delta n_v - \Delta n_e + \Delta n_f = -2$).*

Quelle que soit la décomposition de la sphère, on obtiendra toujours $\mathcal{X} = 2$. Considérons à présent un tore, voir figure 3.21 (C), décomposé comme sur la figure 3.21 (D). Il a 16 sommets, 32 arêtes et 16 faces, ainsi $\mathcal{X} = 16 - 32 + 16 = 0$.

Maintenant, nous aimerions déterminer la caractéristique d'Euler–Poincaré d'un double tore, voir figure 3.21 (E), à savoir, une surface de genre $g = 2$. Une première possibilité serait de procéder comme nous l'avons fait pour la sphère et le tore, à savoir « mailler » le double tore et compter les nombres de sommets, d'arêtes et de faces. Une autre possibilité, plus intéressante, est d'analyser comment la caractéristique d'Euler–Poincaré $\mathcal{X}$ change si nous modifions une surface initiale en y créant une anse supplémentaire, comme illustré sur la figure 3.22.

Cette opération résulte en une variation Δn_v du nombre de sommets, et respectivement des variations $\Delta n_e, \Delta n_f$ des nombres d'arêtes et de faces. À son tour, $\mathcal{X}$ devient $\mathcal{X} + \Delta\mathcal{X}$, où $\Delta\mathcal{X} = \Delta n_v - \Delta n_e + \Delta n_f$. Dans la première étape (A), deux faces sont supprimées, ainsi n_f est décrémenté deux fois

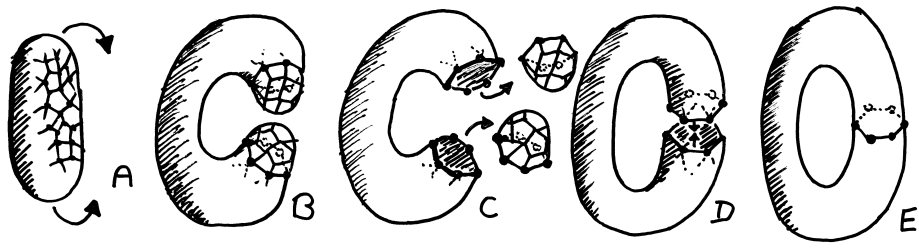


FIGURE 3.23 – *Augmenter le genre par « chirurgie topologique » : le cas général.*

($\Delta n_f = -2$), de même que $\mathcal{X} = n_v - n_e + n_f$. Dans la deuxième étape (B), les deux nouveaux bords sont fusionnés. Ceci fusionne également les sommets et les arêtes présents (en nombres identiques) sur ces deux bords, ainsi n_v et n_e sont diminués du même nombre (il y a le même nombre de sommets et d'arêtes sur chacun des bords, à savoir 4 dans l'exemple présent). Les variations de n_v et n_e s'annulent donc quand elles sont prises en compte dans $\Delta\mathcal{X}$, ainsi cette deuxième étape ne change pas $\mathcal{X}$.

En résumé, l'opération de création d'une nouvelle anse diminue $\mathcal{X}$ de 2. Ici, nous avons vu un cas particulier (création d'une anse en supprimant deux faces carrées et en identifiant les bords), on peut se demander si $\mathcal{X}$ diminue toujours de 2, quelle que soit la manière dont l'anse est créée. La figure 3.23 montre le cas général, obtenu en supprimant du maillage initial (A) deux disques topologiques (B,C) ayant le même nombre n de sommets sur leur bord ($n = 6$ sur la figure) et en fusionnant les bords. Si on fait le bilan de la variation de $\mathcal{X}$, l'étape (C) crée n sommets et n arêtes pour chaque bord, donc $2n$ sommets et $2n$ arêtes au total, dont les contributions s'annulent puisque les nombres de sommets et d'arêtes interviennent avec des signes opposés dans $\mathcal{X}$. Ensuite, l'étape (D) supprime les deux disques topologiques, chacun de caractéristique $\mathcal{X} = 1$, donc $\mathcal{X}$ diminue de 2. Enfin, la dernière étape (E) fusionne les $2n$ sommets et $2n$ arêtes des deux bords, supprimant ainsi n sommets et n arêtes. Encore une fois, les contributions des variations des nombres de sommets et d'arêtes s'annulent. En résumé, $\mathcal{X}$ est décrémenté de 2. Ceci est bien indépendant de la manière dont les deux disques topologiques supprimés sont maillés, et du nombre de sommets sur les bords créés, la seule condition étant que les deux bords créés aient le même nombre de sommets. En conséquence, en partant d'une sphère, de genre 0 et de caractéristique d'Euler–Poincaré $\mathcal{X} = 2$ (voir figure 3.21), à chaque fois qu'une anse est ajoutée, g est augmenté de un (par définition de g) et $\mathcal{X}$ est diminuée de deux. En résumé, pour une surface connexe sans bord, nous pouvons en déduire que :

$$\mathcal{X} = n_v - n_e + n_f = 2 - 2g$$

Pour le double tore de la figure 3.21 (E), on a $g = 2$ et $\mathcal{X} = 2 - 2g = -2$.

Considérons à présent le cas plus général d'une surface ayant plusieurs composantes connexes et plusieurs bords. Avec le même type d'argument, il est relativement facile de déterminer comment cette relation évolue. Puisque pour une sphère topologique $\mathcal{X} = 2$, chaque nouvelle composante connexe ajoute 2 à la caractéristique d'Euler–Poincaré. Considérons maintenant l'opération de « chirurgie topologique » qui crée un bord en supprimant une face. Cette opération diminue n_f de 1, donc elle diminue également $\mathcal{X}$ de 1. En résumé, pour une surface présentant plusieurs composantes connexes et plusieurs bords :

$$\mathcal{X} = n_v - n_e + n_f = 2n_{cnx} - 2g - n_{bd},$$

où n_{cnx} dénote le nombre de composantes connexes et n_{bd} le nombre de bords.

Dans la suite de cette section, nous considérerons des *disques topologiques*, à savoir des surfaces de genre 0 avec une seule composante connexe et un bord exactement. Un disque topologique est caractérisé par :

$$g = 0 \quad ; \quad n_{cnx} = 1 \quad ; \quad n_{bd} = 1$$

En utilisant l'identité précédente, pour un disque topologique, il est facile de vérifier que $\mathcal{X} = 1$. Pour tester si un maillage donné est bien un disque topologique, nous testons tout d'abord la valeur de $\mathcal{X} = n_v - n_e + n_f$. Ensuite, il est nécessaire de déterminer si la surface est connexe. Il est possible d'utiliser un algorithme de parcours récursif à partir d'une face arbitraire et de vérifier si toutes les faces de la surface ont été parcourues. Finalement, pour déterminer le nombre de bords, de manière similaire il est possible de parcourir toutes les arêtes autour de l'un des bords et vérifier ensuite si toutes les arêtes du bord ont été parcourues.

3.3.3 Le théorème du plongement barycentrique de Tutte

Nous nous intéressons à présent au théorème du plongement barycentrique de Tutte [194], qui fournit une caractérisation d'une famille de paramétrisations de maillages valides. D'autre part, il peut être utilisé pour dériver un algorithme permettant de calculer une telle paramétrisation [74]. Sous une forme simple, ce théorème peut être énoncé de la manière suivante :

Théorème 3.3.1. *Considérons un maillage 3-connexe qui est un disque topologique et dont les sommets sont munis de coordonnées 2D. Si les sommets du bord*

se trouvent sur un polygone convexe (au sens strict ou pas, à savoir ayant potentiellement des sommets alignés), et si chaque sommet interne est une combinaison convexe de ses voisins, alors il n'existe pas d'intersection entre les arêtes, et chaque face est strictement convexe.

Un maillage est dit *3-connexe* s'il reste connexe après qu'on ait supprimé une paire de sommets quelconque et les arêtes incidentes à ces sommets. Cette condition est nécessaire pour éviter certaines dégénérescences (triangles plats) qui peuvent apparaître par exemple lorsqu'il existe une longue arête entre deux sommets du bord.

Formellement, les sommets internes sont des combinaisons convexes de leurs voisins si :

$$\begin{aligned} \sum_{j \in N_i} w_{ij} u_j &= u_i, \quad i = 1, \ldots, (n_v - n_{vbd}) \\ \sum_{j \in N_i} w_{ij} v_j &= v_i, \quad i = 1, \ldots, (n_v - n_{vbd}) \\ u_i &= b_i^u, \quad i = (n_v - n_{vbd} + 1), \ldots, n_v \\ v_i &= b_i^v, \quad i = (n_v - n_{vbd} + 1), \ldots, n_v \end{aligned} \tag{3.16}$$

où n_{vbd} dénote le nombre de sommets sur le bord, et où les sommets sont ordonnés de sorte que les sommets internes apparaissent en premier. Les b_i^u et b_i^v désignent les coordonnées des sommets du bords, fixées. Les coefficients donnés w_{ij} associés aux arêtes orientées (i, j) sont strictement positifs et satisfont $\sum_j w_{ij} = 1$. Les coefficients w_{ij} peuvent ne pas être symétriques (en général, $w_{ij} \neq w_{ji}$). En d'autres termes, la condition (3.16) signifie que chaque sommet doit être dans l'enveloppe convexe de ses voisins (d'où le terme *combinaison convexe*). N_i dénote l'ensemble des voisins du sommet i, à savoir, les sommets connectés à i par une arête.

À partir de la donnée des positions des sommets du bord $(u_i, v_i), i = n_v - n_{vbd} + 1, \ldots, n_v$ et les poids w_{ij}, il est possible de calculer une paramétrisation en résolvant les deux systèmes linéaires (3.16), comme suggéré par Floater [74] et expliqué dans la sous-section suivante. Avant de présenter une méthode de calcul pour calculer les coordonnées (u_i, v_i), le reste de cette section concerne la preuve du théorème de Tutte.

La preuve originale par Tutte [194] s'appuie sur des fondements conséquents de théorie des graphes. Plus tard, une autre preuve a été proposée par Gortler, Gotsmann and Thurston [87]. Cette dernière preuve, présentée ici, est bien plus simple et ne s'appuie que sur des notions élémentaires. Afin de mieux saisir l'intuition de sa structure, nous présentons ici la preuve en partant des conclusions et en remontant vers les prémisses, et omettons les détails pour ne garder que les arguments principaux.

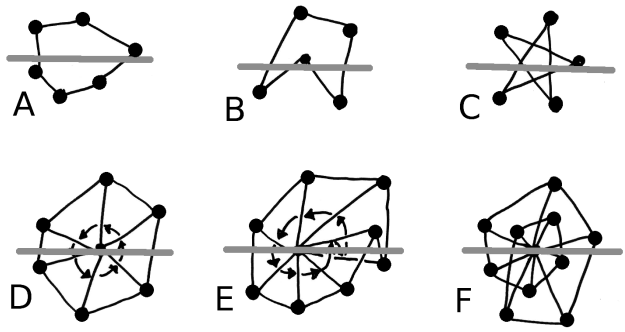


FIGURE 3.24 – *Une face convexe (A) et des faces non convexes (B,C) ; un sommet cyclique (wheel) (D) et des sommets non cycliques (non-wheel) (E,F).*

Le principal argument de la preuve, que nous verrons plus loin, est que si (3.16) est satisfaite, alors toutes les faces sont strictement convexes, et tous les voisinages des sommets ont une certaine propriété (sont *cycliques*). Il est alors facile d'en déduire qu'aucune paire de faces n'a d'intersection.

La figure 3.24 (A)–(C) montre des exemples de faces convexes et non convexes. Toute droite en position générique [6] (en gris) a 0 ou 2 intersections avec les arêtes d'un polygone convexe (A), or un polygone non convexe peut générer plus de 2 intersections (B,C).

Un sommet v est *cyclique* (*wheel vertex*) si les angles orientés entre chaque paire d'arêtes consécutives qui émanent de v sont tous positifs et de somme égale à 2π. La figure 3.24 montre des exemples de sommets non cycliques : (E) l'un des angles est négatif et (F) la somme des angles est de 4π au lieu de 2π.

Toute droite en position générique qui passe par le centre d'un sommet cyclique a 2 intersections avec les coins incidents au sommet (D). Pour les sommets non cycliques, il existe des droites avec un plus grand nombre d'intersections (E,F).

Nous allons maintenant montrer que si (3.16) est satisfaite, alors toutes les faces sont convexes et tous les sommets sont cycliques, en étudiant la relation entre une droite de vecteur directeur (α, β), les faces et les voisinages des sommets. Nous montrerons ensuite que quel que soit le vecteur directeur (α, β) de la droite, des configurations telles que B,C,E,F ne peuvent pas se présenter.

Considérons la configuration de la figure 3.25, avec une droite de vecteur directeur (α, β). La figure est orientée de manière à ce que le vecteur

6. à savoir, qui ne passe pas par un des sommets, de sorte que toutes les intersections soient franches.

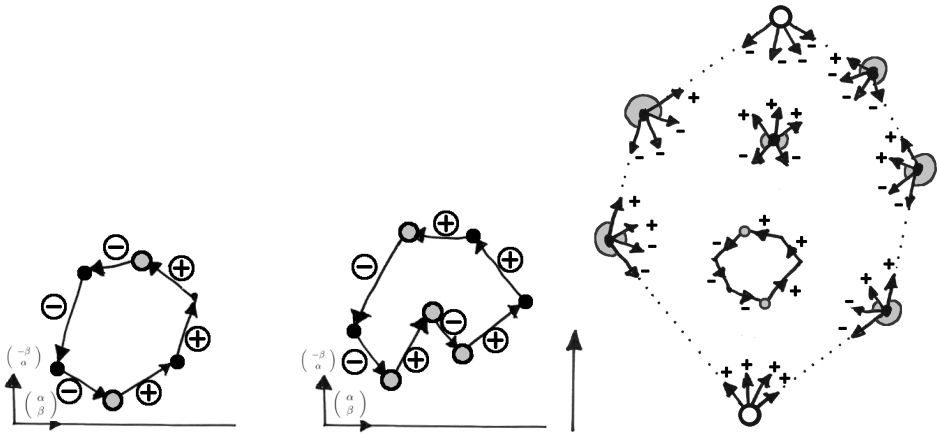


FIGURE 3.25 – *À gauche et au centre : le nombre maximum d'intersections entre un polygone et une droite est borné par le nombre de changements de signe. À droite : structure globale des changements de signe dans les sommets et les faces d'un maillage dont les coordonnées (u, v) satisfont les pré-conditions du théorème de Tutte.*

(α, β) soit horizontal. Nous classifions chaque arête orientée (i, j) en tant que montante $(+)$ ou descendante $(-)$, suivant le signe du produit scalaire $(\mathbf{u}_j - \mathbf{u}_i) \cdot \mathbf{n}$, où $\mathbf{n} = (-\beta, \alpha)$ dénote le vecteur normal de la droite.

Les arêtes ayant une intersection avec une droite de vecteur directeur (α, β) apparaissent en paires $(+,-)$, donc le nombre d'intersections est borné par le nombre de changements de signe (les sommets correspondant à des changements de signe sont mis en évidence en gris clair sur la figure). Un polygone convexe a exactement deux changements de signe pour tout (α, β) en position générique (non colinéaire à une arête), alors qu'un polygone non convexe admet des directions (α, β) avec un plus grand nombre de changements de signe. Le même raisonnement s'applique également aux voisinages des sommets (auquel cas la propriété de cyclicité joue le même rôle que la propriété de convexité des faces).

Considérons à présent un maillage qui est un disque topologique (cf. sous-section précédente), que nous transformons en une sphère topologique en ajoutant une grande face qui ferme son bord. Le principal argument de la preuve est que si les coordonnées (u_i, v_i) satisfont les pré-conditions du théorème de Tutte (sommets du bord sur un polygone convexe et sommets internes satisfaisant (3.16)), alors pour tout vecteur directeur (α, β), la structure des changements de signe pour l'ensemble du maillage est telle qu'illustré sur la figure 3.25 (à droite) : tous les sommets internes et toutes les faces ont deux changements de signe, deux sommets

du bord n'ont aucun changement de signe (le sommet le plus haut et le plus bas), et tous les autres $n_{vbd} - 2$ sommets du bord ont deux changements de signe. Sur cette figure[7], les flèches correspondent aux arêtes orientées émanant de chaque sommet et aux arêtes orientées formant le bord des faces. Les changements de signe sont mis en évidence en gris. Comme il y a une face extérieure infinie (celle qui a été ajoutée pour transformer le disque topologique en une sphère topologique), certains des coins correspondant aux changements de signe sont symbolisés à l'extérieur du bord.

Afin de démontrer que la structure induite par la solution de (3.16) correspond à cette configuration, Gortler *et al.* [87] ont observé que l'ensemble des valeurs $\mathbf{n} \cdot (\mathbf{u}_j - \mathbf{u}_i)$ associées aux arêtes orientées du maillage peut être considéré comme un « champ de vecteurs discret »[8], qui obéit à un équivalent discret du théorème de Poincaré–Hopf [90]. Ce dernier théorème met en relation la topologie d'un champ de vecteurs (représentée ici par les changements de signe) avec la topologie de la surface sur laquelle ce champ de vecteurs est défini (ici une sphère topologique). En deux mots, ce théorème indique que le nombre de singularités du champ de vecteurs correspond à la caractéristique d'Euler–Poincaré de la surface. Dans notre configuration, nous avons une sphère de caractéristique $\mathcal{X} = 2$, donc il y a deux singularités (le sommet le plus haut et le sommet le plus bas). Plusieurs versions discrètes du théorème de Poincaré–Hopf ont été proposées [156, 157]. La version proposée par Gortler *et al.* dans [87] s'énonce de la manière suivante :

Théorème 3.3.2. *Soit z_{ij} un ensemble de nombres réels attachés aux arêtes orientées d'un maillage de genre g sans bord tel que pour toute arête orientée (i, j), $z_{ij} \neq 0$ et $z_{ji} = -z_{ij}$. L'indice ind des sommets et des faces satisfait l'égalité suivante :*

$$\sum_{v \in 1,\ldots,n_v} ind(v) + \sum_{f \in 1,\ldots,n_f} ind(f) = \mathcal{X} = 2 - 2g.$$

Les entiers $ind(v) = (2 - sc(v))/2$ (et $ind(f) = (2 - sc(f))/2$) sont appelées respectivement *indice* du sommet v dans z (et indice de la face f dans z). Les entiers $sc(v)$ (et $sc(f)$) dénotent le nombre de changements de signe de z rencontrés en parcourant les arêtes orientées émanant de v (et respectivement les arêtes orientées autour de f). L'indice d'un sommet ou d'une face peut prendre les valeurs suivantes :

— $ind = 1$: z_{ij} a le même signe pour toutes les arêtes orientées émanant du sommet / tournant autour de la face. Un sommet d'indice 1

7. nous utilisons une convention graphique différente de celle de l'article original [87].
8. ou plus exactement la version discrète d'une 1-forme différentielle.

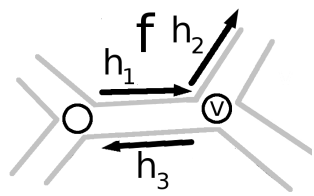


FIGURE 3.26 – *Chaque arête orientée introduit exactement soit un changement de signe dans le sommet duquel elle émane, soit un changement de signe dans la face qu'elle touche.*

est appelé une *source* $(+)$ ou un *puits* $(-)$. Une face d'indice 1 est appelée un *vortex* ;

— $ind = 0$: un sommet ou une face d'indice 0 est dit(e) *non singulier(e)* ;

— $ind < 0$: un sommet ou une face d'indice négatif est appelé(e) une *selle*.

Démonstration.

$$\begin{aligned}
&\sum_{v\in 1,\dots,n_v} ind(v) + \sum_{f\in 1,\dots,n_f} ind(f) \\
&= \tfrac{1}{2} \sum_{v\in 1,\dots,n_v} (2 - sc(v)) + \tfrac{1}{2} \sum_{f\in 1,\dots,n_f} (2 - sc(f)) \\
&= n_v + n_f - \tfrac{1}{2} \left(\sum_{v\in 1,\dots,n_v} sc(v) + \sum_{f\in 1,\dots,n_f} sc(f) \right) \\
&= n_v + n_f - n_e.
\end{aligned}$$

La dernière étape du calcul ci-dessus utilise l'observation suivante : le nombre total de changements de signe dans les faces et dans les sommets est égal au nombre d'arêtes orientées du maillage, à savoir $2n_e$ (il y a deux arêtes orientées par arête). Chaque arête orientée introduit exactement soit un changement de signe dans un sommet soit un changement de signe dans une face : examinons l'arête h_1 de la figure 3.26. Son arête opposée h_3 est de signe opposé. Pour h_2 il y a deux cas possibles :

— h_1 et h_2 ont le même signe, alors le coin h_2, h_3 est un changement de signe dans le sommet v,

— ou h_1 et h_2 ont des signes opposés, alors le coin h_1, h_2 est un changement de signe dans la face f.

■

En utilisant cette version discrète du théorème de Poincaré–Hopf, nous pouvons à présent examiner la structure des changements de signe

engendrée par un ensemble de valeurs $\mathbf{n} \cdot (\mathbf{u}_j - \mathbf{u}_i)$ associées aux arêtes orientées (i, j) d'un maillage, où les points 2D $\mathbf{u}_i = (u_i, v_i)$ sont donnés et satisfont l'équation de Tutte (3.16). Nous pouvons dès à présent affirmer qu'aucune face ne peut être un vortex (à savoir d'indice 1, ou encore sans changement de signe), car ceci impliquerait qu'en partant d'un sommet et en tournant autour de la face nous ne reviendrions pas à la position initiale.

De manière similaire, les sommets ne peuvent pas être d'indice 1 pour la raison suivante : comme les $\mathbf{u}_i$ satisfont l'équation de Tutte, il est facile de vérifier que les valeurs z_{ij} associées aux arêtes orientées émanant d'un sommet interne i satisfont l'égalité :

$$\sum_{j \in N_i} w_{ij} z_{ij} = 0.$$

Comme tous les coefficients w_{ij} sont strictement positifs et comme les z_{ij} sont tous non nuls, le signe de z_{ij} doit changer, donc un sommet interne i ne peut être ni une source, ni un puits. En résumé,

$$\begin{array}{rcll} ind(f) & \leqslant & 0 & \forall f = 1, \ldots, n_f \\ ind(v) & \leqslant & 0 & \forall v = 1, \ldots, (n_v - n_{vbd}). \end{array}$$

Nous rappelons le théorème de Poincaré–Hopf discret (avec $\mathcal{X} = 2$) :

$$\sum_{f=1,\ldots,n_v} ind(v) + \sum_{f=1,\ldots,n_f} ind(f) = 2.$$

Les sommets du bord sont sur un polygone convexe. Par rapport à la direction (α, β) considérée, il y a exactement deux sommets extrêmes, en blanc sur la figure 3.25 (droite). Toutes les arêtes émanant d'un de ces sommets extrêmes sont de même signe. Ainsi, les deux sommets extrêmes sont une source et un puits tous deux d'indice 1. Les autres sommets du bord ont deux changements de signe (et sont donc d'indice 0). Restent à déterminer les indices des faces et des sommets internes. En se rappelant que les indices des sommets et faces internes sont tous négatifs ou nuls, la seule possibilité pour que $\mathcal{X}$ soit de valeur 2 est que toutes les faces et tous les sommets internes soient d'indice 0.

En conclusion, quelle que soit la direction (α, β), toutes les faces et tous les sommets internes sont d'indice 0 (non singuliers), donc toutes les faces sont convexes et tous les sommets internes sont cycliques.
Ceci conclut la preuve du théorème 3.3.1.

Tout au long de la preuve, nous avons supposé que tous les objets géométriques considérés étaient en position générique. L'article initial [87]

inclut des lemmes de perturbation (que nous ne détaillerons pas ici) qui permettent de traiter les cas dégénérés où une arête est colinéaire avec la direction (α, β) considérée. L'article original inclut également des résultats pour des maillages comportant plusieurs bords internes éventuellement non convexes, et étudie également le cas d'objets de genre supérieur. Dans ce cours, nous nous limiterons au cas d'un disque topologique avec un bord convexe. Pour les cas plus généraux, le lecteur pourra consulter l'article initial, et les articles de synthèse et ouvrages [27, 31, 97].

3.3.4 Résolution numérique de l'équation de Tutte

Étant donné un disque topologique, la position 2D de ses sommets du bord sur un polygone convexe, et des poids positifs w_{ij}, il est possible d'obtenir les (u_i, v_i) définissant une paramétrisation valide (sans intersection de triangles) en résolvant (3.16). Nous expliquons ici plusieurs méthodes permettant de résoudre de tels systèmes linéaires. Sous sa forme originale, l'équation de Tutte correspond au système linéaire suivant :

$$\left\{ \begin{array}{rcl} u_i & = & \sum\limits_{j \in N_i} w_{ij} u_j \\ v_i & = & \sum\limits_{j \in N_i} w_{ij} v_j \end{array} \right. \qquad \forall i = 1, \ldots, (n_v - n_{vbd}).$$

Le terme de droite contient à la fois des inconnues (les u_j et v_j associés aux sommets internes) et des valeurs constantes (les u_j et v_j associés aux sommets du bord). Pour réorganiser cette équation sous une forme qui regroupe les valeurs constantes dans le terme de droite, nous étendons la définition des w_{ij} en adoptant la convention suivante :

$w_{ii} = -1$ et $w_{ij} = 0$ si $j \notin N_i$. Nous pouvons alors écrire :

$$\left\{ \begin{array}{rcl} \sum\limits_{j=1,\ldots,(n_v - n_{vbd})} w_{ij} u_j & = & - \sum\limits_{j=(n_v - n_{vbd}+1),\ldots,n_v} w_{ij} u_j \\ \sum\limits_{j=1,\ldots,(n_v - n_{vbd})} w_{ij} v_j & = & - \sum\limits_{j=(n_v - n_{vbd}+1),\ldots,n_v} w_{ij} v_j \end{array} \right. \qquad \forall i = 1, \ldots, (n_v - n_{vbd})$$

En sachant que les u_i et les v_i ne sont pas mélangés dans les équations, ça revient à résoudre deux systèmes linéaires séparés. Sous forme matricielle, ces deux systèmes linéaires s'écrivent :

$$\begin{array}{rcl} W\mathbf{X}^u & = & \mathbf{Y}^u \\ W\mathbf{X}^v & = & \mathbf{Y}^v \end{array}$$

où W est une matrice $(n_v - n_{vbd}) \times (n_v - n_{vbd})$, et $\mathbf{X}^u$,$\mathbf{Y}^u$,$\mathbf{X}^v$,$\mathbf{Y}^v$ sont des vecteurs de dimension $(n_v - n_{vbd})$.

Nous abordons à présent la résolution numérique de ces systèmes linéaires. Considérons un système sous la forme $WX = Y$, avec $n = (n_v - n_{vbd})$. La relaxation de Gauss–Seidel est une méthode qui est très simple à implanter et qui permet de calculer une approximation de la solution. Intuitivement, elle peut se comprendre en considérant l'équation sous la forme d'un système d'équations linéaires :

$$\begin{cases} w_{11}x_1 + w_{12}x_2 + \ldots & +w_{1n}x_n & = & y_1 \\ & & \vdots & \\ w_{i1}x_1 + w_{i2}x_2 + \ldots + w_{ii}x_i + \ldots & +w_{in}x_n & = & y_i \\ & & \vdots & \\ w_{n1}x_1 + w_{n2}x_2 + \ldots & +w_{nn}x_n & = & y_n \end{cases}$$

L'algorithme va effectuer des itérations successives sur les n équations en partant d'une approximation initiale $x_i = 0 \quad \forall i = 1, \ldots, (n_v - n_{vbd})$. Pour chaque équation, la variable x_i sera mise à jour en « faisant comme si » la valeur de toutes les autres variables x_j pour $j \neq i$ dans le vecteur solution était connue. Ainsi, la mise à jour de x_i s'écrit :

$$x_i \leftarrow \frac{1}{w_{ii}} \left(y_i - \sum_{j \in 1,\ldots,(n_v - n_{vbd}), j \neq i} w_{ij} x_j \right).$$

En remplaçant la composante y_i du membre de droite par son expression $-\sum_{j=n_v-n_{vbd}+1,\ldots,n_v} w_{ij}x_j$ et le coefficient w_{ii} par sa valeur -1, dans notre cas particulier, la mise à jour de la variable x_i s'écrit :

$$x_i \leftarrow -\left(- \sum_{j=(n_v-n_{vbd}+1,\ldots,n_v)} w_{ij}x_j - \sum_{j=1,\ldots,(n_v-n_{vbd})} w_{ij}x_j \right)$$

ou encore :

$$x_i \leftarrow \sum_{j \in N_i} w_{ij} x_j.$$

En résumé, résoudre le système d'équations linéaires de Tutte par la relaxation de Gauss–Seidel revient simplement à appliquer des itérations successives, où chaque itération consiste à déplacer chaque sommet au barycentre pondéré de ses voisins. Cette méthode est très simple à implanter, et donne des résultats en un temps raisonnable pour des petits maillages (ayant quelques milliers d'éléments). Pour des maillages plus conséquents, il est possible d'utiliser des algorithmes de résolution numérique plus

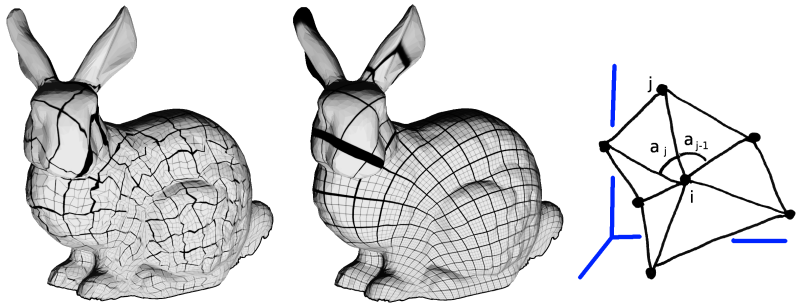


FIGURE 3.27 – *Visualisation de la paramétrisation obtenue avec les poids uniformes (à gauche) et les coordonnées valeur moyenne (au milieu). À droite : les angles impliqués dans la définition des coordonnées valeur moyenne.*

sophistiqués, tels que le gradient bi-conjugué, une généralisation de l'algorithme que nous allons voir en section 3.2 qui permet de traiter des matrices non symétriques, comme suggéré dans [74]. Si la consommation mémoire n'est pas un problème, les solveurs directs creux tels que SUPERLU [120] peuvent être une alternative encore plus efficace. Voir également le chapitre « aspects numériques » du livre [31].

Choix des poids

Suivant l'application, il est souvent souhaitable de créer une paramétrisation minimisant les déformations entre l'espace 2D paramétrique et la surface 3D. En utilisant les poids uniformes $w_{ij} = 1/|N_i|$ où $|N_i|$ dénote le nombre de voisins du sommet i, nous obtenons une paramétrisation certes valide, mais introduisant de fortes distorsions. La figure 3.27 (à gauche) illustre ces déformations en « peignant » une grille régulière dans l'espace 2D paramétrique et en transportant celle-ci sur la surface 3D par la paramétrisation.

Il est possible d'utiliser pour les poids w_{ij} les « coordonnées valeur moyenne » (Mean Value Coordinates) [75, 96], qui prennent en compte la géométrie de la surface 3D initiale, et qui sont positives comme requis par les pré-conditions du théorème de Tutte. Les poids w_{ij} dérivés des coordonnées valeur moyenne ϕ_{ij} ont pour expression :

$$\begin{aligned} \phi_{ij} &= \frac{\tan(a_{j-1}/2)+\tan(a_j/2)}{\|\mathbf{p}_j-\mathbf{p}_i\|} \\ w_{ij} &= \phi_{ij} / \textstyle\sum_{j\in N_i} \phi_{ij} \end{aligned}$$

où les angles a_{j-1} et a_j sont tels que sur la figure 3.27 (à droite).

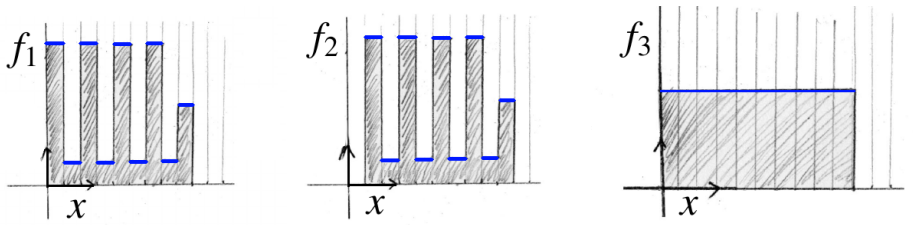


FIGURE 3.28 – *Comparer des fonctions : on aimerait dire que f_1 est plus proche de f_2 que de f_3, mais la norme L_2 ne « voit pas » que f_2 correspond à f_1 légèrement décalée sur l'axe des x.*

3.4 Transport optimal

Cette section présente une introduction au transport optimal, et résume une série de conférences de B. Lévy données entre 2014 et 2017 (lors de l'école d'été de l'Institut Fourier, d'ateliers BIRS, IPAM et CRM). La présentation reste à un niveau élémentaire, correspondant à une vision informatique du problème, avec l'objectif principal de comprendre la structure du raisonnement, pour aller ensuite jusqu'à un algorithme efficace implantable en machine.

Le transport optimal, étudié initialement par Monge [134], est un problème très général qui peut être utilisé comme modèle pour une grande classe de domaines d'application. En particulier, il est une formulation naturelle pour plusieurs questions fondamentales de graphisme par ordinateur [127, 129, 30], car il permet de définir de nouvelles manières de *comparer*, de mesurer des *distances* entre des fonctions et d'*interpoler* entre plusieurs fonctions.

Comparer des fonctions. Considérons les fonctions f_1, f_2 et f_3 de la figure 3.28. Nous avons choisi une fonction f_1 dont le graphe oscille beaucoup, et une fonction f_2 obtenue par translation du graphe de f_1 sur l'axe des x. La fonction f_3 correspond à la valeur moyenne de f_1 (ou de f_2). Si nous mesurons les distances relatives entre ces fonctions en utilisant la norme L_2 classique, à savoir $d_{L_2}(f,g) = \sqrt{\int (f(x) - g(x))^2 dx}$, nous trouverons que f_1 est plus proche de f_3 que de f_2. Le transport optimal permet de définir une distance qui va prendre en compte le fait que le graphe de f_2 puisse être obtenu par une translation (comme ici), ou encore une déformation du graphe de f_1, et ainsi considérer que f_1 est plus proche de f_2 que de f_3.

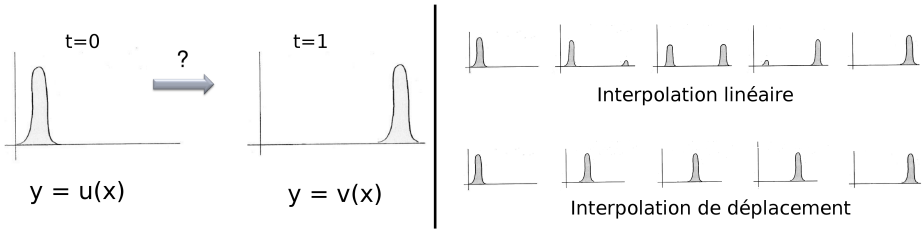


FIGURE 3.29 – *Interpoler entre deux fonctions : l'interpolation linéaire fait disparaître une bosse pendant que l'autre bosse apparaît ; l'interpolation de déplacement, issue du transport optimal, déplace la bosse.*

Interpoler entre des fonctions. Considérons à présent les fonctions u et v de la figure 3.29. Ici nous supposons que u correspond à un certain phénomène physique mesuré à un temps initial $t = 0$ et v au même phénomène mesuré au temps final $t = 1$. Le problème consiste à reconstruire ce qui s'est passé entre $t = 0$ et $t = 1$. Si nous utilisons l'interpolation linéaire (figure 3.29 en haut à droite), nous verrons la bosse de gauche diminuer progressivement pendant que celle de droite apparaît, ce qui ne serait pas très réaliste si les fonctions représentent par exemple une onde en train de se propager. Le transport optimal permet de définir un autre type d'interpolation (interpolation de déplacement de Mc Cann, figure 3.29 en bas à droite), qui va transformer progressivement le graphe de u en le graphe de v.

Le transport optimal permet de munir un espace de fonctions [9] d'une géométrie, et ainsi de calculer des distances dans cet espace, ou d'interpoler entre différentes fonctions, ou encore de calculer des barycentres entre différentes fonctions, et ceci dans un contexte très général. Ainsi, le transport optimal apparaît comme un outil fondamental dans de nombreux domaines appliqués. Ainsi, des applications en graphisme par ordinateur ont été proposées, pour comparer et interpoler des objets de natures diverses [30], pour générer des lentilles permettant de concentrer la lumière d'une manière prescrite [130, 171]. D'autre part, le transport optimal fournit de nouveaux outils pour discrétiser des équations aux dérivées partielles, et ainsi définir de nouveaux algorithmes de résolution numérique [22]. Ce type de schéma numérique permet par exemple de simuler des fluides [80], avec des applications spectaculaires en graphisme par ordinateur [57].

Le reste de cette section s'inspire en partie de [199], [168], [40] et [6]

9. ou d'objets plus généraux, des mesures, voir plus loin.

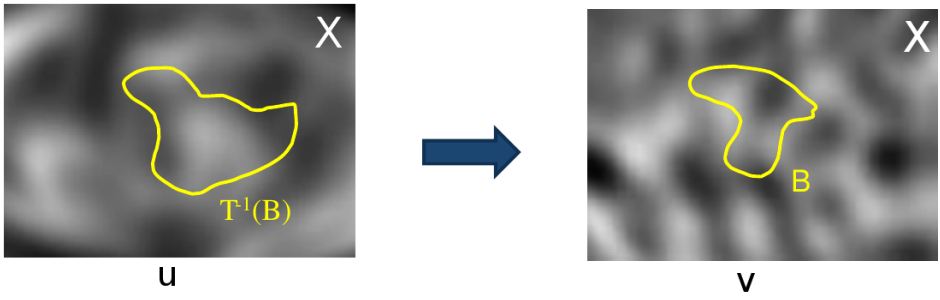


FIGURE 3.30 – *Étant donnés deux terrains, dont les fonctions altitude u et v sont ici symbolisées par des niveaux de gris, le problème de Monge consiste à transformer un terrain en l'autre en déplaçant de la matière selon une application T. Cette application doit satisfaire une contrainte de préservation de matière.*

mais reste à un niveau élémentaire. L'objectif ici est de donner une intuition des différents concepts mis en jeu, et surtout de la manière générale dont ils s'articulent les uns avec les autres, pour aller jusqu'à un algorithme utilisable en pratique.

3.4.1 Le problème de Monge

Le problème du transport optimal a été introduit et étudié par Monge [134] peu avant la Révolution française. Nous commencerons par donner une intuition du problème, puis introduirons rapidement la notion de mesure, nécessaire pour poser le problème dans sa généralité et surtout pour l'analyser.

Intuition. La motivation initiale de Monge pour étudier ce problème était très pratique, et concernait un problème de terrassier : comment transformer un terrain ayant un relief initial en un relief cible souhaité, tout en minimisant le travail du terrassier ?

Monge avait formulé le problème de la manière suivante :

$$\inf_{T:X\to X} \int_X c(x, T(x))u(x)dx$$

sous la contrainte :

$$\forall B \subset X, \int_{T^{-1}(B)} u(x)dx = \int_B v(x)dx$$

où X est un sous-ensemble de $\mathbb{R}^2$, u et v sont deux fonctions positives

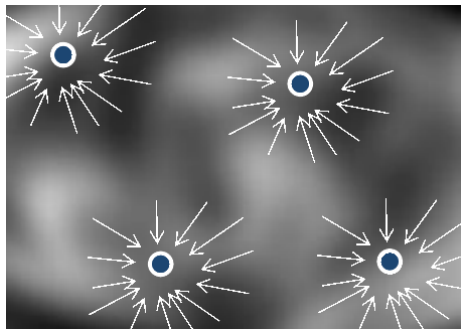


FIGURE 3.31 – *Transport depuis une fonction (niveaux de gris) vers un ensemble de points discret (ronds bleus).*

définies sur X et telles que $\int_X u(x)dx = \int_X v(x)dx$, et $c(\cdot,\cdot)$ est une distance convexe (la distance Euclidienne dans le problème de Monge initial).

Les fonctions u et v représentent l'altitude du relief courant et celle du relief à réaliser (symbolisées par des niveaux de gris sur la figure 3.30). Le problème consiste à trouver (si elle existe) la (ou les) fonction(s) T de X dans X qui permette(nt) de transformer le relief courant u en le relief à réaliser v, tout en minimisant le produit de la quantité de terre transportée $u(x)$ par la distance parcourue $c(x, T(x))$. Clairement, la matière doit être préservée pendant le transport, et donc la quantité totale de terre doit être la même dans le terrain source et dans le terrain cible (les intégrales de u et de v sur X doivent correspondre). Cette contrainte globale de préservation de matière doit être complétée par une contrainte locale, qui impose que la quantité de matière reçue dans tout sous-ensemble B de X doit correspondre à ce qui y a effectivement été transporté, à savoir la matière initialement présente dans la pré-image de B par T. Sans cette contrainte, il serait possible de créer et d'annihiler localement de la matière d'une manière qui se compenserait dans le bilan global. Une application T qui satisfait cette contrainte est appelée une application de transport.

Le problème de Monge avec des mesures. Nous supposons maintenant qu'au lieu d'un « relief à réaliser », nous souhaitons transporter la terre vers un ensemble de points (que nous noterons Y à présent), représentant par exemple un ensemble d'usines permettant d'exploiter une ressource, voir figure 3.31. Chaque usine souhaite recevoir une certaine quantité de matière. Ainsi, la fonction v représentant le « relief à réaliser » est remplacée par une fonction sur un ensemble fini de points. Si la fonction v s'annule partout sauf sur un ensemble fini de points, alors son intégrale sur X s'annule,

donc nous ne pourrons pas utiliser des fonctions pour représenter ce cas de figure. Nous allons utiliser à la place des *mesures*, et associer à chaque usine une *masse de Dirac* pondérée par la quantité de matière à transporter vers l'usine.

À partir de maintenant, nous allons utiliser des mesures μ et ν pour représenter le « relief courant » et le « relief à réaliser ». Dans la précédente sous-section, nous considérions un transport de X dans lui-même. À présent, les mesures μ et ν seront supportées par des ensembles X et Y potentiellement différents (dans le cas précédent, un sous-ensemble de $\mathbb{R}^2$ et un ensemble discret de points). Ceci permet non seulement d'étudier le problème quand la source ou la destination contient des masses de Dirac, mais aussi d'avoir des mesures concentrées sur des surfaces, ce qui permet directement de formaliser des objets informatiques (maillages).

Le lecteur souhaitant en savoir plus sur la théorie de la mesure se référera à [187]. Pour garder une longueur de chapitre raisonnable, nous n'introduirons pas ici la définition d'une mesure, mais pour le moment, nous pourrons nous contenter d'utiliser le petit tableau suivant, qui permet intuitivement de traduire du « langage des fonctions » vers le « langage des mesures » :

fonction u	mesure μ
$\int_B u(x)dx$	$\mu(B)$ ou $\int_B d\mu$
$\int_B f(x)u(x)dx$	$\int_B f(x)d\mu$
$u(x)$	N/A

(à noter : évaluer en un point une mesure μ associée à une fonction u ne donne pas $u(x)$ mais 0, ce qui est indiqué par N/A dans le tableau).

Dans sa version avec des mesures, le problème de Monge s'énonce de la manière suivante :

$$(M) \qquad \inf_{T:X\to Y} \int_X c(x, T(x))d\mu \text{ sous la contrainte : } \nu = T\sharp\mu \tag{3.17}$$

où X et Y sont des Boréliens (à savoir des espaces mesurables), μ et ν sont deux mesures sur X et Y telles que $\mu(X) = \nu(Y)$, et $c(\cdot,\cdot)$ est une distance convexe. La contrainte $\nu = T\sharp\mu$, qui se lit « T pousse μ sur ν », correspond à la contrainte de préservation de masse locale. Étant données une mesure μ sur X et une application T de X dans Y, la mesure $T\sharp\mu$ sur Y, appelée « le poussé en avant de μ par T », est telle que $T\sharp\mu(B) = \mu(T^{-1}(B))$ pour tout Borélien $B \subset Y$. La contrainte de préservation de masse locale signifie donc que $\mu(T^{-1}(B)) = \nu(B)$ pour tout Borélien $B \subset Y$.

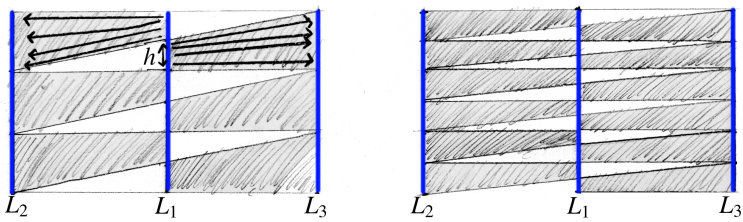


FIGURE 3.32 – *Une illustration classique du problème de l'existence de solutions au problème de Monge : il n'existe aucune application de transport optimal entre un segment L_1 et deux segments parallèles L_2 et L_3 (il est toujours possible d'en trouver un meilleur en remplaçant h par $h/2$).*

Cette contrainte rend le problème très difficile. Essayons d'imaginer comment la prendre en compte dans un programme d'ordinateur : la contrainte concerne tous les sous-ensembles B de Y, comment peut-on imaginer ne serait-ce qu'un algorithme qui vérifie si elle est satisfaite pour une configuration donnée ? Nous allons voir une série de transformations en des problèmes équivalents, où les contraintes deviennent linéaires, pour arriver à la fin à un simple problème d'optimisation convexe, qu'on peut résoudre numériquement avec des méthodes classiques.

Ceci n'est pas la seule difficulté : la fonctionnelle du problème de Monge a une structure non-symétrique, qui pose quelques difficultés pour l'étude de l'existence de solutions du problème (M). Cette dissymétrie provient du fait que T doive être une application, ceci permet de fusionner de la matière (si $T(x_1) = T(x_2)$ pour deux points différents x_1 et x_2), mais pas de scinder de la matière (pour cela, il faudrait que T puisse envoyer un même point x vers deux points différents y_1 et y_2). La figure 3.32 illustre ce problème : supposons qu'on souhaite calculer le transport optimal entre un segment vertical L_1 (symbolisant un « mur de terre ») et deux segments parallèles L_2 and L_3 (symbolisant deux « tranchées » de profondeur égale à la moitié de la hauteur du mur de terre), vers lesquelles on souhaite déplacer la terre du mur pour aplanir l'ensemble. Il est possible de décomposer L_1 en segments de longueur h, envoyés alternativement vers L_2 et L_3 (figure 3.32 à gauche). Pour toute longueur h, il est toujours possible de trouver une meilleure application T, à savoir avec une valeur plus faible de la fonctionnelle dans (3.17), en subdivisant L_1 en des segments plus petits (figure 3.32 à droite). La meilleure manière de procéder consiste à envoyer en tout point de L_1 *la moitié* de la terre vers L_2 et L_3, ce qui ne peut pas être représenté par une application. Ainsi, la meilleure solution du problème (M) n'est pas une application. D'une manière générale, ce

problème survient dès que la mesure source μ a de la masse concentrée sur une variété de dimension $d-1$ [126] (comme le segment L_1).

3.4.2 La relaxation de Kantorovich

Pour s'affranchir de cette difficulté, et définir un problème qui ait un espace de solutions plus large, Kantorovich a proposé une relaxation du problème (M), où la masse peut être à la fois scindée et fusionnée. L'idée consiste à chercher une mesure γ définie sur l'espace produit $X \times Y$ de la manière suivante :

$$(K) \qquad \begin{array}{l} \inf\limits_{\gamma} \left\{ \int\limits_{X\times Y} c(x,y)d\gamma \mid \gamma \geqslant 0 \text{ et } \gamma \in \Pi(\mu,\nu) \right\} \\ \text{avec :} \\ \Pi(\mu,\nu) = \{\gamma \in \mathcal{P}(X\times Y) \mid (P_1)\sharp\gamma = \mu \ ; \ (P_2)\sharp\gamma = \nu\} \end{array} \tag{3.18}$$

où (P_1) et (P_2) dénotent les deux projections $(x,y) \in X \times Y \mapsto x$ et $(x,y) \in X \times Y \mapsto y$ respectivement.

Les mesures poussées en avant par les deux projections $(P_1)\sharp\gamma$ et $(P_2)\sharp\gamma$ sont appelés *marginales* de γ. Les mesures γ dans $\Pi(\mu,\nu)$, à savoir qui ont μ et ν comme marginales, sont appelées des *plans de transport*. Les deux contraintes sur les marginales $(P_1)\sharp\gamma = \mu$ et $(P_2)\sharp\gamma = \nu$ définissant l'ensemble des plans de transports $\Pi(\mu,\nu)$ peuvent s'écrire également :

$$\begin{array}{rcl} (P_1)\sharp\gamma = \mu & \iff & \forall B \subset X, \int_B d\mu = \int_{B\times Y} d\gamma \\ (P_2)\sharp\gamma = \nu & \iff & \forall B \subset Y, \int_B d\nu = \int_{X\times B} d\gamma \end{array} \tag{3.19}$$

(d'après la définition de $(P_1)\sharp\gamma$ et $(P_2)\sharp\gamma$). Intuitivement, ces deux contraintes signifient respectivement que tout ce qui part d'un sous-ensemble de X doit correspondre à ce qu'il y avait initialement dans ce sous-ensemble, et que tout ce qui arrive dans un sous-ensemble de Y doit correspondre à ce que l'on souhaitait y trouver.

Parmi les plans de transport, ceux qui sont de la forme $(Id \times T)\sharp\mu$ correspondent à une application de transport T :

Remarque 3.4.1. *Si $(Id \times T)\sharp\mu \in \Pi(\mu,\nu)$, alors T pousse μ sur ν.*

Démonstration. $(Id \times T)\sharp\mu$ est dans $\Pi(\mu,\nu)$, ainsi $(P_2)\sharp(Id \times T)\sharp\mu = \nu$, ou encore $((P_2) \circ (Id \times T))\,\sharp\mu = \nu$, d'où $T\sharp\mu = \nu$. ■

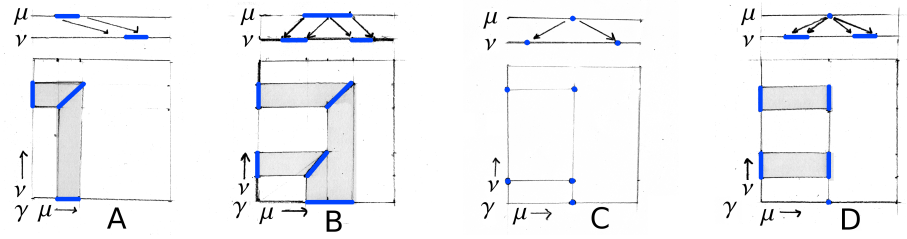


FIGURE 3.33 – *Quatre exemples de plans de transport en 1D. A : un segment est translaté. B : un segment est scindé en deux segments. C : une masse de Dirac est scindée en deux masses de Dirac. D : une masse de Dirac est répartie sur deux segments. Les deux premiers (A et B) sont sous la forme $(Id \times T)\sharp\mu$ où T est une application de transport, alors que le troisième et quatrième (C et D) ne le sont pas, parce que chacun d'eux scinde une masse de Dirac.*

Nous pouvons ainsi observer que si un plan de transport a la forme $\gamma = (Id \times T)\sharp\mu$, alors (K) devient

$$\min\left\{\int_{X\times Y} c(x,y)d\left((Id\times T)\sharp\mu\right)\right\} \quad = \quad \min\left\{\int_X c(x,T(x))d\mu\right\}$$

(on retrouve la même fonctionnelle que dans le problème de Monge initial).

Pour donner une compréhension intuitive de cette notion de plan de transport, nous montrons quatre exemples en 1D sur la figure 3.33 (le plan de transport est alors en $1D \times 1D = 2D$). Intuitivement, le plan de transport γ peut être vu comme une « table » indicée par x et y qui indique la quantité de matière de x transportée vers y. Plus exactement, la mesure γ sur $X \times Y$ est non-nulle sur les sous-ensembles qui contiennent des points (x, y) tels que de la matière soit transportée de x vers y. Quand γ dérive d'une application de transport T, à savoir quand γ est de forme $(Id \times T)\sharp\mu$, alors on peut aussi considérer γ comme le « graphe de T », comme dans les deux premiers exemples (A) et (B) [10].

Les plans de transport des deux autres exemples (C) et (D) n'ont pas d'application de transport associée, car ils scindent des masses de Dirac. Le plan de transport associé au cas de la figure 3.32 est de même nature (mais en $2D \times 2D = 4D$). Il n'admet pas d'écriture sous la forme $(Id \times T)\sharp\mu$ car il scinde la masse concentrée en L_1 vers L_2 et L_3.

10. pour l'exemple (B), l'application de transport T n'est pas définie au centre du segment, mais ceci ne pose pas de problème car on peut ôter de X tout sous-ensemble de mesure nulle, et dans cet exemple il n'y a pas de mesure concentrée au centre du segment.

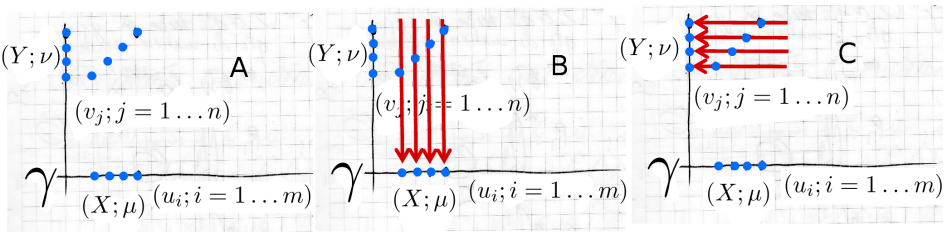


FIGURE 3.34 – *Une version discrète du problème de Kantorovich.*

À partir de ce point, une approche standard pour attaquer ce genre de problème d'existence est de trouver une certaine *régularité* à la fois dans la fonctionnelle et dans l'espace des plans de transport admissibles, à savoir prouver que la fonctionnelle est suffisamment « lisse » et trouver un ensemble compact de plans de transport admissibles. Comme l'ensemble des plans de transport admissibles contient au moins la mesure produit $\mu \otimes \nu$, il est non vide, et l'existence peut être prouvée à l'aide d'un argument topologique qui exploite la régularité de la fonctionnelle et la compacité de l'ensemble (voir [198] chapitre 2). Une fois l'existence d'un plan de transport établie, une autre question intéressante concerne l'existence d'une application de transport associée. Malheureusement, le problème (K) n'a pas directement la structure requise par ce chemin de raisonnement. Toutefois, nous pouvons observer que (K) est un problème d'optimisation linéaire sous contraintes linéaires, ce qui suggère l'utilisation de certains outils, tels que la formulation duale, qui a été développée par Kantorovich. Cette formulation duale permet de mettre en évidence une structure intéressante, qui permet de répondre aux questions précédentes (existence d'un plan de transport, et savoir s'il existe une application de transport correspondant à ce plan de transport).

3.4.3 Le problème dual de Kantorovich

La dualité de Kantorovich s'applique au problème (K), sous sa forme générale (avec des mesures). Afin de faciliter la compréhension du raisonnement, nous allons considérer une version discrète du problème (K), qui permet de le modéliser à l'aide de vecteurs et de matrices (plutôt que des mesures et des opérateurs), et ainsi de mieux cerner la structure de (K), à savoir la nature linéaire de la fonctionnelle et des contraintes. Ceci permet également d'expliquer le raisonnement de construction du dual en manipulant des objets habituels (matrices, vecteurs), mais la structure globale du raisonnement reste la même dans le cas général.

Problème de Kantorovich discret. La figure 3.34 montre la version discrète du transport entre deux segments 1D de la figure 3.33. Les mesures μ et ν sont remplacées par des vecteurs $U = (u_i)_{i=1...m}$ et $V = (v_j)_{i=1...n}$. Le plan de transport γ devient un ensemble de coefficients γ_{ij}, dont chacun indique la part de la quantité u_i qui sera transportée vers v_j. Le problème de Kantorovich discret s'écrit de la manière suivante :

$$\min_{\gamma} < C, \gamma > \text{ sous les contraintes } \begin{cases} P_1\gamma = U \\ P_2\gamma = V \\ \gamma_{i,j} \geqslant 0 \quad \forall i, j \end{cases} \tag{3.20}$$

où γ est le vecteur de $\mathbb{R}^{m \times n}$ de tous les coefficients γ_{ij} (la matrice des γ_{ij} « déroulée » dans un vecteur), et C le vecteur de $\mathbb{R}^{m \times n}$ des coefficients c_{ij} indiquant le coût de transport du point i au point j (par exemple la distance Euclidienne). La fonction objectif est simplement le produit scalaire, noté $< C, \gamma >$, entre le vecteur C des coûts et le vecteur γ. La fonction objectif est **linéaire en** γ. Les contraintes sur les marginales (3.19) imposent dans cette version discrète que les sommes des γ_{ij} sur les colonnes correspondent aux u_i (figure 3.33-B) et que les sommes sur les lignes correspondent aux v_j (figure 3.33-C). Intuitivement, tout ce qui part d'un point i doit correspondre à u_i, à savoir la quantité de matière initialement présente en i, et tout ce qui arrive à un point j doit correspondre à v_j, à savoir la quantité de matière souhaitée en j. Ces deux contraintes sont également **linéaires en** γ, et peuvent s'écrire à l'aide de matrices P_1 et P_2, de dimensions $m \times mn$ et $n \times mn$ respectivement.

Construction du dual de Kantorovich dans le cas discret. Nous introduisons à présent, de manière tout à fait arbitraire pour le moment, la fonction L suivante :

$$L(\varphi, \psi) = < C, \gamma > - < \varphi, P_1\gamma - U > - < \psi, P_2\gamma - V >$$

qui prend en paramètre des vecteurs φ de $\mathbb{R}^m$ et ψ de $\mathbb{R}^n$, construite à partir de la fonction objectif $< C, \gamma >$ de laquelle on soustrait les produits scalaires de φ et ψ avec les vecteurs correspondant au degré de violation des contraintes. Nous pouvons observer que :

$$\begin{aligned} \sup_{\varphi,\psi}[L(\varphi, \psi)] &= < C, \gamma > \text{ si } P_1\gamma = U \text{ et } P_2\gamma = V \\ &= +\infty \text{ sinon.} \end{aligned}$$

En effet, si par exemple une des composantes i de $P_1\gamma$ est non-nulle, on peut rendre L aussi grande qu'on veut en choisissant bien le coefficient φ_i.

Considérons à présent :

$$\inf_{\gamma \geqslant 0} \left[\sup_{\varphi,\psi} [L(\varphi,\psi)] \right] = \inf_{\substack{\gamma \geqslant 0 \\ P_1\gamma = U \\ P_2\gamma = V}} [< C, \gamma >] .$$

L'égalité entre les deux termes découle du fait que pour minimiser $\sup[L(\varphi,\psi)]$, γ n'a pas d'autre choix que de satisfaire les contraintes (voir observation précédente). Nous obtenons ainsi une nouvelle écriture (terme de gauche) du problème de Kantorovich discret (terme de droite). Examinons à présent cette nouvelle écriture en remplaçant L par son expression :

$$\inf_{\gamma \geqslant 0} \left[\sup_{\varphi,\psi} [< C, \gamma > - < \varphi, P_1\gamma - U > - < \psi, P_2\gamma - V >] \right] = \quad (3.21)$$

$$\sup_{\varphi,\psi} \left[\inf_{\gamma \geqslant 0} [< C, \gamma > - < \varphi, P_1\gamma - U > - < \psi, P_2\gamma - V >] \right] = \quad (3.22)$$

$$\sup_{\varphi,\psi} \left[\inf_{\gamma \geqslant 0} [< \gamma, C - P_1^t\varphi - P_2^t\psi > + < \varphi, U > + < \psi, V >] \right] = \quad (3.23)$$

$$\sup_{\substack{\varphi,\psi \\ P_1^t\varphi + P_2^t\psi \leqslant C}} [< \varphi, U > + < \psi, V >] . \quad (3.24)$$

La première étape (3.22) consiste à échanger inf et sup. Ensuite, on réarrange les termes (3.23). En réinterprétant cette équation comme un problème d'optimisation sous contraintes (avec un raisonnement similaire à celui du paragraphe précédent), on obtient finalement le problème d'optimisation (3.24). Dans la contrainte $P_1^t\varphi + P_2^t\psi \leqslant C$, l'inégalité est à considérer composante par composante. Le problème (3.24) se réécrit alors :

$$\begin{array}{l} \sup_{\varphi,\psi} [< \varphi, U > + < \psi, V >] \\ \text{sous la contrainte : } \varphi_i + \psi_j \leqslant c_{ij} \quad \forall i, j. \end{array} \quad (3.25)$$

Par rapport au problème primal (3.20) qui a $m \times n$ variables (les valeurs du plan de transport γ_{ij} pour chaque couple de points (i, j)), ce problème dual a $m + n$ variables (les composantes ψ_i et ψ_j attachées aux points source et aux points destination). Nous verrons plus loin comment réduire encore le nombre de variables, mais avant cela, nous allons revenir au continu (à savoir, avec des fonctions, des mesures et des opérateurs).

Le dual de Kantorovich en continu. Le même raisonnement peut être appliqué au problème de Kantorovich continu 3.18, pour aboutir au problème (DK) suivant :

$$(DK) \qquad \sup_{\varphi,\psi} \left[\int_X \varphi d\mu + \int_Y \psi d\nu \right] \qquad (3.26)$$
$$\text{sous la contrainte :}$$
$$\varphi(x) + \psi(y) \leqslant c(x,y) \quad \forall (x,y) \in X \times Y$$

où maintenant φ et ψ sont des fonctions définies sur X et Y [11].

L'image classique qui donne un sens intuitif à ce problème dual est de considérer qu'au lieu de transporter nous-même la terre, nous faisons appel à une entreprise qui va réaliser le travail à notre place. L'entreprise a une manière spéciale de fixer son prix : la fonction $\varphi(x)$ correspond au tarif pour charger la terre en x, et $\psi(y)$ au tarif pour décharger en y. L'entreprise cherche à maximiser son profit (c'est pourquoi le dual utilise un sup plutôt qu'un inf), mais elle ne peut pas facturer plus cher que ce que cela nous coûterait si nous effectuions le travail nous-même (contrainte).

L'existence pour (DK) reste difficile à étudier, car l'ensemble des fonctions φ, ψ qui satisfont la contrainte n'est pas compact. Toutefois, il est possible de révéler plus de structure dans le problème, en introduisant la notion de *c-conjuguée*, qui va permettre de trouver une classe de fonctions admissibles présentant suffisamment de régularité :

Définition 3.4.2. *Soit une fonction $f : X \to \mathbb{R} \cup \{-\infty, +\infty\}$, la c-conjuguée f^c est définie sur Y par :*

$$f^c(y) := \inf_{x \in X} [c(x,y) - f(x)]$$

— *Si une fonction φ est telle qu'il existe une fonction f telle que $\varphi = f^c$, alors φ est dite* c-concave*;*
— *$\mathbf{\Psi}_c(X)$ dénote l'ensemble des fonctions c-concaves sur X.*

Nous allons maintenant remarquer deux propriétés de (DK) qui vont nous permettre de nous restreindre à la classe des fonctions c-concaves pour chercher φ et ψ :

Remarque 3.4.3. *Si le couple (φ, ψ) est admissible pour (DK), alors (φ, φ^c) est également admissible.*

11. les fonctions φ et ψ doivent être dans $L^1(\mu)$ et $L^1(\nu)$. La démonstration de l'équivalence avec le problème (K) requiert un petit peu plus de précautions que dans le cas discret, en particulier l'étape (3.22) (échange du sup et de l'inf), qui utilise un résultat d'analyse convexe (dû à Rockafellar), voir [199, chapitre 5].

Démonstration.

$$\begin{cases} \forall (x,y) \in X \times Y, \varphi(x) + \psi(y) \leqslant c(x,y) \\ \varphi^c(y) = \inf\limits_{x \in X} [c(x,y) - \varphi(x)] \end{cases}$$
$$\begin{array}{rcl} \varphi(x) + \varphi^c(y) & = & \varphi(x) + \inf_{x' \in X} [c(x',y) - \varphi(x')] \\ & \leqslant & \varphi(x) + c(x,y) - \varphi(x) \\ & \leqslant & c(x,y). \end{array}$$

■

Remarque 3.4.4. *Si le couple (φ, ψ) est admissible pour (DK), alors nous obtenons un* meilleur candidat *en remplaçant ψ par φ^c :*

Démonstration.

$$\left.\begin{array}{rcl} \varphi^c(y) & = & \inf\limits_{x \in X} [c(x,y) - \varphi(x)] \\ \forall x \in X, \quad \psi(y) & \leqslant & c(x,y) - \varphi(x) \end{array}\right\} \quad \Rightarrow \quad \psi(y) \leqslant \varphi^c(y),$$

■

ou en termes de notre image intuitive, en remplaçant ψ par φ^c, l'entreprise peut faire payer plus cher le client et le tarif reste acceptable (à savoir la contrainte reste satisfaite). Ainsi, nous avons :

$$\inf(K) \quad = \sup_{\varphi \in \mathbf{\Psi}_c(X)} \int_X \varphi d\mu + \int_Y \varphi^c d\nu \quad = \sup_{\psi \in \mathbf{\Psi}_c(Y)} \int_X \psi^c d\nu + \int_Y \psi d\mu$$

Nous ne donnerons pas ici le détail de la preuve de l'existence. Le lecteur peut se référer à [199, chapitre 4]. L'idée est que nous sommes à présent dans une situation bien meilleure, puisque l'ensemble des fonctions admissibles $\mathbf{\Psi}_c(Y)$ est compact [12].

La valeur optimale donne une information intéressante, à savoir le coût pour transformer μ en ν. Ceci permet de définir une distance entre des distributions, et permet ainsi de les comparer, ce qui est utile dans plusieurs applications.

12. à condition que l'on fixe la valeur d'une des fonctions de ψ en un point de Y afin de supprimer l'invariance par addition d'une constante.

3.4.4 Du problème de Kantorovich à l'application de transport.

Sous-différentiel. Supposons maintenant que nous souhaitions en plus du coût connaître la manière de transformer μ en ν associée au coût, à savoir, quand elle existe, l'application T de X dans Y dont le plan de transport associé $(Id \times T)\sharp\mu$ minimise la fonctionnelle du problème de Monge. Un résultat caractérise le support de γ, à savoir l'ensemble $\partial_c\psi$ des paires de points (x, y) connectés par le plan de transport :

Théorème 3.4.5.

$$\forall (x, y) \in \partial_c\psi, \nabla\psi(x) - \nabla_x c(x, y) = 0$$

où $\partial_c\psi = \{(x, y)|\varphi(x) + \psi(y) = c(x, y)\}$ *dénote le* sous-différentiel *de* ψ.

Démonstration. Voir [199] chapitres 9 et 10. ■

Nous donnons ici un résumé de l'argument heuristique donné au début du même chapitre de [199], qui permet d'avoir une intuition : considérons un point (x, y) du sous-différentiel $\partial_c\psi$, qui satisfait donc :

$$\varphi(y) + \psi(x) = c(x, y). \tag{3.27}$$

Par définition, $\varphi(y) = \psi^c(y) = \inf_x c(x, y) - \psi(x)$, ainsi $\forall \widetilde{x}, \varphi(y) \leqslant c(\widetilde{x}, y) - \psi(\widetilde{x})$, ou encore :

$$\varphi(y) + \psi(\widetilde{x}) \leqslant c(\widetilde{x}, y). \tag{3.28}$$

En substituant (3.28) dans (3.27), nous obtenons $\psi(\widetilde{x}) - \psi(x) \leqslant c(\widetilde{x}, y) - c(x, y)$ pour tout $\widetilde{x}$.

Imaginons maintenant que $\widetilde{x}$ suive une trajectoire paramétrée par ϵ et prenant son origine en x. Nous pouvons calculer le gradient en x par rapport à une direction arbitraire w en considérant la limite quand ϵ tend vers zéro dans la relation $\frac{\psi(\widetilde{x})-\psi(x)}{\epsilon} \leqslant \frac{c(\widetilde{x},y)-c(x,y)}{\epsilon}$. Ainsi nous avons $\nabla\psi(x) \cdot w \leqslant \nabla_x c(x, y) \cdot w$. Nous pouvons également effectuer le même calcul avec la direction $-w$ au lieu de w, et nous obtenons alors : $\forall w, \nabla\psi(x) \cdot w = \nabla_x c(x, y) \cdot w$, ainsi $\forall (x, y) \in \partial_c\psi, \nabla\psi(x) - \nabla_x c(x, y) = 0$.
Note : les calculs ci-dessous sont purement formels et ne constituent pas une preuve. La preuve complète requiert une analyse plus fine, utilisant une notion de différentiabilité généralisée, ainsi que des outils de l'analyse convexe.

Dans le cas du coût L_2, à savoir avec $c(x, y) = 1/2\|x - y\|^2$, cette relation devient $\forall (x, y) \in \partial_c\psi, \nabla\psi(x) + y - x = 0$, ainsi, quand l'application

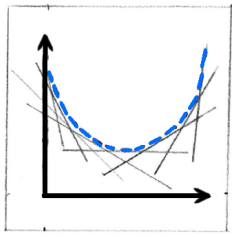


FIGURE 3.35 – *L'enveloppe supérieure d'une famille de fonctions affines est une fonction convexe.*

de transport optimal T existe, elle est donnée par $T(x) = x - \nabla\psi(x) = \nabla(\|x\|^2/2 - \psi(x))$. Non seulement ceci nous donne une expression de T, ce qui est déjà intéressant en soi, mais de plus, ceci permet de caractériser T comme le gradient d'une fonction *convexe* (voir également le théorème de factorisation polaire de Brenier [35]), ce qui est une propriété intéressante, car elle implique que deux « particules transportées » $x_1 \mapsto T(x_1)$ et $x_2 \mapsto T(x_2)$ ne peuvent pas collisionner. Nous regardons maintenant comment prouver ces deux assertions (T gradient d'une fonction convexe et absence de collisions) dans le cas du transport L_2 ($c(x,y) = \|x-y\|^2$).

Remarque 3.4.6. *Si $c(x,y) = 1/2\|x-y\|^2$ et $\psi \in \mathbf{\Psi}_c(X)$, alors $\bar{\psi} : x \mapsto \bar{\psi}(x) = \|x\|^2/2 - \psi(x)$ est une fonction convexe (c'est une équivalence si $X = Y = \mathbb{R}^d$).*

Démonstration.

$$\begin{aligned}
\psi(x) &= \varphi^c(x) \\
&= \inf_y \tfrac{\|x-y\|^2}{2} - \varphi(y) \\
&= \inf_y \tfrac{\|x\|^2}{2} - x\cdot y + \tfrac{\|y\|^2}{2} - \varphi(y) \\
-\bar{\psi}(x) &= \psi(x) - \tfrac{\|x\|^2}{2} = \inf_y \left[-x\cdot y + \left(\tfrac{\|y\|^2}{2} - \varphi(y)\right)\right] \\
\bar{\psi}(x) &= \sup_y \left[x\cdot y - \left(\tfrac{\|y\|^2}{2} - \varphi(y)\right)\right].
\end{aligned}$$

La fonction $x \mapsto x\cdot y - \left(\frac{\|y\|^2}{2} - \varphi(y)\right)$ est affine en x, donc le graphe de $\bar{\psi}$ est l'enveloppe supérieure d'une famille d'hyperplans, donc $\bar{\psi}$ est convexe (voir figure 3.35). ■

Remarque 3.4.7. *Considérons les trajectoires de deux particules paramétrées par $t \in [0,1]$, $t \mapsto (1-t)x_1 + tT(x_1)$ et $t \mapsto (1-t)x_2 + tT(x_2)$. Si $x_1 \neq x_2$ et $0 < t < 1$ alors il n'existe pas de collision entre les deux particules.*

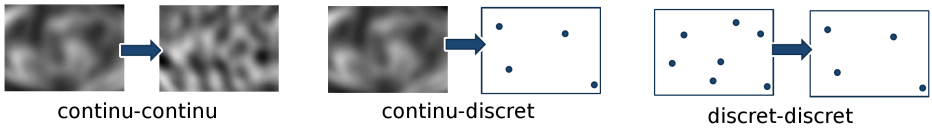


FIGURE 3.36 – *Différents types de transports, suivant que les mesures μ ou ν soient continues ou discrètes.*

Démonstration. Par contradiction, supposons la présence d'une collision, à savoir qu'il existe $t \in (0,1)$ et $x_1 \neq x_2$ tels que :

$$\begin{array}{lcl}
(1-t)x_1 + tT(x_1) & = & (1-t)x_2 + tT(x_2) \\
(1-t)x_1 + t\nabla\bar{\psi}(x_1) & = & (1-t)x_2 + t\nabla\bar{\psi}(x_2) \\
(1-t)(x_1-x_2) + t(\nabla\bar{\psi}(x_1) - \nabla\bar{\psi}(x_2)) & = & 0 \\
\forall v, (1-t)v\cdot(x_1-x_2) + tv\cdot(\nabla\bar{\psi}(x_1) - \nabla\bar{\psi}(x_2)) & = & 0 \\
\qquad \text{prenons } v = (x_1-x_2) & & \\
(1-t)\|x_1-x_2\|^2 + t(x_1-x_2)\cdot(\nabla\bar{\psi}(x_1) - \nabla\bar{\psi}(x_2)) & = & 0.
\end{array}$$

La dernière étape aboutit à une contradiction, parce que le terme de gauche est la somme de deux nombres strictement positifs (en rappelant la définition de la convexité de $\bar{\psi} : \forall x_1 \neq x_2, (x_1-x_2)\cdot(\nabla\bar{\psi}(x_1) - \nabla\bar{\psi}(x_2)) > 0$). ■

3.4.5 Transport continu, discret ou semi-discret

Les développements présentés dans les sous-sections précédentes sont valables pour tout couple de mesures source et cible μ et ν, qui peuvent être aussi bien dérivées de fonctions continues qu'être des mesures discrètes (sommes de masses de Dirac). La figure 3.36 présente trois configurations intéressantes à étudier, ayant des propriétés spécifiques conduisant à des algorithmes différents pour calculer le transport. Nous donnons ici quelques indications sur les cas *continu* $\rightarrow$ *continu* et *discret* $\rightarrow$ *discret*, et développerons plus en détail le cas *continu* $\rightarrow$ *discret* dans la sous-section suivante.

Le cas continu $\rightarrow$ continu et l'équation de Monge-Ampère. Nous rappelons que quand l'application de transport optimal existe, dans le cas du coût L_2 (à savoir, $c(x,y) = \|x-y\|^2$), elle peut être déduite de la fonction ψ en utilisant la relation $T(x) = \nabla\bar{\psi} = x - \nabla\psi$. La formule du changement de variables dans une intégrale sur un sous-ensemble B de X s'écrit :

$$\forall B \subset X, \int_B \mu(x)d\mu = \mu(B) = \nu(T(B)) = \int_B |\det J_T(x)|\, T(x)d\nu \qquad (3.29)$$

où J_T dénote la matrice Jacobienne de T et det le déterminant.

Si μ et ν ont toutes les deux une densité u et v, à savoir $\forall B, \mu(B) = \int_B u(x)dx$ et $\nu(B) = \int_B v(x)dx$, alors on peut (formellement) considérer (3.29) en un point de X :

$$\forall x \in X, u(x) = |\det J_T(x)|\, v(T(x)). \tag{3.30}$$

En injectant $T = \nabla\bar{\psi}$ et $J_T = H\bar{\psi}$ dans (3.30), on obtient :

$$\forall x \in X, u(x) = |\det H\bar{\psi}(x)|\, v(\nabla\bar{\psi}(x)), \tag{3.31}$$

où $H\bar{\psi}$ dénote la matrice Hessienne de $\bar{\psi}$. L'équation (3.31) est connue comme *l'équation de Monge-Ampère*. Il s'agit d'une équation hautement non-linéaire, et ses solutions quand elles existent ont souvent des singularités [13]. À noter que les calculs effectués ci-dessous sont purement formels, et l'étude des solutions de l'équation de Monge-Ampère requiert l'utilisation d'outils plus élaborés. En particulier, il est possible de définir plusieurs types de solutions faibles (solutions de viscosité, solutions au sens de Brenier ...). Plusieurs algorithmes pour calculer des solutions numériques de l'équation de Monge-Ampère ont été proposés, voir par exemple l'algorithme de Benamou-Brenier [21], qui utilise une formulation de type dynamique des fluides, et voir également [144].

Le cas discret → discret. Si μ est une somme de m masses de Dirac et ν une somme de n masses de Dirac, alors on cherche les $m \times n$ coefficients γ_{ij} qui donnent pour tout couple de points i de l'espace de départ et j de l'espace d'arrivée la quantité de matière transportée de i vers j. Ceci correspond au plan de transport dans le problème de Kantorovich discret (voir 3.4.3). Ce type de problème (dit *problème d'assignation*) peut être résolu par différentes méthodes de programmation linéaire [38]. Ces méthodes peuvent être considérablement accélérées en ajoutant un terme de régularisation, qui s'interprète comme l'entropie du plan de transport [117]. Cette version régularisée du problème peut être résolue par des algorithmes numériques très efficaces [54].

Le cas continu → discret (également appelé *semi-discret*) correspond à une fonction continue transportée vers une somme de masses de Dirac (voir exemples de fonctions c-concaves dans [81]). Le transport semi-discret a des relations intéressantes avec des notions de géométrie algorithmique, et

13. de la même manière que l'équation eikonale dont la solution est le champ distance, qui a une singularité sur l'axe médian.

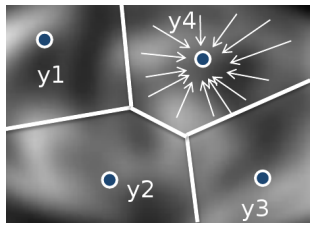


FIGURE 3.37 – *Transport semi-discret : les niveaux de gris symbolisent la quantité d'une ressource devant être transportée vers 4 usines, qui vont se partager le territoire en fonction de ce que chaque usine souhaite collecter.*

des ensembles de polyèdres convexes, qui ont été étudiés par Alexandrov [5] et plus tard par Aurenhammer, Hoffman et Aranov [17]. La section suivante est consacrée au transport semi-discret.

3.4.6 Transport semi-discret

Nous supposons à présent que la mesure source μ est continue (de densité u), et que la mesure cible ν est une somme de masses de Dirac (mesure empirique). Un exemple de ce type de configuration est le cas d'une ressource dont la quantité disponible en un point est représentée par une fonction u, devant être collectée par un ensemble d'usines, comme sur la figure 3.37. On suppose que chacune des n usines doive collecter une quantité ν_j fixée à l'avance, dont la somme correspond à la quantité totale de ressource disponible ($\sum_{j=1}^{n} \nu_j = \int_X u(x)dx$).

Problème de Monge semi-discret. Le problème de Monge s'écrit alors de la manière suivante :

$$(M) \qquad \inf_{T:X\to Y} \int_X c(x, T(x))u(x)dx \text{ sous la contrainte : } \forall j, \int_{T^{-1}(y_j)} = \nu_j.$$

Une application de transport T associe à chaque point x de X l'un des points y_j. Ainsi, on peut partitionner X, en associant à chaque y_j la région $T^{-1}(y_j)$ contenant les points qui vont être transportés en y_j par T. La contrainte impose que la quantité de ressource collectée sur chacune des régions $T^{-1}(y_j)$ corresponde à la quantité souhaitée ν_j.

Examinons à présent la forme que prend le problème dual de Kantorovich. En termes de mesure, la mesure source μ est associée à la densité u, et la mesure cible ν est une somme de masses de Dirac $\nu = \sum_{j=1}^{n} \nu_j \delta_{y_j}$,

supportée par l'ensemble des points $Y = \{y_j\}$. Nous rappelons que le problème dual de Kantorovich s'écrit dans le cas général de la manière suivante :

$$\sup_{\psi \in \mathbf{\Psi^c}(Y)} \left[\int_X \psi^c(x) d\mu + \int_Y \psi(y) d\nu \right]. \tag{3.32}$$

Dans notre cas semi-discret, la fonctionnelle devient une fonction de n variables, qui prend la forme suivante :

$$F(\psi) = F(\psi_1, \psi_2, \dots \psi_n) = \int_X \psi^c(x) u(x) dx + \sum_{j=1}^{n} \psi_j \nu_j = \tag{3.33}$$

$$\int_X \inf_{y_j \in Y} \left[c(x, y_j) - \psi_j \right] u(x) dx + \sum_{j=1}^{n} \psi_j \nu_j = \tag{3.34}$$

$$\sum_{j=1}^{n} \int_{\mathrm{L}ag^c_\psi(y_j)} \left(c(x, y_j) - \psi_j \right) u(x) dx + \sum_{j=1}^{n} \psi_j \nu_j. \tag{3.35}$$

La première étape (3.33) prend en compte la nature des mesures μ et ν dans notre cas. En particulier, on remarquera que la mesure ν est définie par les réels ν_j associés aux points y_j, et que la fonction ψ est définie par les réels ψ_j correspondant à sa valeur en chaque point y_j. L'intégrale $\int_Y \psi(y) d\nu$ devient le produit scalaire $\sum_j \psi_j \nu_j$. Ainsi, la fonctionnelle correspondant au problème dual de Kantorovich devient une fonction F de n variables (les ψ_j). Remplaçons maintenant la c-conjuguée ψ^c par son expression, de manière à obtenir (3.34). L'intégrale du terme de gauche peut être réorganisée, en regroupant ensemble les points de X pour lesquels le même y_j minimise $c(x, y_j) - \psi_j$, ce qui donne (3.35), où la *cellule de Laguerre* $\mathrm{L}ag^c_\psi(y_j)$ est définie par :

$$\mathrm{L}ag^c_\psi(y_j) = \left\{ x \in X \quad | \quad c(x, y_j) - \psi_j \leqslant c(x, y_k) - \psi_k \quad \forall k \right\}.$$

Le diagramme de Laguerre, formé par l'union des cellules de Laguerre, est une structure classique en géométrie algorithmique. Dans le cas du coût L_2, à savoir $c(x, y) = \|x - y\|^2$, il correspond au *diagramme de puissance*, qui a été étudié par Aurenhammer à la fin des années 80 [16], et qui a la particularité d'avoir des frontières de cellules rectilignes.

Concavité de F. La fonction objectif F est un cas particulier du dual de Kantorovich, et hérite donc naturellement de ses propriétés, telles que sa concavité (dont nous n'avons pas encore parlé), intéressante à la fois d'un point de vue théorique pour étudier l'existence et l'unicité, mais également

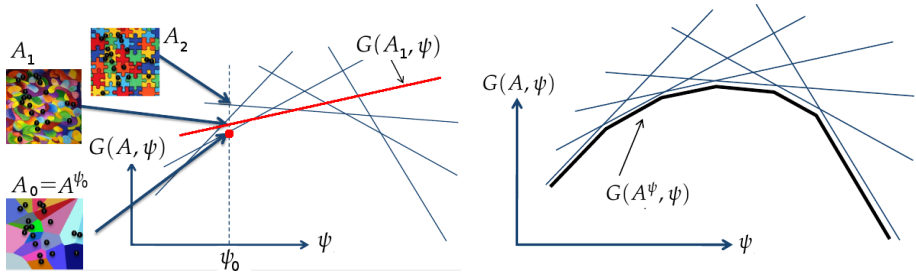


FIGURE 3.38 – *La fonction objectif du problème dual de Kantorovich est concave, parce que son graphe est l'enveloppe inférieure d'une famille de fonctions affines.*

d'un point de vue pratique. Dans le cas semi-discret, la concavité de F est plus simple à prouver que dans le cas général. Nous reprenons ici la preuve d'Aurenhammer *et. al* [17], qui conduit à un algorithme efficace [129], [118], [108].

Théorème 3.4.8. *La fonction objectif F du problème dual de Kantorovich semi-discret* (3.32) *est concave.*

Démonstration. Considérons la fonction G suivante :

$$G(A,[\psi_1,\dots\psi_n]) = \int_X \Big(c(x,y_{A(x)}) - \psi_{A(x)}\Big)\, u(x)dx, \tag{3.36}$$

paramétrée par une *affectation* $A : X \to [1\dots n]$, à savoir une fonction associant à tout x de X le numéro de l'un des y_j. Si on note $A^{-1}(j) = \{x|A(x) = j\}$, alors G peut également s'écrire :

$$\begin{aligned} G(A,\psi) &= \sum_j \int_{A^{-1}(j)} \big(c(x,y_j) - \psi_j\big)\, u(x)dx \\ &= \sum_j \int_{A^{-1}(j)} c(x,y_j)u(x)dx - \sum_j \psi_j \int_{A^{-1}(j)} u(x)dx. \end{aligned} \tag{3.37}$$

Le premier terme ne dépend pas des ψ_j, et le second est une combinaison linéaire des ψ_j, ainsi, pour un A fixé, $\psi \mapsto G(A,\psi)$ est une fonction affine de ψ. La figure 3.38 illustre l'allure du graphe de G pour différentes affectations. L'axe des abscisses symbolise l'ensemble des composantes du vecteur ψ (de dimension n) et les ordonnées la valeur de $G(A,\psi)$. Pour une affectation A fixée, le graphe de G est un hyperplan (symbolisé ici par une droite).

Parmi les affectations A possibles, nous distinguons A^ψ qui associe à un point x l'indice j de la cellule de Laguerre à laquelle x appartient [14], à

14. A n'est pas défini sur l'ensemble des frontières des cellules de Laguerre, mais ceci ne pose pas de problème car cet ensemble est de mesure nulle.

savoir :

$$A^{\psi}(x) = \arg\min_{j} \left[c(x, y_j) - \psi_j \right]$$

Pour une valeur donnée du vecteur $\psi = \psi^0$, parmi tous les A possibles, A^{ψ^0} est celui qui minimise la valeur de $G(A, \psi^0)$, car il minimise l'intégrande en chacun des points (voir figure 3.38 à gauche). Ainsi, G étant affine par rapport à ψ, le graphe de la fonction $\psi \to G(A^{\psi}, \psi)$ est l'enveloppe inférieure d'une famille de droites (figure 3.38 à droite), donc $\psi \to G(A^{\psi}, \psi)$ est concave. Finalement, la fonction objectif du problème dual de Kantorovich F peut se mettre sous la forme $F(\psi) = G(A^{\psi}, \psi) + \sum_j \nu_j \psi_j$, à savoir la somme d'une fonction concave et d'une fonction linéaire, elle est donc également concave. ■

L'application de transport optimal semi-discret. Examinons à présent le sous-différentiel $\partial_c \psi$, à savoir l'ensemble des points (x, y) liés par le plan de transport optimal. Le sous-différentiel est défini par $\partial_c \psi = \left\{ x, y_j \quad | \quad \psi^c(x) + \psi_j = c(x, y_j) \right\}$. Considérons un point x de X à l'intérieur de la cellule de Laguerre $\mathrm{Lag}^{\psi}(y_j)$, ainsi que le sous-différentiel $[\partial_c \psi](x)$ en x :

$$[\partial_c \psi](x) = \{ y_k \quad | \quad \psi^c(x) + \psi_k = c(x, y_k) \} = \tag{3.38}$$

$$\left\{ y_k \quad | \quad \inf_{y_l} [c(x, y_l) - \psi_l] + \psi_k = c(x, y_k) \right\} = \tag{3.39}$$

$$\left\{ y_k \quad | \quad c(x, y_j) - \psi_j + \psi_k = c(x, y_k) \right\} = \tag{3.40}$$

$$\left\{ y_k \quad | \quad c(x, y_j) - \psi_j = c(x, y_k) - \psi_k \right\} = \tag{3.41}$$

$$\left\{ y_j \right\}. \tag{3.42}$$

Dans la première étape (3.39) nous remplaçons ψ^c par sa définition, puis utilisons le fait que x soit dans la cellule de Laguerre de y_j (3.40), et enfin, le seul point de Y satisfaisant (3.41) est y_j parce que nous avons supposé *x à l'intérieur* de la cellule de Laguerre de y_j.

En résumé, l'application de transport optimal T transporte un point x vers le point y_j associé à la cellule de Laguerre $\mathrm{Lag}^{\psi}(y_j)$ qui contient x, avec $\psi_1, \ldots \psi_n$ l'unique vecteur qui maximise F et tel que ψ soit c-concave. Il est possible de montrer que ψ est c-concave si et seulement si aucune cellule de Laguerre n'est vide de matière, à savoir, l'intégrale de u ne s'annule sur aucune cellule de Laguerre.

Nous allons maintenant calculer les dérivées premières et secondes de la fonction objectif. Elles sont utiles pour l'écriture d'algorithmes d'optimisation numérique.

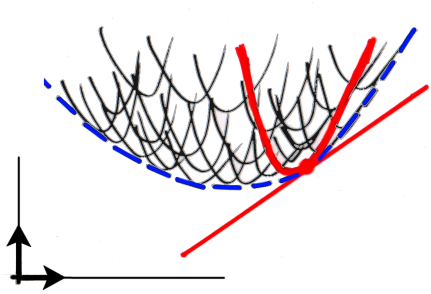


FIGURE 3.39 – *Illustration du « théorème de l'enveloppe » : la tangente en un point à l'enveloppe inférieure d'une famille de fonctions est la tangente à la courbe la plus basse en ce point.*

Dérivées premières de la fonction objectif. Du fait de sa concavité, F admet un maximum unique en ψ^*, caractérisé par $\nabla F(\psi^*) = 0$. Examinons à présent la forme du gradient $\nabla F = \nabla \left(G(A\psi, \psi) + \sum_{j=1}^{n} \psi_j \nu_j \right)$. À première vue, le terme en G va être difficile à dériver, car il implique des intégrales sur des domaines, les $A^{-1}(y_j)$, qui varient en fonction du vecteur de paramètres ψ, nécessitant des calculs un peu fastidieux, réalisables à l'aide du théorème de Reynolds (voir par exemple [123] Page 7). Toutefois, dans notre cas particulier, en se souvenant que le terme de gauche est l'enveloppe inférieure d'une famille de fonctions, on peut utiliser le *théorème de l'enveloppe* [131], illustré sur la figure 3.39 : en un point, le gradient de l'enveloppe inférieure d'une famille de fonctions est égal au gradient de la fonction ayant la valeur la plus petite en ce point. Ainsi, on peut ignorer les variations des cellules de Laguerre en fonction de ψ. En remplaçant $G(A_\psi, \psi)$ par son expression (3.37), on obtient :

$$\begin{aligned}
\frac{\partial}{\partial \psi_j} G(A_\psi, \psi) &= \frac{\partial}{\partial \psi_j} \left(\sum_k \int_{A_\psi^{-1}(k)} c(x, y_k) u(x) dx - \sum_k \psi_k \int_{A_\psi^{-1}(k)} u(x) dx \right) \\
&= - \int_{A_\psi^{-1}(j)} u(x) dx \\
&= - \int_{\mathrm{Lag}^\psi(y_j)} u(x) dx.
\end{aligned}$$

En se souvenant que la fonction objectif F dont on veut calculer le gradient a pour expression $F(\psi) = G(A_\psi, \psi) + \sum_j \nu_j \psi_j$, il vient :

$$\frac{\partial F}{\partial \psi_j} = \nu_j - \int_{\mathrm{Lag}^\psi(y_j)} u(x) dx. \tag{3.43}$$

Comme la fonction objectif F est concave, elle admet un unique maximum. En ce maximum, toutes les composantes du gradient s'annulent, ce qui implique que la quantité de matière récoltée en y_j, à savoir l'intégrale de u sur la cellule de Laguerre de y_j, correspond à la quantité initialement souhaitée, à savoir ν_j.

Dérivées secondes de la fonction objectif. Les coefficients de la matrice Hessienne $\partial^2 F/\partial\psi_i\partial\psi_j$ sont un petit peu plus difficiles à calculer, cette fois on n'échappe pas aux intégrales sur des domaines variables, ce qui nécessite d'appliquer le théorème de Reynolds. Nous ne détaillerons pas le calcul ici pour des raisons de place, mais donnons le résultat final.

Pour un coût c arbitraire, les dérivées secondes sont données par :

$$\begin{aligned}\frac{\partial^2 F}{\partial\psi_i\partial\psi_j} &= -\int_{\mathrm{Lag}^\psi(y_i)\cap\mathrm{Lag}^\psi(y_j)} \|\nabla_x c(x,y_i)-\nabla_x c(x,y_j)\|^{-1}u(x)dx \quad \forall j\neq i\\ \frac{\partial^2 F}{\partial\psi_i^2} &= -\sum_{j\neq i}\frac{\partial^2 F}{\partial\psi_i\partial\psi_j}.\end{aligned} \tag{3.44}$$

Pour le cas particulier du coût L_2, à savoir $c(x,y)=\|x-y\|^2$, les dérivées secondes deviennent :

$$\frac{\partial^2 F}{\partial\psi_i\partial\psi_j}=\frac{\int_{\mathrm{Lag}^\psi(y_i)\cap\mathrm{Lag}^\psi(y_j)} u(x)dx}{\|y_j-y_i\|^2} \quad \forall j\neq i. \tag{3.45}$$

Un algorithme de calcul du transport optimal semi-discret L_2. Avec la définition de $F(\psi)$, l'expression de ses dérivées premières (gradient ∇F) et secondes (matrice Hessienne $\nabla^2 F=\left(\partial^2 F/\partial\psi_i\partial\psi_j\right)_{ij}$), nous pouvons à présent décrire un algorithme de calcul du transport optimal semi-discret, fondé sur une version particulière [108] de la méthode d'optimisation de Newton [141] :

Données : un maillage supportant la densité source u
les points $(y_j)_{j=1}^n$
les quantités $(\nu_j)_{j=1}^n$

Résultat : le diagramme de Laguerre $\mathrm{L}ag^\psi$ tel que :
$$\int_{\mathrm{L}ag^\psi(y_j)} u(x)dx = \nu_j \quad \forall j$$

---

(1) $\psi \leftarrow [0 \dots 0]$
(2) Tant que la convergence n'est pas atteinte
(3) Calculer ∇F et $\nabla^2 F$
(4) Trouver $p \in \mathbb{R}^n$ tel que $\nabla^2 F(\psi)p = -\nabla F(\psi)$
(5) Trouver α
(6) $\psi \leftarrow \psi + \alpha p$
(7) Fin tant que

La mesure source est donnée par sa densité, à savoir une fonction u positive, linéaire par morceaux, supportée par un maillage triangulaire (en 2D) ou à base de tétraèdres (en 3D) sur un domaine X. La mesure cible est discrète, supportée par l'ensemble de points $Y = (y_j)_{j=1}^n$. Chaque point cible souhaite recevoir ν_j. Bien sûr, les quantités de matière se correspondent, à savoir $\int_X u(x)dx = \sum_j \nu_j$. L'algorithme calcule pour chaque point de la mesure cible le sous-ensemble de X qui lui sera associé par le transport optimal T, $T^{-1}(y_j) = \mathrm{L}ag^\psi(y_j)$, à savoir la cellule de Laguerre de y_j. Le diagramme de Laguerre est complètement déterminé par le vecteur ψ qui maximise F.

À la ligne (2), le critère de convergence classique pour un algorithme de Newton utilise la norme du gradient de F. Ici, le gradient de F a un sens géométrique, puisque ses composantes $\partial F/\partial \psi_j$ correspondent à la différence entre la quantité de matière souhaitée ν_j en j et celle effectivement obtenue $\int_{\mathrm{L}ag^\psi(y_j)} u(x)dx$. Nous pouvons alors comparer la plus grande erreur commise, donnée par la plus grande composante de ∇F en valeur absolue, avec la plus petite mesure souhaitée en un y_j. Ainsi, nous considérons que la convergence est atteinte si $\max |\nabla F_j| < \epsilon \min_j \nu_j$, pour un ϵ donné.

La ligne (3) calcule les coefficients du gradient et de la matrice Hessienne de F, en utilisant (3.43) et (3.44). Ces calculs impliquent des intégrales sur les cellules de Laguerre et sur leurs bords. Pour le coût L_2, à savoir avec $c(x, y) = \|x - y\|^2$, les bords des cellules de Laguerre sont

rectilignes, ce qui simplifie grandement les calculs des coefficients de la Hessienne (3.45). De plus, dans ce cas, il est possible d'utiliser des algorithmes efficaces de calcul du diagramme de Laguerre [32, 200]. Leur implantation est disponible dans plusieurs bibliothèques de programmation telles que GEOGRAM [15] et CGAL [16]. Ensuite, il faut calculer l'intersection entre chaque cellule de Laguerre et le maillage qui supporte la densité u, ce qui peut être réalisé à l'aide d'algorithmes dédiés [118], dont l'implantation est également disponible dans GEOGRAM.

La ligne (4) trouve le pas de Newton p en résolvant un système linéaire. Nous utilisons l'algorithme du gradient conjugué (voir §3.2). Là encore, le sens géométrique des composantes de ∇F permet de déterminer le paramètre de convergence.

La ligne (5) détermine le paramètre de descente α. Un résultat dû à Mérigot et Kitagawa [108] assure la convergence de l'algorithme de Newton si la mesure de la plus petite cellule de Laguerre reste supérieure à un certain seuil (à savoir la moitié de la mesure de la plus petite cellule à l'état initial). Ainsi, en partant de $\alpha = 1$, l'algorithme divise α par deux jusqu'à ce que cette condition soit satisfaite.

Refaisons à présent un petit retour en arrière vers la définition originale du problème de Monge (3.17). La contrainte initiale devant être respectée par l'application de transport T était très difficile, car elle caractérisait les pré-images par T de tout sous-ensemble de Y. Il est tout à fait remarquable qu'après les différentes étapes (relaxation de Kantorovich, dualité, c-convexité), le problème final devienne aussi simple, à savoir l'optimisation d'une fonction convexe régulière, dont on sait très bien calculer le gradient et la Hessienne dans le cas semi-discret.

Résultats, exemples et application du transport L_2 semi-discret. La figure 3.40 montre quelques exemples de transport 2D, entre une densité uniforme et un ensemble de points (A). Chaque cellule de Laguerre a la même aire. En considérant le même ensemble de points, mais cette fois avec une densité variable $u(x, y) = 10(1 + \sin(2\pi x)\sin(2\pi y))$ (image B), on obtient un autre diagramme de Laguerre, dont les cellules ont la même valeur de l'intégrale de la densité u. (C) : les coefficients du gradient et de la Hessienne de F font intervenir des intégrales de la densité u sur les cellules

15. http://alice.loria.fr/software/geogram/doc/html/index.html
16. http://www.cgal.org

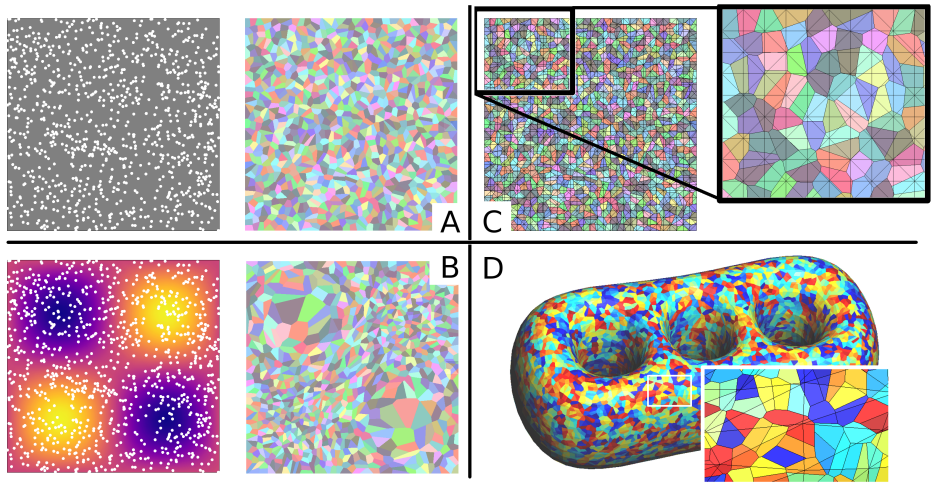


FIGURE 3.40 – *A : transport entre une densité uniforme et un ensemble de points aléatoires ; B : transport entre une densité variable et le même ensemble de points ; C : intersections de maillages ; D : transport entre une mesure supportée par une surface et un ensemble de points 3D.*

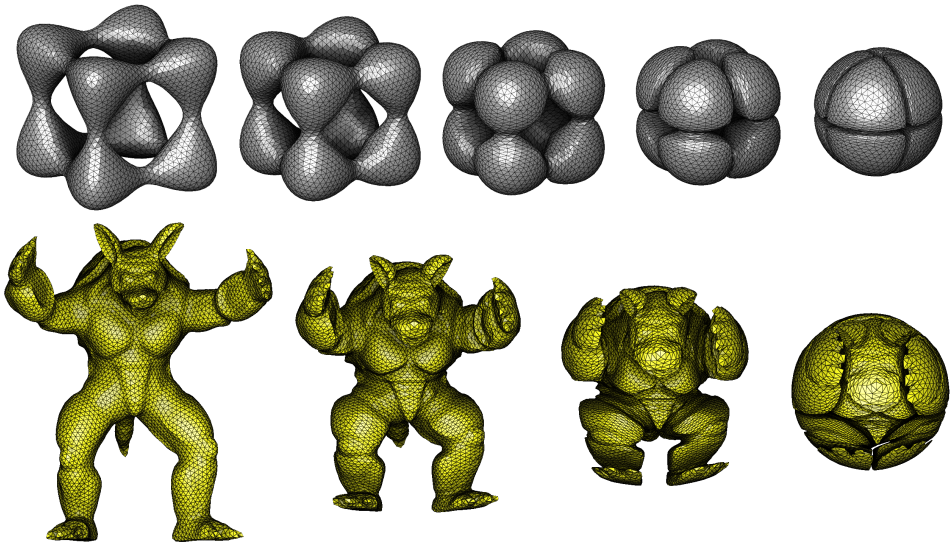


FIGURE 3.41 – *Interpolation de volumes 3D par transport optimal.*

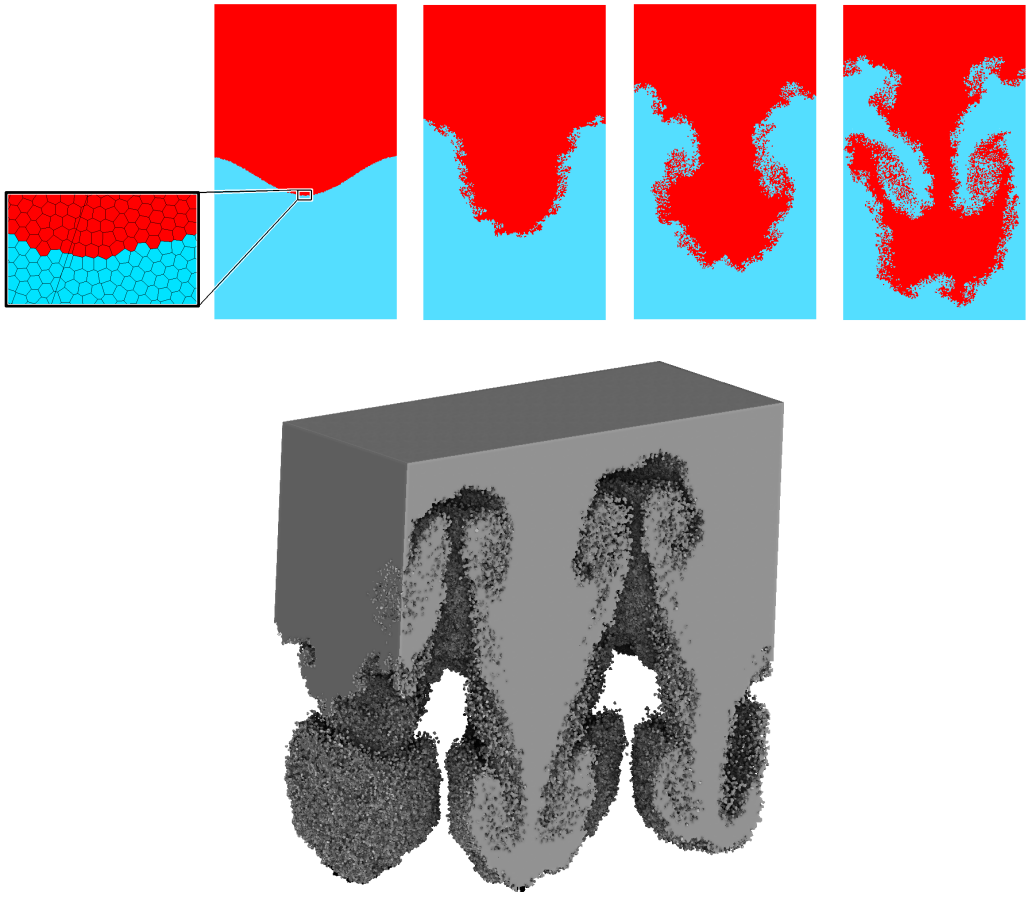


FIGURE 3.42 – *Une application du transport optimal à la mécanique des fluides : simulation numérique de l'instabilité de Taylor-Rayleigh, en 2D (en haut) et en 3D (en bas).*

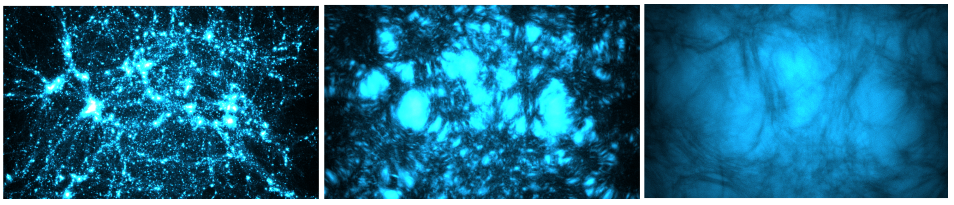


FIGURE 3.43 – *Une application du transport optimal en astrophysique. À gauche : données de départ, simulation d'un cube de 600 millions d'années lumières de côté, 16 millions d'amas de galaxies (Roya Mohayaee, Institut d'Astrophysique de Paris). À droite : reconstruction du mouvement vers les conditions initiales (densité uniforme).*

de Laguerre et sur leurs bords. La densité u est représentée par un maillage triangulaire, et interpolée linéairement sur les triangles. Les intégrales de u sur les cellules de Laguerre sont évaluées en forme close, en calculant l'intersection entre les cellules de Laguerre et le maillage support de u. (D) : le même algorithme peut calculer le transport entre une mesure supportée par une surface 3D et un ensemble de points en 3D.

La figure 3.41 montre deux exemples de déformations de volumes par transport optimal. Le même algorithme est utilisé, mais cette fois avec des diagrammes de Laguerre en 3d, et des calculs d'intersection avec un maillage à base de tétraèdres qui supporte la densité source. Les étapes intermédiaires sont générées par interpolation linéaire des positions (interpolation de Mc Cann).

La figure 3.42 montre une application à la simulation numérique en mécanique des fluides. Un fluide incompressible lourd (en rouge) est placé au dessus d'un fluide incompressible plus léger (en bleu). Les deux fluides tendent à échanger leurs positions, ce qui est rendu difficile par l'incompressibilité. Le résultat est la création de tourbillons de plus en plus rapides (instabilité de Taylor-Rayleigh). La méthode de simulation numérique utilisée ici [80] permet de calculer directement les trajectoires de particules (formulation Lagrangienne) tout en prenant en compte l'incompressibilité, ce qui est réputé difficile en formulation Lagrangienne. Ici l'incompressibilité s'exprime assez naturellement, par la préservation du volume des cellules de Laguerre. À droite de la figure : le même type de calcul réalisé en 3D, avec 10 millions de particules, générant des structures de tourbillons plus compliquées (vues en coupe ici). Le calcul 3D utilise des algorithmes géométriques efficaces [118] pour le calcul du diagramme de Laguerre et des intersections.

Les applications en physique semblent prometteuses, en particulier parce que l'algorithme semi-discret se comporte très bien. Il est maintenant envisageable de résoudre un problème de transport à chaque pas de temps. Avec une implantation optimisée utilisant à la fois un processeur multicœur pour la partie géométrique et un GPU pour la résolution des systèmes linéaires, l'algorithme semi-discret met quelques dizaines de secondes pour résoudre un problème à 10 millions d'inconnues. Ceci ouvre la voie à la résolution numérique de problèmes difficiles. Par exemple, en astrophysique, le problème de la reconstruction de l'état primordial de l'univers [36] consiste à tenter de « remonter dans le temps » à partir de données d'observation de la répartition des amas de galaxie, pour « rejouer le film du Big Bang » à l'envers. Le transport semi-discret pourrait être un algorithme numérique très efficace pour résoudre ce problème. La figure 3.43

montre des premiers résultats, appliqués ici à des données de simulation numérique de l'Institut d'Astrophysique de Paris (Roya Mohayaee).

3.5 Notes bibliographiques

Pour approfondir les différentes sections de ce chapitre, le lecteur pourra consulter les articles et ouvrages suivants :

— Résolution de systèmes linéaires (section 3.2) : les ouvrages de référence en optimisation numérique [141] et en algèbre linéaire [86] ;

— Paramétrisation de surfaces (section 3.3) : les chapitres 2 et 4 de l'ouvrage [31] ;

— Transport optimal (section 3.4) : les deux ouvrages incontournables [198, 199], et l'ouvrage plus récent développant un point de vue plus orienté vers les mathématiciens appliqués [169]. Nous mentionnons également l'élégant théorème de factorisation polaire de Brenier [35], qui établit le lien entre gradient d'une fonction convexe et applications préservant la mesure.

Remerciements. Les auteurs souhaitent remercier Jarek Rossignac, Kai Hormann, Quentin Mérigot, Yann Brenier, Jean-David Benamou, Nicolas Bonneel et Filippo Santambrogio pour de nombreuses discussions qui ont inspiré certains contenus de ce cours, et les relecteurs, Paul Zimmermann et Emmanuel Jeandel, pour leur relecture approfondie et leurs nombreuses corrections.

Chapitre 4

Analyse statique des réseaux booléens

Loïc Paulevé
Adrien Richard

Les réseaux booléens sont des systèmes dynamiques discrets regroupant un nombre fini de variables binaires qui évoluent, dans un temps discret et par interactions mutuelles, selon une loi prédéfinie. Les interactions entre variables sont souvent sommairement décrites par un graphe dirigé : le graphe d'interaction. Les réseaux booléens sont en particulier très utilisés en biologie pour modéliser la dynamique des réseaux de neurones, de régulation génique, et des voies de signalisation cellulaire. Dans ce chapitre, nous aborderons deux approches complémentaires pour capturer des caractéristiques importantes de la dynamique des réseaux booléens par analyse statique. Nous présenterons, d'une part, une étude des points fixes par une approche combinatoire se basant sur le graphe d'interaction et, d'autre part, une étude des trajectoires par interprétation abstraite.

4.1 Introduction

Soit un ensemble de n variables binaires, indexées de 1 à n. Un réseau booléen spécifie pour chaque variable une fonction de $\{0,1\}^n$ dans $\{0,1\}$ qui associe à chaque configuration du réseau (valuation possible de toutes les variables) une valeur pour cette variable.

Partant d'une configuration initiale, l'application itérative de ces fonctions induit une évolution de la configuration du réseau. Cette évolution dépend de l'ordre et de la synchronisation de ces applications. Les séquences

de configurations obtenues peuvent alors présenter des comportements périodiques et des configurations qui apparaissent de manière unique. L'ensemble des variables et leur domaine étant finis, toute application infinie des fonctions fait converger le réseau soit vers une configuration qui ne peut plus être modifiée (point fixe), soit vers une répétition d'un ensemble de configurations.

Les réseaux booléens appartiennent à la grande famille des systèmes complexes : le comportement global émergent du réseau est très difficile à prédire à partir de la spécification des fonctions locales d'évolution des composants. Ils peuvent être apparentés avec les automates cellulaires, les réseaux d'automates et les réseaux de Petri.

Les réseaux booléens sont abondamment étudiés pour la modélisation des réseaux biologiques. Historiquement cela est dû à deux raisons. D'une part, ils sont une abstraction pratique de certaines classes d'équations linéaires par morceaux, aussi très utilisées pour la modélisation des réseaux biologiques [83]. D'autre part, la vision « on/off » de l'état des variables du système s'avère adéquate pour modéliser de manière qualitative le comportement des réseaux de gènes ou de neurones. Ainsi ce type de modèle a été plébiscité par différents biologistes théoriciens dans les années 1960 [99, 105, 190]. Outre les applications en biologie, on peut également citer des applications pour la modélisation de phénomènes sociologiques [197, 85]. Côté informatique, certaines problématiques en théorie de l'information peuvent être formalisées avec les réseaux booléens, notamment le problème du codage réseau linéaire [78].

Ce chapitre porte sur l'étude de deux propriétés importantes de la dynamique des réseaux booléens : leurs points fixes (configurations ne pouvant plus évoluer) et leurs trajectoires (séquences de configurations possibles). Comme nous l'illustrerons en conclusion, ces analyses ont des applications en biologie des systèmes.

Plus particulièrement, ce chapitre se concentre sur *l'analyse statique* de ces propriétés. De manière générale, une analyse statique vise à déduire formellement certaines propriétés d'un modèle à partir de sa spécification sans exécuter ce dernier. C'est-à-dire dans notre cas, sans calculer explicitement les différentes évolutions de la configuration d'un réseau booléen.

En pratique, les analyses statiques s'appuient sur des abstractions de la spécification du modèle qui vont mettre en exergue des caractéristiques liées aux propriétés recherchées. Dans ce chapitre, nous présenterons deux abstractions des réseaux booléens : le graphe d'interaction, qui résume les dépendances, positives et négatives, entre les variables du système et les

impliquants premiers d'itérations, qui résument les changements possibles de configurations.

Le premier avantage des analyses statiques est qu'elles permettent de déduire des propriétés qui peuvent être soit trop coûteuses, soit impossibles, à calculer en exécutant simplement le modèle. Les analyses statiques permettent également de raisonner sur des familles (ou classes) de modèles. En effet, parce qu'elles s'appuient sur des abstractions, les conclusions des analyses sont identiques pour tout modèle concret partageant la même abstraction.

Plan du chapitre

Les définitions de base sont données dans la section 4.2, présentant formellement les notions de réseau booléen, d'itérations, et de graphe d'interaction.

La section 4.3 détaille le lien entre certaines propriétés du graphe d'interaction et le nombre maximum et minimum de points fixes des réseaux booléens correspondants. Ces résultats font apparaître l'importance des cycles positifs ou négatifs du graphe d'interaction, et leur façon d'être connectés les uns aux autres.

La section 4.4 montre une abstraction des transitions pour dériver des conditions nécessaires ou suffisantes pour des propriétés d'existence de trajectoires.

Les analyses statiques présentées permettent d'aborder l'étude formelle de très grands réseaux booléens, et d'établir certaines prédictions pour leur contrôle. La section 4.5 illustre ainsi quelques applications de l'analyse statique des réseaux booléens pour des problématiques actuelles en biologie des systèmes.

4.2 Définitions de base et notations

Un *réseau booléen* à n composantes est une fonction $f : \{0,1\}^n \to \{0,1\}^n$. On note $\mathcal{N} = \{1, \ldots, n\}$. On appelle *configuration* de f un vecteur $x \in \{0,1\}^n$. La i-ème composante de x est noté x_i. La i-ème composante de $f(x)$ est notée $f_i(x)$. Ainsi, on peut voir f_i comme une fonction de $\{0,1\}^n$ vers $\{0,1\}$.

Étant données deux configurations $x, y \in \{0,1\}^n$, on note $\Delta(x,y)$ l'ensemble des composantes $i \in \mathcal{N}$ telles que $x_i \neq y_i$. Ainsi, $|\Delta(x,y)|$ est la *distance de Hamming* entre x et y. On associe à l'ensemble des configurations l'ordre partiel usuel composante à composante : $x \leqslant y$ si et seulement si

$x_i \leqslant y_i$ pour tout $i \in \mathcal{N}$. L'ensemble des configurations muni de cet ordre est donc un *treillis booléen*. On note $x + y$ l'addition modulo 2 composante à composante. Les composantes qui prennent la valeur 1 dans la configuration $x + y$ sont donc exactement les composantes dans $\Delta(x, y)$. On note e_i le i-ème vecteur de base : toutes les composantes sont à 0 sauf la i-ème, qui vaut 1.

Une itération est un changement de configuration x par la mise à jour d'une ou plusieurs de ses composantes en suivant $f(x)$. Trois modes de mise à jour sont classiquement considérés : le synchrone (ou parallèle), où toutes les composantes sont mises à jour ; l'asynchrone, où une seule composante est mise à jour ; et le général, où une ou plusieurs composantes sont mises à jour.

Définition 4.2.1 (Itérations)**.** *Les itérations forment une relation binaire irréflexive* $\to \subseteq \{0,1\}^n \times \{0,1\}^n$*. Les itérations* synchrones *de f sont données par* $ITS(f)$*,* asynchrones *par* $ITA(f)$*, et* générales *par* $ITG(f)$ *:*

$$ITS(f) := \{x \to f(x) \mid x \in \{0,1\}^n, f(x) \neq x\}$$
$$ITA(f) := \{x \to y \mid x, y \in \{0,1\}^n, x \neq y, \Delta(x,y) = \{i\}, y_i = f_i(x)\}$$
$$ITG(f) := \{x \to y \mid x, y \in \{0,1\}^n, x \neq y, \forall i \in \Delta(x,y), y_i = f_i(x)\}$$

Par définition, $ITS(f) \subseteq ITG(f)$ et $ITA(f) \subseteq ITG(f)$. La comparaison des différents modes de mise à jour fait l'objet de nombreux travaux, par exemple [9, 142].

Définition 4.2.2 (Point fixe)**.** *Un point fixe de f est une configuration $x \in \{0,1\}^n$ telle que $f(x) = x$.*

Définition 4.2.3 (Attracteurs)**.** *Un attracteur f est un ensemble* minimal *non-vide de configurations $X \subseteq \{0,1\}^n$ clos par la relation d'itération $IT\square$ choisie ($\forall x \in X, x \to y \in IT\square(f) \Rightarrow y \in X$).*

On remarque qu'un attracteur est de la forme $X = \{x\}$ si et seulement si x est un point fixe de f.

Un *graphe signé* est un couple $G = (V, E)$ où $E \subseteq V \times V \times \{-1, 1\}$. Naturellement, V est l'ensemble des sommets, et E l'ensemble des arcs. Les arcs sont dirigés et portent un signe positif ou négatif. Notons qu'il est possible d'avoir à la fois un arc positif et un arc négatif d'un sommet sur un autre. G a un graphe non-signé sous-jacent $|G|$ obtenu en ignorant les signes de façon naturelle. Lorsque les notions en jeux n'impliquent pas les signes, on identifie G et $|G|$ sans confusion possible. On dira par exemple que G est acyclique pour signifier que $|G|$ l'est. Les cycles et chemins de G sont vus comme des sous-graphes, et sont donc sans répétition de sommet.

Le signe d'un chemin ou d'un cycle est le produit du signe de ses arcs. Ainsi, un chemin ou un cycle est *positif* si il contient un nombre pair d'arcs négatifs, et *négatif* si il contient un nombre impair d'arcs négatifs.

Définition 4.2.4 (Graphe d'interaction)**.** *Le graphe d'interaction d'un réseau booléen f à n composantes est le graphe signé $G(f)$ défini comme suit : l'ensemble des sommets est $\mathcal{N}$, et pour tout $i, j \in \mathcal{N}$, il existe un arc positif (resp. négatif) de j sur i si et seulement si il existe une configuration x telle que $x_j = 0$ et*

$$f_i(x + e_j) - f_i(x)$$

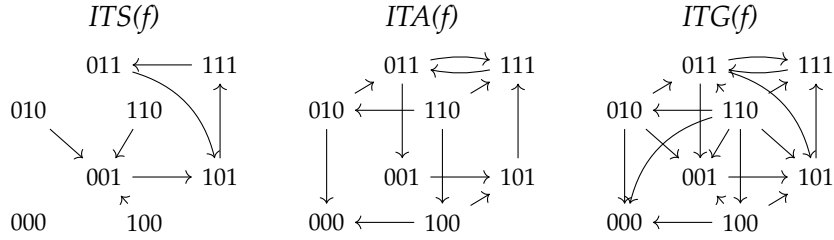
est positif (resp. négatif).

Ainsi, il y a un arc de j sur i si et seulement si f_i dépend de la j-ème variable. La version non signée du graphe d'interaction indique uniquement les dépendances, et les signes ajoutent une information supplémentaire sur la nature de ces dépendances.

Exemple 4.2.5. *Soit le réseau booléens à 3 composantes suivant :*

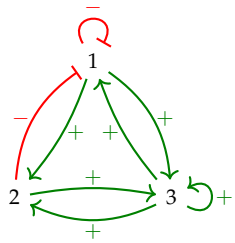
$$\begin{aligned} f_1(x) &:= x_3 \wedge (\neg x_1 \vee \neg x_2) \\ f_2(x) &:= x_3 \wedge x_1 \\ f_3(x) &:= x_1 \vee x_2 \vee x_3 \end{aligned}$$

Les itérations selon les différents modes de mise à jour considérés dans ce chapitre sont représentées ci-dessous sous forme de graphes dirigés :



Le réseau f possède 1 point fixe, 000; les itérations synchrones (ITS) possèdent en plus un attracteur composé de $\{011, 101, 111\}$; les itérations asynchrones et générales (ITA et ITG) possèdent en plus un attracteur composé de $\{001, 101, 111, 011\}$. Ces attracteurs correspondent aux composantes fortement connexes terminales des graphes respectifs.

Le graphe d'interaction $G(f)$ est le graphe dirigé suivant, où les arcs positifs sont dessinés en vert avec une étiquette $+$, et les arcs négatifs sont dessinés en rouge avec une étiquette $-$ et se terminant avec une barre.



4.3 Étude des points fixes par une approche combinatoire

Dans bien des contextes, lorsqu'un système complexe est étudié, il est bien plus facile de récolter des informations sur son graphe d'interaction que sur sa dynamique. Cette situation est très fréquente dans le contexte des réseaux de gènes : souvent, le graphe d'interaction est assez bien compris, alors que la dynamique, qui est très difficile à observer, ne l'est pas (cf. Section 4.5.1). On est donc amené à se poser la question suivante : *que peut-on dire sur la dynamique du système à partir de son graphe d'interaction seulement ?*

De nombreuses propriétés dynamiques peuvent être étudiées sous cet angle. Parmi elles, le nombre de points fixes a reçu une grande attention. Les points fixes sont robustes au mode d'itération considéré, et ont souvent des interprétations fortes en pratique. Dans le contexte des réseaux de neurones dotés d'une mémoire associative, le nombre de points fixes correspond à la taille de la mémoire. Dans le contexte des réseaux de gènes, les points fixes correspondent à des patterns stables d'expressions des gènes que l'on peut souvent assimiler à des fonctions ou types cellulaires bien déterminés (cf. Section 4.5.2).

Dans cette section, nous allons donc étudier l'influence du graphe d'interaction sur le nombre de points fixes, en présentant certains résultats de base. On se concentrera essentiellement sur le nombre maximum et minimum de points fixes pour un graphe d'interaction donné. Ce graphe d'interaction sera représenté par un graphe signé G à n sommets, notés de 1 à n, et on notera $F(G)$ l'ensemble des réseaux booléens *sur* G, c'est-à-dire, l'ensemble des réseaux booléens qui ont G pour graphe d'interaction. Nous allons donc étudier les quantités suivantes :

$$\max(G) := \max_{f \in F(G)} |\mathrm{Fixe}(f)|$$
$$\min(G) := \min_{f \in F(G)} |\mathrm{Fixe}(f)|$$

où Fixe(f) désigne l'ensemble des points fixes de f.

Avant de commencer, nous avons besoin d'un résultat qui sera utilisé tout au long de cette section. Si G a un arc positif (resp. négatif) de j sur i on dit que j est un *activateur* (resp. *inhibiteur*) de i. On note $\leqslant_i^G$ l'ordre partiel sur $\{0,1\}^n$ défini par $x \leqslant_i^G y$ si et seulement si $x_j \leqslant y_j$ pour tous les activateurs j de i et $x_j \geqslant y_j$ pour tous les inhibiteurs j de i. Autrement dit $x \leqslant_i^G y$ si et seulement si on passe de x à y en ajoutant des activateurs de i et en retirant des inhibiteurs de i. D'après la proposition suivante, si $f \in F(G)$, alors chaque composante f_i est monotone par rapport à $\leqslant_i^G$.

Proposition 4.3.1. *Si $f \in F(G)$ alors, pour tout $i \in \mathcal{N}$ et $x, y \in \{0,1\}^n$,*

$$x \leqslant_i^G y \Rightarrow f_i(x) \leqslant f_i(y).$$

Démonstration. Par induction sur la distance de Hamming entre x et y, notée $d(x,y)$. Le cas de base $d(x,y) = 0$ est trivial, on suppose donc que $d(x,y) > 0$. Soit $j \in \Delta(x,y)$. Si $x_j < y_j$ alors, comme $x \leqslant_i^G y$, G n'a pas d'arc négatif de j sur i et donc $f_i(x + e_j) \geqslant f_i(x)$. Comme $x + e_j \leqslant_i^G y$ et $d(x + e_j, y) = d(x,y) - 1$, par induction on a $f_i(x + e_j) \leqslant f_i(y)$, et donc $f_i(x) \leqslant f_i(y)$. De façon similaire, si $x_j > y_j$ alors G n'a pas d'arc positif de j sur i et donc $f_i(y + e_j) \leqslant f_i(y)$. Comme $x \leqslant_i^G y + e_j$ et $d(x, y + e_j) = d(x,y) - 1$, par induction on a $f_i(x) \leqslant f_i(y + e_j)$, et donc $f_i(x) \leqslant f_i(y)$. ■

4.3.1 Absence de cycle positif ou négatif

Nous pouvons maintenant entrer dans le vif du sujet avec l'observation de base suivante : si G est acyclique, alors tous les réseaux booléens sur G convergent en au plus n itérations (synchrones).

Théorème 4.3.2 (Robert [164, 165])**.** *Si G est acyclique, alors f^n (la composition de f avec elle-même n fois) est une fonction constante pour tout $f \in F(G)$.*

Démonstration. Soit $i_1 i_2 \dots i_n$ un tri topologique de G : il n'y a pas d'arc de i_l à i_k dès que $l \geqslant k$. Ainsi, chaque f_{i_k} ne dépend que de composantes i_l avec $l < k$. Soient x et y deux configurations quelconques. On montre, par induction sur k allant de 1 à n, que $f_{i_k}^m(x) = f_{i_k}^m(y)$ pour tout $m \geqslant k$, ce qui implique clairement le théorème. Comme f_{i_1} ne dépend d'aucune composante, f_{i_1} est une fonction constante et a évidemment $f_{i_1}^m(x) = f_{i_1}^m(y)$ pour tout $m \geqslant 1$. Soit $1 < k \leqslant m$. Par induction, $f_{i_l}^{m-1}(x) = f_{i_l}^{m-1}(y)$ pour tout $l < k$. Comme f_{i_k} ne dépend que de composantes i_l avec $l < k$, on en déduit que $f_{i_k}(f^{m-1}(x)) = f_{i_k}(f^{m-1}(y))$. Autrement dit, $f_{i_k}^m(x) = f_{i_k}^m(y)$, ce qui complète l'induction. ■

Exercice 4.3.3. *D'après ce théorème, si G est acyclique et $f \in F(G)$, alors $ITS(f)$ est acyclique. Montrer que $ITA(f)$ et $ITG(f)$ le sont également.*

Si f^k est une constante pour un certain k, alors f a évidemment un unique point fixe. On en déduit le corollaire suivant.

Corollaire 4.3.4. *Si G est acyclique, alors* $\min(G) = \max(G) = 1$.

La réciproque est fausse : si G est obtenu à partir d'un cycle positif et d'un cycle négatif en identifiant un sommet, alors on montre facilement que l'on a tout de même $\min(G) = \max(G) = 1$.

Le théorème suivant montre qu'il est possible d'obtenir le corollaire précédent par une approche plus fine, en faisant intervenir le signe des cycles. Il peut être vu comme une adaptation au cas booléen des deux conjectures du biologiste René Thomas. On peut attribuer ce théorème à Aracena [7] (voir aussi [8]). De nombreuses généralisations ont été proposées, en particulier dans [161, 162].

Théorème 4.3.5 (Version booléenne des conjectures de Thomas)**.**

1. *Si G n'a pas de cycle positif alors* $\max(G) \leqslant 1$.
2. *Si G n'a pas de cycle négatif alors* $\min(G) \geqslant 1$.

Remarque 4.3.6. *On peut montrer, de façon plus générale, les deux propriétés suivantes : si G n'a pas de cycle positif alors, pour tout $f \in F(G)$, $ITA(f)$ a au plus un attracteur ; si G n'a pas de cycle négatif alors, pour tout $f \in F(G)$, $ITA(f)$ n'a pas d'attracteur cyclique.*

Le premier point du Théorème 4.3.5 est une conséquence immédiate du lemme suivant, qui sera réutilisé plusieurs fois dans la suite. Si U est un ensemble de sommets dans G, alors $G[U]$ désigne le sous-graphe de G induit par U (c'est-à-dire obtenu en retirant les sommets hors de U avec les arcs attachés).

Lemme 4.3.7. *Si $f \in F(G)$ a deux points fixes distincts x et y, alors $G[\Delta(x, y)]$ a un cycle positif.*

Démonstration. Prenons un quelconque $i \in \Delta(x, y)$. Si $x_i < y_i$ on a $f_i(x) = x_i < f_i(y) = y_i$ ce qui veut dire, d'après la proposition 4.3.1, que $y \leqslant_i^G x$ est faux : G a un arc positif de j sur i avec $x_j < y_j$ ou un arc négatif de j sur i avec $x_j > y_j$. G a donc un arc de j sur i de signe $y_j - x_j$. On montre de façon similaire que si $x_i > y_i$ alors G a un arc de j sur i de signe $x_j - y_j$. Ainsi, dans tous les cas, il existe un sommet j dans $\Delta(x, y)$ tel que G a un arc de j sur i de signe

$$s_{ji} := (y_j - x_j)(y_i - x_i).$$

On en déduit que $G[\Delta(x,y)]$ a un cycle $i_0 i_1 \dots i_{p-1} i_0$ où le signe de l'arc de i_k sur i_{k+1} est $s_{i_k i_{k+1}}$, où $k+1$ est calculé modulo p. Le signe du cycle est donc, en prenant encore une fois l'addition modulo p,

$$\prod_{k=0}^{p-1} s_{i_k i_{k+1}} = \prod_{k=0}^{p-1} (y_{i_k} - x_{i_k}) \prod_{k=0}^{p-1} (y_{i_{k+1}} - x_{i_{k+1}}) = \left(\prod_{k=0}^{p-1} (y_{i_k} - x_{i_k}) \right)^2 = 1.$$

■

Pour le second point du Théorème 4.3.5 nous allons utiliser la version dirigée suivante d'un théorème très classique sur les graphes signés, le théorème d'Harary [92].

Théorème 4.3.8. *Si G est fortement connexe et sans cycle négatif, alors il existe une partition de ses sommets en deux parties A et B (l'une des deux pouvant être vide) telle qu'un arc de G est positif si et seulement si son sommet initial et son sommet terminal sont dans la même partie.*

Un réseau f est *monotone* si pour toutes configurations x et y

$$x \leqslant y \Rightarrow f(x) \leqslant f(y).$$

Il est facile de voir qu'un réseau f sur G est monotone si et seulement si G n'a que des arcs positifs. De plus, d'après le lemme suivant, si f est un réseau sur un graphe G fortement connexe et sans cycle négatif, alors chaque f est monotone à une translation près (qui ne dépend que de G).

Lemme 4.3.9. *Si G est fortement connexe et sans cycle négatif alors il existe une configuration z telle que, pour tout $f \in F(G)$, le réseau $x \mapsto h(x) := f(x+z) + z$ est monotone.*

Démonstration. Soient A et B une partition des sommets de G comme dans le Théorème 4.3.8. Soit z la configuration définie par $z_i = 1$ si $i \in A$ et $z_i = 0$ sinon. Soit i une composante et x une configuration avec $x_i = 0$. Pour montrer que h est monotone, il suffit de montrer que $h(x) \leqslant h(x + e_i)$. Soit j une quelconque composante. Si i et j sont dans la même partie, on a $z_i = z_j$, et comme G n'a pas d'arc négatif de i sur j on a $f_j(x+z) + z_j \leqslant f_j(x+z+e_i) + z_j$, et donc $h_j(x) \leqslant h_j(x+e_i)$. Si i et j sont dans des parties différentes, on a $z_i \neq z_j$, et comme G n'a pas d'arc positif de i sur j on a $f_j(x+z) + z_j \leqslant f_j(x+z+e_i) + z_j$, et donc $h_j(x) \leqslant h_j(x+e_i)$. ■

Lemme 4.3.10. *Si G est fortement connexe et sans cycle négatif, alors*

$$\min(G) \geqslant 2.$$

Démonstration. Soit $f \in F(G)$ et soit une configuration z telle que le réseau $h(x) := f(x+z)+z$ est monotone ; cette configuration existe d'après le Lemme 4.3.9. Soit H le graphe d'interaction de h. Il est facile de voir que H s'obtient à partir de G en rendant positifs tous les arcs. Donc H est fortement connexe. Soit $\mathbf{0}$ le vecteur à n composantes, toutes nulles. Pour toute configuration x on a $\mathbf{0} \leqslant_i^H x$ (puisque $\leqslant_i^H$ coïncide avec l'ordre partiel usuel $\leqslant$) et donc $h_i(\mathbf{0}) \leqslant h_i(x)$. On en déduit que si $h_i(\mathbf{0}) = 1$ alors h_i est une fonction constante, qui a 1 pour seule image. Mais alors i est une source de H, ce qui est faux. Donc $h_i(\mathbf{0}) = 0$ pour toute configuration i, de sorte que $h(\mathbf{0}) = \mathbf{0}$. On montre de façon similaire que $h(\mathbf{1}) = \mathbf{1}$, où $\mathbf{1}$ est le vecteur à n composantes, toutes égales à 1. On déduit alors que $\mathbf{0} + z = z$ et $\mathbf{1} + z$ (*i.e.* la négation de z) sont des points fixes de f. ■

Nous sommes maintenant en mesure de démontrer le second point du Théorème 4.3.5.

Démonstration du second point du Théorème 4.3.5. Par hypothèse, G est sans cycle négatif. Soit $f \in F(G)$. On montre que f admet au moins un point fixe par induction sur le nombre n de composantes. Le cas $n = 1$ étant trivial, on suppose $n > 1$. De plus, si G est fortement connexe, d'après le Lemme 4.3.10, f a au moins deux points fixes, donc on suppose que G n'est pas fortement connexe. Il existe alors une partition (I, J) des sommets en deux parties non-vides telle que G n'a pas d'arc de J à I. On suppose, sans perte de généralité, que $I = \{1, \dots, m\}$ pour un certain $1 \leqslant m < n$. Soit f^I le réseau booléen à m composantes défini par

$$f^I(x) = (f_1^I(x), \dots, f_m^I(x)) := (f_1(x, \mathbf{0}), \dots, f_m(x, \mathbf{0}))$$

où $\mathbf{0}$ est le vecteur contenant $n - m$ zéros. Il est facile de voir que le graphe d'interaction de f^I est inclus dans G. Il est donc sans cycle négatif et donc, par induction, f^I a un point fixe $\alpha \in \{0,1\}^m$. Soit maintenant le réseau booléen f^J à $n - m$ composantes défini par

$$f^J(x) = (f_1^J(x), \dots, f_{n-m}^I(x)) := (f_{m+1}(\alpha, x), \dots, f_n(\alpha, x))$$

De même, le graphe d'interaction de f^J est inclus dans G et donc, par induction, f^J a un point fixe $\beta \in \{0,1\}^{n-m}$. La configuration (α, β) est alors un point fixe de f. En effet, pour tout $i \in I$, f_i ne dépend que de composantes dans I (puisque G n'a pas d'arc de J sur I) et donc

$$f_i(\alpha, \beta) = f_i(\alpha, \mathbf{0}) = f_i^I(\alpha) = \alpha_i = x_i$$

De plus, pour tout $i \in J$ on a

$$f_i(\alpha, \beta) = f_{i-m}^I(\beta) = \beta_{i-m} = x_i.$$

■

4.3.2 Borne du feedback positif

Nous avons vu que le nombre de points fixes est très petit − au plus 1 − lorsque G est acyclique ou, de façon plus générale, lorsque G est sans cycle positif (c'est le premier point du Théorème 4.3.5). Il est naturel de se demander si il existe des généralisations de la forme "si G n'est pas trop différent d'un graphe acyclique, ou d'un graphe sans cycle positif, alors le nombre de points fixes n'est pas trop grand". Ceci pose la question d'une "distance" à l'acyclicité. Plusieurs notions sont envisageables : le nombre de cycles, le nombre de sommets appartenant à un cycle, etc. Dans notre contexte, le nombre transversal, défini ci-dessous, est une mesure de la cyclicité particulièrement pertinente.

Un *Feedback Vertex Set* (FVS) est un ensemble de sommets qui intersecte tous les cycles : la suppression de ces sommets donne un graphe acyclique. De façon similaire, un *FVS positif* est un ensemble de sommets qui intersecte tous les cycles positifs : la suppression de ces sommets donne un graphe sans cycle positif. On note $\tau(G)$ la taille minimale d'un FVS et $\tau^+(G)$ la taille minimale d'un FVS positif. On appelle $\tau(G)$ et $\tau^+(G)$ le *nombre transversal* et le *nombre transversal positif* de G. On a, par exemple, $\tau(G) = 0$ dans le cas acyclique et $\tau^+(G) = 0$ lorsque tous les cycles sont négatifs. On a toujours $\tau^+(G) \leqslant \tau(G)$, avec évidemment égalité lorsque tous les cycles sont positifs.

On peut maintenant énoncer et démontrer la *borne du feedback positif*, qui est une simple implication du Lemme 4.3.7.

Théorème 4.3.11 (Aracena [7])**.**

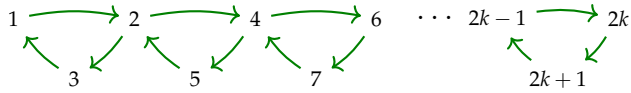
$$\max(G) \leqslant 2^{\tau^+(G)}.$$

Démonstration. Soient I un FVS de G de taille $\tau^+(G)$ et $f \in F(G)$. Soient x et y deux points fixes distincts de f. D'après le Lemme 4.3.7, $G[\Delta(x,y)]$ a un cycle positif, et donc $\Delta(x,y)$ intersecte I. Cela signifie que la restriction x_I de x sur I est différente de la restriction de y_I de y sur I. Nous avons donc montré que l'opération de restriction $x \mapsto x_I$ est une injection de $\mathrm{Fixe}(f)$ dans l'ensemble $\{0,1\}^I$, qui est de taille $2^{\tau^+(G)}$. On a donc évidemment $|\mathrm{Fixe}(f)| \leqslant 2^{\tau^+(G)}$. ■

Remarque 4.3.12. *On peut montrer, plus généralement, que pour tout $f \in F(G)$, $ITA(f)$ a au plus $2^{\tau(G)}$ attracteurs.*

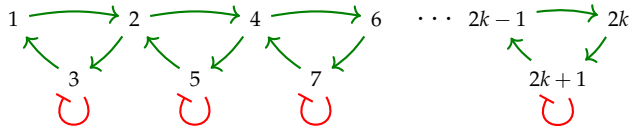
Cette borne est très perfectible. Pour s'en rendre compte, considérons le graphe G_k décrit dans la figure ci-dessous : il correspond à une chaîne

de k cycles de longueur 3 et contient $n = 2k + 1$ sommets. De plus, tous les arcs sont positifs.



Il est facile de voir que $\tau(G_k) = \tau^+(G_k) = \lfloor \frac{k}{2} \rfloor$. Un exercice un peu plus délicat consiste à montrer que $\max(G)$ est égal au nombre d'ensembles indépendants maximaux que G possède (un ensemble indépendant de G est un ensemble U de sommets tel que $G[U]$ n'a pas d'arc), de sorte que $\max(G) = P(k)$ où $P(k)$ est le nombre de Padovan défini récursivement par $P(0) = P(1) = P(2) = 1$ et $P(k) = P(k-2) + P(k-3)$. Il est connu que $P(k)$ est l'entier le plus proche de p^{k-1}/s où $p = 1.3247\ldots$ est le nombre plastique, et $s = 1.0453\ldots$ l'unique racine réelle de $s^3 - 2s^2 + 23s - 23 = 0$ (le nombre plastique joue pour la suite de Padovan le même rôle que le nombre d'or pour la suite de Fibonacci). On a donc, pour k suffisamment grand, $\max(G_k) \geqslant 2^{k/2.5}$. Ceci montre que la borne du feedback positif, $\max(G_k) \leqslant 2^{\lfloor k/2 \rfloor}$, est assez précise dans ce cas.

Considérons maintenant le graphe d'interaction G'_k obtenu à partir de G_k en ajoutant une boucle négative sur chaque maillon de la chaîne, comme illustré ci-dessous.



On a encore $\tau(G'_k) = \tau^+(G'_k) = \lfloor \frac{k}{2} \rfloor$, la borne du feedback positif est donc la même. Elle est cependant très mauvaise, car il est possible de montrer que $\max(G'_k) = 1$ pour tout $k \geqslant 1$. On voit donc ici que l'ajout de cycles négatifs réduit drastiquement le nombre de points fixes. La borne du feedback positif, qui ignore les cycles négatifs, devient alors mauvaise.

On en vient à se demander si l'on peut améliorer la borne en prenant en compte les cycles négatifs. C'est un problème subtil, car les cycles négatifs ont une influence très variable sur le nombre de points fixes. Dans certaines situations, ils ont tendance à être néfastes pour la présence de nombreux points fixes ; l'exemple ci-dessus l'illustre. Mais dans d'autres situations, les cycles négatifs sont bénéfiques pour la présence de nombreux points fixes : l'exemple des cliques positives et négatives, étudiées ci-dessous, illustre bien ce phénomène.

4.3.3 Clique positive et négative

On note K_n^+ la clique positive à n sommets : il y a un arc positif de i à j si et seulement si $i \neq j$, et il n'y a pas d'arc négatif. La clique négative K_n^- est définie de façon similaire : il y a un arc négatif de i à j si et seulement si $i \neq j$, et il n'y a pas d'arc positif. Le nombre transversal et le nombre transversal positif valent $n-1$ dans les deux cas. Pour la clique positive comme négative, la borne du feedback positif est donc 2^{n-1}. On peut cependant faire beaucoup mieux :

Théorème 4.3.13 (Gadouleau-Richard-Riis [79])**.**

$$\frac{\binom{n}{\lfloor \frac{n}{2} \rfloor}}{n} \leqslant \max(K_n^+) \leqslant \frac{2^{n+1}}{n+2} \leqslant \max(K_n^-) = \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Ce théorème est intéressant de deux points de vue.

Premièrement, on déduit de ce résultat que pour tout entier k fixé, si n est suffisamment grand, alors $\max(K_{n+k}^+) < \max(K_n^-)$. Ainsi, K_n^- permet la présence de plus de points fixes que K_{n+k}^+ alors qu'il possède moins de cycles positifs et a un nombre transversal positif plus petit ($n-1$ contre $n+k-1$). On voit donc ici que les cycles négatifs de K_n^- (qui correspondent aux cycles de longueur impaire), absents dans K_{n+k}^+, sont favorables à la présence de nombreux points fixes, comme annoncé à la fin de la section précédente.

Deuxièmement, la preuve de ce théorème fait appel aux trois théorèmes suivants ; les deux premiers ont été établis dans le contexte des codes correcteurs d'erreurs en théorie de l'information, et le troisième est le bien connu lemme de Sperner en théorie des ensembles. Ceci montre que l'étude du nombre de points fixes nécessite diverses techniques, et qu'elle est connectée à d'autres domaines à la frontière entre les mathématiques et l'informatique. On appelle *poids* d'une configuration x, et on note $|x|$, le nombre de composantes i telles que $x_i = 1$. Pour $1 \leqslant w \leqslant n$ on note $B(n,w)$ l'ensemble des configurations de $\{0,1\}^n$ de poids w.

Théorème 4.3.14 (Borne de Graham et Sloane [88])**.** *Soit $1 \leqslant w \leqslant n$. Il existe un sous-ensemble $A \subseteq B(n,w)$ tel que deux configurations distinctes de A diffèrent en au moins 4 composantes, et*

$$|A| \geqslant \frac{\binom{n}{w}}{n}.$$

Théorème 4.3.15 (Borne de Varshamov [196])**.** *Soit $1 \leqslant d \leqslant n$, et soit un sous-ensemble $A \subseteq \{0,1\}^n$ tel que, pour toutes configurations x et y distinctes de*

A, il existe au moins d composantes i telles que $x_i > y_i$ ou d composantes i telles que $x_i < y_i$. Alors

$$|A| \leqslant \frac{2^{n+1}}{\sum_{k=0}^{d-1} \binom{\lfloor \frac{n}{2} \rfloor}{k} + \binom{\lceil \frac{n}{2} \rceil}{k}}.$$

Théorème 4.3.16 (Lemme de Sperner [179]). *Si A est une antichaîne de $\{0,1\}^n$ (deux configurations distinctes de A sont toujours incomparables), alors*

$$|A| \leqslant \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Démonstration de la première inégalité du Théorème 4.3.13. Soit $w := \lfloor n/2 \rfloor$ et $R := \binom{n}{w}/n$. Si $R \leqslant 2$ alors la conclusion est évidente (et elle découle du Lemme 4.3.10). On suppose donc que $R \geqslant 3$ ce qui implique $n \geqslant 6$. Soit $A \subseteq B(n,w)$ tel que deux configurations distinctes de A diffèrent en au moins 4 composantes et $|A| \geqslant R$; un tel ensemble existe d'après la Borne de Graham et Sloane. Si x est une configuration et i une composante, alors $x^{i1} := x$ si $x_i = 1$ et $x^{i1} := x + e_i$ si $x_i = 0$. Ainsi, x^{i1} est obtenu en plaçant un 1 sur la $i$ème composante de x. On définit x^{i0} de façon similaire.

Considérons le réseau booléen f à n composantes défini par

$$f_i(x) = \begin{cases} 1 \text{ si } x^{i1} \in A \text{ ou si } |x^{i1}| > w \text{ et } x^{i0} \notin A, \\ 0 \text{ sinon} \end{cases}$$

pour toute composante i. Il est facile de voir que toutes les configurations de A sont des points fixes et que f_i ne dépend pas de la composante i. Ainsi, le graphe d'interaction G de f n'a pas de boucle.

Montrons que f est monotone. Supposons, par contradiction, qu'il existe deux configurations x et y et une composante i telles que $x \leqslant y$ et $f_i(x) > f_i(y)$. On a donc $|x| < |y|$. Comme f_i ne dépend pas de la composante i on peut supposer, sans perte de généralité, que $x_i = y_i = 1$. Si $|x^{i1}| > w$ alors $|y^{i1}| > w+1$ et donc $|y^{i0}| > w$ d'où $y^{i0} \notin A$. Mais alors $f_i(y) = 1$, ce qui est faux. On en déduit que $|x^{i1}| \leqslant w$ et comme $f_i(x) = 1$ cela implique $x^{i1} \in A$. Donc $|y^{i1}| > w$. Si $y^{i0} \in A$ on a $|y^{i1}| = w+1 = |x^{i1}| + 1$, donc il existe une composante $j \neq i$ telle que $y^{i1} = x^{i1} + e_j$. Mais alors $y^{i0} = x + e_j + e_i$ et $x^{i1} = x$ sont deux éléments de A qui ne diffèrent qu'en deux composantes seulement, ce qui contredit le choix de A. On a donc $y^{i0} \notin A$ d'où $f_i(y) = 1$, ce qui est faux. f est donc monotone.

Ainsi, tous les arcs de G sont positifs, et pour montrer que $G = K_n^+$ il suffit de montrer que pour toutes composantes distinctes i et j, G a un arc de j sur i. Soit X l'ensemble des configurations x de poids w telles que $x_i < x_j$, $x \notin A$ et $x + e_i + e_j \notin A$. Montrons que $X \neq \emptyset$. En effet, il existe

une configuration x de poids w telle que $x_i < x_j$ et $x \notin A$. Supposons que $x + e_i + e_j \in A$. Soient trois composantes k, l, m, distinctes entre elles et distinctes de i et j, telles que $x_k < x_l = x_m$; ces composantes existent car $n \geqslant 6$ et $w = \lfloor n/2 \rfloor$. On a $x + e_i + e_j + e_k + e_l \notin A$ et $x + e_i + e_j + e_k + e_m \notin A$. Comme $x + e_k + e_l$ et $x + e_k + e_m$ ne diffèrent qu'en deux composantes seulement, l'une de ces deux configurations au moins n'est pas dans A, et donc l'une de ces deux configurations au moins est dans X. Ainsi, $X \neq \emptyset$. Soit donc $x \in X$. On a $|x^{i1}| = |x| + 1 = w + 1$ et $x^{i0} = x \notin A$ donc $f_i(x) = 1$. De plus $(x + e_j)^{i1} = x + e_i + e_j$ n'est pas dans A et est de poids w, donc $f_i(x + e_j) = 0$. G a donc bien un arc de j sur i. ■

Démonstration de la seconde inégalité du Théorème 4.3.13. On montre, plus généralement, que si G est un graphe d'interaction qui n'a que des arcs positifs et aucune boucle, alors $\max(G) \leqslant 2^{n+1}/n + 2$. Soit $f \in F(G)$ et soient x et y deux points fixes distincts de f. Si $f_i(x) = x_i < y_i = f_i(y)$ alors, d'après la Proposition 4.3.1, $y \leqslant_i^G x$ est faux, et donc G a un arc de j sur i avec $x_j < y_j$, et $j \neq i$ puisque G n'a pas de boucle. On montre de façon similaire, si $f_i(x) = x_i > y_i = f_i(y)$ alors il existe $j \neq i$ tel que $x_j > y_j$. Donc pour tous points fixes distincts x et y il existe au moins 2 composantes i telles que $x_i < y_i$ ou au moins 2 composantes i telles que $x_i > y_i$. On déduit alors de la Borne de Varshamov, appliquée au cas $d = 2$, que

$$|\mathrm{Fixe}(f)| \leqslant \frac{2^{n+1}}{\sum_{k=0}^{1} \binom{\lfloor \frac{n}{2} \rfloor}{k} + \binom{\lceil \frac{n}{2} \rceil}{k}} = \frac{2^{n+1}}{n+2}.$$

■

Démonstration de l'égalité dans le Théorème 4.3.13. Soit G un graphe d'interaction qui n'a que des arcs négatifs. Soit $f \in F(G)$ et soient $x, y \in \mathrm{Fixe}(f)$. Si $x \leqslant y$ alors, comme tous les arcs sont négatifs, on a $f(x) \geqslant f(y)$ et comme x et y sont fixes on obtient $x \geqslant y$, de sorte que $x = y$. Ainsi, deux points fixes sont comparables si et seulement si ils sont égaux, ce qui signifie que $\mathrm{Fixe}(f)$ est une antichaîne. D'après le Lemme de Sperner, on a donc $\max(G) \leqslant \binom{n}{\lfloor n/2 \rfloor}$, et en particulier, $\max(K_n^-) \leqslant \binom{n}{\lfloor n/2 \rfloor}$.

On montre maintenant l'inégalité inverse. Soit f le réseau booléen à n composantes défini par $f_i(x) = 1$ si et seulement si il existe au moins $n/2$ composantes $j \neq i$ telles que $x_j = 0$ ($i = 1, \ldots, n$). Il est facile de voir que $f \in F(K_n^-)$ et que toutes les configurations de poids $\lfloor n/2 \rfloor$ sont des points fixes. Donc $\max(K_n^-) \geqslant |\mathrm{Fixe}(f)| \geqslant \binom{n}{\lfloor n/2 \rfloor}$. ■

Exercice 4.3.17. *L'étude de* $\min(K_n^+)$ *et* $\min(K_n^-)$ *est bien plus simple, et l'exercice consiste à montrer que l'on a :* $\min(K_n^+) = 2$ *pour tout* $n \geqslant 2$*;* $\min(K_n^-) = 1$ *pour* $2 \leqslant n \leqslant 3$*; et* $\min(K_n^-) = 0$ *pour tout* $n \geqslant 4$.

4.3.4 Influence des cycles disjoints dans le cas monotone

Nous avons vu que la borne du feedback positif est perfectible car elle ne prend pas en compte les cycles négatifs, dont l'influence sur le nombre maximum de points fixes est difficile à cerner. Mais la borne du feedback positif est-elle au moins bonne lorsque tous les cycles sont positifs ? Nous allons montrer que ce n'est pas le cas. Nous avons pour cela besoin d'un paramètre de graphe supplémentaire.

Étant donné un graphe d'interaction G, on note $\nu(G)$ la taille maximale d'un ensemble de cycles deux à deux disjoints (c'est-à-dire ne partageant aucun sommet). On note également $\nu^+(G)$ la taille maximale d'un ensemble de cycles positifs deux à deux disjoints. On a, évidemment, $\nu(G) = \nu^+(G)$ si tous les cycles sont positifs, et $\nu(G) = 0$ si et seulement si G est acyclique. De plus, $\nu^+(G) \leqslant \nu(G) \leqslant \tau(G)$ et $\nu^+(G) \leqslant \tau^+(G) \leqslant \tau(G)$. On dit que deux cycles C et C' sont *indépendants* si ils sont disjoints et si il n'y a aucun arc de C à C' et aucun arc de C' à C.

Lemme 4.3.18. *Si $f \in F(G)$ et* Fixe(f) *a une chaîne de $k+1$ points fixes, alors*

$$\nu(G) \geqslant k.$$

Démonstration. Soit $x^1 \leqslant x^2 \leqslant \cdots \leqslant x^{k+1}$ une telle chaîne. D'après le Lemme 4.3.7, $G[\Delta(x^l, x^{l+1})]$ a un cycle pour tout $1 \leqslant l \leqslant k$, que l'on note C^l. Comme les k ensembles $\Delta(x^1, x^2), \Delta(x^2, x^3), \ldots, \Delta(x^k, x^{k+1})$ sont disjoints deux à deux, les cycles $C^1, \ldots, C^k$ sont disjoints, et donc $\nu(G) \geqslant k$. ■

Lemme 4.3.19. *Si G est sans arc négatif, de degré entrant maximal au plus* 2*, et n'a pas deux cycles indépendants, alors*

$$\max(G) \leqslant \nu(G) + 1.$$

Démonstration. Soit $f \in F(G)$. On note N_i les voisins entrants de i dans G. Comme le degré entrant de i est au plus 2, et que tous les arcs sont positifs, on a soit $f_i(x) = \min(\{x_j : j \in N_i\} \cup \{1\})$ soit $f_i(x) = \max(\{x_j : j \in N_i\} \cup \{0\})$. Dans le premier cas, on dit que i est de type ET et dans le second cas on dit que i est de type OU.

Supposons, afin d'obtenir une contradiction, que x et y sont deux points fixes incomparables de f. Soit I l'ensemble des composantes i telles que $x_i < y_i$, et J l'ensemble des composantes i telles que $x_i > y_i$. Comme x et y sont incomparables I et J ne sont pas vides. Si $i \in I$ alors $f_i(x) = x_i < y_i = f_i(y)$ donc $y \leqslant_i^G x$ est faux : G a un arc de j sur i avec $x_j < y_j$. Donc le degré entrant minimal $G[I]$ est au moins 1. Donc $G[I]$ a un cycle C^I. On montre de même que $G[J]$ a un cycle C^J. Soit $i \in I$. Si i est de type ET,

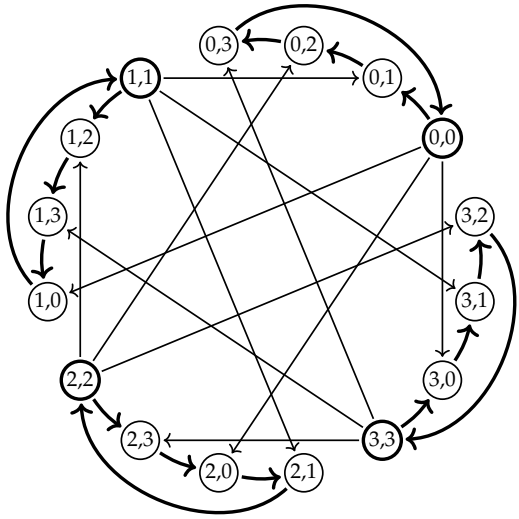


FIGURE 4.1 – K_4^*

comme $f_i(y) = 1$ on a $y_j = 1$ pour tout $j \in N_i$ et donc G n'a pas d'arc de J sur i. Si i est de type OU, comme $f_i(x) = 0$ on a $x_j = 0$ pour tout $j \in N_i$ et donc G n'a pas d'arc de J sur i. On en déduit que G n'a pas d'arc de J sur I, et on montre de même que G n'a pas d'arc de I sur J. Les cycles C^I et C^J sont donc indépendants, une contradiction.

Donc les points fixes de f sont deux à deux comparables. Autrement dit, $\mathrm{Fixe}(f)$ est une chaîne, et d'après le Lemme 4.3.18 on a $|\mathrm{Fixe}(f)| \leqslant \nu + 1$. ■

Théorème 4.3.20 (Aracena-Richard-Salinas [10])**.** *Pour tout $k \geqslant 1$, il existe un graphe d'interaction G fortement connexe ne possédant que des arcs positifs tel que*

$$\max(G) - 1 \leqslant \nu(G) = \tau(G) = k.$$

Démonstration. Soit $I = \{0, \ldots, k-1\}$. Soit K_k^* le graphe d'interaction défini comme suit : l'ensemble des sommets est l'ensemble $I \times I$; pour tout $i, j \in I$ il existe un arc positif de (i, j) à $(i, j+1)$, avec une addition modulo k ; pour tout $i, j \in I$, si $i \neq j$ alors il existe un arc positif de (i, i) sur (j, i). Le graphe K_4^* est illustré dans le Figure 4.1. On montre sans difficulté que $\nu(K_k^*) = \tau(K_k^*) = k$ et que K_k^* n'a pas deux cycles indépendants. Ainsi, d'après le Lemme 4.3.19, on a $\max(K_k^*) \leqslant k + 1$. ■

On voit donc que la borne du feedback positif est très perfectible même lorsque tous les arcs sont positifs. Est-il donc possible de l'améliorer dans ce cas ? C'est en effet possible, en utilisant le Lemme 4.3.18, le célèbre théorème de Knaster-Tarski, et la généralisation suivante bien connue du Lemme de Sperner établie par Erdős (on retrouve le Lemme de Sperner en prenant $k = 2$).

Théorème 4.3.21 (Knaster-Tarski [109, 188])**.** *Si f est un réseau booléen monotone alors $|\mathrm{Fixe}(f)|$ est un treillis non-vide.*

Théorème 4.3.22 (Erdős [70])**.** *Si $A \subseteq \{0,1\}^n$ est sans chaîne de taille $k+1$ alors $|A|$ est au plus la somme des k plus grands coefficients binomiaux d'ordre n :*

$$|A| \leqslant \sum_{l=\lfloor \frac{n-k+1}{2} \rfloor}^{\lfloor \frac{n+k-1}{2} \rfloor} \binom{n}{l}.$$

Théorème 4.3.23 (Aracena-Richard-Salinas [10])**.** *Si G n'a que des arcs positifs, alors $\max(G)$ est au plus 2 plus la somme des $\nu(G) - 1$ plus grands coefficients binomiaux d'ordre $\tau(G)$:*

$$\max(G) \leqslant 2 + \sum_{l=\lfloor \frac{n-\nu(G)+2}{2} \rfloor}^{\lfloor \frac{n+\nu(G)-2}{2} \rfloor} \binom{\tau(G)}{l}.$$

Démonstration. Soit $f \in F(G)$ et soit I un FVS de taille $\tau(G)$. Nous allons montrer que

$$\forall x, y \in \mathrm{Fixe}(f), \qquad x \leqslant y \iff x_I \leqslant y_I. \tag{4.1}$$

Il suffit évidemment de montrer que $x_I \leqslant y_I$ implique $x \leqslant y$, puisque l'autre direction est triviale. Soit $i_1, \ldots, i_k$ un tri topologique de $G - I$. On pose $I^0 := I$ et $I^l := I \cup \{i_1, \ldots, i_l\}$ pour tout $1 \leqslant l \leqslant k$. On montre, par induction sur l allant de 0 à k que $x_{I^l} \leqslant y_{I^l}$. Le cas de base $l = 0$ est donné par hypothèse, donc on suppose $l \geqslant 1$. Comme f_{i_l} ne dépend que de composantes j dans I^{l-1}, comme $x_{I^{l-1}} \leqslant y_{I^{l-1}}$ par induction, et comme f est monotone, on a $f_{i_l}(x) \leqslant f_{i_l}(y)$. Comme x et y sont fixes on obtient $x_{i_l} \leqslant y_{i_l}$ et donc $x_{I^l} \leqslant y_{I^l}$ ce qui complète l'induction.

Soit

$$A := \{x_I : x \in \mathrm{Fixe}(f)\}.$$

D'après (4.1), A est un ensemble partiellement ordonné isomorphe à $\mathrm{Fixe}(f)$. Ainsi, d'après le théorème de Tarski, A est un treillis non-vide, et d'après le Lemme 4.3.18, A n'a pas de chaîne de taille $\nu(G) + 1$. Comme

A est un treillis, il possède un unique élément minimal a^- et un unique élément maximal a^+. Il est clair que toutes les chaînes de taille maximale de A contiennent à la fois a^- et a^+. Donc $A' = A \setminus \{a^-, a^+\}$ n'a pas de chaîne de taille $\nu(G) - 1$. Comme $A' \subseteq \{0,1\}^I$, d'après le Théorème 4.3.22, $|A'| = |A| - 2$ est au plus la somme des $\nu(G) - 1$ plus grands coefficients binomiaux d'ordre $|I| = \tau(G)$. ■

Notons $\beta(G)$ la borne du théorème précédent. Il est facile de voir que $\beta(G) \leqslant 2^{\tau(G)}$ et que l'égalité est atteinte si et seulement si $\nu(G) = \tau(G)$, et donc si $\max(G) = 2^\tau$ alors $\nu(G) = \tau(G)$. En fait, l'écart entre $\beta(G)$ et 2^τ est d'autant plus grand que l'écart entre $\nu(G)$ et $\tau(G)$ est important. Il est cependant difficile de démontrer l'existence de graphes qui réalisent un écart important entre $\nu(G)$ et $\tau(G)$. Le meilleur résultat dans cette direction est dû à Seymour [172] : pour tout n, il existe un graphe G à n sommets tel que $\tau(G)/\nu(G) \geqslant \frac{1}{30}\log\nu(G)$. (Notons que pour le graphe complet dirigé K_n, qui possède $n^2 - n$ arcs et aucune boucle, on a seulement $\tau(K_n)/\nu(K_n) \geqslant 2$.) Par ailleurs, on sait que pour $\nu(G)$ fixé, $\tau(G)$ ne peut pas être arbitrairement grand (la preuve est très complexe) :

Théorème 4.3.24 (Reed-Robertson-Seymour-Thomas [159]). *Il existe une fonction $h : \mathbb{N} \to \mathbb{N}$ telle que, pour tout graphe G,*

$$\tau(G) \leqslant h(\nu(G)).$$

On en déduit que, pour tout graphe d'interaction G, il est possible de borner $\max(G)$ en fonction des cycles disjoints seulement :

$$\max(G) \leqslant 2^{\tau^+(G)} \leqslant 2^{\tau(G)} \leqslant 2^{h(\nu(G))}.$$

Peut-on borner $\max(G)$ en fonction des cycles *positifs* uniquement ? Il semblerait malheureusement qu'il n'existe pas de fonction $h^+ : \mathbb{N} \to \mathbb{N}$ telle que $\tau^+(G) \leqslant h^+(\nu^+(G))$ … mais cela n'empêche pas nécessairement l'existence d'une fonction h^+ telle que $\max(G) \leqslant 2^{h^+(\nu^+(G))}$ pour tout graphe d'interaction G !

4.4 Étude des trajectoires par interprétation abstraite

Une trajectoire π est une séquence finie d'itérations de f suivant le mode de mise à jour choisi. On note $|\pi|$ la longueur de cette séquence ; la i-ème itération est donnée par π_i. Nécessairement, $\forall i \in \{1, \ldots, |\pi| - 1\}$, si $\pi_i = x \to y$, alors il existe $z \in \{0,1\}^n$ tel que $\pi_{i+1} = y \to z$.

L'étude des trajectoires se heurte à un problème combinatoire dû au nombre exponentiel (en n, le nombre de composantes du réseau) d'itérations possibles. Cette section présente une approche par interprétation abstraite pour raisonner sur ces trajectoires.

Cette section présente tout d'abord une abstraction des itérations par leurs impliquants premiers (4.4.1) pour calculer une sur-approximation des itérations accessibles depuis un ensemble de configurations $X \subseteq \{0,1\}^n$ (4.4.2). Nous présentons ces abstractions en suivant un cadre classique de l'interprétation abstraite, reposant sur des correspondances de Galois entre les domaines concrets (itérations) et abstraits (impliquants premiers) et une sémantique de point fixe.

Lorsque l'étude des trajectoires est focalisée sur l'accessibilité d'une itération ou d'une configuration particulière, nous pouvons définir la notion de *trajectoire causalement minimale* (4.4.3). Nous montrons que l'ensemble des itérations formant les trajectoires causalement minimales peut être capturé efficacement par cette abstraction.

Enfin, en s'appuyant sur une représentation sous forme de graphe des impliquants premiers d'itération, nous pouvons également caractériser un sous-ensemble des trajectoires possibles (4.4.4).

Nous concluons cette section en discutant de quelques généralisations possibles de cette approche (4.4.5).

Notations supplémentaires Dans cette section, nous utilisons des formules propositionnelles, notées entre crochets, pour exprimer des contraintes sur les configurations du réseau. Ces formules sont uniquement des conjonctions de littéraux portant sur la valeur des composantes d'une configuration, dénotée par la variable v. Ainsi, étant donnés $i \in \mathcal{N}$ et $b \in \{0,1\}$, la formule $[\mathrm{v}_i = b]$ est vraie si et seulement si la valeur de la composante i de la configuration est b. Une conjonction C est un ensemble de littéraux de la forme $[\mathrm{v}_i = b]$ où $i \in \mathcal{N}$ et $b \in \{0,1\}$. Étant donné $x \in \{0,1\}^n$, nous notons $C(x)$ son évaluation pour la configuration x :

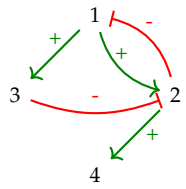
$$C(x) \Leftrightarrow \forall [\mathrm{v}_i = b] \in C, x_i = b \ .$$

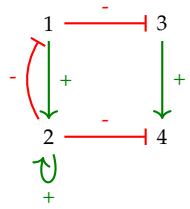
Étant donnée une fonction λ d'un ensemble vers un ensemble, on note $\mathrm{lfp}\,\lambda$ son plus petit point fixe au sens de l'inclusion $\subseteq$; dans cette section il sera toujours unique. Étant donné un ensemble X, on note $\mathrm{lfp}_X \lambda$ son plus petit point fixe incluant X ; dans cette section il sera toujours unique.

Exemples suivis Dans cette section, nous illustrerons les résultats sur deux réseaux différents à 4 composantes.

$$f_1(x) := \neg x_2 \qquad g_1(x) := \neg x_2$$
$$f_2(x) := x_1 \wedge \neg x_3 \qquad g_2(x) := x_1 \vee x_2$$
$$f_3(x) := x_1 \qquad g_3(x) := \neg x_1$$
$$f_4(x) := x_2 \qquad g_4(x) := \neg x_2 \wedge x_3$$

Leurs graphes d'interaction sont les suivants :





4.4.1 Abstraction des itérations

L'ensemble des itérations est de taille exponentielle en n, quel que soit le mode de mise à jour choisi. Cependant, on peut remarquer que l'application d'une même fonction peut générer un grand nombre d'itérations. Prenons par exemple un réseau à $n > 2$ composantes pour lequel $f_1(x) = x_2$. À partir de toute configuration x où $x_1 = 0$ et $x_2 = 1$, il existe une itération asynchrone vers la même configuration sauf pour la composante 1 qui vaut 1. Il existe ainsi 2^{n-2} itérations différentes représentant la seule application de f_1 pour augmenter la valeur de la composante 1.

Dans cette section, nous proposons d'étudier une abstraction des itérations qui considère les conditions minimales sur une configuration pour les changements de valeurs induits par l'itération.

Définition 4.4.1 (Impliquant premier d'itération)**.** *Étant donnés une itération $x \to y$ et $i \in \mathcal{N}$ où $x_i \neq y_i$, la conjonction $C \subseteq \{[\mathrm{v}_j = x_j] \mid j \in \mathcal{N}\}$ est un* impliquant premier de l'itération $x \to y$ par i *si et seulement si*

$$\forall z \in \{0,1\}^n, z_i = x_i \wedge C(z) \Rightarrow f_i(z) = y_i$$

et C est minimal. On note $\langle \mathrm{v}_i : x_i \rightsquigarrow y_i, C\rangle$ un tel impliquant premier d'itération.

Ainsi, $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle$ se lit : pour changer la valeur a à b de la composante i, il est suffisant qu'une configuration $x \in \{0,1\}^n$ où $x_i = a$ vérifie $C(x)$. Remarquons que la cardinalité de C est au plus le degré entrant de i dans le graphe d'interaction $G(f)$.

Nous pouvons alors définir une fonction d'abstraction α qui associe à un ensemble d'itérations leurs impliquants premiers :

Définition 4.4.2 (Abstraction des itérations)**.** *Étant donné un ensemble d'itérations* $T \subseteq \{0,1\}^n \times \{0,1\}^n$,

$$\alpha(T) := \{\langle \mathrm{v}_i : x_i \rightsquigarrow y_i, C\rangle \mid i \in \mathcal{N}, x \to y \in T, x_i \neq y_i, \\ C \textit{ est un impliquant premier de } x \to y \textit{ par } i\} \ .$$

La fonction de concrétisation qui associe un ensemble d'impliquants premiers d'itération à un ensemble d'itérations suivant le mode de mise à jour général peut se définir comme suit :

Définition 4.4.3 (Concrétisation des impliquants premiers d'itération)**.** *Étant donné un ensemble* P *d'impliquants premiers d'itération,*

$$\gamma(P) := \{x \to y \mid x, y \in \{0,1\}^n, x \neq y, \alpha(\{x \to y\}) \subseteq P\} \ .$$

Cette concrétisation assure que les itérations générées ne révèlent pas d'impliquants premiers d'itération autres que P (condition $\alpha(\{x \to y\}) \subseteq P$). Nous pouvons illustrer cet effet sur le petit exemple suivant : prenons un réseau avec 3 composantes, où $f_1(x) := x_2 \vee x_3$, et $P = \{\langle \mathrm{v}_1 : 0 \rightsquigarrow 1, [\mathrm{v}_2 = 1]\rangle\}$. Une définition alternative pourrait générer toute itération $x \to y$ telle que $x_1 = 0 \wedge x_2 = 1$, ce qui donnerait 2 itérations concrètes : $010 \to 110$ et $011 \to 110$. Or, l'abstraction de cette dernière itération révèle l'impliquant premier d'itération $\langle \mathrm{v}_1 : 0 \rightsquigarrow 1, [\mathrm{v}_3 = 1]\rangle$. Avec la définition proposée, on obtient bien $\gamma(P) = \{010 \to 110\}$.

Les fonctions *monotones* α et γ forment une *correspondance de Galois* entre les ensembles d'itérations et les ensembles d'impliquants premiers d'itération, en considérant la relation d'ordre $\subseteq$ d'inclusion entre ensembles.

Définition 4.4.4 (Correspondance de Galois)**.** *Soient* $(A, \sqsubseteq_A)$ *et* $(B, \sqsubseteq_B)$ *deux ensembles partiellement ordonnés. Les fonctions* $\alpha : A \to B$ *et* $\gamma : B \to A$ *forment une correspondance de Galois si et seulement si* α *et* γ *sont monotones,* $\forall a \in A, a \sqsubseteq_A \gamma(\alpha(a))$*, et* $\forall b \in B, \alpha(\gamma(b)) \sqsubseteq_B b$*.*

Propriété 4.4.5 (Correspondance de Galois)**.** *Pour tout ensemble d'itérations* $T \subseteq IT\square(f)$*,* $T \subseteq \gamma(\alpha(T))$*, et pour tout ensemble* P *d'impliquants premiers d'itération,* $\alpha(\gamma(P)) \subseteq P$ *.*

Démonstration. ($T \subseteq \gamma(\alpha(T))$) Soit une itération $x \to y \in T$. Pour chaque $i \in \mathcal{N}$ tel que $x_i \neq y_i$ il existe au moins un impliquant premier d'itération

$\langle \mathrm{v}_i : x_i \rightsquigarrow y_i, C\rangle$ où $C(x)$ et donc appartenant à $\alpha(T)$. Ainsi, $x \to y \in \gamma(\alpha(T))$.

($\alpha(\gamma(P)) \subseteq P$) On remarque que pour tout $T = \{x \to y\} \cup T'$, $\alpha(T) = \alpha(\{x \to y\}) \cup \alpha(T')$. La définition de γ s'assure que pour tout $x \to y \in \gamma(P)$, on a $\alpha(\{x \to y\}) \subseteq P$. ■

Une propriété équivalente est que $\alpha(T) \subseteq P \Leftrightarrow T \subseteq \gamma(P)$.

Une correspondance de Galois est une manière de formaliser le lien entre un domaine concret (les itérations) et abstrait (leurs impliquants premier). La composition $\gamma \circ \alpha$ est dite *extensive*, et garantit que l'abstraction est correcte car sa concrétisation retrouve bien les éléments initiaux. La composition $\alpha \circ \gamma$ est dite *réductive* et indique que la concrétisation ne perd pas en précision vis-à-vis de l'abstraction.

On peut remarquer une propriété un peu plus forte : $\alpha(\gamma(P)) = P$. En effet, pour tout impliquant premier d'itération $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle$, il existe au moins un x tel que $x_i = a \wedge C(x)$, ainsi $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \alpha(\gamma(P))$. Les fonctions α et γ forment donc une *insertion de Galois*.

Étant donné un réseau booléen f, l'ensemble complet de ses impliquants premiers d'itération peut être défini directement, sans passer par les ensembles explicites d'itération. En effet, $\langle \mathrm{v}_i : 1 - b \rightsquigarrow b, C\rangle$ est un impliquant premier d'itération si et seulement si $[\mathrm{v}_i = 1 - b \wedge C]$ est un impliquant premier (IP) de $[f_i(\mathrm{v}) = b]$.

Définition 4.4.6. *L'abstraction d'un réseau booléen f en impliquants premiers d'itération est définie par*

$$\begin{aligned}\alpha(f) := &\{\langle \mathrm{v}_i : 0 \rightsquigarrow 1, C\rangle \mid i \in \mathcal{N}, [\mathrm{v}_i = 0 \wedge C] \text{ IP de } [f_i(\mathrm{v}) = 1]\}\\ &\cup\{\langle \mathrm{v}_i : 1 \rightsquigarrow 0, C\rangle \mid i \in \mathcal{N}, [\mathrm{v}_i = 1 \wedge C] \text{ IP de } [f_i(\mathrm{v}) = 0]\}\ .\end{aligned}$$

La propriété suivante souligne que $\alpha(f)$ est complet, et que le mode de mise à jour n'influe pas sur le résultat de l'abstraction :

Propriété 4.4.7. $\alpha(f) = \alpha(ITG(f)) = \alpha(ITA(f)) = \alpha(ITS(f))$.

L'intérêt principal de ce domaine abstrait est sa taille comparée aux ensembles d'itérations : $|ITS(f)| \in O(2^n)$; $|ITA(f)| \in O(n \cdot 2^n)$; $|ITG(f)| \in O(2^n \cdot (2^n - 1))$. Le nombre d'impliquants premiers d'itération, et donc d'impliquants premiers, est exponentiel suivant le nombre de termes. Si une composante du réseau a d régulateurs (degré entrant dans le graphe d'interaction), alors elle a au plus 2^d impliquants premiers d'itération. Ainsi $|\alpha(f)| \in O(n \cdot 2^d)$ dans le cas général. Si les f_i sont monotones, alors la borne se raffine à $\binom{d}{\lfloor d/2 \rfloor}$. Cette abstraction est donc intéressante si d est bien plus petit que n.

Exemple 4.4.8. *Prenons le réseau exemple f à 4 composantes défini page 177, où notamment $f_2(x) := x_1 \wedge \neg x_3$ et $f_3(x) = x_1$. Nous obtenons $|ITS(f)| = 15$, $|ITA(f)| = 32$, $|ITG(f)| = 59$. Soit l'itération $1000 \to 1110 \in ITG(f)$:*

$$\alpha(\{1000 \to 1110\}) = \{\langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1], [\mathrm{v}_3 = 0]\}\rangle, \langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle\} \ .$$

De manière similaire,

$$\alpha(\{0100 \to 0000\}) = \{\langle \mathrm{v}_2 : 1 \rightsquigarrow 0, \{[\mathrm{v}_1 = 0]\}\rangle\} \ .$$

Appliqué directement à f, $|\alpha(f)| = 9$, avec en particulier les impliquants premiers d'itération pour la désactivation de 2 : $\langle \mathrm{v}_2 : 1 \rightsquigarrow 0, \{[\mathrm{v}_1 = 0]\}\rangle$ et $\langle \mathrm{v}_2 : 1 \rightsquigarrow 0, \{[\mathrm{v}_3 = 1]\}\rangle$.

Appliqué au réseau exemple g, où notamment $g_2(x) := x_1 \vee x_2$, le seul impliquant premier d'itération pour 2 est pour son activation : $\langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle$, sa désactivation étant impossible.

Abstraction des configurations Dans les analyses qui suivent, nous allons également avoir besoin de définir la relation d'abstraction entre un ensemble de configurations et un ensemble de littéraux.

L'abstraction consiste simplement à lister tous les littéraux correspondant aux valeurs des composantes des configurations données ; et la concrétisation consiste à générer toutes les configurations formées uniquement par ces littéraux.

Définition 4.4.9. *Étant donné un ensemble de configurations $X \subseteq \{0,1\}^n$,*

$$\mathring{\alpha}(X) := \{[\mathrm{v}_i = x_i] \mid i \in \mathcal{N}, x \in X\} \ .$$

Définition 4.4.10. *Étant donné un ensemble L de littéraux,*

$$\mathring{\gamma}(L) := \{x \in \{0,1\}^n \mid \forall i \in \mathcal{N}, [\mathrm{v}_i = x_i] \in L\} \ .$$

Les opérateurs ainsi définis forment une correspondance de Galois entre les ensembles de configurations et les ensembles de littéraux, toujours en considérant la relation d'ordre $\subseteq$ d'inclusion entre ensembles.

Propriété 4.4.11 (Correspondance de Galois)**.** *Pour tout ensemble de configurations $X \subseteq \{0,1\}^n$, $X \subseteq \mathring{\gamma}(\mathring{\alpha}(X))$, et pour tout ensemble L de littéraux, $\mathring{\alpha}(\mathring{\gamma}(L)) \subseteq L$.*

Exemple 4.4.12.

$$\mathring{\alpha}(\{0100, 1100\}) = \{[\mathrm{v}_1 = 0], [\mathrm{v}_1 = 1], [\mathrm{v}_2 = 1], [\mathrm{v}_3 = 0], [\mathrm{v}_4 = 0]\}$$
$$\mathring{\gamma}(\mathring{\alpha}(\{0100, 1100\})) = \{0100, 1000, 1100\}$$

4.4.2 Sur-approximation des trajectoires

Partant d'un ensemble de configurations $X \subseteq \{0,1\}^n$, nous nous intéressons à l'ensemble des trajectoires possibles partant de n'importe quelle configuration de X. Cet ensemble est infini s'il existe des répétitions possibles.

Une représentation plus compacte de cet ensemble de trajectoires est tout simplement l'ensemble des itérations qui les composent. Cet ensemble est fini (c'est un sous-ensemble du mode choisi, ici nous choisissons le mode général) et permet de reconstruire toutes les trajectoires possibles.

Nous définissons $A_X(T)$ la fonction qui ajoute à l'ensemble d'itérations T les itérations de $ITG(f)$ qui peuvent poursuivre une trajectoire composée uniquement des itérations T. Pour cela, nous collectons l'ensemble des configurations atteintes avec une itération de T ($\phi(T)$), et nous ajoutons toutes les itérations partant soit de X, soit de cet ensemble.

Définition 4.4.13. *Étant donné un ensemble d'itérations $T \subseteq ITG(f)$,*

$$A_X(T) := T \cup \{z \to y \in ITG(f) \mid z \in X \cup \phi(T)\}$$

où

$$\phi(T) := \{y \mid x \to y \in T\} \ .$$

En appliquant cette fonction récursivement à partir de l'ensemble vide, c'est-à-dire $(A_X \circ A_X \circ \cdots \circ A_X)(\emptyset)$, on obtient l'ensemble des itérations formant exactement l'ensemble des trajectoires possibles depuis X. Comme il n'existe qu'un nombre fini d'itérations et que A_X est monotone, la récursion de A_X converge toujours vers un point fixe. De plus, par le théorème du point fixe de Kleene [183], on remarque que ce point fixe est le plus petit point fixe de A_X, noté $\mathrm{lfp}\, A_X$.

Propriété 4.4.14. *Étant donné $X \subseteq \{0,1\}^n$, $x \to y \in \mathrm{lfp}\, A_X$ si et seulement si il existe une trajectoire π d'itérations parmi $ITG(f)$ depuis $x \in X$ où il existe $p \in \{1, \ldots, |\pi|\}$ avec $\pi_p = x \to y$.*

Comme discuté dans la sous-section précédente, calculer A_X est coûteux car il existe un nombre exponentiel (en n) d'itérations. Nous cherchons donc à calculer le pendant abstrait de $\mathrm{lfp}\, A_X$, c'est-à-dire $\alpha(\mathrm{lfp}\, A_X)$, où le domaine est plus petit.

L'idée est la suivante : nous allons définir une version abstraite de A_X qui, au lieu de prendre en argument un ensemble d'itérations, prend en argument un ensemble d'impliquants premiers d'itération, et va calculer une extension de cet ensemble de manière similaire.

Étant donné un ensemble de littéraux L, la fonction $A^\#_L(P)$ collecte les impliquants premiers d'itération $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle$ où pour chaque littéral $[\mathrm{v}_j = d] \in C$, soit $[\mathrm{v}_j = d] \in L$, soit il existe un impliquant premier d'itération dans P de la forme $\langle \mathrm{v}_j : e \rightsquigarrow d, C'\rangle$.

Définition 4.4.15. *Étant donné un ensemble d'impliquants premiers d'itération P,*

$$A^\#_L(P) := P \cup \{\langle \mathrm{v}_j : a \rightsquigarrow b, C\rangle \in \alpha(f) \mid C \subseteq L \cup \phi^\#(P)\}$$

où

$$\phi^\#(P) := \{[\mathrm{v}_i = b] \mid \exists \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in P\}$$

Comme pour A_X, on remarque que le plus petit point fixe de $A^\#_L$ correspond au point fixe obtenu par son itération successive appliquée initialement à l'ensemble vide.

Pour démontrer que cette fonction $A^\#_L$ permet de calculer une abstraction du plus petit point fixe de A_X, nous montrons que, avec $L = \mathring{\alpha}(X)$, le résultat de $A^\#_L(P)$ inclut le résultat de $\alpha(A_X(\gamma(P)))$.

Lemme 4.4.16. $\alpha(A_X(\gamma(P))) \subseteq A^\#_{\mathring{\alpha}(X)}(P)$

Démonstration. Soit $z \to y \in A_X(\gamma(P))$. Si $z \in X$, on vérifie $\forall \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \alpha(\{z \to y\})$, $C \subseteq \mathring{\alpha}(X)$ et donc $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in A^\#_{\mathring{\alpha}(X_0)}(P)$. Si $z \in \phi(T)$, nécessairement $\mathring{\alpha}(\{z\}) \subseteq \mathring{\alpha}(\{x\}) \cup \phi^\#(P)$. Ainsi, $\forall \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \alpha(\{z \to y\})$, $C \subseteq \mathring{\alpha}(X) \cup \phi^\#(P)$, et donc $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in A^\#_{\mathring{\alpha}(X_0)}(P)$. ■

Cette propriété nous assure ainsi que les itérations successives de $A^\#_L$ calculent un sur-ensemble de l'abstraction des itérations successives de A_X, comme illustré par la figure 4.2. On obtient alors le théorème suivant sur l'approximation du point fixe de A_X :

Théorème 4.4.17. *Étant donné un ensemble de configurations $X \subseteq \{0,1\}^n$,*

$$\alpha(\mathrm{lfp}\, A_X) \subseteq \mathrm{lfp}\, A^\#_{\mathring{\alpha}(X)}$$

Cette abstraction permet de donner une condition nécessaire pour des propriétés d'accessibilité depuis X, par exemple :

- il existe une trajectoire depuis X contenant l'itération $x \to y$ seulement si $\alpha(\{x \to y\}) \subseteq \mathrm{lfp}\, A^\#_{\mathring{\alpha}(X)}$;
- il existe une trajectoire menant à une configuration satisfaisant une conjonction de littéraux C seulement si $C \subseteq \mathring{\alpha}(X) \cup \phi^\#(\mathrm{lfp}\, A^\#_{\mathring{\alpha}(X)})$.

Ces propriétés se vérifient en un temps linéaire avec la taille de $\alpha(f)$.

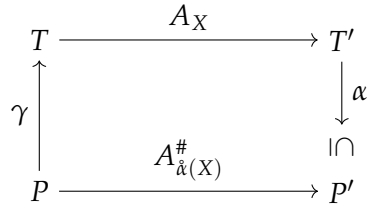


FIGURE 4.2 – *Schéma de l'abstraction de A_X par $A^{\#}_{\mathring{\alpha}(X)}$.*

Exemple 4.4.18. *Prenons le réseau exemple g à 4 composantes défini page 177. Avec $X_1 = \{0101\}$, nous obtenons*

$$\text{lfp}\, A^{\#}_{\mathring{\alpha}(X_1)} = \{\langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 0]\}\rangle, \langle \mathrm{v}_4 : 1 \rightsquigarrow 0, \{[\mathrm{v}_2 = 1]\}\rangle\}$$

Avec $X_2 = \{1000\}$, nous obtenons

$$\begin{aligned}\text{lfp}\, A^{\#}_{\mathring{\alpha}(X_2)} = \alpha(g) = \{&\langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle,\\ &\langle \mathrm{v}_1 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 0]\}\rangle, \langle \mathrm{v}_1 : 1 \rightsquigarrow 0, \{[\mathrm{v}_2 = 1]\}\rangle,\\ &\langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 0]\}\rangle, \langle \mathrm{v}_3 : 1 \rightsquigarrow 0, \{[\mathrm{v}_1 = 1]\}\rangle,\\ &\langle \mathrm{v}_4 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 0], [\mathrm{v}_3 = 1]\}\rangle,\\ &\langle \mathrm{v}_4 : 1 \rightsquigarrow 0, \{[\mathrm{v}_2 = 1]\}\rangle, \langle \mathrm{v}_4 : 1 \rightsquigarrow 0, \{[\mathrm{v}_3 = 0]\}\rangle\}\end{aligned}$$

Ce dernier exemple montre l'effet de sur-approximation apporté par notre abstraction : il n'existe aucune trajectoire permettant d'activer 4 *depuis* 1000*, cependant l'abstraction du point fixe contient bien l'abstraction d'une itération activant* 4*.*

4.4.3 Raffinement pour les trajectoires causalement minimales

L'étude des trajectoires dans les réseaux booléens est souvent motivée par l'étude de l'atteinte d'une valeur b pour une composante i en particulier : par exemple l'activation d'un gène ou d'un facteur de transcription connu comme important. Dans ce cas de figure, nous pouvons nous intéresser à l'ensemble des trajectoires menant à toute configuration satisfaisant le littéral $[\mathrm{v}_i = b]$, et par exemple observer des points communs entre toutes ces trajectoires (points de passage obligés, etc.).

Plus précisément, nous nous intéressons aux trajectoires *causalement minimales* pour l'atteinte de $[\mathrm{v}_i = b]$, c'est-à-dire aux trajectoires *acycliques* où chaque mise à jour de composante dépend soit d'une itération précédente, soit de la configuration initiale.

Définition 4.4.19 (Trajectoire causalement minimale)**.** *Une trajectoire π est causalement minimale pour $[\mathrm{v}_i = b]$ depuis x_0 si et seulement si il n'existe pas*

de trajectoire $\varpi \neq \pi$, $|\varpi| \leqslant |\pi|$, *depuis* x_0 *vers* z *telle que* $z_i = b$ *et telle qu'il existe une injection* $\sigma : \{1, \ldots, |\varpi|\} \to \{1, \ldots, |\pi|\}$ *vérifiant* $\forall p, q \in \{1, \ldots, |\varpi|\}, p < q \Leftrightarrow \sigma(p) < \sigma(q)$ *et, avec* $\varpi^p = x \to y$ *et* $\pi^{\sigma(p)} = x' \to y'$, $\Delta(x, y) \subseteq \Delta(x', y')$.

Ces trajectoires font ainsi appel à un ensemble minimal de mises à jour pour atteindre $[\mathrm{v}_i = b]$: toute autre trajectoire utilise un sur-ensemble de ces mises à jour. On remarque que l'ensemble des trajectoires causalement minimales est fini.

Exemple 4.4.20. *Prenons le réseau exemple f à 4 composantes défini page 177, pour lequel nous nous intéressons aux trajectoires causalement minimales pour l'atteinte de* $[\mathrm{v}_4 = 1]$ *depuis la configuration* 1000.

La trajectoire acyclique $\pi = (1000 \to 1100; 1100 \to 1110; 1110 \to 1111)$ *n'est pas minimale car il existe une trajectoire* $\varpi = (1000 \to 1100; 1100 \to 1101)$ *avec l'injection* $\sigma = \{1 \mapsto 1; 2 \mapsto 3\}$ *telle que* $\Delta(\varpi_1) = \Delta(\pi_1)$ *et* $\Delta(\varpi_2) = \Delta(\pi_3)$. *La trajectoire* ϖ *est quant à elle l'unique trajectoire minimale.*

À l'aide de notre abstraction par impliquants premiers d'itération, nous pouvons calculer une sur-approximation des itérations formant les trajectoires causalement minimales.

Commençons par observer que si il existe $x \in X$ tel que $x_i \neq b$, alors une trajectoire atteignant $[\mathrm{v}_i = b]$ depuis x va forcément utiliser une itération modifiant la valeur de la composante i de $1 - b$ vers b. D'après notre résultat de la sous-section précédente, cette itération appartient à l'ensemble $P = \{\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \mathrm{lfp}\, A^{\#}_{\mathring{a}(X)}\}$.

L'idée est alors d'ajouter à cet ensemble P les impliquants premiers d'itération qui sont nécessaires pour pouvoir appliquer les itérations abstraites par P : étant donné un impliquant premier d'itération $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in P$, pour chaque littéral $[\mathrm{v}_j = d] \in C$, si il existe une configuration $x \in X$ où $x_j \neq d$ ($x_j = 1 - d$), ou si il existe un autre impliquant premier d'itération dans P qui nécessite $[\mathrm{v}_j = 1 - d]$, alors nous ajoutons tout impliquant premier d'itération $\langle \mathrm{v}_j : 1 - d \rightsquigarrow d, C'\rangle \in \mathrm{lfp}\, A^{\#}_L$. Ce processus est alors répété jusqu'à son point fixe.

Définition 4.4.21 (Clôture minimale par causalité)**.** *Étant donnés un ensemble de configurations* $X \subseteq \{0, 1\}^n$ *et un ensemble d'impliquants premiers d'itération* $P \subseteq \mathrm{lfp}\, A^{\#}_{\mathring{a}(X)}$, $\mathrm{lfp}_P \nabla^{\#}_{\mathring{a}(X)}$ *est la clôture minimale par causalité de P depuis X où*

$$\nabla^{\#}_L(P) := P \cup \{\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \mathrm{lfp}\, A^{\#}_L \mid [\mathrm{v}_i = b] \in \mu^{\#}(P), [\mathrm{v}_i = a] \in L \cup \mu^{\#}(P)\}$$

où

$$\mu^{\#}(P) := \bigcup \{C \mid \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in P\}$$

Le calcul de $\mathrm{lfp}_P \nabla^{\#}_{\mathring{\alpha}(X)}$ est de complexité linéaire avec la taille de $\alpha(f)$.

L'ensemble obtenu forme alors une sur-approximation des itérations formant les trajectoires causalement minimales :

Théorème 4.4.22. *Toute trajectoire de $ITG(f)$ causalement minimale pour l'atteinte de $[\mathrm{v}_i = b]$ depuis une configuration de X utilise seulement des itérations de $\gamma(\mathrm{lfp}_P \nabla^{\#}_{\mathring{\alpha}(X)})$ où $P = \{\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in \mathrm{lfp}\, A^{\#}_{\mathring{\alpha}(X)}\}$.*

Une preuve possible, que nous ne détaillerons pas ici, est donnée dans [146], et démontre que si une trajectoire utilise une itération n'appartenant pas à l'ensemble $\gamma(\mathrm{lfp}_P \nabla^{\#}_{\mathring{\alpha}(X)})$, alors elle n'est pas causalement minimale.

Exemple 4.4.23. *Prenons le réseau exemple f à 4 composantes défini page 177, pour lequel nous nous intéressons aux trajectoires causalement minimales pour l'atteinte de $[\mathrm{v}_4 = 1]$ depuis la configuration* 1000. *On remarque que* $\mathrm{lfp}\, A^{\#}_{\mathring{\alpha}(X)} = \alpha(f)$ *avec* $X = \{1000\}$. *Avec* $P = \{\langle \mathrm{v}_4 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 1]\}\rangle\}$, *nous obtenons le raffinement suivant :*

$$\mathrm{lfp}_P \nabla^{\#}_{\mathring{\alpha}(X)} = \{\langle \mathrm{v}_4 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 1]\}\rangle, \langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle\}$$

4.4.4 Sous-approximation des trajectoires en asynchrone

Les deux résultats précédents produisent une sur-approximation des trajectoires. Ces analyses sont utiles pour donner des conditions nécessaires à l'existence de trajectoires, et pour raisonner sur leur ensemble.

Dans cette sous-section, nous abordons la question d'une sous-approximation des trajectoires asynchrones, c'est-à-dire de conditions suffisantes à l'existence de trajectoires concrètes formées par $ITA(f)$.

L'approche que nous présentons utilise toujours l'abstraction par impliquants premiers d'itération, et montre que lorsqu'un ensemble de tels impliquants satisfait une certaine propriété, alors cela garantit l'existence de trajectoires qui peuvent être reconstruites à partir de l'abstraction.

Un ingrédient important de cette sous-approximation est la mise en avant de *relations de dépendance* entre les impliquants premiers d'itération : si $\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle$ et $\langle \mathrm{v}_j : d \rightsquigarrow e, C'\rangle$ sont tels que $[\mathrm{v}_j = e] \in C$, alors la résolution du premier impliquant peut dépendre du second. Ces relations se modélisent sous la forme d'un graphe dirigé, c'est-à-dire une relation binaire entre les impliquants premiers d'itération.

Définition 4.4.24. *Étant donné un ensemble P d'impliquants premiers d'itération, $E(P) \subseteq P^2$ est la relation binaire suivante :*

$$E(P) := \{(\langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle, \langle \mathrm{v}_j : d \rightsquigarrow e, C'\rangle) \in P^2 \mid [\mathrm{v}_j = e] \in C\}$$

Pour tout élément $p \in P$, $\Lambda_{E(P)}(p)$ est l'ensemble des éléments en relation avec p dans la clôture transitive de $E(P)$:

$$\Lambda_{E(P)}(p) := \mathrm{lfp}\,(Q \mapsto Q \cup \{q' \mid (q, q') \in E(P), p = q \vee q \in Q\})$$

On dit que $E(P)$ est acyclique si $\forall p \in P$, $p \notin \Lambda_{E(P)}(p)$.

Dans le cas où $E(P)$ est acyclique, pour tout $p \in P$, on note $M(p)$ l'ensemble des impliquants présents sur tous les chemins entre toutes les racines de p et p (dans le cas où $E(P)$ est un arbre, $M(p)$ est tout simplement l'ensemble des ancêtres de p). Intuitivement, les impliquants dans $M(p)$ sont les responsables, indépendamment et à eux seuls, de la dépendance envers p.

Définition 4.4.25. *Étant donné un ensemble P d'impliquants premiers d'itération tel que $E(P)$ est acyclique, pour tout $p \in P$,*

$$M(p) := \{p\} \cup \textstyle\bigcap_{q:(q,p)\in E(P)} M(q)$$

Le théorème suivant donne une condition suffisante pour qu'un ensemble P d'impliquants premiers d'itération puisse se concrétiser en une trajectoire à partir d'une configuration $x \in \{0,1\}^n$ donnée. La première condition s'assure de l'acyclicité des relations dans P, ce qui permet un raisonnement par induction. La seconde condition s'assure que pour tout impliquant d'itération $p = \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle$, pour tout littéral $[\mathrm{v}_j = d] \in C$, si $x_j \neq d$ ou si il existe un impliquant hors $M(p)$ de la forme $\langle \mathrm{v}_j : d \rightsquigarrow 1-d, D\rangle$, alors il doit exister au sein de P un impliquant pour une itération permettant d'atteindre la valeur d de la composante j, autrement dit, il doit exister $\langle \mathrm{v}_j : 1-d \rightsquigarrow d, C'\rangle \in P$. Enfin, la troisième condition s'assure qu'il existe un ordre entre les littéraux de C de telle sorte que les dépendances d'un littéral ne contredisent pas les littéraux inférieurs.

Théorème 4.4.26. *Soient $x \in \{0,1\}^n$ une configuration et $P \subseteq \alpha(f)$ un ensemble d'impliquants premiers d'itération qui satisfait les propriétés suivantes :*

1. *$E(P)$ est acyclique ;*
2. $\forall p = \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in P$,
   $\forall [\mathrm{v}_j = d] \in C, (x_j \neq d \vee [\mathrm{v}_j = 1-d] \in \phi^{\#}(P \setminus M(p)))$
   $\Rightarrow \exists \langle \mathrm{v}_j : 1-d \rightsquigarrow d, C'\rangle \in P$;
3. *$\forall \langle \mathrm{v}_i : a \rightsquigarrow b, C\rangle \in P$, la clôture transitive de la relation binaire $\preceq \subseteq C^2$ définie par*

$$[\mathrm{v}_j = d] \preceq [\mathrm{v}_k = e] \overset{\Delta}{\Leftrightarrow} (j = k \wedge d = e) \vee \exists p = \langle \mathrm{v}_j : 1 - d \rightsquigarrow d, C' \rangle \in P, \; q = \langle \mathrm{v}_k : e \rightsquigarrow 1 - e, C'' \rangle \in P : q \in \Lambda_{E(P)}(p)$$

est une relation d'ordre.

Pour tout $\langle \mathrm{v}_i : a \rightsquigarrow b, C \rangle \in P$, il existe une séquence d'itérations asynchrones de x à une configuration $y \in \{0,1\}^n$ telle que $y_i = b$ et $\forall [\mathrm{v}_j = d] \in C : i \neq j, y_j = d$.

Démonstration. Par induction. *Cas de base.* Si P est un singleton, $P = \{\langle \mathrm{v}_i : a \rightsquigarrow b, C \rangle\}$, alors nécessairement, $\forall [\mathrm{v}_j = d] \in C$, $x_j = d$. Ainsi, il existe une trajectoire composée d'une seule itération pour chaque itération dans $\gamma(P)$ (qui n'est jamais vide).

Induction. $E(P)$ étant acyclique, on choisit $p = \langle \mathrm{v}_i : a \rightsquigarrow b, C \rangle \in P$ tel qu'il n'existe aucun $q \in P$ avec $(q, p) \in E(P)$ (p est une racine). L'ensemble $P \setminus \{p\}$ satisfait les conditions du théorème, et par hypothèse d'induction, $\forall q = \langle \mathrm{v}_j : d \rightsquigarrow e, C' \rangle \in P, p \neq q$, il existe une trajectoire de x vers une configuration y où $y_j = e$. D'après la condition 3 du théorème, il existe un ordre $\preceq$ entre les littéraux C. On note ainsi $C = \{C_1, \ldots, C_t\}$ ces littéraux avec $\forall r, s \in \{1, \ldots, t\}, r \leqslant s \Leftrightarrow C_r \preceq C_s$. Partant avec $C_1 = [\mathrm{v}_j = d]$, soit $x_j = d$, soit, par la condition 2 du théorème, il existe $q^1 = \langle \mathrm{v}_j : 1 - d \rightsquigarrow d, C' \rangle \in P$ et par hypothèse d'induction, il existe une trajectoire depuis x menant à une configuration z^1 vérifiant $[\mathrm{v}_j = d]$. Cette trajectoire utilise uniquement des itérations de $T^1 = \gamma(\Lambda_{E(P)}(q^1))$. Avec $C_2 = [\mathrm{v}_k = e]$, si $z^1_k \neq e$, nous pouvons appliquer le même raisonnement pour l'existence de $q^2 = \langle \mathrm{v}_k : 1 - e \rightsquigarrow e, C'' \rangle \in P$ et d'une trajectoire depuis z^1 vers une configuration z^2 vérifiant $[\mathrm{v}_k = e]$ et utilisant uniquement des itérations de $T^2 = \gamma(\Lambda_{E(P)}(q^2))$. De plus, comme $C_1 \preceq C_2$, nous avons la garantie qu'aucune itération de T^2 ne modifie la composante j, ainsi, z^2 vérifie également $[\mathrm{v}_j = d]$. Ce raisonnement est itéré jusqu'à C_t. La trajectoire ainsi obtenue arrive dans une configuration z^t telle que $C(z^t)$. Cette trajectoire peut alors être étendue avec l'itération correspondante dans $\gamma(\{p\})$ pour obtenir une configuration y où $y_i = b$ et $\forall [\mathrm{v}_j = d] \in C$, $y_j = d$. ■

Étant donné un ensemble P d'impliquants premiers d'itération, vérifier si P satisfait les conditions du théorème de sous-approximation s'effectue en temps linéaire en la taille de P. Étant donné un impliquant premier d'itération p, trouver si il existe un ensemble P contenant p et vérifiant les conditions du théorème peut se formuler comme un problème de satisfaction de contraintes booléennes (SAT), où les variables sont les littéraux $[\mathrm{v}_j = d]$ impliqués dans la résolution de p, et leurs valeurs sont les différents impliquants premiers d'itération de la forme $\langle \mathrm{v}_j : 1 - d \rightsquigarrow d, C' \rangle$.

Exemple 4.4.27. *Prenons le réseau exemple g à 4 composantes défini page 177.*

L'ensemble $P_1 \subseteq \alpha(g)$, dont $E(P_1)$ est représenté ci-dessous satisfait les critères du théorème de sous-approximation depuis $x = 1000$

$$\langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 0]\}\rangle$$
$$\searrow$$
$$\langle \mathrm{v}_1 : 1 \rightsquigarrow 0, \{[\mathrm{v}_2 = 1]\}\rangle$$
$$\searrow$$
$$\langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle$$

On remarque qu'aucun impliquant d'itération pour $\mathrm{v}_1 : 0 \rightsquigarrow 1$ n'est présent malgré l'impliquant $p = \langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle$ car $M(p) = P_1$, ainsi la condition 2 est bien satisfaite. En appliquant le raisonnement par induction, nous pouvons construire à partir de P_1 la trajectoire $(1000 \to 1100; 1100 \to 0100; 0100 \to 0110)$.

On cherche maintenant à construire $P_2 \subseteq \alpha(b)$ tel que $\langle \mathrm{v}_4 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 0], [\mathrm{v}_3 = 1]\}\rangle \in P_2$. Nécessairement $P_1 \subseteq P_2$ (condition 2). Ainsi, il est nécessaire d'avoir un impliquant pour une itération de la forme $\mathrm{v}_2 : 1 \rightsquigarrow 0$ dû à la présence de $\langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1]\}\rangle$ (condition 2), or il n'existe pas de tel impliquant. Ainsi, notre sous-approximation ne permet pas de conclure sur l'existence d'une trajectoire depuis x pour activer la composante 4.

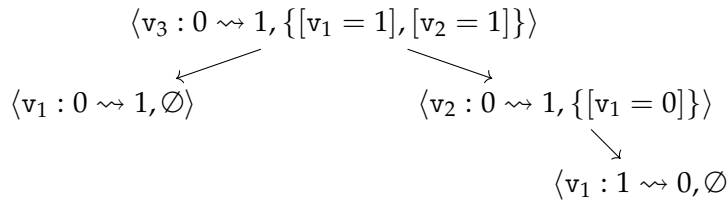
Nous illustrons la condition 3 sur deux autres exemples.

Exemple 4.4.28. *Prenons le réseau booléen f' à 3 composantes défini par $f'_1(x) := \neg x_1 \wedge \neg x_2$, $f'_2(x) := \neg x_1 \wedge \neg x_2$, et $f'_3(x) = x_1 \wedge x_2$; et cherchons $P \subseteq \alpha(f')$ contenant $\langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1], [\mathrm{v}_2 = 1]\rangle$ vérifiant les conditions du théorème depuis la configuration 000. En appliquant la condition 2, nous obtenons l'ensemble suivant P, dont nous représentons $E(P)$:*

$$\langle \mathrm{v}_3 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 1], [\mathrm{v}_2 = 1]\}\rangle$$
$$\swarrow \qquad\qquad \searrow$$
$$\langle \mathrm{v}_1 : 0 \rightsquigarrow 1, \{[\mathrm{v}_2 = 0]\}\rangle \qquad\qquad \langle \mathrm{v}_2 : 0 \rightsquigarrow 1, \{[\mathrm{v}_1 = 0]\}\rangle$$
$$\swarrow \qquad\qquad\qquad\qquad \searrow$$
$$\langle \mathrm{v}_2 : 1 \rightsquigarrow 0, \emptyset\rangle \qquad\qquad\qquad\qquad \langle \mathrm{v}_1 : 1 \rightsquigarrow 0, \emptyset\rangle$$

Or la condition 3 n'est pas vérifiée : en effet, $[\mathrm{v}_1 = 1] \preceq [\mathrm{v}_2 = 1]$ par la branche gauche de l'arbre ci-dessus ; et $[\mathrm{v}_2 = 1] \preceq [\mathrm{v}_1 = 1]$ par sa branche droite. Ainsi $\preceq$ n'est pas une relation d'ordre ; donc P ne satisfait pas la condition 3. Et en effet : il n'existe aucune trajectoire asynchrone *permettant d'atteindre une configuration dont la composante 3 vaut 1.*

Exemple 4.4.29. *Prenons le réseau booléen g' à 3 composantes défini par $g'_1(x) := \neg x_1$, $g'_2(x) := \neg x_1$, $g'_3(x) := x_1 \wedge x_2$; et cherchons $P \subseteq \alpha(g')$ contenant $\langle \mathtt{v}_3 : 0 \rightsquigarrow 1, \{[\mathtt{v}_1 = 1], [\mathtt{v}_2 = 1]\}\rangle$ vérifiant les conditions du théorème depuis la configuration 000. En appliquant la condition 2, nous obtenons l'ensemble suivant P', dont nous représentons $E(P')$:*



La condition 3 est ici vérifiée car le seul couple non réflexif de $\preceq$ est $[\mathtt{v}_2 = 1] \preceq [\mathtt{v}_1 = 1]$; ainsi $\preceq$ est une relation d'ordre qui indique l'ordre dans lequel les itérations doivent être effectuées : d'abord activer 2, puis 1, pour pouvoir activer 3, l'activation de 1 ne modifiant pas la valeur de la composante 2. Nous pouvons ainsi construire la trajectoire concrète suivante : $(000 \to 010; 010 \to 011; 011 \to 111)$.

4.4.5 Discussion

Propriétés des trajectoires Les deux derniers résultats portent sur l'étude de trajectoires dont la finalité est l'accessibilité d'une configuration où une composante (i) a une valeur donnée (b). On remarque que cette spécification permet également de considérer des configurations partielles et des séquences de configurations partielles en s'autorisant à rajouter des variables dans notre système. Ainsi, pour étudier les trajectoires amenant à toute configuration $x \in \{0,1\}^n$ vérifiant $C(x)$, où C est une conjonction de littéraux, on ajoute au réseau une composante $m = n+1$ telle que $f_m(x) = 1 \Leftrightarrow C(x)$; et on applique les résultats précédents avec $i = m$ et $b = 1$. Pour étudier les trajectoires passant d'abord par une configuration vérifiant C_1, puis C_2 on ajoute deux composantes $m_1 = n+1$ et $m_2 = n+2$ au réseau avec $f_{m_1}(x) = 1 \Leftrightarrow C_1(x)$ et $f_{m_2}(x) = 1 \Leftrightarrow x_{m_1} = 1 \wedge C_2(x)$; et on applique les résultats précédents avec $i = m_2$ et $b = 1$ (en considérant ces nouvelles composantes à la valeur 0 initialement).

Généralisations L'interprétation abstraite définie dans cette section peut se généraliser aux *réseaux discrets* où le domaine des variables est fini et discret, mais pas forcément binaire. Elle se généralise également aux réseaux d'automates et réseaux de Petri 1-bornés où, au lieu de spécifier des fonctions déterministes pour calculer les mises à jour, sont définies directement les conditions pour les changements de valeurs des composantes

des configurations [148, 77, 146]. Ainsi, pour ces derniers formalismes, le calcul des impliquants d'itération n'est pas nécessaire, car ils sont donnés explicitement dans la spécification du modèle. On peut en effet remarquer que l'aspect *premier* des impliquants d'itération est une considération d'optimalité pour l'interprétation abstraite définie : les résultats tiennent toujours dans le cas où des impliquants d'itération non minimaux sont utilisés, tant que leur ensemble forme une couverture complète des conditions d'itération.

En pratique Le principal avantage de cette interprétation abstraite est le passage d'un ensemble d'itérations de taille exponentielle selon le nombre de composantes du réseau à l'ensemble de leurs impliquants premiers de complexité exponentielle selon le nombre de régulateurs de chaque composante. Pour les réseaux biologiques, le nombre de régulateurs est généralement bien plus petit que le nombre de composantes. Quand des réseaux comprennent plusieurs centaines, voire milliers de composantes, le nombre maximum de régulateurs par composante est au pire de l'ordre de la dizaine. De plus, les fonctions booléennes sont généralement monotones (un régulateur est soit un inhibiteur, soit un activateur, mais rarement les deux), et ont souvent une forme proche de disjonctions entre la présence des activateurs, conjugué avec l'absence des inhibiteurs. Ainsi, le nombre d'impliquants premiers d'itération est bien en deçà de la borne maximale de 2^d ou $\binom{d}{\lfloor d/2 \rfloor}$ dans le cas monotone, où d est le degré entrant des sommets dans le graphe d'interaction.

La vérification de l'existence de trajectoires est très difficile en pratique dans les grands réseaux booléens, à cause de l'explosion combinatoire des itérations possibles. D'ailleurs, le problème de l'atteignabilité est PSPACE-complet pour les réseaux booléens avec les itérations asynchrones. L'interprétation abstraite présentée ici a rendu possible l'analyse de réseaux de taille jusqu'alors inabordable par les outils classiques de *model checking*. Pour de nombreux réseaux, les sur- et sous-approximations présentées ici s'avèrent souvent suffisantes pour conclure formellement sur l'existence ou absence de trajectoires [149, 77, 145].

Hormis la vérification de l'existence de certaines trajectoires, notre interprétation abstraite permet de spécifier des problèmes de contrôle sous la forme de problèmes SAT sur un nombre de variables restreint, et donc applicables à l'analyse de réseaux de grande taille. Nous pouvons par exemple citer l'identification de points communs entre toutes les trajectoires menant à un ensemble de configurations donné (*cut sets*) [147] ; l'identification de mutations (composantes forcées à une valeur donnée) pour contrôler l'ac-

cessibilité d'un ensemble de configurations [145] ; ou encore l'identification d'itérations responsables de la perte d'accessibilité de configurations [71].

4.5 Quelques applications en biologie des systèmes

Outre leur apport sur la compréhension fondamentale des systèmes dynamiques complexes, une part importante des résultats théoriques sur les réseaux booléens aide à produire des méthodes formelles et efficaces pour l'étude des modèles de réseaux biologiques. Nous présentons brièvement dans cette section quelques-unes de ces méthodes.

4.5.1 Identification de réseaux booléens

En biologie, il n'y a pas de code source (ni même de binaire) disponible à partir duquel un modèle formel peut être automatiquement dérivé, ou intensivement comparé. Les modèles de réseaux biologiques sont élaborés à partir de la connaissance très partielle sur les interactions entre les molécules étudiées issue de la lecture d'articles scientifiques ou de bases de données, elles-mêmes peuplées suite à la lecture d'articles.

Pour la majorité des applications en biologie, la connaissance actuelle ne permet pas de spécifier complètement un réseau booléen. En pratique, les modélisations partent souvent du graphe d'interaction. Ce dernier rassemble toutes les interactions découvertes expérimentalement entre les variables choisies du système. Mais les fonctions booléennes sous-jacentes sont rarement connues. Par exemple, dans le cas où il serait connu que deux variables influencent directement et positivement la valeur d'une troisième variable, il est rarement connu si l'influence est additive (disjonction) ou multiplicative (conjonction). Pour parfaire l'incertitude du modeleur, le graphe d'interaction peut être incomplet (interactions inconnues), et mélange souvent des interactions découvertes dans des types cellulaires et conditions expérimentales très différents (souris, éléphant, humain…).

L'identification de réseaux booléens est ainsi un problème récurrent et épineux pour les applications en biologie des systèmes. L'objectif est alors, étant donné un graphe d'interaction $\mathcal{G}$ d'identifier le ou les réseaux booléens f tels que $G(f) \subseteq \mathcal{G}$ et qui vérifient des contraintes sur ses points fixes, et des contraintes sur ses trajectoires. Ces contraintes peuvent être issues de connaissances générales sur le système, ou de données expérimentales mesurant l'activité des gènes dans un état stationnaire (point fixe), ou mesurant l'expression de gènes ou protéines au cours du temps (trajectoires).

Le problème majeur de l'identification de réseaux réside dans la grande combinatoire des modèles possibles et dans la complexité de la vérification des propriétés recherchées. Si une variable est directement influencée par d variables, il y a grossièrement 2^{2^d} fonctions booléennes possibles (pour les fonctions monotones, cela correspond au *Dedekind number*, inconnu pour $d > 8$).

Les analyses statiques des réseaux permettent alors de restreindre efficacement cet espace de recherche en apportant des contraintes supplémentaires comme la présence de certains cycles, ou des conditions nécessaires pour l'existence de trajectoires. Les méthodes les plus efficaces actuellement combinent analyses statiques et problèmes de satisfaction de contraintes booléennes (SAT) ou apparentés, comme l'Answer Set Programming [52, 143].

Le sujet de l'identification de réseau est un domaine de recherche très actif où se mêlent intelligence artificielle, apprentissage automatique, et méthodes formelles, et où beaucoup de progrès restent à accomplir. Notre domaine manque également cruellement de méthodes permettant de raisonner sur des familles de modèles : en effet, il est extrêmement rare qu'un unique réseau booléen vérifie les propriétés demandées. En pratique, ce sont souvent des milliers ou millions de modèles possibles, non distinguables expérimentalement. Or peu d'approches permettent actuellement de traiter formellement ces ensembles de modèles.

4.5.2 Reprogrammation cellulaire

Au fil de ses divisions, et en fonction de l'environnement, une cellule pluripotente (par exemple une cellule souche) peut se spécialiser en différents types de cellules (peau, graisse, os, ...). Ces différenciations s'observent aussi au niveau de l'expression de certains gènes, qui ne vont être exprimés que dans certains sous-types cellulaires. Pendant longtemps, il était supposé qu'une fois une cellule différenciée, elle ne pouvait pas revenir en arrière et se transformer en un autre type de cellule. Mais des expériences récentes ont montré qu'il était possible de changer le type de certaines cellules, en déstabilisant son réseau de gènes, pour transformer par exemple de la peau en neurone, ou de la graisse en os [184, 192, 185]. On parle alors de dé-différenciation (retour à un état pluripotent) et de trans-différenciation (changement de type cellulaire). La figure 4.3 illustre ces processus.

De nombreux travaux portent actuellement sur l'étude de la *reprogrammation cellulaire* à l'aide des réseaux booléens [1, 48, 202, 49]. L'objectif est de prédire automatiquement des perturbations à appliquer à la cellule pour

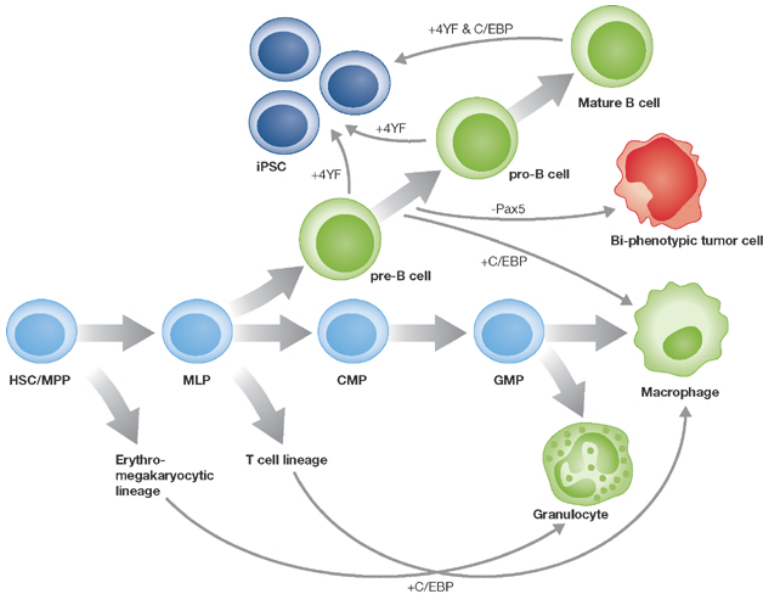


FIGURE 4.3 – *Représentation schématique des différenciations cellulaires au cours d'une partie de l'hématopoïèse (processus de production des cellules sanguines), et quelques mécanismes de reprogrammation connus via l'action de facteurs de transcriptions indiqués sur les arcs ; iPSC : cellules souches induites (dé-différenciation). Reproduit depuis [160].*

provoquer un changement de type. Ces perturbations sont typiquement des mutations de certains gènes, afin de bloquer ou de forcer leur expression.

Une des hypothèses utilisées est que les états différenciés stables de la cellule correspondent à des points fixes (ou plus généralement à des attracteurs cycliques) du réseau booléen associé. La reprogrammation des réseaux booléens revient alors à trouver les perturbations à appliquer à un point fixe pour rendre possible l'existence d'une trajectoire vers un point fixe différent.

Les réseaux booléens utilisés pour les problématiques de reprogrammation sont généralement de très grande taille, allant de plusieurs dizaines à plusieurs centaines, voire milliers de variables. La prédiction de perturbations pour la reprogrammation pose ici un problème combinatoire du nombre de perturbations candidates à tester.

Les analyses statiques permettent alors de restreindre l'espace de recherche des perturbations candidates, que ce soit d'un point de vue formel pour éviter d'étudier des candidats qui ne peuvent pas provoquer une reprogrammation [124] ; ou d'un point de vue pratique pour guider

la recherche d'une solution possible. Par exemple, dans [53], les auteurs exploitent les cycles positifs du graphe d'interaction pour prédire certaines perturbations pour provoquer un changement de point fixe. Des travaux en cours utilisent également les abstractions des trajectoires pour déduire, à l'aide de résolution de problèmes SAT, les perturbations nécessaires pour atteindre un état ciblé.

Remerciements. Une partie des résultats présentés ont été obtenus en collaboration avec Julio Aracena et Lilian Salinas (Universidad de Concepción, Chile) et Maximilien Gadouleau (Durham University, UK) sur le nombre de points fixes dans les réseaux booléens ; Morgan Magnin, Olivier Roux et Maxime Folschette (LS2N, Nantes) sur les abstractions des trajectoires ; Carito Guziolowski (LS2N, Nantes), Max Ostrowski (Potsdam University), Anne Siegel (IRISA, Rennes) sur les aspects d'identification de réseaux ; Antonio del Sol, Sascha Zickenrott (LCSB Luxembourg), Laurence Calzone, Andrei Zinovyev (Institut Curie), Stefan Haar et Hugues Mandon (Inria Saclay/LSV, Cachan) sur les aspects de reprogrammation cellulaire.

Chapitre 5

Expérimentation mathématique avec Sage

Vincent Delecroix

Ce cours est une introduction au logiciel de mathématiques SageMath (ou Sage). L'objectif de ces notes est double :

1. *vous permettre de débuter avec Sage et vous donner accès à une liste de références vers des ressources plus avancées,*
2. *avoir des pointeurs pour faire des expérimentations autour des autres cours de cette école. En particulier les cours sur la satisfaisabilité (chapitre 2) via la résolution de problèmes linéaires et SAT, celui sur la géométrie (chapitre 3) via les polytopes et enfin celui sur les probabilités (chapitre 1) via la génération aléatoire de structures discrètes.*

Sage est un logiciel qui évolue rapidement depuis sa création en 2005. Les informations contenues dans ce document concernent la version Sage 8.0 datant de juillet 2017.

5.1 Histoire et idéologie

Cette courte section donne quelques points de repère sur l'écosystème du logiciel Sage. Il ne vous apprendra pas à utiliser Sage et vous pouvez sauter à la section suivante si c'est cela que vous cherchez.

Le début de l'histoire. Sage est un logiciel de mathématiques qui a été créé autour de 2005 par William Stein. Sa motivation originale était de faire interagir des bibliothèques et logiciels pour faire des calculs autour

des formes modulaires (voir par exemple le livre [181] datant de 2006 et contenant les premiers exemples de code Sage). La première version de Sage (qui s'appelait alors Manin) faisait interagir PARI/GP (bibliothèque et logiciel de théorie des nombres), Pyrex (compilateur permettant d'interfacer simplement du code C en Python), GMP (pour les entiers en précision arbitraire) et NTL (bibliothèque pour la théorie des nombres). Sage, ou plutôt Manin, offrait aussi une interface à Mathematica et Magma, deux logiciels propriétaires.

Peu après, sous l'impulsion de David Joyner et David Kohel qui rejoignirent l'équipe de développement, une interface à GAP (pour pouvoir travailler avec les groupes de permutations), Maxima (pour le calcul symbolique) et Singular (pour les polynômes multivariés) sont crées. La structure interne de Manin évolue considérablement. Le nom change de Manin à SAGE comme acronyme à "Software for Algebra and Geometry Experimentation". Aujourd'hui, le logiciel s'appelle Sage ou SageMath et l'acronyme a été laissé de côté.

Pour plus de détails bibliographiques sur les débuts, vous pouvez consulter le document [182].

Aujourd'hui. Sage est actuellement développé par des centaines de personnes et permet l'interaction d'environ 200 bibliothèques et logiciels. Il a un cycle de développement rapide avec entre 4 et 5 versions par an ces dernières années, voir Figure 5.1. Ce choix de cycle rapide permet aux utilisateurs de profiter des améliorations rapidement et simplement.

année	mois	version
2013	déc.	**6.0**
2014	jan.	6.1
	mai	6.2
	août	6.3
	nov.	6.4
2015	fév.	6.5
	avr.	6.6
	mai	6.7
	juil.	6.8
	oct.	6.9
2016	jan.	**7.0**
	mars	7.1
	mai	7.2
	août	7.3
	oct.	7.4
2017	jan.	7.5
	mars	7.6
	juil.	**8.0**

FIGURE *5.1 – Dates de sorties des versions de Sage depuis la version 6.0.*

L'écosystème Sage. Comme il a été mentionné dans les deux paragraphes précédents, la partie principale de Sage est une interface commune à des bibliothèques spécialisées. Il contient aussi du code propre essentiellement écrit en Python (1.5 millions de lignes) et Cython (0.5 million de lignes).

Notons que le projet Pyrex faisant partie de Sage à ses débuts a été repris par certains développeurs de Sage pour donner naissance à Cython (un des mainteneurs est R. Bradshaw, employé de Google et ancien développeur de Sage). Ce dernier est largement utilisé en dehors de Sage. Les développeurs sont ainsi encouragés à participer au développement d'outils dont la portée dépasse simplement l'usage interne de Sage.

Licence. Dès sa création, un des objectifs du projet SageMath est de proposer un logiciel de mathématiques générales sous licence libre (et donc gratuit). Des logiciels propriétaires (souvent chers) sont surreprésentés dans la recherche ainsi que les cursus universitaires. Sage est publié sous licence GPL [1].

Python. Une autre particularité de Sage est d'avoir adopté le langage de programmation Python plutôt que d'en créer un nouveau (de même que Maxima est basé sur Lisp, mais contrairement à Mathematica ou PARI/GP qui nécessitent d'apprendre leur propre langage). Le langage Python est très facile à prendre en main même pour quelqu'un qui n'a jamais programmé. Cette simplicité du langage est probablement pour beaucoup dans la popularité de Sage. Par exemple, le code suivant devrait être lisible par la plupart des scientifiques.

```
sage: V = ZZ^3
sage: v1 = V([1, 3, 5])
sage: v2 = V([-1, 1, 0])
sage: v1.dot_product(v2).is_prime()
True
```

Bien que pour utiliser Sage il vous faut écrire en Python, les bibliothèques utilisées en internes sont écrites en C/C++, Lisp ou Fortran.

Il est à noter que Python n'est pas un langage performant (attendez vous à des boucles jusqu'à 60 fois plus lentes que dans un langage compilé [2]). Cependant ces écueils de Python sont faciles à identifier et, lorsque c'est nécessaire, il est facile d'insérer du code en C/C++ en utilisant Cython.

1. voir par exemple `https://fr.wikipedia.org/wiki/Licence_publique_générale_GNU`.

2. Les comparatifs ne manquent pas, voir par exemple `http://www.osnews.com/story/5602/Nine_Language_Performance_Round-up_Benchmarking_Math_File_I_O`.

Développement. Le développement de Sage est assez unique car il regroupe plusieurs centaines de développeurs sur un pied d'égalité. Chacun est libre de proposer une modification du code via un ticket sur le site `https://trac.sagemath.org`. Le ticket passe ensuite par un relecteur qui est un des autres développeurs. Suite à des discussions entre l'auteur et le relecteur des modifications peuvent être apportées au ticket. Lorsqu'un consensus est atteint le code est alors inclus dans Sage et sera disponible dans la prochaine version.

Il est relativement aisé de contribuer à Sage et toutes les utilisatrices y sont invitées.

Brève liste de logiciels et bibliothèques. Nous donnons une petite liste des bibliothèques et logiciels utilisés par Sage et qui sont cités dans ce document. Vous trouverez une liste exhaustive sur le site `http://www.sagemath.org/`. Nous choisissons de présenter seulement les 4 logiciels de calcul formels à partir desquels Sage a été construit à ces débuts (GAP, Maxima, PARI/GP et Singular) ainsi que les librairies prenant une importance plus grande dans le cadre de ce texte.

Cryptominisat (`https://github.com/msoos/cryptominisat`)
Bibliothèque C et module Python pour résoudre les problèmes SAT. Il s'agit d'un paquetage optionnel de Sage.

Cython (`http://cython.org/`)
Compilateur d'extension C/C++ pour Python (successeur de Pyrex).

GAP (`http://www.gap-system.org/`)
Logiciel de calcul formel avec un très fort développement vers la théorie des groupes. Les premières versions datent de 1988 [3].

GMP/MPIR (`https://gmplib.org/` et `http://mpir.org/`)
Librairie C pour les entiers en précision arbitraire (avec de nombreuses optimisations). La fourche MPIR de GMP a été crée pour offrir des optimisations pour plus d'architectures et un support Windows.

Maxima (`http://maxima.sourceforge.net/`)
Logiciel en licence libre pour la manipulation d'expressions symboliques. Le projet découle de Macsyma qui a commencé dans les années 1960 !

NTL (`http://www.shoup.net/ntl/`)
Librairie C++ pour la théorie des nombres.

Python (`https://www.python.org/`)
Langage de programmation sur lequel Sage se base.

3. Voir `http://www.gap-system.org/Doc/History/history.html`.

Pyrex
Compilateur anciennement utilisé par Sage pour créer des extensions Python en C.

PARI/GP (`https://pari.math.u-bordeaux.fr/`)
PARI est une bibliothèque C de théorie des nombres et GP un interpréteur qui permet de l'utiliser. Il est distribué sous licence libre et ses débuts datent de 1983 [4].

Parma Polyhedra Library (PPL) (`http://bugseng.com/products/ppl/`)
Bibliothèque pour les problèmes linéaires en nombres rationnels (polytopes, optimisation, etc).

Singular (`http://www.singular.uni-kl.de/`)
Bibliothèque et logiciel de calcul spécialisé dans les systèmes polynomiaux et les bases de Gröbner. Le projet a commencé en 1984 [5].

5.2 Installer et démarrer Sage

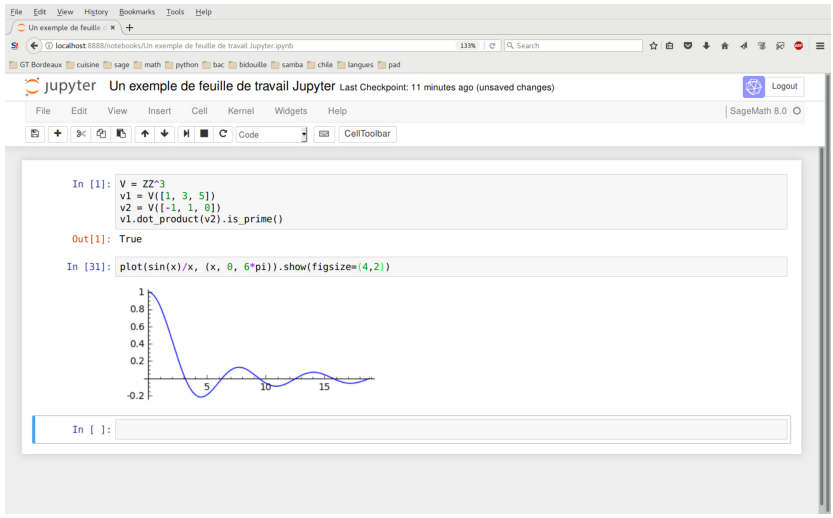
Sage peut s'utiliser d'au moins cinq façons

1. sur son propre ordinateur via la console (la commande pour lancer l'exécutable est appelée `sage`),

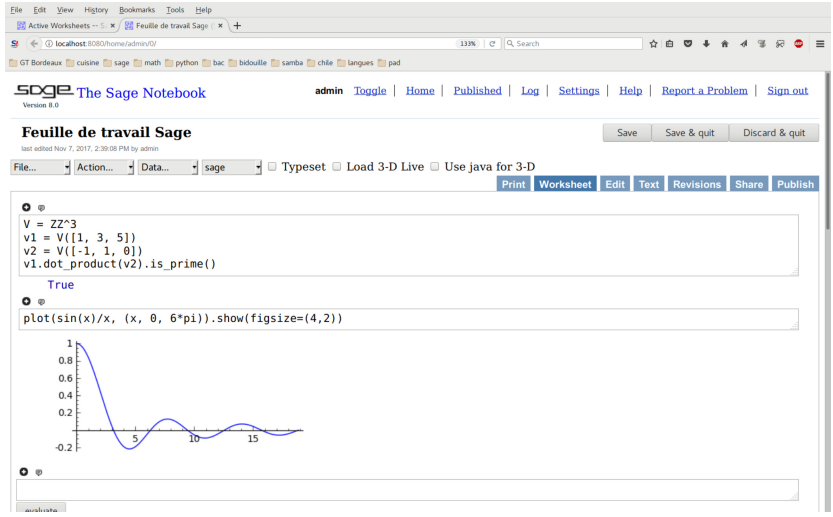
```
[vincent@mangouste ~]$ sage
 SageMath version 8.0, Release Date: 2017-07-21
 Type "notebook()" for the browser-based notebook interface.
 Type "help()" for help.

sage: V = ZZ^3
sage: v1 = V([1, 3, 5])
sage: v2 = V([-1, 1, 0])
sage: v1.dot_product(v2).is_prime()
True
sage:
```

4. Voir `http://pari.math.u-bordeaux.fr/timeline.html`
5. Voir `http://www.singular.uni-kl.de/index.php/background/history.html`

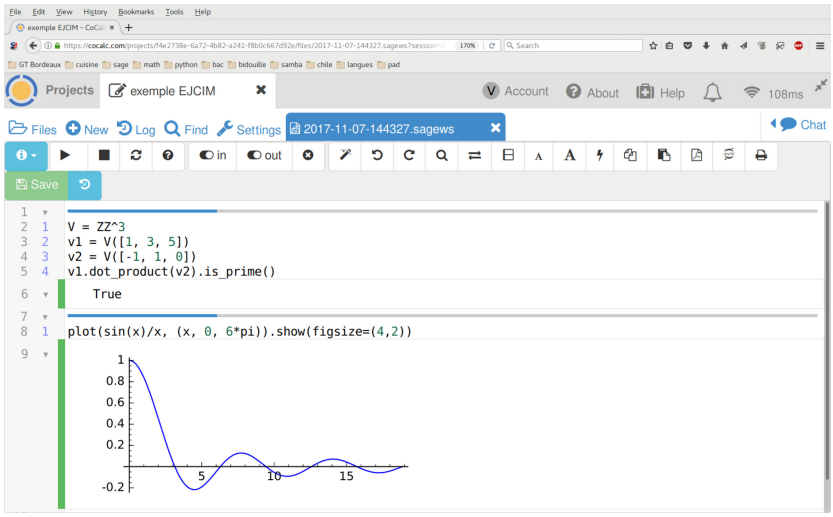
2. sur son propre ordinateur via Jupyter (interface graphique par défaut à partir de Sage 8.0) – la commande pour lancer Jupyter est `sage -n jupyter`,



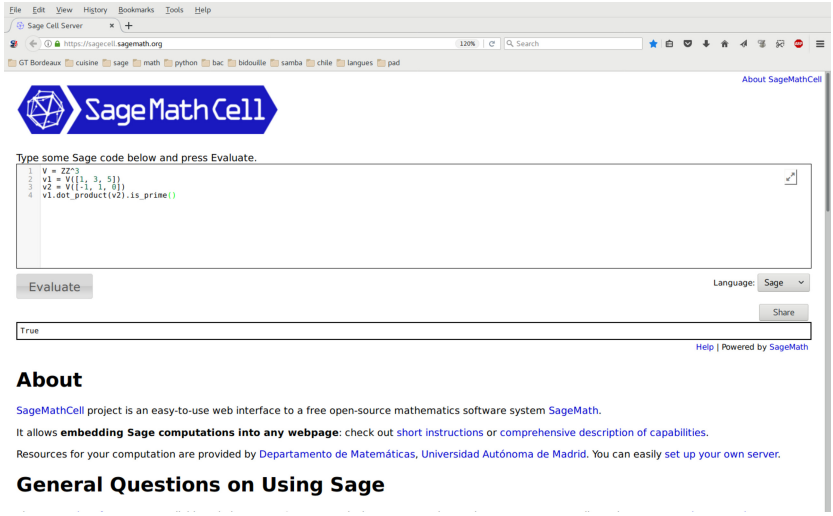
3. sur son propre ordinateur via le Sage Notebook (interface graphique par défaut jusqu'à Sage 7.6 et qui va progressivement disparaître) – la commande pour lancer le Sage Notebook est `sage -n sagenb`,



4. en ligne sur Cocalc (`https://cocalc.com/`),



5. en ligne via une cellule Sage `https://sagecell.sagemath.org/`).



La manière d'installer Sage sur votre machine dépend de votre système.

1. Debian ($\geqslant$ Stretch, 9.2) et Ubuntu ($\geqslant$ 17.04, Zesty) : installez le paquetage `sagemath`.
2. Fedora : il y a un support limité pour un paquetage `sagemath` non

officiel (voir `http://fedoraproject.org/wiki/SIGs/SciTech/SAGE` et `https://ask.sagemath.org/question/27158`).

3. Archlinux : installez le paquetage `sagemath`, voir `https://wiki.archlinux.org/index.php/SageMath`.
4. Gentoo : consultez `https://github.com/cschwan/sage-on-gentoo/`.
5. Windows : utilisez `https://github.com/embray/sage-windows/`.
6. Pour tous les autres systèmes, téléchargez le binaire correspondant. sur `http://www.sagemath.org/download`

Formats de fichiers, sauvegardes. Pour sauvegarder votre travail de programmation vous avez plusieurs possibilités.

extension	explication
`.py`	fichiers Python, pour lire un fichier Python depuis la console ou dans Jupyter vous pouvez utiliser `%runfile` ou `%attach`
`.pyx`	fichier source Cython, fonctionne également avec `%runfile` et `%attach`
`.sage`	fichiers Sage. Équivalent à un fichier Python sauf lorsque vous exécuterez votre fichier dans un shell via `$ sage mon_fichier.sage` auquel cas le préparseur de Sage est appliqué
`.ipynb`	feuille de travail ("notebook") Jupyter (le nom ipynb découle de "IPython notebook")
`.sws`	feuille de travail Sage (obsolète)
`.rst`	fichier au format ReST, permet d'écrire des fichiers mélangeant du texte et du code Sage. Conversion possible vers pdf ou feuille de travail Jupyter.

5.3 Cinq principes de base

Casse. Il y a deux conventions de nommage dans Sage

— notation serpent ("snake case") : `vector`, `matrix`, `is_prime`, ...

— notation chameau ("camel case") : `VectorSpace`, `PolynomialRing`, `Matrix`, ...

La notation chameau est réservée aux noms des objets (les espaces vectoriels, les anneaux de polynômes, les matrices, ...). La notation serpent est a priori réservée aux fonctions (comme `is_prime`) mais il y a parfois des fonctions qui construisent des objets (comme `vector` et `matrix`).

Orientation objet. Python est un langage objet. La plupart des fonctions qui s'appliquent à un objet `x` sont appelées via `x.f()` plutôt que `f(x)`.

```
sage: 42.bits()
[0, 1, 0, 1, 0, 1]
sage: bits(42)
Traceback (most recent call last)
...
NameError: name 'bits' is not defined
```

Parfois les deux syntaxes sont possibles

```
sage: 101.is_prime()
True
sage: is_prime(101)
True
```

Complétion automatique. Pour trouver une fonction, une manière pratique consiste à utiliser la complétion automatique par la touche tabulation. Cela consiste à écrire une partie d'un mot et appuyer sur la touche ⟨tab⟩.

```
sage: prim<tab>
           prime_divisors  prime_powers    primes
           prime_factors   prime_range     primes_first_n
           prime_pi        prime_to_m_part primitive_root
```

Lorsqu'une seule complétion est possible le mot est automatiquement complété. Cette remarque s'applique aussi aux méthodes d'un objet préalablement construit

```
sage: n = 42
sage: n.f<tab>
          n.factor
          n.factorial
          n.floor
```

Documentation. Il est possible d'accéder à la documentation directement dans la console ou une feuille de travail. Pour cela, on ajoute un point d'interrogation à la fin avant d'appuyer sur `<entrée>`.

```
sage: bernoulli?
Signature:      bernoulli(n, algorithm='default', num_threads=1)
Docstring:
   Return the n-th Bernoulli number, as a rational number.

   INPUT:

   * "n" - an integer

   * "algorithm":

     * "'default'" -- use 'flint' for n <= 300000, and 'bernmm'
       otherwise (this is just a heuristic, and not guaranteed
       to be optimal on all hardware)

     * "'arb'" -- use the arb library

     * "'flint'" -- use the FLINT library
```

Pour accéder au code source, on utilise deux points d'interrogation au lieu d'un.

Langage Python. Pour aller plus loin en programmation dans Sage il est indispensable de bien connaître la syntaxe et les structure de données Python. Python est avant tout un langage impératif et on retrouve les flots de contrôle `if/elif/else`, `for` et `while`. Par ailleurs, Python possède quelques structures de données avancées dont voici la liste.

nom	type	délimiteurs	modifiable
liste	`list`	`[` et `]`	oui
tuple	`tuple`	`(` et `)`	
chaîne	`str`	`'`, `"`, `'''`, `"""`	non
ensemble	`set`	`{` et `}`	oui
ensemble gelé	`frozenset`		non
dictionnaire	`dict`	`{` et `}`	oui

Pour apprendre Python, je recommande

— le tutoriel en français de la documentation officielle Python `https://docs.python.org/fr/2/tutorial/`,

— les premiers chapitres des "Scipy lecture notes" (en anglais) `http://www.scipy-lectures.org/`

— le tutoriel thématique "Tutorial : Programming in Python and Sage" de la documentation officielle de Sage (en anglais)

`http://doc.sagemath.org/html/en/thematic_tutorials/tutorial-programming-python.html`.
— le tutoriel "How to Think Like a Computer Scientist" (en anglais) `http://interactivepython.org/runestone/static/thinkcspy/index.html`

5.4 Ressources

Livres.
— Calcul Mathématique avec Sage [41] : ouvrage de référence en français qui couvre les différents aspects du calcul formel et scientifique. Couvre grosso-modo les mathématiques rencontrées dans un cursus de licence. Mentionnons également qu'une version en anglais pour Sage 8.0 est disponible. Toutes les versions et correctifs sont disponibles à `https://members.loria.fr/PZimmermann/sagebook/`.
— Sage for Undergraduates [19] : ouvrage en anglais qui couvre le programme de licence de mathématiques avec beaucoup de détails concernant le graphisme.

Documentation en ligne. Voir la fin de la Section précédente concernant le langage Python.

Autre ressources en ligne.
— Un forum de questions/réponses `http://ask.sagemath.org/`.
— Une liste de discussion `https://groups.google.com/forum/#!forum/sage-support`.

5.5 Optimisation linéaire et problème SAT

Nous allons voir dans cette section comment utiliser les optimiseurs linéaires et les solveurs SAT dans Sage. En particulier, on verra comment ils peuvent permettre de résoudre simplement (du point de vue du programmeur) certains algorithmes sur les graphes.

5.5.1 Optimisation linéaire

Un problème d'optimisation (linéaire) est un problème qui consiste à trouver le maximum d'une fonction $f(x) = a_1x_1 + \ldots + a_dx_d$ sous des

contraintes de la forme

$$b_1x_1 + b_2x_2 + \ldots + b_dx_d + c \leqslant 0.$$

Optimisation linéaire avec Sage. Prenons l'exemple suivant : optimiser $f(x_0, x_1, x_2) = 3x_0 + 2x_1 + x_2$ sous les contraintes

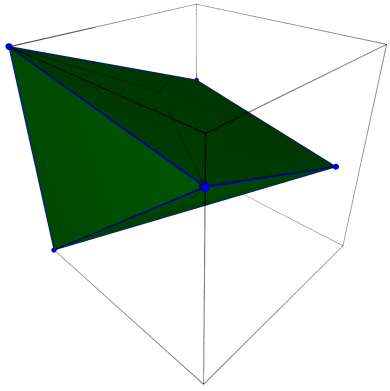
$$\begin{aligned} x_0 \geqslant 0, \quad x_1 \geqslant 0, \quad x_2 \geqslant 0, \\ 7x_0 - x_1 - x_2 \leqslant 0, \\ 5x_1 - 3x_2 \leqslant 0, \\ 9x_0 + 8x_1 + 7x_2 - 27 \leqslant 0. \end{aligned}$$

Dans Sage, ce problème se traduit de la manière suivante

```
sage: M = MixedIntegerLinearProgram()
sage: x = M.new_variable(nonnegative=True)
sage: M.set_objective(3*x[0] + 2*x[1] + x[2])
sage: M.add_constraint(7*x[0] - x[1] - x[2] <= 0)
sage: M.add_constraint(5*x[1] - 3*x[2] <= 0)
sage: M.add_constraint(9*x[0] + 8*x[1] + 7*x[2] - 27 <= 0)
```

L'ensemble des contraintes forme un polyèdre de $\mathbb{R}^3$ que l'on peut dessiner via les commandes suivantes

```
sage: P = M.polyhedron()
sage: P.plot()
```



Pour résoudre le problème `M` et récupérer le point pour lequel la fonction est maximisée on utilise

```
sage: M.solve()         # résolution (renvoie l'optimum)
5.62268041237
sage: vals = M.get_values(x)  # valeurs des variables
sage: vals
{0: 0.4453608247422578, 1: 1.169072164949047, 2: 1.9484536082467576}
```

La valeur de retour de la méthode `get_values` est un dictionnaire (une association clé-valeur) : les clés sont `1`, `2` et `3` et on y accède via les crochets comme avec une liste.

```
sage: vals[1]
1.169072164949047
```

Remarquons que l'optimum est bien un des sommets de `P`

```
sage: P.vertices()
(A vertex at (0.0, 0.0, 0.0),
 A vertex at (0.4453608247, 1.169072165, 1.948453608),
 A vertex at (0.4655172414, 0.0, 3.25862069),
 A vertex at (0.0, 1.372881356, 2.288135593),
 A vertex at (0.0, 0.0, 3.857142857))
```

Dans sa version ci-dessus, le problème d'optimisation linéaire est résolu avec des nombres flottants. Il est également possible de résoudre ce même problème avec des nombres rationnels (exacts). La seule chose à changer ci-dessus est de remplacer la ligne `M = MixedIntegerLinearProgram()` par `M = MixedIntegerLinearProgram(solver='PPL')`. Dans ce cas, la résolution du problème donnera

```
sage: M.solve()
2727/485
sage: numerical_approx(_)
5.62268041237113
sage: M.get_values(x)
{0: 216/485, 1: 567/485, 2: 189/97}
sage: [v.numerical_approx() for v in M.get_values(x).values()]
[0.445360824742268, 1.16907216494845, 1.94845360824742]
```

Notez l'utilisation de la variable spéciale `_` qui correspond au résultat de la commande qui précède.

Il est également possible de faire une recherche d'optimum sur les points entiers. Dans ce cas là, il suffit de remplacer la ligne `x = M.new_variable(nonnegative=True)` par la ligne `x = M.new_variable(nonnegative=True, integer=True)`. On obtient alors

```
sage: M.solve()
4.0
sage: M.get_values(x)
{0: 0.0, 1: 1.0, 2: 2.0}
```

Notez que résoudre un problème linéaire a une complexité polynomiale (à dimension fixée). Cependant, le problème 3-SAT se réduit (polynomialement) à l'existence de points entiers dans un polytope. La recherche d'optimum sur les points entiers est donc un problème NP-complet.

Exemple d'optimisation combinatoire via de l'optimisation entière. On cherche à résoudre le problème suivant : étant donné un graphe, trouver un ensemble de sommets de cardinalité minimum tel que chaque arête est adjacente à un point de cet ensemble (un tel ensemble est dit *couvrant*).

En prenant une variable x_v pour chaque sommet, ce problème peut se traduire par la minimisation de $\sum_{v \text{ sommet}} x_v$ sous les contraintes suivantes

$$\begin{aligned} & x_v \in \{0,1\} \\ & x_u + x_v \geqslant 1 \quad \text{pour chaque arête } (u,v) \end{aligned}$$

On peut alors vérifier que le graphe de Petersen possède un ensemble couvrant de taille 6.

```
sage: G = graphs.PetersenGraph()
sage: M = MixedIntegerLinearProgram(maximization=False)
sage: x = M.new_variable(nonnegative=True, binary=True)
sage: M.set_objective(sum(x[v] for v in G))
sage: for (u,v) in G.edges(labels=False):
....:     M.add_constraint(x[u] + x[v] >= 1)
sage: M.solve()
6.0
```

Ainsi le graphe de Petersen a un ensemble couvrant de cardinalité 6 (et il n'y en a pas de plus petit). Notez l'utilisation des boucles `for` dans le code ci-dessus pour itérer sur les sommets du graphe dans `sum(x[v] for v in G)` et sur ses arêtes dans `for (u,v) in G.edges(labels=False)`.

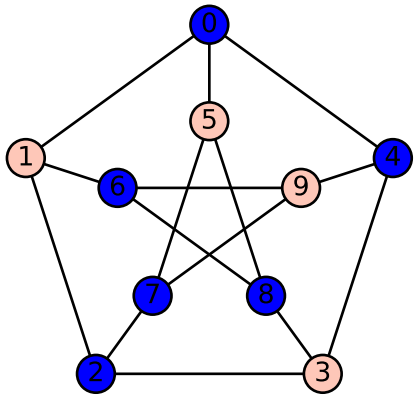
Pour récupérer l'ensemble des sommets de l'ensemble couvrant on utilise comme précédemment la méthode `get_values`.

```
sage: S = [i for (i,j) in M.get_values(x).items() if j]
sage: S
[0, 2, 4, 6, 7, 8]
```

La méthode `items` d'un dictionnaire permet de faire une itération sur les paires clé-valeur.

Vous pouvez alors visualiser la solution avec

```
sage: G.plot(vertex_colors={'blue': S})
```



Exercice : Un ensemble indépendant d'un graphe est un sous-ensemble de sommets qui ne contient aucune paire de sommets adjacents. Calculer un ensemble indépendant du graphe de Petersen de cardinalité maximale.

5.5.2 Solveurs SAT

Interface Sage aux solveurs SAT. Dans Sage, il est possible d'utiliser des solveurs SAT. Considérons la clause logique suivante

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_2)$$

Créer le problème correspondant à ϕ avec Sage se fait de la manière suivante

```
sage: S = SAT()
sage: S.add_clause((1,  -2,  3))
sage: S.add_clause((1, 2, -3))
sage: S.add_clause((2, 3))
sage: S.add_clause((-1, -3))
sage: S.add_clause((1, -2))
```

Notez que les variables sont encodées par des entiers positifs et qu'on utilise $-i$ pour la négation de la i-ième variable. On peut alors résoudre le problème via

```
sage: solver()
(None, True, True, False)
```

La valeur en position i du tuple indique la valeur de x_i. Comme les indices des tuples commencent à 0 et les variables du problème sont indicées par $\{1, 2, \ldots\}$ on peut tout simplement ignorer la première valeur `None`. Ainsi, la solution `(None, True, True, False)` correspond à

$$x_1 = \text{True} \quad x_2 = \text{True} \qquad x_3 = \text{False}.$$

Exemple d'utilisation de SAT en combinatoire. Nous illustrons maintenant le problème de coloration des graphes en formulant une clause SAT. Considérons un graphe $G = (V, E)$ et un entier positif k. Nous voulons savoir s'il existe un k-coloriage de G, c'est à dire une fonction $f : V \to \{1, 2, \ldots, k\}$ telle que pour chaque arête (u, v) de G on ait $f(u) \neq f(v)$.

Nous construisons une instance SAT associé à ce problème de la façon suivante. On considère des variables $x_{u,c}$ pour chaque paire consistante d'un sommet $u \in V$ et d'une couleur $c \in \{1, \ldots, k\}$. On considère alors deux familles de clauses

$$\forall u \in V, \qquad \phi_u^{(k)} := \quad x_{u,1} \vee x_{u,2} \vee \ldots x_{u,k} \tag{5.1}$$

$$\forall (u, v) \in E, \forall i \in \{1, \ldots, k\} \qquad \psi_{u,v,i}^{(k)} := \neg x_{u,i} \vee \neg x_{v,i} \tag{5.2}$$

et leur conjonction

$$\theta^{(k)} := \bigwedge_{u \in V} \phi_u^{(k)} \wedge \bigwedge_{(u,v) \in E, i \in \{1,\ldots,k\}} \psi_{u,v,i}^{(k)}$$

Exercice : Le graphe G est k-coloriable si et seulement si la formule $\theta^{(k)}$ est satisfiable.

On peut alors vérifier que le graphe de Petersen est 3-coloriable (les sommes du graphe sont les entiers $\{0, 1, \ldots, 9\}$ et les couleurs par $\{0, 1, \ldots, k-1\}$ et on encode $x_{u,i}$ par $u + 10i + 1$)

```
sage: G = graphs.PetersenGraph()
sage: solver = SAT()
sage: k = 3
....: for u in G.vertices():
....:     solver.add_clause([u + 10*i + 1 for i in range(k)])
```

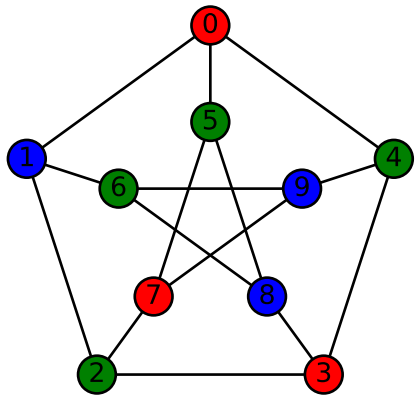
```
sage: for u,v in G.edges(False):
....:     for i in range(k):
....:         solver.add_clause([-(u + 10*i + 1), -(v + 10*i + 1)])
sage: ans = solver()
sage: ans
(None, True, False, False, ...,  False, True, True)
```

On peut convertir le résultat `ans` en une partition (chaque élément de la partition représentant une couleur)

```
sage: p = [[], [], []]
sage: for u in range(10):
....:     for i in range(k):
....:         if ans[u + 10*i + 1]:
....:             p[col].append(u)
sage: p
[[0, 3, 7], [2, 4, 5, 6], [1, 8, 9]]
```

Et obtenir une image de la coloration.

```
sage: G.plot(vertex_colors={'red': p[0], 'green': p[1], 'blue': p[2]})
```



5.6 Les polytopes

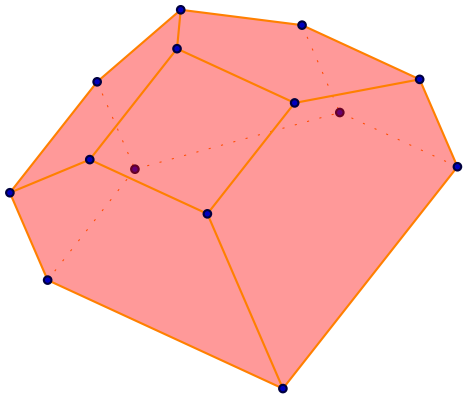
Proche des problèmes d'optimisation, Sage possède de nombreuses primitives pour les calculs sur les polytopes. Pour créer un polytope comme l'enveloppe convexe d'un nombre fini de points, on utilise la commande `Polyhedron`.

```
sage: P = Polyhedron(vertices=[[1,0,1],[1,0,0],[1,1,0],
....:        [0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],
....:        [0,-1,0]]).polar()
sage: P
A 3-dimensional polyhedron in QQ\^{}3 defined
as the convex hull of 14 vertices
```

Vous pouvez ensuite naviguer entre les propriétés du polytope que vous venez de construire en utilisant la complétion automatique.

```
sage: P.<tab>
        P.adjacency_matrix                P.base_extend
        P.affine_hull                     P.base_ring
        P.ambient_dim                     P.bipyramid
        P.ambient_space                   P.bounded_edges
        P.barycentric_subdivision         P.bounding_box
       ...
```

On peut par exemple faire de jolies images :



Ou convertir le polytope en contraintes d'un problème d'optimisation

```
sage: P.to_linear_program()
Linear Program (no objective, 3 variables, 9 constraints)
```

Notez qu'un polytope peut-aussi être défini à partir de sa liste de faces (des hyperplans).

```
sage: Polyhedron(ieqs=[[1,1,0],[1,-1,0],[1,0,1],[1,0,-1]])
A 2-dimensional polyhedron in QQ^2 defined
as the convex hull of 4 vertices
```

Quel est le polytope construit par la commande précédente ?

5.7 Génération aléatoire

Nous proposons maintenant quelques expérimentations autour des probabilités. Un des objectifs est de construire des générateurs aléatoires d'objets combinatoires à partir de primitives très simples.

5.7.1 Premières manipulations

Nous utiliserons seulement des fonctions de génération quasi-aléatoires élémentaires suivantes

— `random()` : un flottant uniforme dans $[0, 1]$

— `randint(i,j)` : un entier uniforme dans $\{i, i+1, \ldots, j\}$

Pour faire le lien avec les variables aléatoires du Chapitre 1 vous devez considérer que des appels successifs à la fonction `random()` est une *réalisation* d'une suite de variables indépendantes uniforme.

Une construction pratique de Python est la *liste en compréhension*. C'est ce qui correspond en mathématiques à la notation ensembliste

$$S = \{x : \ x \in \mathbb{N}, x^2 > 3\}.$$

Voici l'exemple des nombres premiers < 20 et d'un échantillon de loi uniforme

```
sage: l = [n for n in range(1, 20) if is_prime(n)]
sage: l = [random() for _ in range(10)]
```

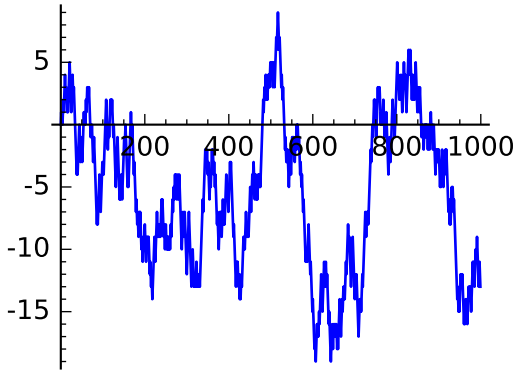
Une autre manière de construire des listes est via la méthode `append` (qui ajoute un élément à une liste). La petite boucle suivante simule une marche aléatoire de 1000 pas sur $\mathbb{Z}$ dont chaque pas consiste à choisir entre $+1$ et -1 avec probabilité $1/2$

```
sage: x = 0   # position courante
sage: W = []  # marche
sage: for i in range(1000):
....:     W.append(x)
....:     x = x + 1 - 2*randint(0,1)
```

Notez la boucle avec `range` qui permet de répéter 1000 fois une opération.

On peut visualiser la suite `W` en utilisant la commande

```
sage: list_plot(W, plotjoined=True)
```



Exercice :

1. On peut définir les nombres binomiaux $B(n,m) = \frac{n!}{m!(n-m)!}$ par l'équation
$$(1+x)^n = \sum_{m=0}^{n} B(n,m)x^m. \tag{5.3}$$
2. En utilisant la relation $(1+x)^n = (1+x)(1+x)^{n-1}$, montrer que les nombres binomiaux vérifient la récurrence
$$B(n,m) = B(n-1,m) + B(n-1,m-1). \tag{5.4}$$
3. Quel est le rapport entre les nombres binomiaux et la marche aléatoire sur $\mathbb{Z}$?

Pour fabriquer les listes des nombres binomiaux $B(n,m)$ à n fixé, la récurrence (5.4) se traduit en des boucles imbriquées

```
sage: B = [1]   # liste pour n=0
sage: for n in range(1, 5):
....:     B.append(1)
....:     for i in range(n-1, 0, -1):
....:         B[i] = B[i] + B[i-1]
....:     print(B)
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
```

Notez la fonction `range(n - 1, 0, -1)` qui permet de faire une boucle de `n - 1` (inclus) à `0` (exclus) avec un pas `-1`.

Exercice : (nombres binomiaux, nombres de Stirling et nombres eulériens)

1. Pour les valeurs de $n = 3, 5, 10, 30, 100$ faire un graphique de la suite $m \mapsto B(n, m)$ (utiliser `list_plot` comme on l'a fait précédemment et possiblement `graphics_array` pour inclure plusieurs graphiques dans une image).
2. Les nombres de Stirling (de première espèce non signés) $S(n, m)$ sont définis par l'équation

$$x^{(n)} := x(x+1)(x+2)\dots(x+n-1) = \sum_{m=0}^{n} S(n,m)x^m. \quad (5.5)$$

3. En utilisant l'équation $x^{(n)} = (x+n)x^{(n-1)}$, montrer que les nombres de Stirling satisfont la récurrence

$$S(n,m) = (n-1)S(n-1,m) + S(n-1,m-1). \quad (5.6)$$

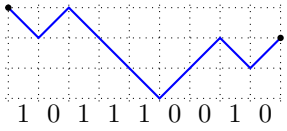
4. Modifier le programme pour les binomiaux pour qu'il génère les nombres de Stirling.
5. Quel est le lien entre les nombres de Stirling et les permutations ?
6. Pour les valeurs de $n = 3, 5, 10, 30, 100$ faire un graphique de la suite $m \mapsto S(n, m)$.
7. Montrer que les nombres eulériens $A(n, m)$ (voir Chapitre 1) vérifient

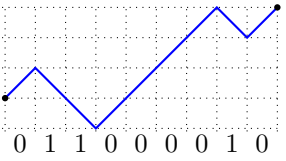
$$A(n,m) = (m+1)A(n-1,m) + (n-m)A(n-1,m-1). \quad (5.7)$$

8. Pouvez-vous donner une "formule" pour les nombres eulériens de la même forme que (5.3) ou (5.5) ?
9. Utiliser la récurrence (5.7) pour engendrer les nombres eulériens.
10. Pour les valeurs de $n = 3, 5, 10, 30, 100$ faire un graphique de la suite $m \mapsto A(n, m)$.
11. Que constatez-vous ? Quels sont les liens entre les graphiques de $B(n, m)$, $S(n, m)$ et $A(n, m)$?

5.7.2 Excursion brownienne discrète

On se propose maintenant d'implanter la génération aléatoire d'analogues discrets de l'excursion brownienne. On considère des marches aléatoires sur $\mathbb{Z}$ représentées par des mots finis de $\{0,1\}^*$ (0 indique un pas montant et 1 un pas descendant). Plus formellement, la marche associée à $s = s_1 s_2 \dots s_n \in \{0,1\}^n$ est la suite d'entier $(w_0, w_1, \dots, w_n) \in \mathbb{Z}^{n+1} = 0$ définie par $w_0 = 0$ et pour tout $i \in \{1, \dots, n\}$ $w_i = w_{i-1} + 1$ si $s_i = 0$ et $w_i = w_{i-1} - 1$ si $s_i = 1$. On pourra voir deux exemples de mots s ainsi que les marches associées ci-dessous.





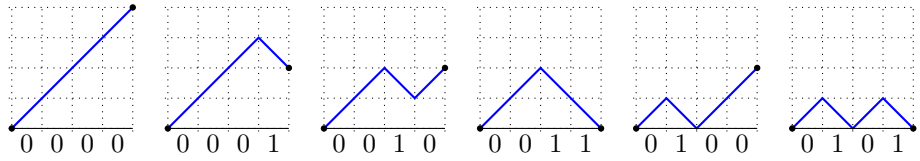
Étant donné un mot s dans $\{0,1\}^n$, le *vecteur d'évaluation* de s est le vecteur $(|s|_0, |s|_1)$ des nombres d'occurrences de 0 et 1 dans w. En reprenant les exemples ci-dessus, les vecteurs d'évaluation de 101110010 et 011000010 sont respectivement $(4,5)$ et $(6,3)$. On remarquera que le point final de la marche w_n est égal à $|s|_0 - |s|_1$.

Exercice (génération aléatoire à évaluation fixée) : Nous proposons d'implanter une chaîne de Markov non-homogène pour générer uniformément un mot de $\{0,1\}^*$ dont le vecteur d'évaluation (n_0, n_1) est donné. Pour cela, on choisit la première lettre avec une loi de Bernoulli de paramètre $p = n_1/(n_0 + n_1)$. Si la première lettre est un 0 alors on recommence avec le nouveau vecteur d'évaluation $(n_0 - 1, n_1)$ sinon, on recommence à partir de $(n_0, n_1 - 1)$.

1. Justifier cette méthode.
2. Programmer la dans Sage.
3. Faire des graphique pour les évaluations $(30,30)$, $(30,10)$ et $(1000,1000)$.
4. Décrire la renormalisation appropriée pour qu'on ait une convergence en loi vers l'excursion brownienne.
5. Refaire le graphique pour l'évaluation $(1000,1000)$ à l'échelle de l'excursion brownienne.

Une marche est dite positive si le mot associé s de $\{0,1\}^n$ est tel que pour tout préfixe u de s on ait $|u|_0 \geqslant |u|_1$. Autrement dit, toutes les étapes de la marche w sont $\geqslant 0$. On note $\mathcal{P}_n \subset \{0,1\}^n$ l'ensemble des marches

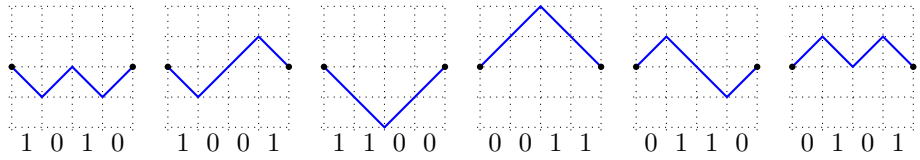
positives de longueur n et $\mathcal{P}_n^{(k)}$ celles parmi $\mathcal{P}_n$ qui finissent en k. Ci-dessous sont dessinées les 6 marches positives de longueur 4.



Sur ces exemple, les coordonnées d'arrivée sont respectivement 4, 2, 2, 0, 2 et 0.

On dit qu'une marche est *équilibrée* si le mot s vérifie $|s|_0 = |s|_1$. Autrement dit, il s'agit d'un élément dont l'évaluation est (m, m) pour un certain entier $m \geqslant 0$. Ou encore, ce sont les marches qui terminent en $w_{2m} = 0$.

On note $\mathcal{B}_{2m}^{(k)} \subset \{0,1\}^{2m}$ les mots équilibrés de $\{0,1\}^{2m}$ pour lesquels la marche correspondante possède exactement k pas descendants de 0 vers -1. Ci-dessous sont dessinées les 6 marches équilibrées de longueur 4.



Sur ces exemples, les nombre de descentes de 0 vers -1 sont respectivement 2, 1, 1, 0, 1, 0.

Un *mot de Dyck* est un élément de $\mathcal{P}_{2m} \cap \mathcal{B}_{2m}$. Finalement, étant donné un mot $u \in \{0,1\}^*$ on note $\overline{u}$ le mot consistant à échanger les lettres 0 et 1.

Exercice (bijection $\mathcal{P}_{2m}^{2k} \simeq \mathcal{B}_{2m}^{(k)}$) :

1. Montrer que tout mot w de $\mathcal{P}_{2m}^{(2k)}$ se décompose de manière unique en $w = d_0 1 d_1 1 d_2 1 \ldots 1 d_2 k$ où les facteurs d_i sont des mots de Dyck.
2. À un mot w de $\mathcal{P}_{2m}^{(2k)}$ décomposé en $w = d_0 1 d_1 1 \ldots 1 d_{2k}$ comme dans la question précédente, on associe $\phi(w) = d_0 0 \overline{d_1} 1 d_2 0 \ldots 0 \overline{d_{2k-1}} 1 d_{2k}$.
3. Montrer que ϕ induit une bijection de $\mathcal{P}_{2m}^{(2k)}$ vers $\mathcal{B}_{2m}^{(k)}$ et implanter cette fonction dans Sage.
4. Décrire la bijection inverse et l'implanter dans Sage.

Exercice : (génération des marches positives par rejet) On s'intéresse à une autre méthode de génération aléatoire uniforme de marches positives. On va générer des marches de taille n et faire du rejet anticipé lorsqu'elle n'est pas positive. Plus précisément, on génère des bits aléatoires les uns après les autres. Si la marche associée arrive en -1 on efface tout et on recommence. On s'arrête lorsque la longueur de la suite est n.

1. Implanter cette méthode de génération aléatoire pour $\mathcal{P}_n$.
2. Donner une génération aléatoire alternative pour $\mathcal{B}_{2m}$ en utilisant la génération aléatoire de $\mathcal{P}_{2m}$ et la bijection de l'exercice précédent.
3. Déterminer la moyenne du nombre de bits aléatoires nécessaires pour chacune des deux méthodes. Quelle méthode est la plus économe en aléa ?

Bibliographie

[1] W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaudon, V. Soumelis, C. Chaouiya, and D. Thieffry. Model checking to assess t-helper cell plasticity. *Frontiers in Bioengineering and Biotechnology*, 2, 2015.

[2] D. Aldous. Brownian excursions, critical random graphs and the multiplicative coalescent. *Ann. Probab.*, 25(2):812–854, 1997.

[3] D. J. Aldous. Exchangeability and related topics. In *École d'été de probabilités de Saint-Flour, XIII—1983*, volume 1117 of *Lecture Notes in Math.*, pages 1–198. Springer, Berlin, 1985.

[4] D. J. Aldous and J. Pitman. Brownian bridge asymptotics for random mappings. *Random Structures Algorithms*, 5(4):487–512, 1994.

[5] A. D. Alexandrov. *Intrinsic geometry of convex surfaces (translation of the 1948 Russian original)*. CRC Press, 2005.

[6] L. Ambrosio and N. Gigli. A user's guide to optimal transport. *Modelling and Optimisation of Flows on Networks, Lecture Notes in Mathematics*, pages 1–155, 2013.

[7] J. Aracena. Maximum number of fixed points in regulatory Boolean networks. *Bulletin of Mathematical Biology*, 70(5):1398–1409, 2008.

[8] J. Aracena, J. Demongeot, and E. Goles. Positive and negative circuits in discrete neural networks. *IEEE Transactions of Neural Networks*, 15:77–83, 2004.

[9] J. Aracena, E. Goles, A. Moreira, and L. Salinas. On the robustness of update schedules in Boolean networks. *Biosystems*, 97(1):1 – 8, 2009.

[10] J. Aracena, A. Richard, and L. Salinas. Number of fixed points and disjoint cycles in monotone boolean networks. *SIAM Journal on Discrete Mathematics*, 31(3):1702–1725, 2017.

[11] R. Arratia, A. D. Barbour, and S. Tavaré. On random polynomials over finite fields. *Math. Proc. Cambridge Philos. Soc.*, 114(2):347–368, 1993.

[12] R. Arratia, A. D. Barbour, and S. Tavaré. *Logarithmic combinatorial structures: a probabilistic approach*. European Mathematical Society (EMS), Zürich, 2003.

[13] G. Audemard, B. Benhamou, and P. Siegel. La méthode d'avalanche AVAL : une méthode énumérative pour SAT. In *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC)*, pages 17–25, 1999.

[14] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In *SAT'2008*, 2008.

[15] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009.

[16] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM J. Comput.*, 16(1):78–96, 1987.

[17] F. Aurenhammer, F. Hoffmann, and B. Aronov. Minkowski-type theorems and least-squares partitioning. In *Symposium on Computational Geometry*, pages 350–357, 1992.

[18] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *SAT'2003*, pages 341–355, 2003.

[19] G. V. Bard. *Sage for Undergraduates*. American Mathematical Society, Providence, RI, 2015.

[20] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in sat modulo theories. In *Logic for Programming, Artificial Intelligence, and Reasoning: 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006. Proceedings*, 2006.

[21] J.-D. Benamou and Y. Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[22] J.-D. Benamou, G. Carlier, Q. Mérigot, and E. Oudet. Discretization of functionals involving the Monge-Ampère operator. *arXiv*, Aug. 2014. [math.NA] http://arxiv.org/abs/1408.4336.

[23] J. Bertoin and J. Pitman. Path transformations connecting Brownian bridge, excursion and meander. *Bull. Sci. Math.*, 118(2):147–166, 1994.

[24] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds, 1999.

[25] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[26] P. Billingsley. On the distribution of large prime divisors. *Period. Math. Hungar.*, 2:283–289, 1972.

[27] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. T. Silva, M. Tarini, and D. Zorin. Quad-mesh generation and processing: A survey. *Comput. Graph. Forum*, 32(6):51–76, 2013.

[28] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for nelson-oppen and rewrite-based decision procedures. In *Proceedings of the Third International Joint Conference on Automated Reasoning*, IJCAR'06, Berlin, Heidelberg, 2006.

[29] A. Boneh and M. Hofri. The coupon-collector problem revisited—a survey of engineering problems and computational methods. *Comm. Statist. Stochastic Models*, 13(1):39–66, 1997.

[30] N. Bonneel, M. van de Panne, S. Paris, and W. Heidrich. Displacement interpolation using Lagrangian mass transport. *ACM Trans. Graph.*, 30(6):158, 2011.

[31] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing*. A K Peters, 2010.

[32] A. Bowyer. Computing Dirichlet tessellations. *Comput. J.*, 24(2):162–166, 1981.

[33] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Computer Aided Verification, 17th International Conference, Edinburgh, Scotland, UK, July 6-10*, pages 335–349, 2005.

[34] A. R. Bradley and Z. Manna. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media, 2007.

[35] Y. Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on Pure and Applied Mathematics*, 44:375–417, 1991.

[36] Y. Brenier, U. Frisch, M. Henon, G. Loeper, S. Matarrese, R. Mohayaee, and A. Sobolevskii. Reconstruction of the early universe as a convex optimization problem. *arXiv*, September 2003. arXiv:astro-ph/0304214v3.

[37] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: a comparative analysis. *Ann. Math. Artif. Intell.*, 55(1-2):63–99, 2009.

[38] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.

[39] M. Buro and H. K. Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.

[40] L. Caffarelli. The Monge-Ampère equation and optimal transportation, an elementary review. *Optimal transportation and applications (Martina Franca, 2001), Lecture Notes in Mathematics*, pages 1–10, 2003.

[41] A. Casamayou. *Calcul mathématique avec Sage*. 2013.

[42] P. Chassaing and P. Flajolet. Hachage, arbres, chemins & graphes. *Gazette des Mathématiciens*, 95:29–49, January 2003.

[43] P. Chassaing and G. Louchard. Phase transition for parking blocks, Brownian excursion and coalescence. *Random Structures Algorithms*, 21(1):76–119, 2002.

[44] P. Chassaing and J.-F. Marckert. Parking functions, empirical processes, and the width of rooted labeled trees. *Electron. J. Combin.*, 8(1):Research Paper 14, 19, 2001.

[45] B. Chauvin, T. Klein, J.-F. Marckert, and A. Rouault. Martingales and profile of binary search trees. *Electron. J. Probab.*, 10:no. 12, 420–435, 2005.

[46] B. Chauvin and A. Rouault. Connecting Yule process, bisection and binary search tree via martingales. *Journal of the Iranian Statistical Society*, 3, 2004.

[47] A. Choi, N. Zaitlen, B. Han, K. Pipatsrisawat, A. Darwiche, and E. Eskin. Efficient genome wide tagging by reduction to sat. In K. Crandall and J. Lagergren, editors, *Algorithms in Bioinformatics*, volume 5251 of *Lecture Notes in Computer Science*, pages 135–147. Springer Berlin / Heidelberg, 2008.

[48] D. P. A. Cohen, L. Martignetti, S. Robine, E. Barillot, A. Zinovyev, and L. Calzone. Mathematical modelling of molecular pathways enabling tumour cell invasion and migration. *PLoS Comput Biol*, 11(11):e1004571, Nov 2015.

[49] S. Collombet, C. van Oevelen, J. L. Sardina Ortega, W. Abou-Jaoudé, B. Di Stefano, M. Thomas-Chollier, T. Graf, and D. Thieffry. Logical modeling of lymphoid and myeloid cell specification and transdifferentiation. *Proceedings of the National Academy of Sciences*, 114(23):5792–5799, 2017.

[50] S. Conchon and S. Krstić. Strategies for combining decision procedures. *Theoretical Computer Science*, 354(2):187–210, 2006.

[51] S. Cook. The complexity of theorem proving procedures. In *ACM Symposium of Theory of Computing*, pages 151–158, 1971.

[52] F. Corblin, E. Fanchon, L. Trilling, C. Chaouiya, and D. Thieffry. *Automatic Inference of Regulatory and Dynamical Properties from Incomplete Gene Interaction and Expression Data*, pages 25–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[53] I. Crespo, T. M. Perumal, W. Jurkowski, and A. del Sol. Detecting cellular reprogramming determinants by differential stability analysis of gene regulatory networks. *BMC Syst Biol*, 7(1):140, 2013.

[54] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2292–2300, 2013.

[55] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *JACM*, 5:394–397, 1962.

[56] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7:201–215, 1960.

[57] F. de Goes, C. Wallez, J. Huang, D. Pavlov, and M. Desbrun. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.*, 34(4):50:1–50:11, 2015.

[58] L. M. de Moura and N. Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[59] L. M. de Moura and D. Jovanovic. A model-constructing satisfiability calculus. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference,Rome, Italy, January 20-22*, pages 1–12, 2013.

[60] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure. In *Proc. of the 4th Int'l Conf. on Principles of KR&R*, pages 134–145, 1994.

[61] L. Devroye. A note on the height of binary search trees. *J. Assoc. Comput. Mach.*, 33(3):489–498, 1986.

[62] L. Devroye. Méthodes probabilistes dans l'étude de certaines classes d'arbres aléatoires. Cours donné aux Journées ALEA 2004, CIRM (Luminy, France), 19 - 23 Janvier, 2004.

[63] L. Devroye. Applications of Stein's method in the analysis of random binary search trees. In *Stein's method and applications*, volume 5 of *Lect. Notes Ser. Inst. Math. Sci. Natl. Univ. Singap.*, pages 247–297. Singapore Univ. Press, Singapore, 2005.

[64] Dimacs. Second challenge on satisfiability testing organized by the center for discrete mathematics and computer science of rutgers university. 1993.

[65] M. Drmota. An analytic approach to the height of binary search trees. *Algorithmica*, 29(1-2):89–119, 2001.

[66] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In *Proceedings of IJCAI'2001*, pages 248–253, 2001.

[67] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA*, pages 81–94, 2006.

[68] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[69] N. Eén and N. Sörenson. An extensible sat-solver. In *SAT'2003*, pages 333–336, 2003.

[70] P. Erdös. On a lemma of Littlewood and Offord. *Bulletin of the American Mathematical Society*, 51(12):898–902, 1945.

[71] L. F. Fitime, O. Roux, C. Guziolowski, and L. Paulevé. Identification of bifurcation transitions in biological regulatory networks using Answer-Set Programming. *Algorithms for Molecular Biology*, 12(1):19, 2017.

[72] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3):207–229, 1992.

[73] P. Flajolet, P. Poblete, and A. Viola. On the analysis of linear probing hashing. *Algorithmica*, 22(4):490–515, 1998. Average-case analysis of algorithms.

[74] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[75] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[76] D. Foata. Distributions eulériennes et mahoniennes sur le groupe des permutations. In *Higher combinatorics (Proc. NATO Advanced Study*

Inst., Berlin, 1976), volume 31 of *NATO Adv. Study Inst. Ser., Ser. C: Math. Phys. Sci.*, pages 27–49. Reidel, Dordrecht-Boston, Mass., 1977. With a comment by Richard P. Stanley.

[77] M. Folschette, L. Paulevé, M. Magnin, and O. Roux. Sufficient conditions for reachability in automata networks with priorities. *Theoretical Computer Science*, 608, Part 1, From Computer Science to Biology and Back:66 – 83, 2015.

[78] M. Gadouleau, A. Richard, and E. Fanchon. Reduction and fixed points of boolean networks and linear network coding solvability. *IEEE Transactions on Information Theory*, 62(5):2504–2519, 2016.

[79] M. Gadouleau, A. Richard, and S. Riis. Fixed points of Boolean networks, guessing graphs, and coding theory. *SIAM Journal on Discrete Mathematics*, 29(4):2312–2335, 2015.

[80] T. O. Gallouët and Q. Mérigot. A Lagrangian scheme à la Brenier for the incompressible euler equations. *Foundations of Computational Mathematics*, May 2017.

[81] W. Gangbo and R. J. McCann. The geometry of optimal transportation. *Acta Math.*, 177(2):113–161, 1996.

[82] M. Garey and D. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[83] L. Glass and S. Kauffman. Logical analysis of continuous, non-linear biochemical control networks. *Journal of Theoretical Biology*, 39(1):103–129, 1973.

[84] A. Goldberg. On the Complexity of the Satisfiability Problem. Technical Report 16, New York University, 1979.

[85] E. Goles and L. Gómez. Combinatorial game associated to the one dimensional schelling's model of social segregation. *Natural Computing*, Mar 2017.

[86] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[87] S. J. Gortler, C. Gotsman, and D. Thurston. Discrete one-forms on meshes and applications to 3D mesh parameterization. *Computer Aided Geometric Design*, 23(2):83–112, 2006.

[88] R. Graham and N. Sloane. Lower bounds for constant weight codes. *IEEE Transactions on Information Theory*, 26(1):37–43, 1980.

[89] A. Griggio. A practical approach to satisability modulo linear integer arithmetic. *JSAT*, 8(1/2):1–27, 2012.

[90] V. Guillemin and A. Pollack. *Differential topology*. Englewood Cliffs, N.J. Prentice-Hall, 1974.

[91] Y. Hamadi, S. Jabbour, and L. Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2009.

[92] F. Harary. On the notion of balance of a signed graph. *The Michigan Mathematical Journal*, 2(2):143–146, 1953.

[93] M. Heule and H. Van Maaren. Effective incorporation of double look-ahead procedures. In *SAT'07: Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, pages 258–271, Berlin, Heidelberg, 2007. Springer-Verlag.

[94] W. Hodges. *A shorter model theory*. Cambridge University Press, New York, NY, USA, 1997.

[95] L. Holst. On birthday, collectors', occupancy and other classical urn problems. *Internat. Statist. Rev.*, 54(1):15–27, 1986.

[96] K. Hormann and M. S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25(4):1424–1441, 2006.

[97] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: theory and practice video files associated with this course are available from the citation page. In *34. International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2007, San Diego, California, USA, August 5-9, 2007, Courses*, page 1, 2007.

[98] J. Huang. The effect of restarts on the efficiency of clause learning. In *IJCAI'2007*, pages 2318–2323, 2007.

[99] F. Jacob and J. Monod. On the regulation of gene activity. *Cold Spring Harbor Symposia on Quantitative Biology*, 26:193–211, 1961.

[100] M. Järvisalo, A. Biere, and M. Heule. Blocked clause elimination. In *TACAS*, pages 129–144, 2010.

[101] R. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, pages 167–187, 1990.

[102] D. Jovanovic and L. M. de Moura. Cutting to the chase - solving linear integer arithmetic. *J. Autom. Reasoning*, 51(1):79–108, 2013.

[103] O. Kallenberg. *Foundations of modern probability*. Probability and its Applications (New York). Springer-Verlag, New York, second edition, 2002.

[104] H. Katebi, K. A. Sakallah, and I. L. Markov. Symmetry and satisfiability: An update. In *SAT*, pages 113–127, 2010.

[105] S. A. Kauffman. Metabolic stability and epigenesis in randomly connected nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[106] J. F. C. Kingman. Random discrete distribution. *J. Roy. Statist. Soc. Ser. B*, 37:1–22, 1975.

[107] J. F. C. Kingman. The population structure associated with the Ewens sampling formula. *Theoret. Population Biology*, 11(2):274–283, 1977.

[108] J. Kitagawa, Q. Mérigot, and B. Thibert. A newton algorithm for semi-discrete optimal transport. *CoRR*, abs/1603.05579, 2016.

[109] B. Knaster and A. Tarski. Un théoreme sur les fonctions d'ensembles. *Ann. Soc. Polon. Math*, 6(133):2013134, 1928.

[110] D. E. Knuth. *The art of computer programming. Vol. 3*. Addison-Wesley, Reading, MA, 1998. Sorting and searching, Second edition.

[111] D. Kroening and O. Strichman. *Decision procedures*, volume 5. Springer, 2008.

[112] S. Krstić and S. Conchon. Canonization for disjoint unions of theories. *Information and Computation*, 199(1-2):87–106, May 2005.

[113] R. Lassaigne and M. de Rougemont. *Logique et complexité*. Hermès, 1996. collection informatique.

[114] D. Le Berre, O. Roussel, and O. Roussel. The sat competitions (2002–2009). www.satcompetition.org, 2002-2009.

[115] D. Le Berre, O. Roussel, and L. Simon. SAT competition, 2009. http://www.satcompetition.org/.

[116] D. LeBerre and L. Simon, editors. *Special Volume on the SAT 2005 competitions and evaluations*, volume 2. Journal on Satisfiability, Boolean Modeling and Computation, 2006.

[117] C. Leonard. A survey of the Schrödinger problem and some of its connections with optimal transport. *arXiv*, August 2013. [math.PR] http://arxiv.org/abs/1308.0215.

[118] B. Lévy. A numerical algorithm for L_2 semi-discrete optimal transport in 3d. *ESAIM M2AN (Mathematical Modeling and Analysis)*, 2015.

[119] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI'97*, pages 366–371, 1997.

[120] X. S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.*, 31(3):302–325, Sept. 2005.

[121] T. Lindvall. *Lectures on the coupling method*. Dover Publications, Inc., Mineola, NY, 2002.

[122] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[123] E. Maitre. Modèles et calculs d'interfaces. Notes de Cours M2R Mathématiques appliquées, UJF, Grenoble, 2010.

[124] H. Mandon, S. Haar, and L. Paulevé. Relationship between the Reprogramming Determinants of Boolean Networks and their Interaction Graph. In E. Cinquemani and A. Donzé, editors, *Hybrid Systems Biology: 5th International Workshop, HSB 2016, Grenoble, France, October 20-21, 2016, Proceedings*, volume 9957 of *Lecture Notes in Computer Science*, pages 113–127. Springer International Publishing, 2016.

[125] J.-F. Marckert and A. Mokkadem. The depth first processes of Galton-Watson trees converge to the same Brownian excursion. *Ann. Probab.*, 31(3):1655–1678, 2003.

[126] R. J. McCann. Existence and uniqueness of monotone measure-preserving maps. *Duke Mathematical Journal*, 80(2):309–323, 1995.

[127] F. Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011.

[128] L. Mercier. *Grands graphes et grand arbres aléatoires : Analyse du comportement asymptotique*. PhD thesis, Université de Lorraine, Mai 2016.

[129] Q. Mérigot. A multiscale approach to optimal transport. *Comput. Graph. Forum*, 30(5):1583–1592, 2011.

[130] Q. Mérigot, J. Meyron, and B. Thibert. Light in power: A general and parameter-free algorithm for caustic design. *CoRR*, abs/1708.04820, 2017.

[131] P. Milgrom and I. Segal. Envelope Theorems for Arbitrary Choice Sets. *Econometrica*, 70(2):583–601, March 2002.

[132] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. *National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.

[133] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transition'. *Nature*, 400:133–137, 1998.

[134] G. Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences (1781)*, pages 666–704, 1784.

[135] M. Moskewicz, C. Conor, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, Juin 2001.

[136] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[137] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[138] R. Nieuwenhuis and A. Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Computer Aided Verification, 17th International Conference, Edinburgh, Scotland, UK, July 6-10*, pages 321–334, 2005.

[139] R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. In *Term Rewriting and Applications, 16th International Conference, Nara, Japan, April 19-21*, pages 453–468, 2005.

[140] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). *J. ACM*, 53(6):937–977, 2006.

[141] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[142] M. Noual and S. Sené. Synchronism versus asynchronism in monotonic boolean automata networks. *Natural Computing*, Jan 2017.

[143] M. Ostrowski, L. Paulevé, T. Schaub, A. Siegel, and C. Guziolowski. Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. *Biosystems*, 149:139 – 153, 2016.

[144] N. Papadakis, G. Peyré, and E. Oudet. Optimal Transport with Proximal Splitting. *SIAM Journal on Imaging Sciences*, 7(1):212–238, Jan. 2014.

[145] L. Paulevé. Pint: a static analyzer for transient dynamics of qualitative networks with IPython interface. In *CMSB 2017 - 15th conference on Computational Methods for Systems Biology*, volume 10545 of *Lecture Notes in Computer Science*, pages 309–316. Springer International Publishing, 2017.

[146] L. Paulevé. Reduction of Qualitative Models of Biological Networks for Transient Dynamics Analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017. In press.

[147] L. Paulevé, G. Andrieux, and H. Koeppl. Under-approximating cut sets for reachability in large scale automata networks. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[148] L. Paulevé, M. Folschette, M. Magnin, and O. Roux. Analyses statiques de la dynamique des réseaux d'automates indéterministes. *Technique et Science Informatiques (TSI)*, 34(4):463–484, 2015.

[149] L. Paulevé, M. Magnin, and O. Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. *Mathematical Structures in Computer Science*, 22(04):651–685, 2012.

[150] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *proceedings of SAT*, pages 294–299, 2007.

[151] K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers with restarts. In *Principles and Practice of Constraint Programming - CP 2009*, 2009.

[152] B. Pittel. On growing random binary trees. *J. Math. Anal. Appl.*, 103(2):461–480, 1984.

[153] B. Pittel. Linear probing: the probable largest search time grows logarithmically with the number of records. *J. Algorithms*, 8(2):236–249, 1987.

[154] B. Pittel. Note on the heights of random recursive trees and random m-ary search trees. *Random Structures Algorithms*, 5(2):337–347, 1994.

[155] M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[156] N. Ray, B. Vallet, L. Alonso, and B. Lévy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1), 2009.

[157] N. Ray, B. Vallet, W. Li, and B. Lévy. N-symmetry direction field design. *ACM Trans. Graph.*, 27(2), 2008.

[158] B. Reed. The height of a random binary search tree. *J. ACM*, 50(3):306–332, 2003.

[159] B. Reed, N. Robertson, P. Seymour, and R. Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

[160] G. Regalo and A. Leutz. Hacking cell differentiation: transcriptional rerouting in reprogramming, lineage infidelity and metaplasia. *EMBO Molecular Medicine*, 5(8):1154–1164, 2013.

[161] E. Remy, P. Ruet, and D. Thieffry. Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335 – 350, 2008.

[162] A. Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378 – 392, 2010.

[163] I. Rish and R. Dechter. Resolution versus search: two strategies for SAT. *J. of Approximate Reasoning*, 2000.

[164] F. Robert. Iterations sur des ensembles finis et automates cellulaires contractants. *Linear Algebra and its Applications*, 29:393–412, 1980.

[165] F. Robert. *Discrete iterations: a metric study*, volume 6 of *Series in Computational Mathematics*. Springer, 1986.

[166] J. Robinson. A machine oriented based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[167] H. Rueß and N. Shankar. Deconstructing Shostak. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 19–28, Copenhagen, Denmark, 2001. IEEE Computer Society.

[168] F. Santambrogio. Introduction to Optimal Transport Theory. In *Optimal Transport, Theory and Applications*, August 2014. [math.PR] http://arxiv.org/abs/1009.3856.

[169] F. Santambrogio. *Optimal Transport for Applied Mathematicians*. Birkhauser, 2015.

[170] L. Saïs, editor. *Problème SAT : progrès et défis*. Hermès / Lavoisier, 2008.

[171] Y. Schwartzburg, R. Testuz, A. Tagliasacchi, and M. Pauly. High-contrast computational caustic design. *ACM Trans. Graph.*, 33(4):74:1–74:11, 2014.

[172] P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

[173] N. Shankar and H. Rueß. Combining Shostak theories. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA)*, volume 2378 of *LNCS*, pages 1–19, Copenhagen, Denmark, 2002. Springer.

[174] J. Shoenfield. *Mathematical logic*. Ak Peters Series. Association for Symbolic Logic, 1967.

[175] G. R. Shorack and J. A. Wellner. *Empirical processes with applications to statistics*. John Wiley & Sons, Inc., New York, 1986.

[176] R. E. Shostak. Deciding combinations of theories. *JACM*, 31(1):1–12, 1984.

[177] J. P. M. Silva and K. A. Sakallah. Grasp a new search algorithm for satisfiability. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[178] L. Simon. *Multirésolution pour le test de consistance et la déduction en logique propositionnelle*. PhD thesis, Université Orsay Paris XI, Décembre 2001.

[179] E. Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

[180] R. P. Stanley and J. Pitman. A polytope related to empirical distributions, plane trees, parking functions, and the associahedron. *Discrete Comput. Geom.*, 27(4):603–634, 2002.

[181] W. Stein. *Modular forms, a computational approach*, volume 79 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2007. With an appendix by Paul E. Gunnells.

[182] W. Stein. Mathematical software and me: A very personal recollection, 2009. disponible à `http://wstein.org/mathsoftbio/history.pdf`.

[183] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

[184] K. Takahashi, K. Tanabe, M. Ohnuki, M. Narita, T. Ichisaka, K. Tomoda, and S. Yamanaka. Induction of pluripotent stem cells from adult human fibroblasts by defined factors. *Cell*, 131:861–872, Nov. 2007.

[185] K. Takahashi and S. Yamanaka. A decade of transcription factor-mediated reprogramming to pluripotency. *Nat Rev Mol Cell Biol*, 17(3):183–193, Feb 2016.

[186] S. Tanny. A probabilistic interpretation of Eulerian numbers. *Duke Math. J.*, 40:717–722, 1973.

[187] T. Tao. *An Introduction to Measure Theory*. American Mathematical Society, 2011.

[188] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[189] G. Tenenbaum. *Introduction to analytic and probabilistic number theory*. American Mathematical Society, Providence, RI, third edition, 2015.

[190] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973.

[191] C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In *Proceedings of the First International Workshop on Frontiers of Combining Systems (FROCOS)*, Applied Logic, pages 103–120. Kluwer Academic Publishers, 1996.

[192] Y. Tokuzawa, K. Yagi, Y. Yamashita, Y. Nakachi, I. Nikaido, H. Bono, Y. Ninomiya, Y. Kanesaki-Yatsuka, M. Akita, H. Motegi, and et al. Id4, a new candidate gene for senile osteoporosis, acts as a molecular switch promoting osteoblast differentiation. *PLoS Genet*, 6(7):e1001019, Jul 2010.

[193] G. Tseitin. *Structures in Constructives Mathematics and Mathematical Logic*, chapter On the complexity of derivations in the propositional calculus, pages 115–125. A.O. Slesenko, 1968.

[194] W. T. Tutte. How to draw a graph. *Proc Lond Math Soc*, 13:743–767, 1963.

[195] A. Val. Tractable classes for directional resolution. In *Proc. 17th (U.S.) Nat. Conf. on Articial Intelligence.*, 2000.

[196] R. Varshamov. Some features of linear codes that correct asymmetric errors. In *Soviet Physics Doklady*, volume 9, page 538, 1965.

[197] J. Vera and E. Goles. Automata networks for memory loss effects in the formation of linguistic conventions. *Cognitive Computation*, 8(3):462–466, Jun 2016.

[198] C. Villani. *Topics in optimal transportation*. Graduate studies in mathematics. American Mathematical Society, Providence (R.I.), 2003.

[199] C. Villani. *Optimal transport : old and new*. Grundlehren der mathematischen Wissenschaften. Springer, Berlin, 2009.

[200] D. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Comput. J.*, 24(2):167–172, 1981.

[201] G. J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization*, pages 185–207, 2003.

[202] J. G. T. Zañudo and R. Albert. Cell fate reprogramming by control of intracellular network dynamics. *PLOS Computational Biology*, 11(4):1–24, 04 2015.

[203] H. Zhang. SATO: an efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI*, pages 272–275, 1997.

Table des figures

Table des matières