La méthode probabiliste : d'un résultat existentiel à un algorithme

Miguel ACOSTA, Jhih-Huang LI
Sous la direction de Pierre PANSU et Nicolas SCHABANEL

17 juin 2011

Exposé de première année

ENS

Table des matières

Introduction

La méthode probabiliste est un outil puissant pour montrer l'existence d'une configuration donnée d'un espace discret vérifiant une certaine propriété. Il suffit souvent de faire des considérations sur un choix fait au hasard pour obtenir un résultat non trivial. Dans ce mémoire, nous nous intéressons d'abord à cette méthode en traitant quelques exemples (section 1). Les exemples et preuves étudiés sont donnés dans [1]. Nous montrons ensuite, dans la deuxième section, le lemme local de Lovász, un résultat très utile dans ce domaine.

Ce lemme a, par exemple, une application surprenante : dans un graphe simple orienté de degré sortant minimal δ et de degré entrant maximal Δ, si $e(\delta\Delta + 1)(1 - \frac{1}{k})^\delta < 1$ (*), alors il existe un cycle de longueur multiple de k. Ceci soulève une question assez naturelle : le cycle existe, mais peut-on le trouver efficacement ?

Dans [2], R. Moser donne un algorithme aléatoire, puis un autre déterministe, qui permettent de trouver une configuration convenable dans le cadre du lemme local de Lovász. Dans la section 3, nous présentons l'algorithme aléatoire, que nous analysons, grâce à la construction des « arbres témoins » dans la section 4. L'essentiel des idées et des preuves est tiré de [2] et [3]. Finalement, nous étudions, à l'aide de la méthode des espérances conditionnelles (introduite dans la première section) la variante déterministe dans la section 5, qui est en temps polynomial.

Nous donnons en appendice deux programmes, un en Caml et un en C, pour trouver un cycle de longueur multiple de k dans un graphe simple orienté vérifiant (*). Ce sont des implémentations de l'algorithme aléatoire de Moser.

1 La méthode probabiliste

1.1 Méthode de base

La méthode probabiliste est souvent utilisée en mathématiques discrètes. Montrer qu'un objet satisfaisant une propriété donnée existe revient à travailler dans un espace de probabilité bien choisi et prouver que la propriété désirée est vérifiée avec probabilité strictement positive. On peut trouver par conséquent un point dans l'espace de probabilité qui donne cette propriété souhaitée.

Plus concrètement, regardons quelques exemples :

Définition 1.1. Pour tout n, on note K_n *le graphe complet à n sommets*, c'est-à-dire entre toute paire de sommets est tracée une et une seule arête.

Définition 1.2. *Le nombre de Ramsey* $R(k,l)$ est le plus petit entier n tel que pour tout 2-coloriage en rouge et bleu des arêtes de K_n, on obtient soit un K_k entièrement rouge, soit un K_l entièrement bleu.

Exemple 1.3. Si $\binom{n}{a}2^{1-\binom{a}{2}} < 1$, alors $R(a,a) > n$. Par conséquent, $R(a,a) > \lfloor 2^{a/2} \rfloor$ si $a \geq 3$.

Démonstration. Considérons un 2-coloriage en rouge et bleu aléatoire, dont les couleurs des arêtes sont déterminées indépendamment l'une de l'autre, avec probabilité $\frac{1}{2}$ pour les deux couleurs. On note E l'ensemble de tous les ensembles de a sommets de K_n. Pour tout $R \in E$, on note A_R l'événement que les arêtes reliant deux sommets dans R soient toutes de même couleur. Par l'indépendance du coloriage, $\mathbb{P}(A_R) = 2^{1-\binom{a}{2}}$. Or, $|E| = \binom{n}{a}$, donc la probabilité qu'au moins un des A_R soit réalisé est majorée par $\binom{n}{a}2^{1-\binom{a}{2}} < 1$. Ainsi avec probabilité strictement positive, aucun des A_R n'est réalisé. Par conséquent, $R(a,a) > n$.

Si $a \geq 3$, on prend $n = \lfloor 2^{a/2} \rfloor$, alors $\binom{n}{a}2^{1-\binom{a}{2}} < 1$. On est donc bien dans l'hypothèse de l'énoncé. □

Définition 1.4. Soit $A \subset \mathbb{N}$ un ensemble d'entiers. On dit que A est *de somme libre* si pour tous $a, b, c \in A$, $a + b \neq c$.

Exemple 1.5. Tout ensemble $B = \{b_1, ..., b_n\}$ de n entiers non nuls contient un sous-ensemble A de somme libre avec $|A| > \frac{1}{3}n$.

Démonstration. Soit $p = 3k + 2$ un nombre premier satisfaisant $p > 2 \max\limits_{1 \leq i \leq n} |b_i|$. On pose $C = \{k+1, ..., 2k+1\}$. Remarquons que C est un sous-ensemble de somme libre dans $\mathbb{Z}/p\mathbb{Z}$ et $\frac{|C|}{p-1} = \frac{k+1}{3k+1} > \frac{1}{3}$. Soit x une variable aléatoire distribuée uniformément sur l'ensemble $D = \{1, ..., p-1\}$. On définit les variables aléatoires $d_1, ..., d_n$ par $d_i \equiv xb_i (\text{mod } p)$. $\mathbb{Z}/p\mathbb{Z}$ étant un corps, les d_i sont toutes uniformément réparties dans D. Donc $\mathbb{P}(d_i \in C) > \frac{1}{3}$. De ce fait, il existe $x \in [\![1, p-1]\!]$ tel que $A = \{b_i \mid xb_i \in C\}$ soit de somme libre et de cardinal strictement supérieur à $\frac{n}{3}$. En effet, si $b_i = b_j + b_k$ alors $xb_i = xb_j + xb_k (\text{mod } p)$, ce qui contredirait la propriété de somme libre dans $\mathbb{Z}/p\mathbb{Z}$; le cardinal est strictement supérieur à $\frac{1}{3}n$ par la somme de probabilité. □

1.2 Linéarité de l'espérance

Nous venons de voir une méthode pour déterminer si un graphe ou un ensemble de nombres vérifie une certaine propriété, juste en tirant au hasard certains paramètres et en calculant la probabilité que la propriété que l'on souhaite soit réalisée.

La méthode décrite dans cette section à l'aide de quelques exemples consiste à calculer l'espérance d'une variable aléatoire X bien choisie dans un espace de probabilité judicieux. Souvent, cette variable peut s'écrire $X = \sum_{i=1}^{n} a_i X_i$, avec les X_i faciles à manipuler. Comme l'espérance est linéaire, $E[X] = \sum_{i=1}^{m} a_i E[X_i]$. On utilise souvent le fait qu'il existe un point de l'espace de probabilité tel que $X \geq E[X]$ et un autre tel que $X \leq E[X]$. Comme on considère aussi, très souvent, des espaces finis, ce fait donne l'existence d'une configuration telle que $X \geq E[X]$ ou $X \leq E[X]$.

Voici deux exemples qui illustrent cette méthode.

Définition 1.6. Pour un graphe orienté G, un *chemin hamiltonien* est la donnée d'une permutation $(s_1, \ldots, s_n)$ des sommets de G telle que, pour tout $1 \leq i \leq n-1$, l'arête (s_i, s_{i+1}) soit dans G.

Définition 1.7. Un *tournoi* T à n éléments est un graphe orienté T avec n sommets et tel que, pour deux sommets u, v de T est tracée une et une seule arête parmi (u, v) et (v, u).

Exemple 1.8 (Chemins hamiltoniens)**.** On va montrer que, pour tout $n \in \mathbb{N}$, il existe un tournoi T à n éléments qui contient au moins $n!2^{-(n-1)}$ chemins hamiltoniens.

Démonstration. Pour le faire, on va calculer l'espérance du nombre de chemins hamiltoniens dans T lorsqu'on tire au hasard, uniformément, l'orientation de chaque arête.

Soient $s_1, \ldots, s_n$ les sommets de T. On tire au hasard les orientations des arêtes. On note X le nombre de chemins hamiltoniens dans T : c'est une variable aléatoire. On peut alors décomposer $X = \sum_{\sigma \in S_n} X_\sigma$ où X_σ est l'indicatrice que $(s_{\sigma(1)}, \ldots, s_{\sigma(n)})$ est un chemin hamiltonien. On sait alors que $E[X] = E\left[\sum_{\sigma \in S_n} X_\sigma\right] = \sum_{\sigma \in S_n} E[X_\sigma] = n!2^{-(n-1)}$, puisque l'espérance de X_σ est la probabilité que $(s_{\sigma(1)}, \ldots, s_{\sigma(n)})$ soit un chemin hamiltonien, c'est-à-dire que les $n-1$ arêtes soient dans le bon sens.

En particulier, il existe un point de l'espace de probabilité pour lequel $X \geq n!2^{-(n-1)}$, c'est-à-dire, un tournoi T avec au moins $n!2^{-(n-1)}$ chemins hamiltoniens. □

Exemple 1.9. Soient $a, n \in \mathbb{N}$. Alors il existe un $2-$coloriage de K_n contenant au plus $\binom{n}{a} 2^{1-\binom{a}{2}}$ K_a monochromatiques.

Démonstration. On fait comme précédemment : on colorie chaque arête de K_n en rouge ou en bleu avec probabilité $\frac{1}{2}$ et indépendamment.

Soit X le nombre de K_a monochromatiques qui apparaissent dans K_n lors du coloriage aléatoire. Notons A l'ensemble des sous-graphes K_a de K_n. On sait alors que $|A| = \binom{n}{a}$ et que $X = \sum_{G \in A} X_G$, où X_G est l'indicatrice que G est monochromatique. Comme un K_a a $\binom{a}{2}$ arêtes, on a $E[X_G] = 2^{1-\binom{a}{2}}$ pour tout $G \in A$ (il y a deux cas parmi $2^{\binom{a}{2}}$ qui sont monochromatiques).

Donc $E[X] = \sum_{G \in A} E[X_G] = \binom{n}{a} 2^{1-\binom{a}{2}}$. Il existe alors un $2-$coloriage de K_n contenant au plus $\binom{n}{a} 2^{1-\binom{a}{2}}$ K_a monochromatiques. □

1.3 Dérandomisation

1.3.1 Méthode générale : espérances conditionnelles

Avant de formaliser la méthode de dérandomisation à l'aide des probabilités conditionnelles, commençons plutôt par un exemple pour éclairer les idées.

Exemple 1.10. Pour tout n entier, il existe un 2-coloriage des arêtes de K_n tel que le nombre de copies de K_4 monochromatiques est au plus $\binom{n}{4}2^{-5}$. Par la méthode des espérances conditionnelles, on peut trouver un tel coloriage en temps polynomial.

Démonstration. L'existence découle de l'exemple 1.9.

Pour avoir un algorithme déterministe et dérandomisé, on procédera comme suit :

1. On numérote les arêtes de K_n et on colorie la première arête en rouge, par exemple.
2. On définit une fonction de poids pour tout K_n colorié partiellement : pour toute copie K de K_4 dans K_n, on note

$$w(K) = \begin{cases} 0 & \text{si au moins une arête est coloriée rouge et une autre bleue} \\ 2^{-5} & \text{si aucune arête n'est coloriée} \\ 2^{r-6} & \text{si } r \geq 1 \text{ arêtes sont coloriées et toutes de même couleur} \end{cases}$$

   On peut remarquer que ce poids représente exactement la probabilité que K soit monochromatique si toutes les arêtes restantes sont coloriées indépendamment de manière uniforme, qui est juste la probabilité conditionnelle que K soit monochromatique sachant le coloriage partiel. Ainsi, par la linéarité de l'espérance, la somme des poids W donne l'espérance du nombre de copies K_4 monochromatique.
3. On minimise ce poids au cours du coloriage. En effet, supposons connues les couleurs des arêtes $e_1, ..., e_{i-1}$, et W le poids correspondant. Il faut maintenant colorier e_i. Notons W_{rouge} le poids correspondant si e_i est coloriée rouge et w_{bleu} si e_i est coloriée bleue. Par définition,

$$W = \frac{W_{rouge} + W_{bleu}}{2}.$$

   Il existe donc un des W_{rouge} et W_{bleu} qui est plus petit ou égal à W. Si $W_{rouge} \leq W_{bleu}$, on pose e_i rouge. Sinon, on pose e_i bleue.

Ce procédé est bien déterministe puisqu'à chaque étape, on calcule W_{rouge} et W_{bleu} puis choisit la couleur qui correspond au plus petit poids. Il est en plus de complexité temporelle polynomiale car le calcul de W_{rouge} et W_{bleu} se fait en temps polynomial (seulement les K contenant l'arête e_i sont concernés et ils ne sont que $\binom{n-2}{2}$) et on a $\binom{n}{2}$ arêtes à colorier. □

Formalisons cette idée :

Proposition 1.11. *Soient $Y_1, ..., Y_m$ des variables aléatoires indépendantes uniformes sur un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$ et à valeurs dans un ensemble fini $\mathcal{Y}$. Soit $\mathcal{A}$ un ensemble d'événements de $\mathcal{F}$ qui dépendent de certaines variables aléatoires parmi $Y_1, ..., Y_m$. Soit s un entier vérifiant $\sum_{A\in\mathcal{A}} \mathbb{P}(A) \leq s$. On peut procéder, en temps polynomial, de la manière suivante pour trouver une évaluation des Y_j telle qu'au plus s événements dans $\mathcal{A}$ soient vérifiés :*

1. *Soit $j \in [\![1, m]\!]$. Supposons connues les valeurs de $y_1, ..., y_{j-1}$, qui vérifient*

$$\sum_{A\in\mathcal{A}} \mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}) \leq s.$$

2. *Pour tout $A \in \mathcal{A}$, on sait que*

$$\mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}) = \frac{1}{|\mathcal{Y}|} \sum_{y\in\mathcal{Y}} \mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}, Y_j = y).$$

   Il existe donc un $y_j \in \mathcal{Y}$ tel que

$$\begin{aligned} \mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}, Y_j = y_j) &= \min_{y\in\mathcal{Y}} \mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}, Y_j = y) \\ &\leq \mathbb{P}(A \mid Y_1 = y_1, ..., Y_{j-1} = y_{j-1}). \end{aligned}$$

3. *Une fois les valeurs $y_1, ..., y_m$ déterminées, la probabilité conditionnelle $\mathbb{P}(A \mid Y_1 = y_1, .., Y_m = y_m)$ devient déterministe, valant 1 si A est vérifié pour l'affectation $(Y_1, ..., Y_m) = (y_1, ..., y_m)$ et 0 sinon. En plus, la somme des probabilités conditionnelles étant décroissante, la somme finale représente exactement le nombre d'événements réalisés avec l'affectation $(Y_1, ..., Y_m) = (y_1, ..., y_m)$. Le résultat en découle.*

1.3.2 Méthode de majoration des espérances conditionnelles

Cependant, le calcul des probabilités conditionnelles $\mathbb{P}(A \mid Y_1 = y_1, ..., Y_j = y_j)$ peut être parfois compliqué. Auquel cas, on peut utiliser la méthode par majoration décrite ci-dessous :

Proposition 1.12. *Soient $Y_1, ..., Y_m$ des variables aléatoires indépendantes uniformes sur un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$ à valeurs dans un ensemble fini $\mathcal{Y}$. Soit $\mathcal{A}$ un ensemble d' événements de $\mathcal{F}$ qui dépendent de certaines variables aléatoires parmi $Y_1, ..., Y_m$. Supposons qu'il existe des fonctions f_j^A faciles à calculer et s vérifiant :*

1. $f_{j-1}^A(y_1, ..., y_{j-1}) \geq \min_{y \in \mathcal{Y}} f_j^A(y_1, ..., y_{j-1}, y)$ *pour tout* $A \in \mathcal{A}$, $j \in [\![1, m]\!]$ *et* $y_1, ..., y_{j-1}, y \in \mathcal{Y}$.
2. $f_j^A(y_1, ..., y_j) \geq \mathbb{P}(A \mid Y_1 = y_1, ..., Y_j = y_j)$ *pour tout* $A \in \mathcal{A}$, $j \in [\![1, m]\!]$ *et* $y_1, ..., y_j \in \mathcal{Y}$.
3. $\sum_{A \in \mathcal{A}} f_0^A \leq s$.

Alors si on choisit, comme décrit dans la proposition précédente, les valeurs de y_j qui minimisent les $f_j^A(y_1, \ldots, y_j)$, on obtient :

$$\sum_{A \in \mathcal{A}} \mathbb{P}(A \mid Y_1 = y_1, ..., Y_m = y_m) \leq \sum_{A \in \mathcal{A}} f_m^A(y_1, ..., y_m) \leq s$$

Comme les $A \in \mathcal{A}$ dépendent uniquement des Y_j ($1 \leq j \leq m$), les probabilités conditionnelles $\mathbb{P}(A \mid Y_1 = y_1, ..., Y_m = y_m)$ valent soit 0, soit 1. On a trouvé donc une évaluation des Y_j pour laquelle au plus s événements dans $\mathcal{A}$ sont satisfaits.

Démonstration. La démonstration est claire. On procède par récurrence en choisissant les y_j qui minimisent les espérances conditionnelles pour aboutir au résultat. □

Cette technique nous sera particulièrement utile dans la section 5.2.2 : on sera dans une situation où les espérances conditionnelles ne sont pas faciles à calculer, mais on disposera d'une majoration convenable et calculable efficacement.

Exemple 1.13. Soit $(a_{i,j})_{1 \leq i,j \leq n}$ une matrice réelle, où $-1 \leq a_{i,j} \leq 1$ pour tout $(i, j) \in [\![1, n]\!]^2$. On peut trouver, en temps polynomial, $\epsilon_1, ..., \epsilon_n \in \{-1, 1\}$ tels que pour tout $i \in [\![1, n]\!]$, $|\sum_{j=1}^n \epsilon_j a_{i,j}| \leq \sqrt{2n \ln(2n)}$.

Démonstration. Considérons l'espace de probabilité symétrique constitué de 2^n points, correspondant aux 2^n évaluations possibles pour le n-uplet $(\epsilon_1, ..., \epsilon_n) \in \{-1, 1\}^n$. On définit $\beta = \sqrt{2n \ln(2n)}$ et on pose A_i l'événement $|\sum_{j=1}^n \epsilon_j a_{i,j}| > \beta$. Maintenant on applique la méthode de majoration des espérances conditionnelles en choisissant judicieusement les majorants afin de trouver un point dans l'espace où aucun A_i ne soit satisfait.

On définit $\alpha = \frac{\beta}{n}$ et on pose $G(x)$ la fonction

$$G(x) = \cosh(\alpha x) = \frac{e^{\alpha x} + e^{-\alpha x}}{2}.$$

On peut d'abord remarquer quelques propriétés de G :

1. En comparant les premiers termes dans le développement de Taylor, on obtient

$$G(x) \leq e^{\frac{\alpha^2 x^2}{2}}.$$

Cette inégalité est stricte dès que x et α sont non nuls.

2. Propriété de la fonction cosinus hyperbolique :

$$G(x)G(y) \leq \frac{G(x+y)+G(x-y)}{2}.$$

On peut maintenant définir les fonctions f_p^i, les majorants des espérances conditionnelles, par :

$$f_p^i(\epsilon_1, ..., \epsilon_p) = 2e^{-\alpha\beta} G\left(\sum_{j=1}^{p} \epsilon_j a_{i,j}\right) \prod_{j=p+1}^{n} G(a_{i,j})$$

pour tous $i, p \in [\![1, n]\!]$ et $(\epsilon_1, ..., \epsilon_p) \in \{-1, 1\}^p$.

Il est clair que ces fonctions se calculent facilement, il ne reste donc plus qu'à vérifier les conditions de la proposition 1.12 où l'on remplace l'inégalité large du troisième point par une inégalité stricte en prenant $s = 1$. En effet, $\sum_{i=1}^{n} f_p^i$ majore l'espérance conditionnelle du nombre d'événements vérifiés parmi les A_i sachant les évaluations des $\epsilon_1, ..., \epsilon_{p-1}$, si elle est strictement inférieure à 1, alors à la fin de l'algorithme ($p = n$), comme ceci représente exactement le nombre d'événements vérifiés, elle doit être nulle.

Faisons la vérification dans l'ordre :

1. On montre que pour tout $i \in [\![1, n]\!]$ et tout p-uplet $(\epsilon_1, ..., \epsilon_p) \in \{-1, 1\}^p$,

$$f_{p-1}^i(\epsilon_1, ..., \epsilon_{p-1}) \geq \min\{f_p^i(\epsilon_1, ..., \epsilon_{p-1}, -1), f_p^i(\epsilon_1, ..., \epsilon_{p-1}, 1)\}.$$

Pour cela, on pose $v = \sum_{j=1}^{p-1} \epsilon_j a_{i,j}$. Par définition de f_p^i et les propriétés de G :

$$\begin{aligned} f_{p-1}^i(\epsilon_1, ..., \epsilon_{p-1}) &= 2e^{-\alpha\beta} G(v) G(a_{i,p}) \prod_{j=p+1}^{n} G(a_{i,j}) \\ &= 2e^{-\alpha\beta} \frac{G(v - a_{i,p}) + G(v + a_{i,p})}{2} \prod_{j=p+1}^{n} G(a_{i,j}) \\ &= \tfrac{1}{2}(f_p^i(\epsilon_1, ..., \epsilon_{p-1}, -1) + f_p^i(\epsilon_1, ..., \epsilon_{p-1}, 1)) \\ &\geq \min\{f_p^i(\epsilon_1, ..., \epsilon_{p-1}, -1), f_p^i(\epsilon_1, ..., \epsilon_{p-1}, 1)\} \end{aligned}$$

2. On montre que pour tout $i \in [\![1, n]\!]$ et tout $(\epsilon_1, ..., \epsilon_{p-1}) \in \{-1, 1\}^p$,

$$f_{p-1}^i(\epsilon_1, ..., \epsilon_{p-1}) \geq \mathbb{P}(A_i \mid \epsilon_1, ..., \epsilon_{p-1}).$$

Prenons le même v que précédemment. Alors,

$$\mathbb{P}(A_i \mid \epsilon_1, ..., \epsilon_{p-1}) \leq \mathbb{P}(v + \sum_{j \geq p} \epsilon_j a_{i,j} > \beta) + \mathbb{P}(-v - \sum_{j \geq p} \epsilon_j a_{i,j} > \beta)$$

Or,

$$\begin{aligned}\mathbb{P}\left(v+\sum_{j\geq p}\epsilon_j a_{i,j}>\beta\right) &= \mathbb{P}\left(e^{\alpha\left(v+\sum_{j\geq p}\epsilon_j a_{i,j}\right)}>e^{\alpha\beta}\right)\\ &\leq e^{\alpha v}e^{-\alpha\beta}E\left(e^{\alpha\left(\sum_{j\geq p}\epsilon_j a_{i,j}\right)}\right)\\ &= e^{\alpha v}e^{-\alpha\beta}\prod_{j\geq p}E\left(e^{\alpha\epsilon_j a_{i,j}}\right)\\ &= e^{\alpha v}e^{-\alpha\beta}\prod_{j\geq p}G(a_{i,j})\end{aligned}$$

De même pour l'autre terme :

$$\mathbb{P}\left(-v-\sum_{j\geq p}\epsilon_j a_{i,j}>\beta\right)\leq e^{-\alpha v}e^{-\alpha\beta}\prod_{j\geq p}G(a_{i,j})$$

Donc :

$$\mathbb{P}(A_i\mid\epsilon_1,...,\epsilon_{p-1})\leq 2e^{-\alpha\beta}G(v)\prod_{j\geq p}G(a_{i,j})=f^i_{p-1}(\epsilon_1,...,\epsilon_{p-1}).$$

3. Il faut montrer que $\sum_{i=1}^n f_0^i<1$.

$$\begin{aligned}\sum_{i=1}^n f_0^i &= \sum_{i=1}^n 2e^{-\alpha\beta}\prod_{j=1}^n G(a_{i,j})\\ &\leq \sum_{i=1}^n 2e^{-\alpha\beta}\prod_{j=1}^n e^{\frac{\alpha^2 a_{i,j}^2}{2}}\\ &\leq \sum_{i=1}^n 2e^{-\alpha\beta}e^{\frac{\alpha^2 n}{2}}\\ &= 2ne^{\frac{\alpha^2 n}{2}-\alpha\beta}=2ne^{-\frac{\alpha^2 n}{2}}=1.\end{aligned}$$

En plus, la première inégalité est stricte sauf quand $a_{i,j}=0$ pour tout i,j. Cependant, la deuxième est stricte sauf quand $a_{i,j}=1$ pour tout i,j. D'où l'inégalité stricte.

La preuve se termine donc.

□

2 Le lemme local de Lovász

Nous venons de voir des méthodes pour monter l'existence d'une propriété en montrant qu'un événement a lieu avec probabilité positive. On peut remarquer que, souvent, quand la taille du problème tend vers $+\infty$, la probabilité que la propriété soit satisfaite dans un tirage au hasard tend vers 1 : et donc que très souvent, un coloriage ou un choix aléatoire conviendra. Il y a aussi un autre cas où c'est facile de montrer qu'une probabilité est strictement positive, mais tend vers 0 quand la taille du problème tend vers $+\infty$: c'est quand il s'agit d'événements indépendants de probabilité strictement positive. À mi-chemin entre ces deux cas, on trouve le résultat prouvé par Erdös et Lovász en 1975, connu sous le nom du lemme local de Lovász : il sert à montrer qu'une probabilité est strictement positive quand on a une faible dépendance entre les événements et certaines conditions sur leur probabilité. C'est un outil puissant, comme on le verra dans les exemples suivants traités à la fin de cette section.

Nous montrons ici le cas général du lemme local, le cas symétrique (plus facile à appliquer) et nous donnerons quelques exemples pour éclairer le fonctionnement de ce résultat.

2.1 Cas général

Théorème 2.1 (Lemme local de Lovász : cas général)**.** *Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace de probabilité. Soient $A_1, \dots, A_n \in \mathcal{F}$ des événements. Soit $E = \{\{i,j\} \subset [\![1,n]\!] \mid i \neq j, A_i \text{ et } A_j \text{ ne sont pas indépendants}\}$. On suppose qu'il existe des réels $x_1, \dots, x_n \in [0,1[$ tels que, pour tout $i \in [\![1,n]\!]$, on ait*

$$\mathbb{P}(A_i) \leq x_i \prod_{\{i,j\} \in E} (1 - x_j)$$

Alors

$$\mathbb{P}(\bigwedge_{i=1}^{n} \overline{A_i}) \geq \prod_{i=1}^{n} (1 - x_i) > 0$$

Démonstration. On montre, par récurrence sur $s < n$, le résultat suivant :
Pour tout $i \in [\![1,n]\!]$, et pour tout $S \subseteq [\![1,n]\!] \backslash \{i\}$ de cardinal s, $\mathbb{P}(A_i \mid \bigwedge_{j \in S} \overline{A_j}) \leq x_i$.

- Si $s = 0$, on a bien, pour tout $i \in [\![1,n]\!]$, $\mathbb{P}(A_i) \leq x_i \prod_{\{i,j\} \in E} (1 - x_j) \leq x_i$
- Soit $0 < s < n$ tel que le résultat soit vrai pour tout $s' < s$.
  Soient $i \in [\![1,n]\!]$ et $S \subseteq [\![1,n]\!] \backslash \{i\}$ de cardinal s. On décompose S en deux sous-ensembles : $S_1 = \{j \in S \mid \{i,j\} \in E\}$ et $S_2 = S \backslash S_1$.
  On a alors
  $$\mathbb{P}(A_i \mid \bigwedge_{j \in S} \overline{A_j}) = \frac{\mathbb{P}(A_i \wedge \bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l})}{\mathbb{P}(\bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l})}$$
  - Majorons le numérateur : $\mathbb{P}(A_i \wedge \bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l}) \leq \mathbb{P}(A_i \mid \bigwedge_{l \in S_2} \overline{A_l}) = \mathbb{P}(A_i)$ puisque A_i est indépendant des événements de S_2. D'où $\mathbb{P}(A_i \wedge \bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l}) \leq x_i \prod_{\{i,j\} \in E} (1 - x_j)$
  - Minorons le dénominateur : Si $S_1 = \{j_1, \dots, j_r\}$, on a $\mathbb{P}(\bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l}) = \mathbb{P}(\overline{A_{j_1}} \mid \bigwedge_{l \in S_2} \overline{A_l}) \times \mathbb{P}(\overline{A_{j_2}} \mid \overline{A_{j_1}} \wedge \bigwedge_{l \in S_2} \overline{A_l}) \times \cdots \times \mathbb{P}(\overline{A_{j_r}} \mid \overline{A_{j_1}} \wedge \cdots \wedge \overline{A_{j_{r-1}}} \wedge \bigwedge_{l \in S_2} \overline{A_l}) \geq (1 - x_{j_1}) \cdots (1 - x_{j_r})$ par hypothèse de récurrence. D'où $\mathbb{P}(\bigwedge_{j \in S_1} \overline{A_j} \mid \bigwedge_{l \in S_2} \overline{A_l}) \geq \prod_{\{i,j\} \in E} (1 - x_j)$
  On a donc $\mathbb{P}(A_i \mid \bigwedge_{j \in S} \overline{A_j}) \leq x_i$.

Il suffit alors de remarquer que

$$\mathbb{P}(\bigwedge_{i=1}^{n} \overline{A_i}) = \mathbb{P}(\overline{A_1}) \times \mathbb{P}(\overline{A_2} \mid \overline{A_1}) \times \cdots \times \mathbb{P}(\overline{A_n} \mid \bigwedge_{i=1}^{n} \overline{A_i}) \geq \prod_{i=1}^{n} (1 - x_i) > 0$$

□

2.2 Cas symétrique et exemples

Le résultat du lemme local de Lovász est assez général et puissant. Cependant, les conditions à vérifier restent difficiles à exprimer en pratique. De plus, on se trouve souvent dans des situations avec une grande symétrie pour les A_i. C'est pourquoi on donne un corollaire de la proposition 2.2, qui a des conditions plus faciles à vérifier, quitte à supposer un problème symétrique.

Corollaire 2.2 (Lemme local de Lovász : cas symétrique)**.** *Soient $(A_i)_{1 \leq i \leq n}$ des événements dans un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$. Supposons qu'il existe $d \in \mathbb{N}$ et $p \in [0,1[$ tels que :*

– *Pour tout* $1 \leq i \leq n$, *il existe au plus* d *événements différents de* A_i *qui ne sont pas indépendants de* A_i.
– *Pour tout* $1 \leq i \leq n$, $\mathbb{P}(A_i) \leq p$.

Si, de plus, $ep(d+1) \leq 1$, *alors* $\mathbb{P}(\bigwedge_{i=1}^{n} \overline{A_i}) > 0$.

Démonstration. Si $d = 0$, les événements sont indépendants l'un de l'autre, donc $\mathbb{P}(\bigwedge_{i=1}^{n} \overline{A_i}) = (1-p)^n > 0$. Sinon, on applique le lemme local de Lovász. On prend le graphe de dépendance $D = (V, E)$, c'est-à-dire $(i, j) \in E \Leftrightarrow A_i$ et A_j ne sont pas indépendants. Ensuite, il suffit de prendre $x_i = \frac{1}{d+1}$ pour tout i. On a ainsi

$$\mathbb{P}(A_i) \leq p \leq \frac{1}{e(d+1)} \leq \frac{1}{d+1}(1 - \frac{1}{d+1})^d$$

car la fonction $x \mapsto (1 - \frac{1}{1+x})^x$ décroît vers $\frac{1}{e}$ lorsque x tend vers $+\infty$. □

Voici deux exemples d'utilisation du cas symétrique du lemme local de Lovász. On reviendra plus tard sur le deuxième pour donner un algorithme qui le résout.

Définition 2.3. Un *hypergraphe* $H = (V, E)$ est la donnée d'un ensemble fini de points V (ses sommets) et d'une collection E de sous-ensembles de V (ses arêtes).

Exemple 2.4 (Un premier exemple)**.** Soit $H = (V, E)$ un hypergraphe. On suppose que chaque arête a au moins k éléments, et qu'elle intersecte au plus d autres arêtes. Si de plus $e(d+1) < 2^{k-1}$, alors il existe un 2-coloriage de V tel que toutes les arêtes de H contiennent des sommets des deux couleurs.

Démonstration. On colorie aléatoirement, et indépendamment, les sommets de l'hypergraphe : bleu avec probabilité $\frac{1}{2}$ et rouge avec probabilité $\frac{1}{2}$. Pour $f \in E$, on note A_f l'événement où f est monochromatique. Clairement, $\mathbb{P}(A_f) = 2 \times 2^{-|f|} \leq 2^{1-k}$ pour tout $f \in E$. De plus, chaque A_f est indépendant de tous les A_g tels que $f \cap g = \emptyset$, et donc de tous les A_g sauf au plus d. Puisque $e(d+1)2^{1-k} < 1$, d'après le cas symétrique du lemme local de Lovász, on a $\mathbb{P}(\bigwedge_{f \in E} \overline{A_f}) > 0$. En particulier, comme notre espace de probabilité est fini, il existe un 2-coloriage de V qui ne réalise aucun événement A_f : c'est-à-dire, il est tel que toutes les arêtes de H contiennent des sommets des deux couleurs. □

Exemple 2.5 (Existence d'un cycle d'une certaine longueur)**.** On se donne $D = (V, E)$ un graphe simple et orienté dont δ désigne le degré sortant minimal et Δ le degré entrant maximal. Soit k un entier naturel. Si $e(\delta\Delta + 1)(1 - \frac{1}{k})^{\delta} < 1$ alors D contient un cycle de longueur multiple de k.

Démonstration. L'idée est la suivante : Les sommets sont coloriés par k couleurs différentes, numérotées de 0 jusqu'à k. Le coloriage qu'on cherche est tel que pour chaque sommet il existe un successeur colorié par la couleur suivante. Maintenant, il suffit de partir d'un sommet quelconque, et on fait un tour en choisissant un successeur de couleur suivante à chaque fois. On s'arrête lorsqu'on revient à un sommet déjà visité, noté v. Le graphe étant fini, ce procédé se termine. On reprend le chemin partant de v et qui se rejoint lui-même, le cycle de longueur multiple de k est ainsi obtenu.

Pour commencer, on a le droit de supposer que δ est atteint pour chaque sommet. En effet, si un sous-graphe de D possède un cycle de longueur multiple de k alors il en est de même pour le graphe d'origine.

On dispose de k couleurs qui sont numérotées de 0 à k et d'une fonction de coloriage aléatoire $f : V \to \{0, 1, ..., k-1\}$ qui à chaque sommet $v \in V$ associe une couleur. Les variables aléatoires

$(f(v))_{v\in V}$ suivent la loi uniforme et sont indépendantes. Pour un $v \in V$ on note A_v l'événement qu'aucun de ses successeurs n'est colorié par la couleur suivante, *ie.* pour tout $(v, w) \in E$ $f(w) \not\equiv f(u) + 1 \pmod k$. On a $\mathbb{P}(A_v) = (1 - \frac{1}{k})^\delta$. Fixons $v \in V$ et cherchons son degré de dépendance. Soit $u \in V$. Si A_u et A_v ne sont pas indépendants, alors soit u est un successeur de v, soit u et v ont un successeur en commun. La première situation a lieu δ fois et la deuxième $(\Delta - 1)\delta$ fois. Donc le degré de dépendance est majoré par $\delta\Delta$. Par conséquent, on vérifie bien l'hypothèse énoncée dans le cas symétrique du lemme local de Lovász. Donc $\mathbb{P}(\bigwedge_{v\in V} \overline{A_v}) > 0$. D'où l'exitence d'un coloriage tel que

$$\forall v \in V, \exists w \in V, (v, w) \in E \text{ et } f(w) \equiv f(u) + 1 (\mathrm{mod} k).$$

Le résultat découle alors de l'idée donnée au début de la preuve. □

3 Mise en algorithme

Nous venons de voir la puissance du lemme local de Lovász pour montrer des résultats d'existence d'une propriété dans un graphe vérifiant une propriété facile à tester. Cependant, et contrairement à ce qui se passe lors des autres utilisations d'une méthode probabiliste, comme les événements sont peu dépendants entre eux, la probabilité qu'un coloriage aléatoire réussisse peut diminuer exponentiellement avec la taille du problème.

Une question se pose alors naturellement : est-ce possible d'exhiber une solution en un temps polynomial par un algorithme ? Ou encore : est-ce possible de le faire d'une façon déterministe ?

La réponse à ces deux questions est affirmative : c'est ce que nous nous proposons de monter dans la suite, en expliquant le raisonnement fait par Moser dans [2] et détaillé dans Spencer [3].

L'algorithme qu'on décrira dans la suite fournit, en plus, une preuve du lemme local de Lovász dans le cas où les événements dépendent d'une famille finie de variables aléatoires à valeurs dans un ensemble fini, qui est, malgré le fait de ne pas être aussi général que le lemme, le cas où l'on se trouve la plupart du temps.

3.1 Cadre

On se place dans le cadre suivant : Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace de probabilité et $\mathcal{A} = \{A_1, \ldots, A_n\} \subset \mathcal{F}$. On suppose qu'il existe des variables aléatoires indépendantes $X_1, \ldots X_m$ $\mathcal{F}$-mesurables, à valeurs dans un ensemble fini telles que, pour tout $j \in [\![1, n]\!]$ A_j dépend uniquement des éléments de $vbl(A_j) \subset \{X_1, \ldots X_m\}$.

Remarque 3.1. L'hypothèse sur l'existence des X_i n'est pas du tout contraignante. En effet, étant donnés les A_j, on peut toujours construire un nombre fini de variables aléatoires X_i, à valeurs dans un ensemble fini, qui déterminent les A_j. (cf. appendice A)

Définition 3.2. Pour un événement A_j, on note

$$\Gamma(A_j) = \{i \in [\![1, n]\!] \mid i \neq j, vbl(A_i) \cap vbl(A_j) \neq \emptyset\}$$

l'ensemble des événements distincts de A_j qui ne sont pas indépendants de A_j, puis $\Gamma^+(A_j) = \Gamma(A_j) \cup \{A_j\}$.

De plus, pour chaque événement A on se donne $x(A) \in [0, 1[$. Les hypothèses du lemme local de Lovász s'écrivent alors :

1. Dans le cas général :

– Pour tout $A \in \mathcal{A}$, $\mathbb{P}(A) \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B))$

2. Puis, dans le cas symétrique (en se donnant $p \in [0, 1[$:
   – Pour tout $A \in \mathcal{A}$, $\mathbb{P}(A) \leq p$
   – Pour tout $A \in \mathcal{A}$, $\Gamma(A)$ est de cardinal $\leq d$.
   – $p \frac{(d+1)^{d+1}}{d^d} \leq ep(d+1) \leq 1$

Si ces hypothèses sont satisfaites, on a

$$\mathbb{P}(\bigwedge_{i=1}^{n} \overline{A_i}) > 0$$

3.2 Algorithme

C'est grâce au travail de Moser et Tardos [2] que nous avons les résultats qui suivront dans le texte. L'algorithme que nous présentons à continuation est décrit dans [2] et expliqué, dans sa version symétrique, dans [3].

On se propose de prouver ce résultat en fournissant un algorithme probabiliste, qui donne une évaluation des variables X_i qui ne satisfait aucune des conditions imposées par les événements A_j.

L'algorithme est assez simple :

Algorithme : MOSER_ ALEATOIRE

Affecter chaque X_i avec une valeur choisie au hasard
TANT QUE un des A_i est satisfait avec l'évaluation courante
        CHOISIR un A parmi les A_i qui sont satisfaits
        Réaffecter chaque $X \in vbl(A)$ avec une valeur choisie au hasard
FIN TANT QUE

Pour que cet algorithme soit bien défini, il suffit d'avoir une fonction déterministe pour CHOISIR. On peut, par exemple, prendre l'événement d'indice minimal qui est satisfait avec l'évaluation courante.

4 Complexité de l'algorithme

L'algorithme de la section 3.2 renvoie manifestement une affectation des X_j qui ne satisfait aucun des événements A_i. La difficulté réside sur la terminaison de l'algorithme. Dans cette section on montrera, à l'aide de la construction des arbres témoins, que l'espérance du temps d'exécution de l'algorithme est linéaire en n. La construction des arbres est donnée dans [2] et [3] ; le calcul de la complexité dans le cas symétrique se trouve dans [3] et celle du cas général dans [2].

4.1 Arbres témoins

On va construire une suite aléatoire d'arbres, appelés arbres témoins, qui témoigne de l'exécution de l'algorithme, et qui sera un outil essentiel dans le calcul de complexité puis pour trouver une version déterministe en temps polynomial de celui-ci (dans la section 5.2.2).

4.1.1 Construction

On construit, à partir d'une exécution de l'algorithme, une suite d'événements Log telle que, pour chaque $t \in \mathbb{N}^*$ $Log(t) = A_j$ si au cours de la $t^{ème}$ boucle TANT QUE on choisit A_j pour changer les évaluations de ses variables. En particulier, l'algorithme se termine si et seulement si la suite Log est finie.

La suite Log décrit tout ce qui se passe lors de l'exécution de l'algorithme, et contient donc toute l'information nécessaire pour le décrire. Cependant, pour voir plus clairement les étapes de l'algorithme qui sont en rapport avec l'événement traîté au temps t, on construit une suite d'arbres $\mathcal{L} = (\tau(j))_{j\in\mathbb{N}^*}$ à partir de la suite Log de la façon suivante :

Définition 4.1. Si τ est un arbre dont chaque sommet est associé à une étiquette dans $\mathcal{A}$.

1. On désigne par $V(\tau)$ l'ensemble des sommets de τ.
2. Pour tout $v \in V(\tau)$ on note $[v]$ l'étiquette associée.

Définition 4.2. Pour $t \in \mathbb{N}^*$, on définit par récurrence descendante sur i l'arbre étiqueté $\tau^{(i)}(t)$ de la manière suivante :

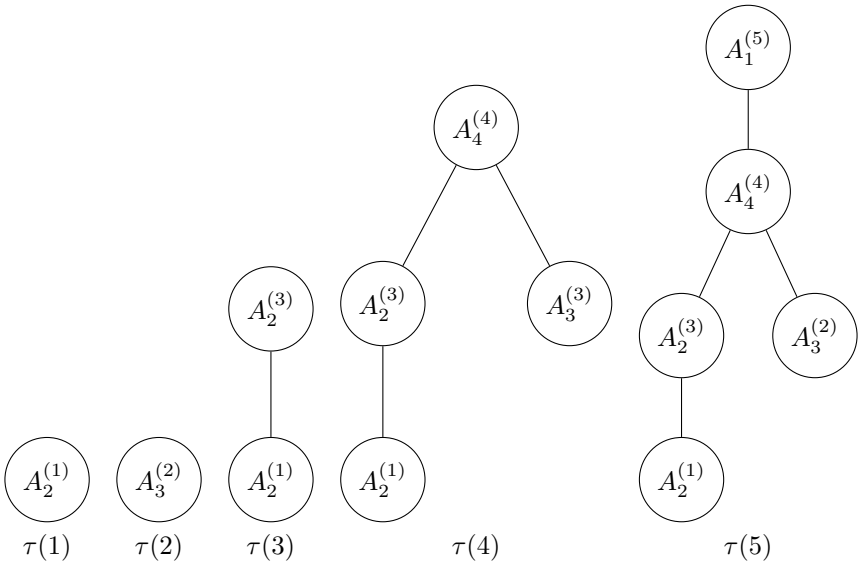
- $\tau^{(t)}(t)$ est un arbre qui a un seul sommet étiqueté par $Log(t)$.
- Pour tout $i \in [\![1, t-1]\!]$.
  Si $Log(i)$ est indépendant de tous les événements des étiquettes de $V(\tau^{(i+1)}(t))$, on pose $\tau^{(i)}(t) = \tau^{(i+1)}(t)$.
  Sinon, on choisit, arbitrairement, un sommet v de $\tau^{(i+1)}(t)$ de profondeur maximale parmi ceux dont l'étiquette n'est pas indépendante $Log(i)$. On définit alors $\tau^{(i)}(t)$ comme $\tau^{(i+1)}(t)$ où on rajoute un fils u, étiqueté par $Log(i)$, au nœ ud v.

On pose alors $\tau(t) = \tau^{(1)}(t)$ l'arbre témoin de l'exécution au temps t.

Exemple 4.3. Ici on donne un exemple de la construction des arbres témoins. Considérons les événements A_1, A_2, A_3, A_4 qui dépendent de certaines des variables aléatoires indépendantes X_1, X_2, X_3, X_4, X_5 de la manière présentée dans le tableau ci-dessous :

	A_1	A_2	A_3	A_4
X_1	1	0	1	1
X_2	1	1	0	0
X_3	0	0	1	0
X_4	0	1	0	0
X_5	0	1	0	1

Dans le tableau, les 1 signifient la dépendance et les 0 l'indépendance.
Supposons que la réévaluation se déroule dans cet ordre $(Log(t))_{1\leq t\leq 5} = (A_2, A_3, A_2, A_4, A_1)$, alors les arbres témoins corréspondant à cette exécution de l'algorithme sont illustrés par les dessins suivants :



Les 5 arbres ci-dessus sont les arbres témoins correspondant aux instants de $t = 1$ à $t = 5$ de gauche à droite. Les indices en haut entre parenthèses représentent l'étape i à laquelle on rajoute l'événement dans $\tau^{(i)}(t)$.

Ainsi, si on fixe un instant t de l'exécution de l'algorithme, l'arbre témoin $\tau(j)$ décrit toutes le réévaluations des variables X_i avant l'instant t et qui jouent un rôle sur le fait que l'événement $Log(t)$ soit satisfait à l'instant t :

Dans l'exemple 4.3, l'arbre $\tau(2)$ ne contient que A_3, car la modification des variables faite sur A_2 à $t = 1$ ne joue aucun rôle sur le fait que A_3 soit satisfait ou non à $t = 2$. Par contre, à $t = 4$, les modifications des évaluations des variables de A_2 et A_3 des étapes précédentes jouent un rôle pour déterminer si A_4 est satisfait ou non à $t = 4$.

4.1.2 Propriétés

Maintenant on a construit la suite d'arbres témoins associée à une exécution, on peut énoncer quelques propriétés de ces arbres qui permettront de caractériser une exécution de l'algorithme.

Tout d'abord, un arbre témoin doit vérifier certaines propriétés assez contraignantes :

Lemme 4.4. *Soit τ un arbre témoin fixé, $Log : \mathbb{N}^* \longrightarrow \mathcal{A}$ la suite (aléatoire) associée à une exécution de l'algorithme et $(\tau(t))_{t\in\mathbb{N}^*}$ la suite d'arbres témoins associée. On a les assertions suivantes :*

1. *Les arbres dans la suite $(\tau(t))_{t\in\mathbb{N}^*}$ sont tous différents.*
2. *Si τ apparaît dans la suite $\mathcal{L} = (\tau(t))_{t\in\mathbb{N}^*}$, alors les sommets se situant au même niveau de τ ont des étiquettes distinctes.*

Démonstration. Raisonnons par l'absurde. Supposons que l'arbre témoin τ apparaît deux fois dans $\mathcal{L}$ aux instants s et t avec $s < t$. On note A la racine de τ, qui est donc la racine de $\tau(s)$ et de $\tau(t)$. A est l'événement dont on a réévalué les variable aux instants s et t lors de l'exécution de l'algorithme. Or, par la méthode de construction des arbres témoins, le nombre de sommets étiquetés par A dans $\tau(s)$ est exactement le nombre de $i \in [\![1, s]\!]$ qui vérifient $Log(i) = A$, car A n'est pas indépendant de lui-même. Il en va de même pour l'arbre $\tau(t)$. Par conséquent, le nombre de fois qu'apparaît A dans $\tau(t)$ est supérieur à celui dans $\tau(s)$. D'où la contradiction.

Soit τ un arbre témoin qui apparaît dans $\mathcal{L}$, on note t l'instant auquel il apparaît, donc on a $\tau(t) = \tau$. Pour $v \in V(\tau)$, on note $d(v)$ la profondeur de v dans l'arbre, c'est-à-dire la distance entre v et la racine ; on note aussi $q(v)$ le plus grand entier q tel que $\tau^{(q)}(t)$ contienne v. Soient $u, v \in \mathcal{A}$ tels que $[u]$ et $[v]$ ne soient pas indépendants (donc différents), on peut supposer, par exemple, $q(u) < q(v)$. Au cours de la construction, lorsque l'on ajoute le sommet u à l'arbre $\tau^{(q(u)+1)}$, on choisit un sommet dont l'étiquette n'est pas indépendante de $[u]$ et qui est de profondeur maximale. On voit donc sans difficulté que $d(u) > d(v)$. En conséquence, les sommets qui se situent au même niveau dans un arbre témoin sont associés à des événements indépendants l'un de l'autre.

□

Le lemme suivant donne une majoration de la probabilité d'apparition d'un arbre témoin τ lors de l'exécution de l'algorithme. Cette majoration s'avère très utile et on s'en servira à plusieurs reprises dans la suite.

Lemme 4.5. *Soit τ un arbre témoin fixé et $Log : \mathbb{N}^* \longrightarrow \mathcal{A}$ la suite (aléatoire) associée à une exécution de l'algorithme. La probabilité que τ apparaisse dans $\mathcal{L} = (\tau(t))_{t\in\mathbb{N}^*}$ est au plus*

$$\prod_{v\in V(\tau)} \mathbb{P}([v])$$

Pour montrer ce résultat, on a besoin de regarder de plus près, et plus formellement, l'exécution de l'algorithme. On a des variables aléatoires indépendantes X_i, auxquelles on attribue des évaluations différentes à chaque étape de la boucle TANT QUE. En d'autres termes :

1. On a, pour chaque $i \in [\![1, m]\!]$ une suite de variables aléatoires indépendantes et identiquement distribuées à X_i, $(X_i^{(k)})_{k\in\mathbb{N}}$.
2. La « première évaluation » correspond à considérer $(X_1, \dots, X_m) = (X_1^{(1)}, \dots, X_m^{(1)})$. C'est l'évaluation au temps d'exécution $t = 1$.
3. « L'évaluation courante » au temps d'exécution $t+1$ correspond au m-uplet $(X_0^{(p_1+\epsilon_1)}, \dots, X_m^{(p_m+\epsilon_m)})$, où $(X_0^{(p_1)}, \dots, X_m^{(p_m)})$ est l'évaluation au temps t et $\epsilon_i = \mathbb{1}_{\{X_i \in vbl(Log(t))\}}$

L'idée est simple, mais s'avère un outil très pratique pour analyser l'algorithme : il s'agit de « préprocesser » les données aléatoires. En d'autres termes, on fait un tirage au hasard au début puis on fait tourner un algorithme déterministe avec ces entrées au lieu de faire des tirages au hasard au cours de l'exécution.

Lors de l'exécution de l'algorithme, la seule donnée aléatoire est celle des évaluations successives des X_i, le reste se faisant de façon déterministe. L'idée du « preprocessing » est la suivante : il suffit de connaître à l'avance les valeurs que prendront les X_i à chaque fois qu'on leur demande d'être réévaluées, en se donnant une famille de suites de valeurs, puis de faire tourner l'algorithme en lisant les valeurs des X_i sur ces suites au lieu de choisir une évaluation au hasard.

Voici un exemple pour illustrer cette idée :

Exemple 4.6. On considère le cas où on a 5 variables aléatoires X_1, X_2, X_3, X_4, X_5, à valeurs dans $\{0, 1\}$ desquelles dépendent les événements qui nous intéressent. L'évaluation courante à l'instant t est donnée par les chiffres encadrés : le passage de $t = k$ à $t = k + 1$ consiste donc à avancer d'une case sur les lignes des variables de A_j si on choisit de traiter A_j au temps $t = k$.

X_1	$\boxed{1}$	0	0	1	0	
X_2	$\boxed{1}$	1	0	0	1	
X_3	$\boxed{0}$	0	1	0	1	$\cdots$
X_4	$\boxed{0}$	1	0	0	1	
X_5	$\boxed{0}$	1	0	1	1	

$t = 0$

$\to X_1$	1		$\boxed{0}$	$\to$	0	1		0	
X_2	1		$\boxed{1}$		0	0		1	
$\to X_3$	$\boxed{0}$	$\to$	0		1	0		1	$\cdots$
$\to X_4$	0		1		0	$\boxed{0}$	$\to$	1	
X_5	0		1		0	$\boxed{1}$		1	

$\to X_1$	1		0	$\to$	$\boxed{0}$	1		0	
X_2	1		$\boxed{1}$		0	0		1	
$\to X_3$	0	$\to$	$\boxed{0}$		1	0		1	$\cdots$
$\to X_4$	0		1		0	0	$\to$	$\boxed{1}$	
X_5	0		1		0	$\boxed{1}$		1	

De $t = k$ à $t = k + 1$

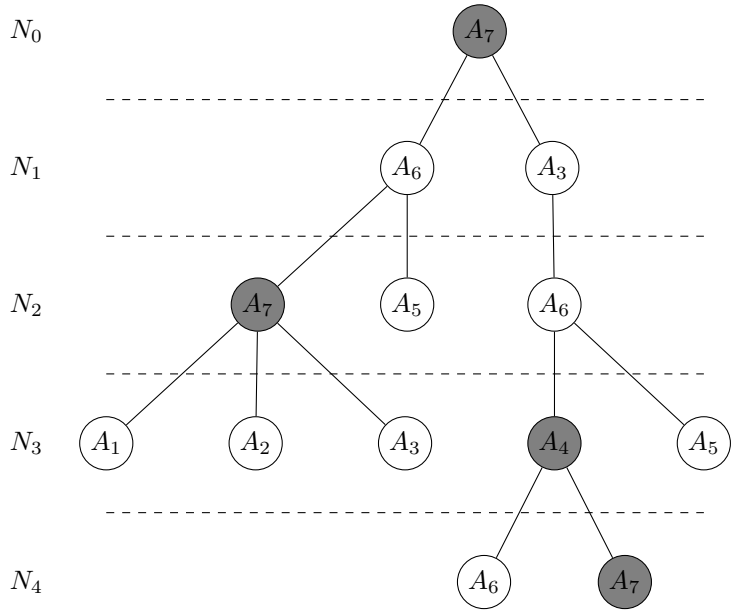
Dans ce cas, A_j dépend des variables X_1, X_3 et X_4.

Définition 4.7. Pour un arbre témoin τ de profondeur k fixé, on définit les entiers $q_{i,j}$ ($1 \leq i \leq m$, $0 \leq j \leq k$) par récurrence :

$$q_{i,k} = 0 \text{ pour tout i}$$
$$q_{i,j-1} = \begin{cases} q_{i,j} & \text{si tous les événements de } N_k \text{ sont indépendants de } X_i \\ q_{i,j} + 1 & \text{sinon} \end{cases}$$

Pour $1 \leq i \leq m$ et $0 \leq j \leq k$, $q_{i,j}$ est le nombre de nœuds dépendant de X_i de profondeur supérieure à j.

Exemple 4.8. On suppose que la variable aléatoire X_1 n'intervient que dans les événements A_4 et A_7. Par conséquent, dans cet arbre, la suite $(q_{1,j})_{0 \leq j \leq 4}$ est définie par : $(q_{1,j})_{0 \leq j \leq 4} = (3, 3, 2, 1, 0)$.



Maintenant, on peut donner une preuve du lemme 4.5 :

Démonstration. On se donne un arbre témoin τ. On remarque que les $q_{i,j}$ définis précédemment ne dépendent que de τ, et donc, à τ fixé, sont déterministes. On peut faire alors les remarques suivantes :

1. Si v apparaît à l'instant t, il est nécessaire que $[v]$ soit vérifié quand tous les $Log(j)$, pour $1 \leq j \leq t-1$, ont été traités.
2. Si l'arbre τ apparaît et v apparaît dans Log à l'instant t, alors tous les $Log(j)$ $(1 \leq j \leq t-1)$ dépendant de $[v]$ sont placés en dessous de v dans τ.
3. Si w est un nœud de τ dans un niveau en dessous de v et dépendant de v, alors $[w]$ est un des $Log(j)$ pour $1 \leq j \leq t-1$
4. Si τ apparaît et si $v \in \tau$ est de profondeur j dépendant de X_i, alors, à l'instant t où v apparaît dans Log, $X_i = X_i^{(q_{i,j})}$. C'est-à-dire, la variable X_i a été appelée $q_{i,j}$ fois depuis le début de l'exécution de l'algorithme.

Tenant compte de ces dernières remarques, on a que :

$$\begin{aligned}
\mathbb{P}(\tau \text{ apparaît dans } \mathcal{L}) &\leq \prod_{v \in V(\tau)} \mathbb{P}(v \mid \exists t \geq 0 \text{ tel que les } w \text{ en dessous de } v \text{ sont exactement les événements dépendant de } v \text{ traités avant } t \text{ et } Log(t) = [v]) \\
&\leq \prod_{v \in V(\tau)} \mathbb{P}(v \mid \text{les } w \in \tau \text{ en dessous de } v \text{ sont les seuls événements dépendant de } v \text{ qui ont été traités avant } v) \\
&\leq \prod_{j=1}^{k} \prod_{v \in N_j} \mathbb{P}(v \mid \forall X_i \in vbl([v]) X_i = X_i^{(q_{i,j})}) \\
&= \prod_{j=1}^{k} \prod_{v \in N_j} \mathbb{P}(v) \\
&= \prod_{v \in V(\tau)} \mathbb{P}(v)
\end{aligned}$$

□

Remarque 4.9. Le raisonnement de la preuve du lemme 4.5 reste vrai quand on remplace $\mathbb{P}$ par la probabilité conditionnelle sachant un événement G, puisqu'on ne considère que des inclusions (conditions nécessaires pour qu'un arbre donné apparaisse). La probabilité qu'un arbre τ apparaisse dans la suite $\mathcal{L} = (\tau_t)_{t \in \mathbb{N}^*}$ sachant un événement G est donc majorée par $\prod_{v \in \tau} \mathbb{P}(v \mid G)$.

En ayant l'outil des arbres témoins, on peut faire le calcul de complexité de l'algorithme. On fera deux calculs différents. Un pour le cas symétrique, qu'on peut trouver dans [3] et qui permet de voir la structure des arbres témoins, et un autre pour le cas général, donnée dans [2], qui se base sur un argument de probabilité d'un processus de Galton-Watson. Les preuves étant élégantes et fondées sur des arguments différents, nous avons décidé de présenter les deux.

4.2 Calcul de complexité (Cas symétrique)

On donne ici une preuve de la complexité de l'algorithme dans le cas symétrique. On fait donc les hypothèses suivantes, pour un $p \in [0, 1[$:

- Pour tout $A \in \mathcal{A}$, $\mathbb{P}(A) \leq p$
- Pour tout $A \in \mathcal{A}$, $\Gamma(A)$ est de cardinal $\leq d$.
- $p\frac{(d+1)^{d+1}}{d^d} \leq ep(d+1) \leq 1$

On considére ici un arbre infini $\mathcal{T}$, où chaque nœud a exactement $d+1$ fils.

Définition 4.10.
1. Pour un arbre fini τ, on note sa taille (le nombre de sommets) $|\tau|$.
2. On note $S(\mathcal{T})$ l'ensemble des sous-arbres (avec racine) de $\mathcal{T}$.
3. Pour $n \in \mathbb{N}$, on note $S_n(\mathcal{T})$ l'ensemble des sous-arbres de $\mathcal{T}$ de hauteur $\leq n$.
4. Pour $A \in \mathcal{F}$, on définit la variable aléatoire L_A comme étant le nombre de fois où l'événement A apparaît dans la suite *Log*.

Lemme 4.11. *Pour $A \in \mathcal{F}$, on a*

$$\mathbb{E}(L_A) \leq \sum_{\tau \in S(\mathcal{T})} p^{|\tau|}$$

.

Démonstration. L_A est exactement le nombre d'arbres T avec racine A qui apparaissent dans la suite $\mathcal{L} = (\tau(t))_{t \in \mathbb{N}^*}$. Or, d'après les lemmes 4.4 et 4.5, ces arbres sont tous distincts, et chacun apparaît avec probabilité $\leq p^{|T|}$. Or, en étiquetant la racine de $\mathcal{T}$ avec A et les fils d'un de ses nœuds v par les événements qui ne sont pas indépendants de $[v]$, on a une injection des arbres qui peuvent apparaître dans $\mathcal{L}$ dans les sous-arbres de T. Donc, par la linéarité de l'espérance, on a $\mathbb{E}(L_A) \leq \sum_{\tau \in S(\mathcal{T})} p^{|\tau|}$. □

Proposition 4.12. *On suppose $d \geq 1$. Si on note $\Phi : x \mapsto p(1+x)^{d+1}$ et $y = \sum_{\tau \in S(\mathcal{T})} p^{|\tau|}$, alors*

1. *Φ admet exactement un point fixe dans $]0, \frac{1}{d}]$ (et éventuellement un autre dans $]\frac{1}{d}, +\infty[$).*
2. *y est le point fixe de Φ qui appartient à $]0, \frac{1}{d}]$. En particulier, y est fini.*

Démonstration.
1. Puisque $d \geq 1$, Φ est strictement convexe et croissante. De plus $\Phi(\frac{1}{d}) = p(\frac{d+1}{d})^{d+1} = \frac{1}{d} p \frac{(d+1)^{d+1}}{d^d} \leq \frac{1}{d}$ et $\Phi(0) = p \geq 0$, d'où l'existence et l'unicité du point fixe.
2. On pose, pour $s \in \mathbb{N}$, $y_s = \sum_{\tau \in S_s(\mathcal{T})} p^{|\tau|}$. On a alors clairement que $y_s \underset{s \to \infty}{\longrightarrow} y$, et que $y_0 = 0$. Pour $s \geq 0$, on considère un sous-arbre de taille $\leq s+1$ de $\mathcal{T}$ comme sa racine, à laquelle sont accrochées des sous-arbres de taille $\leq s$. D'où

$$y_{s+1} = \sum_{i=0}^{d+1} p \binom{d+1}{k} y_s = p(1+y_s)^{d+1} = \Phi(y_s).$$

Comme Φ est croissante, $(y_s)_{s \in \mathbb{N}}$ tend vers le point fixe de Φ qui se trouve dans $]0, \frac{1}{d}]$.

□

On peut maintenant calculer l'espérance du temps d'exécution de l'algorithme.

Proposition 4.13. *En notant $|Log|$ la longueur de la suite Log, on a*

$$\mathbb{E}(|Log|) \leq \frac{n}{d}$$

En particulier, l'algorithme se termine presque sûrement.

Démonstration. On a $\mathbb{E}(|Log|) = \sum_{i=1}^{n} \mathbb{E}(L_{A_i}) \leq \frac{n}{d}$ par la linéarité de l'espérance et la proposition 4.12 □

On utilisera un argument tout à fait différent pour montrer le cas général.

4.3 Calcul de complexité (Cas général)

Ici on va considérer une variante du processus de Galton-Watson. On se donne un événement $A \in \mathcal{A}$ et on définit l'arbre aléatoire généré par ce processus de la manière suivante :

- A l'instant initial, l'arbre ne contient qu'un sommet, qui est sa racine, étiqueté par A.
- A l'instant $t+1$, à tout v sommet né à l'instant t on rajoute un nœud étiqueté par $B \in \Gamma^+([v])$ en tant que fils avec probabilité $x(B)$, de sorte que l'ajout de ces fils soient des événements indépendants.
- On fait l'étape de récurrence jusqu'à ce qu'aucun fils naisse, et l'arbre aléatoire de ce processus est ce dernier arbre.

On va donner une majoration pour la probabilité qu'un arbre témoin τ soit généré par ce processus. Ceci nous servira après pour calculer la complexité de l'algorithme ainsi que l'espérance du nombre d'arbres témoins dans $\mathcal{L}$ de racine étiquetée par un événement de $\mathcal{A}$.

Lemme 4.14. *Soit τ un arbre témoin fixé et d'étiquette A à la racine, alors la probabilité que le processus de Galton-Watson produise exactement l'arbre τ est*

$$p_\tau = \frac{1-x(A)}{x(A)} \prod_{v \in V(\tau)} x'([v]) \quad \text{où} \quad x'(A) = x(A) \prod_{B \in \Gamma^+(A)} (1-x(B)) \text{ pour tout } A \in \mathcal{A}.$$

Démonstration. Pour tout $v \in V(\tau)$, on désigne par $W_v \subseteq \Gamma^+([v])$ le complémentaire de l'ensemble des fils de v dans $\Gamma^+([v])$. On a l'égalité suivante :

$$p_\tau = \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in W_v} (1-x([u])) \right)$$

car les naissances des nœuds sont des événements indépendants et le sommet d'étiquette A apparaît toujours à la racine, d'où la division par $x(A)$. On veut se débarrasser des ensembles W_v. Pour cela, fixons un $v \in V(\tau)$,

$$\prod_{u \in W_v} (1-x([u])) = \prod_{u \text{ fils de } v} \frac{1}{1-x([u])} \prod_{u \in \Gamma^+(v)} (1-x([u])).$$

Comme chaque nœud sauf la racine est exactement un fils de l'un des nœuds, en faisant le produit sur tous les $v \in V(\tau)$, on obtient :

$$p_\tau = \frac{1-x(A)}{x(A)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1-x([v])} \prod_{u \in \Gamma^+(v)} (1-x([u])) \right),$$

qui se réécrit encore :

$$p_\tau = \frac{1-x(A)}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in \Gamma(v)} (1-x([u])) \right).$$

D'où l'égalité cherchée. □

Avec la majoration précédente, on peut estimer l'espérance de longueur de *Log*, ce qui permettra de conclure que l'algorithme randomisé se termine en temps linéaire en espérance.

Proposition 4.15. *En notant $|Log|$ la longueur de la suite Log, on a*

$$\mathbb{E}(|Log|) \leq \sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)}.$$

En particulier, l'algorithme se termine presque sûrement.

Démonstration. Pour tout $A \in \mathcal{A}$, on note N_A le nombre d'apparitions de A dans la suite Log : c'est aussi le nombre d'appels de réévaluation sur les variables de l'événement A. On désigne aussi par $\mathcal{T}_A$ l'ensemble des arbres témoins de racine étiquetée par A. Ainsi,

$$\mathbb{E}(N_A) = \sum_{\tau \in \mathcal{T}_A} \mathbb{P}(\tau \text{ apparaît dans } \mathcal{L}) \leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \mathbb{P}([v]) \leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x'([v]),$$

où la première inégalité vient du lemme 4.5 et la seconde de l'hypothèse 3.1. Puis,

$$\mathbb{E}(N_A) \leq \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau \leq \frac{x(A)}{1 - x(A)}$$

où la première inégalité est la réécriture de la formule précédente et la seconde du fait que le processus Galton-Watson qu'on a considéré ne génère qu'un arbre en une fois. Pour finir, en sommant sur tous les événements $A \in \mathcal{A}$,

$$\mathbb{E}(|Log|) = \sum_{A \in \mathcal{A}} \mathbb{E}(N_A) \leq \sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)}.$$

L'espérance de la longueur de la suite Log est finie, donc la longueur de la suite est finie presque sûrement, c'est-à-dire l'algorithme se termine presque sûrement. □

Remarque 4.16. Si on se place dans le cas symétrique du lemme local, en reprenant la démonstration 3.7, on peut remplacer les $x(A)$ par $\frac{1}{d+1}$ où d désigne le degré de dépendance maximal. Dans ce cas particulier, $\mathbb{E}(|Log|) \leq \frac{n}{d}$.

5 Variante déterministe

On vient de voir un algorithme randomisé de complexité linéaire : la question de la dérandomistion reste à traiter. Dans l'article de Moser [2], on présente une variante déterministe de l'algorithme de la section 3.2, qui est polynomiale en m et n à d fixé. On utilisera les méthodes de la section 1.3 de façon importante.

On montrera d'abord quelques lemmes préliminaires sur les arbres témoins, pour pouvoir ensuite proposer un algorithme déterministe correct et qui se termine. On vérifiera finalement que sa complexité est polynomiale.

5.1 Lemmes préliminaires

Il est possible de dérandomiser notre algorithme sous quelques hypothèses supplémentaires :
– On suppose qu'il existe $\epsilon > 0$ tel que

$$\forall A \in \mathcal{A}, \mathbb{P}(A) \leq (1 - \epsilon) x(A) \prod_{B \in \Gamma(A)} (1 - x(B)).$$

– On peut encore supposer que le maximum des $x(A)$ est strictement inférieur à 1. Sinon, il suffit de poser $\tilde{x}(A) = (1 - \frac{\epsilon}{2})x(A)$ pour tout $A \in \mathcal{F}$, et alors $\forall A \in \mathcal{A}$,

$$\frac{(1-\frac{\epsilon}{2})\tilde{x}(A)\prod_{B\in\Gamma(A)}(1-\tilde{x}(B))}{(1-\epsilon)x(A)\prod_{B\in\Gamma(A)}(1-x(B))} \geq \frac{(1-\frac{\epsilon}{2})\tilde{x}(A)}{(1-\epsilon)x(A)} = \frac{(1-\frac{\epsilon}{2})^2}{1-\epsilon} \geq 1.$$

Et donc,

$$\forall A \in \mathcal{A}, \mathbb{P}(A) \leq (1 - \frac{\epsilon}{2})\tilde{x}(A) \prod_{B\in\Gamma(A)} (1 - \tilde{x}(B)).$$

On note $(X_i^{(j)})_{1\leq i\leq n, j\geq 1}$ la liste de variables aléatoires dont $X_i^{(j)}$ désigne la valeur que prend X_i lors de la j-ième évaluation.

Dans la suite, on ne regardera que les arbres qui ont une chance d'apparaître, c'est pour cela qu'on définit la notion de consistance :

Définition 5.1. On dit que l'arbre témoin τ est consistant avec $(X_i^{(j)})_{1\leq i\leq n, j\geq 1}$ si la τ-vérification passe, c'est-à-dire :

1. On procède par récurrence à partir du niveau le plus profond et on attibue les valeurs correspondantes (*cf.* lemme 4.5 et exemple 4.6) dans $(X_i^{(j)})_{1\leq i\leq n, j\geq 1}$ aux variables aléatoires $(X_i)_{1\leq i\leq n}$.
2. Si tous les événements dans l'arbre τ sont vérifiés, alors on dit que la τ-vérification passe.

Le but est d'éliminer les arbres témoins de taille supérieure à une certaine constante, pour ne garder qu'un nombre fini d'arbres témoins possibles. Ceci impliquera que l'algorithme se termine. On commence par majorer le nombre d'arbres témoins possédant au moins k nœuds.

Lemme 5.2. *L'espérance du nombre d'arbres témoins possédant au moins k nœuds qui sont consistants, notée $Q(k)$, est bornée par $(1-\epsilon)^k \sum_{A\in\mathcal{F}} \frac{x(A)}{1-x(A)}$. En particulier, si les $x(A)$ sont majorés uniformément par une constante strictement inférieure à 1, $Q(k) = \mathcal{O}(n(1-\epsilon)^k)$*

Démonstration. On note $\mathcal{T}_A(k)$ l'ensemble des arbres témoins possédant au moins k nœuds.

$$Q(k) \leq \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \mathbb{P}(\tau \text{ apparaît dans } \mathcal{L}) \leq \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} \mathbb{P}([v])$$

par le lemme 4.5. Or, pour tout $v \in V(\tau)$, $\mathbb{P}([v]) \leq (1-\epsilon)x'([v])$. Donc

$$Q(k) \leq (1-\epsilon)^k \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} x'([v]) = (1-\epsilon)^k \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)} \sum_{\tau\in\mathcal{T}_A(k)} p_\tau$$

où la dernière égalité découle du lemme 4.14. Donc,

$$Q(k) \leq (1-\epsilon)^k \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)}$$

La somme étant bornée, $Q(k) = \mathcal{O}(n(1-\epsilon)^k)$. □

On veut se restreindre à un intervalle où il y a peu d'arbres témoins consistants. On obtient une borne en $\log n$, ce qui permettra, dans la section 5.3, d'avoir un temps polynomial.

Lemme 5.3. *On peut trouver une constante c, indépendante de n, telle que, en espérance, le nombre d'arbres témoins consistants de taille supérieure à* $c\ln(n)$ *soit inférieur à* $\frac{1}{2}$*. En conséquence, avec probabilité au moins* $\frac{1}{2}$*, il n'existe aucun arbre témoin consistant de taille supérieure à* $c\ln(n)$*.*

Démonstration. Par le lemme précédent, $Q(k) = \mathcal{O}(n(1-\epsilon)^k)$. Soit a une constante positive telle que $Q(k) \leq an(1-\epsilon)^k$. On veut majorer $an(1-\epsilon)^k$ par $1/2$.

$$an(1-\epsilon)^k \leq \frac{1}{2} \Leftrightarrow \ln a + \ln n + k\ln(1-\epsilon) \leq -\ln 2 \Leftrightarrow k \geq \frac{-\ln 2 - \ln k - \ln n}{\ln(1-\epsilon)}$$

D'où l'existence d'une constante $c > 0$ telle que $Q(c\ln n) \leq 1/2$. On peut remarquer aussi que cette constante se calcule facilement (en temps constant).
Le deuxième point est une conséquence de l'inégalité de Markov. □

Grâce au lemme précédent, on obtient une borne à partir de laquelle il y a peu d'arbres témoins possibles. Cependant, on ne peut pas considérer tous les arbres de taille supérieure à cette borne car il peut y en avoir une infinité. Le lemme suivant permettra de se restreindre à un intervalle fini pour vérifier qu'il n'y a aucun arbre témoin consistant de taille supérieure à la borne.

Lemme 5.4. *Supposons que le degré de dépendance est majoré par d, i.e. pour tout* $A \in \mathcal{A}, |\Gamma(A)| \leq d$*. Soit* $u \in \mathbb{N}$*. S'il n'existe aucun arbre témoin de taille comprise dans* $[u, (d+1)u]$ *consistant avec l'évaluation* $(X_i^{(j)})_{1\leq i\leq n, j\geq 1}$*, alors il n'existe aucun arbre témoin de taille supérieure ou égale à u consistant avec la même évaluation.*

Démonstration. Raisonnons par contradiction. Supposons que ceci est faux pour un certain u et prenons τ un contre-exemple de taille minimale. Ainsi τ est de taille strictement supérieure à $(d+1)u$ et il n'existe aucun arbre témoin consistant de taille dans $[u, (d+1)u]$. Le degré de dépendance étant borné, chaque nœud de τ possède au plus $d+1$ fils. On note $w_1, ..., w_j, j \leq d+1$ les fils de la racine. On note *Log* la suite aléatoire associée à l'arbre témoin τ, de longueur $|\tau|$. Pour tout $i \in [\![1, j]\!]$, on construit, à partir de la suite *Log*, un arbre témoin $\tau_i = \tau(q(w_i))$, où $q(v)$ est le plus grand entier q tel que $\tau^{(q)}(t)$ contienne v. Il est évident que les τ_i sont consistants avec l'évaluation $(X_i^{(j)})_{1\leq i\leq n, j\geq 1}$. En plus, τ_i contient au moins les nœuds du sous-arbre w_i de τ, dont le cardinal est au moins $\frac{|\tau|-1}{d+1}$, qui est supérieur ou égal à u. Donc τ_i est soit un arbre témoin consistant de taille dans $[u, (d+1)u]$, soit un arbre témoin qui contredit la minimalité de τ. Contradiction. □

5.2 Algorithme déterministe

À l'aide des lemmes qu'on vient de montrer, on peut construire une variante déterministe et en temps polynomial (en m et n à d fixé) de l'algorithme pour trouver une évaluation des X_i qui ne satisfasse aucun $A \in \mathcal{A}$. L'idée est de faire tourner l'algorithme précédent, mais en choisissant de façon déterministe les valeurs que prennent les X_i, de sorte qu'il n'y ait pas d'arbres témoins consistants à partir d'un certain rang. Ceci garantit la terminaison de la variante déterministe. Si les évaluations et le choix des arbres à rendre inconsistants sont faits avec soin, l'algorithme sera en temps polynomial.

L'algorithme se déroule en deux temps :

1. Créer un tableau F d'évaluations des X_i.

2. Faire tourner l'algorithme de la section 3.2 avec les évaluations de F à la place des affectations aléatoires.

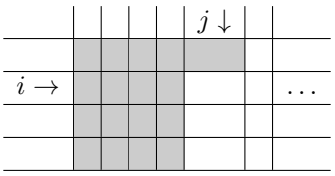
La deuxième étape a déjà été traitée : il reste à construire le tableau (ou les m listes) F.

5.2.1 Calcul avec des espérances conditionnelles

Dans un premier temps, on fera l'hypothèse qu'on sait calculer les espérances conditionnelles pour utiliser la méthode du 1.3.1. Plus précisément :

- On se donne les n événements et les m variables aléatoires $X_1, \ldots, X_m$.
- On calcule la constante c de la section précédente (lemme 5.3).
- On énumère tous les arbres témoins de taille dans $[c\log(n), (d+1)c\log(n)]$ dans une liste L, puis on note $\#L$ le nombre d'arbres consistants dans L.

On définit l'événement $G_{i,j}$ par : $G_{i,j} = \{\forall j < t \ \forall 1 \leq i \leq m \ X_i^{(j)} = \mathsf{F}_{i,j} \ , \ X_1^{(t)} = \mathsf{F}_{1,t}, \ \ldots \ , \ X_{i-1}^{(t)} = \mathsf{F}_{i-1,t}\}$. Cet événement correspond à un remplissage partiel du tableau F : dans $G_{i,j}$, on connaît les valeurs de certains X_k^l, qui sont données par le remplissage de F :



On note aussi, pour $t \in \mathbb{N}$, $E_t = E(\#L \mid G_{m,t})$, l'espérance conditionnelle du nombre d'arbres consistants dans L sachant les t premières évaluations des X_i.

On procède ensuite comme suit :

Algorithme : MOSER_ DETERMINISTE

$t \leftarrow 0$
TANT QUE $E_t > 0$ faire
    POUR i allant de 1 à m faire
        Choisir x_i qui minimise $E(\#L \mid G_{i,t} \ , \ X_i^{(t)} = x_i)$
        $\mathsf{F}_{i,t} \leftarrow x_i$
    Fin POUR
    $t \leftarrow t + 1$
Fin TANT QUE

En d'autres mots, à chaque étape, indexée par t, on choisit des valeurs pour l'évaluation suivante des X_i en minimisant l'espérance conditionnelle du nombre d'arbres consistants dans L. On peut alors faire quelques remarques :

Remarque 5.5.

1. Cette partie de l'algorithme est bel et bien déterministe : les calculs d'espérances conditionnelles n'ont pas besoin de données aléatoires.
2. Comme l'espérance conditionnelle est $\leq \frac{1}{2}$ au départ, et qu'on la minimise à chaque étape, $E_t \leq \frac{1}{2}$ pour tout $t \geq 0$.
3. Comme la taille des arbres dans L est bornée par $c(d+1)\log(n)$, le nombre d'évaluations des X_i qui ont une influence sur la consistance de ces arbres est borné par la même constante. (On pourra donc fixer la taille de F).

4. Une fois décidées les $c(d+1)\log(n)$ premières évaluations, la suite E_t stationne au nombre d'arbres dans L qui sont consistants avec *toutes* les évaluations qui coïncident avec celles données par F à cet instant.
5. Par conséquent, à partir du temps $c(d+1)\log(n)$, E_t est un entier positif $\leq \frac{1}{2}$, donc nul. L'algorithme se termine donc.

Avec cette construction et les lemmes de la section 5.1, on s'assure d'avoir un algorithme qui se termine. En effet, la construction du tableau d'évaluations F et le lemme 5.3 assurent que, en prenant ces valeurs pour les X_i, il n'y aura aucun arbre consistant de taille dans l'intervalle $[c\log(n), (d+1)c\log(n)]$. Or, d'après le lemme 5.4, ceci implique qu'il n'y a pas d'arbre témoin de taille supérieure à $c\log(n)$ consistant avec les évaluations de F. La suite $\mathcal{L}$ de la deuxième étape ne contient donc que des arbres de taille bornée : elle est alors finie. Par conséquent, l'algorithme se termine.

5.2.2 Algorithme polynomial

Dans la section 5.2.1, on a utilisé fortement le calcul des espérances conditionnelles. Cependant, ce calcul n'est pas évident à faire. On va donc ajuster l'algorithme pour que tous les calculs soient simples à effectuer et qu'il tourne en temps polynomial en n.
Grâce à la proposition 1.12, il suffit de trouver un majorant des espérances conditionnelles d'apparition des arbres de L. Ce majorant est donné par le lemme 4.5. On pose alors $f^\tau_{jm+i} = \prod_{v\in\tau} \mathbb{P}([v] \mid G_{i,j})$, avec $\tau \in L$ et $G_{i,j}$ donné par $\mathsf{F}_{u,v} = y_{mv+u}$ pour tous les u, v tels que $mv+u < mj+i$. Si de plus $t = \frac{1}{2}$, on se trouve dans la situation de la proposition 1.12.
En effet, on a bien :

1. $f^\tau_{j-1}(y_1, ..., y_{j-1}) \geq \min_{y\in\mathcal{Y}} f^\tau_j(y_1, ..., y_{j-1}, y)$ pour tout $\tau \in L$ et tous $y_1, \ldots, y_{j-1}$ par définition des probabilités conditionnelles.
2. $f^\tau_j(y_1, ..., y_j) \geq \mathbb{P}(\tau$ apparaît dans $\mathcal{L} \mid Y_1 = y_1, ..., Y_j = y_j)$ pour tout $\tau \in L$ et tous $y_1, \ldots, y_j$ car le produit majore l'espérance conditionnelle (lemme 4.5).
3. $\sum_{\tau\in L} f^\tau_0 \leq \frac{1}{2}$, d'après le choix de c (lemme 5.3).

Si on note, pour $t \in \mathbb{N}$, $S_t = \sum_{\tau\in L} \prod_{v\in V(\tau)} \mathbb{P}([v] \mid G_{m,t})$, on a la version suivante de l'algorithme :

Algorithme : MOSER_ POLYNOMIAL

$t \leftarrow 0$
TANT QUE $S_t > 0$ faire
    POUR i allant de 1 à m faire
        Choisir x_i qui minimise $\sum_{\tau\in L} \prod_{v\in V(\tau)} \mathbb{P}([v] \mid G_{i,t}\ ,\ X_i^{(t)} = x_i)$
        $\mathsf{F}_{i,t} \leftarrow x_i$
    Fin POUR
    $t \leftarrow t+1$
Fin TANT QUE

Les mêmes arguments de la remarque 5.5 permettent de dire que cette version de l'algorithme termine et est correcte. Le seul changement est le nombre qu'on minimise, mais celui-ci majore la probabilité conditionnelle, est décroissant et inférieur à $\frac{1}{2}$. La modification est faite pour que les calculs soient faits en un temps polynomial, comme on montrera dans la section suivante.

5.3 Complexité de la version déterministe

On veut montrer que l'algorithme MOSER_ POLYNOMIAL se termine en temps polynomial. On commence par montrer qu'il n'y a qu'un nombre polynomial d'arbres témoins dans l'intervalle qu'on considère. C'est une condition nécessaire parce qu'on les passe en revue pour construire le tableau F.

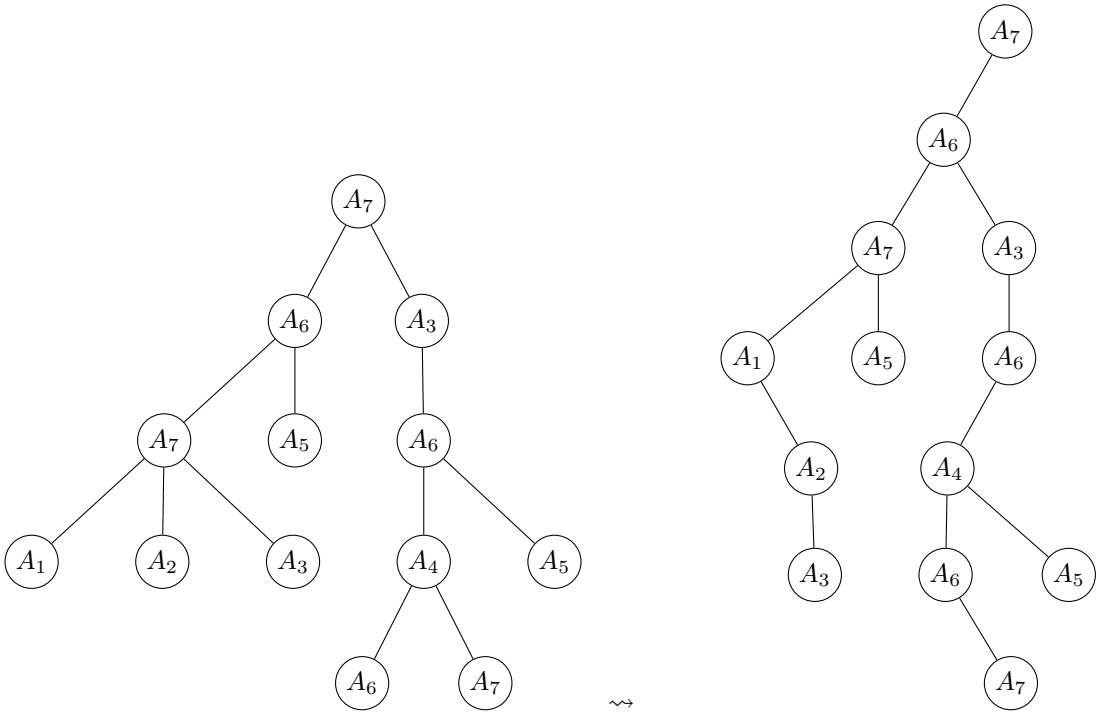
Lemme 5.6. *Il existe un nombre polynomial d'arbres témoins de taille comprise dans l'intervalle* $[c \ln n, (d+1)c \ln n]$.

Etant donné un arbre témoin τ, on peut le transformer en un arbre binarie T contenant les mêmes nœuds. La méthode est la suivante :

1. On donne un ordre des frères dans chaque génération.
2. La racine de T est celle de τ.
3. Si v est un nœud de T, alors le fils à gauche de v dans T est le plus grand fils de v dans τ. Si v est une feuille dans τ, il n'a pas de fils à gauche.
4. Si v est un nœud de T, alors le fils à droite de v dans T est le frère suivant dans τ. Si v est le plus petit dans sa génération, il n'a pas de fils à droite.

On peut remarquer en plus que cette transformation est une bijection.

Exemple 5.7. Ici, on transforme l'arbre de l'exemple 4.8 en arbre binaire. Le « fils à droite » d'un nœud est le premier de ses fils et le « fils à gauche » est le frère suivant. Ainsi, les nœuds A_6 et A_3 des deux premiers niveaux sont les fils de la racine A_7.



Maintenant, on peut donner une preuve au lemme 5.6.

Démonstration. Dans la suite, on note S_N le nombre d'arbres témoins de taille N, qui est majoré par le nombre d'arbres binaires de taille N. On sait qu'il existe $C(N) = \frac{\binom{2N}{N}}{N+1} = \mathcal{O}(\frac{4^N}{N^{3/2}}) = \mathcal{O}(4^N)$ structures différentes d'arbres binaires non étiquetés. Il faut alors établir une majoration pour le nombre d'étiquettes possibles qu'on peut avoir à chaque nœud. Pour la racine, on a n choix : tous les A_i sont possibles. Pour un nœud interne ou une feuille, on a au plus $d+1$ choix : en effet, l'étiquette d'un tel nœud doit être un A_i dépendant de l'étiquette qui correspond, dans l'arbre témoin, à son père. On obtient ainsi une majoration du nombre d'arbres témoin possédant N nœuds : $S_N \leq \mathcal{O}(C(N)n(d+1)^N)$. En sommant sur l'intervalle entier, on parvient à

$$\begin{aligned} S &= \sum_{N=c\ln n}^{(d+1)c\ln n} S_N \leq \sum_{N=c\ln n}^{(d+1)c\ln n} \mathcal{O}((4(d+1))^N n) \\ &= \mathcal{O}((4(d+1))^{c(d+1)\ln n} n) = \mathcal{O}(n^{c(d+1)\ln(4(d+1))+1}) \end{aligned}$$

□

Proposition 5.8. *L'algorithme déterministe de Moser* MOSER_ POLYNOMIAL *a une complexité polynomiale en m et n à d fixé.*

Démonstration. On doit montrer deux choses :

1. La construction du tableau F se fait en temps polynomial.
2. Une fois le tableau construit, l'algorithme tourne en temps polynomial.

On remarque que si le tableau F est construit, il est de taille $mc \ln n$. Donc, lors de l'exécution de l'algorithme sur cette entrée, il y aura au plus $mc \ln n$ boucles. Or chaque boucle consiste à tester s'il y a des A_i vérifiés et, le cas échéant, réaffecter les variables de $vbl(A_i)$, ce qui se fait en temps $\mathcal{O}(n+m)$. On a alors une complexité polynomiale en n et m.

Il reste à montrer que la construction de F se fait en temps polynomial. Par le lemme 5.3, la constante c se calcule en temps linéaire. D'après le lemme 5.6, énumérer les arbres témoins de taille comprise dans l'intevalle $[c \ln n, (d+1)c \ln n]$ se fait en temps polynomial (en faisant un parcours en profondeur). Maintenant, pour remplir une case $\mathsf{F}_{i,j}$, il suffit de calculer les majorations $\prod_{v \in V(\tau)} \mathbb{P}(v \mid G_{i,j})$, d'en faire l'addition et de les comparer. Comme on considère un nombre polynomial d'arbres, tous de taille inférieure à $(d+1)c \ln n$, cette opération se fait en temps polynomial. Finalement, comme il faut remplir $m(d+1)c \ln n$ cases, la construction de F se fait en temps polynomial. □

Références

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley-Blackwell, aug 2008.

[2] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. may 2009.

[3] Joel H. Spencer. Robin Moser makes Lovász local lemma algorithmic !

A Construction des X_i

Dans la section 3.1, nous avons fait une hypothèse sur l'espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$ dans lequel on travaille, à savoir que, étant donnés des événements $A_1, \dots, A_n$ il existe un nombre fini de variables aléatoires indépendantes $X_1, \dots, X_m$ à valeurs dans un ensemble fini telles que les A_j soient déterminés par les X_i. C'est-à-dire, que chaque $\mathbb{1}_{A_j}$ soit une fonction de $X_1, \dots, X_n$. Dans cet appendice, nous montrerons que cette situation tout à fait générale, quitte à faire grossir un peu l'espace de probabilité.

La condition dont on a besoin est d'avoir la possibilté de « découper » toute ensemble mesurable. C'est-à-dire, que notre espace n'ait pas d'atomes. Voici une façon de procéder pour pouvoir supposer cela.

Définition A.1. Soit $(X, \mathcal{F}, \mu)$ un espace mesuré. On dit que $A \in \mathcal{F}$ est un *atome* pour μ si $0 < \mu(A) < \infty$ et pour tout $B \in \mathcal{F}$ tel que $B \subseteq A$, $\mu(B) = 0$ ou $\mu(B) = \mu(A)$.

Lemme A.2. *Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace de probabilité. Alors $\Omega \times [0,1]$, muni de la tribu produit, n'admet pas d'atomes.*

Démonstration. Soit $A \in \mathcal{F} \otimes \mathcal{B}([0,1])$ tel que $\mathbb{P}(A) > 0$. Notons, pour $(\omega, x) \in \Omega \times [0,1]$ et $u \in [0,1]$, $f(u,(\omega,x)) = \mathbb{1}_A(\omega,x)\mathbb{1}_{[0,u]}(x)$. On remarque que $f(u_0,.)$ est mesurable pour tout $u_0 \in [0,1]$, $f(.,(\omega_0,x_0))$ est continue en tout $u \in [0,1] \setminus \{x_0\}$ (et donc presque partout), et que $|f| \leq 1$. Par le théorème des intégrales dépendant d'un paramètre, $F(u) = E[f(u,(\omega,x))]$ est continue. Or $F(0) = 0$ et $F(1) = \mathbb{P}(A)$. Il existe donc $u_0 \in [0,1]$ tel que $F(u_0) = \frac{\mathbb{P}(A)}{2}$. En posant $B = A \cap \Omega \times [0,u_0]$, on a $B \subset A$ et $\mathbb{P}(B) = \frac{\mathbb{P}(A)}{2}$. Donc A n'est pas un atome. □

On aura aussi besoin de choisir des parties de mesure arbitraire. C'est pourquoi nous avons besoin d'un résultat intermédiaire :

Lemme A.3. *Soit $(X, \mathcal{F}, \mu)$ un espace mesuré avec $\mu(X) < \infty$. S'il n'admet pas d'atomes, alors il existe une partition de X en deux parties mesurables de mesure $\frac{\mu(X)}{2}$.*

Démonstration. On raisonne par l'absurde.

On pose $\mathcal{P}$ l'ensemble des partitions de X en ensembles mesurables et contenant une partie de mesure $\geq \frac{\mu(X)}{2}$. Cette partie est alors unique, car on suppose qu'il n'existe pas de partie mesurable de X de mesure $\frac{\mu(X)}{2}$. Pour une partition $P \in \mathcal{P}$, on la notera C_P.

On définit une relation d'ordre partiel sur les éléments de P de la manière suivante : si $A, B \in \mathcal{P}$ ont pour parties de mesure $\geq \frac{\mu(X)}{2}$ C_A et C_B respectivement, alors $A \prec B \Leftrightarrow C_A \supset C_B$ et $\mu(C_A) > \mu(C_B)$. C'est clairement un ordre partiel sur $\mathcal{P}$. Montrons qu'il est inductif. Soit $\mathcal{Q}$ une partie de $\mathcal{P}$ totalement ordonnée. Soit $m = \inf_{Q \in \mathcal{Q}} \mu(C_Q) \geq \frac{\mu(X)}{2}$. On a deux cas :

- Soit m est atteint par une partition Q, qui est alors un majorant de $\mathcal{Q}$ pour $\prec$.
- Soit m n'est pas atteint. Il existe alors une suite $(Q_n)_{n\in\mathbb{N}}$ d'éléments de $\mathcal{Q}$ telle que $(\mu(C_{Q_n}))_{n\in\mathbb{N}}$ décroît strictement vers m. On sait alors que $(C_{Q_n})_{n\in\mathbb{N}}$ est une suite décroissante de parties mesurables de X. On pose $C_\infty = \bigcap_{n\in\mathbb{N}} C_{Q_n}$. C'est une partie mesurable de X telle que $\mu(C_\infty) = m \geq \frac{\mu(C)}{2}$. La partition $R = \{C_\infty, (X \setminus C_\infty)\}$ est alors dans $\mathcal{P}$ et est un majorant de $\mathcal{Q}$. En effet, si $Q \in \mathcal{Q}$, $\mu(C_Q) > m$. Il existe alors $n \in \mathcal{N}$ tel que $\mu(C_Q) > \mu(C_{Q_n})$. Comme $\mathcal{Q}$ est totalement ordonné, on a aussi $C_Q \supset C_{Q_n} \supset C_\infty$. Donc $Q \prec R$.

Par le lemme de Zorn, il existe un élément maximal R dans $\mathcal{P}$ pour $\prec$. On contruit alors, par récurrence, une suite de sous-ensembles mesurables de C_R, de mesure strictement positive qui tend vers 0 :

– On pose $A_0 = C_R$
– Si A_n est construit, ce n'est pas un atome par hypothèse. Il existe donc une partie mesurable $A_{n+1} \subset A_n$ avec $0 < \mu(A_{n+1}) \leq \frac{\mu(A_n)}{2}$.

Il est alors clair que $(A_n)_{n\in\mathbb{N}}$ est une suite décroissante de parties mesurables de mesure strictement positive qui tend vers 0.

Or, $\mu(C_R) > \frac{\mu(X)}{2}$. Il exite donc un entier n tel que $\mu(C_R) > \mu(C_R \setminus A_n) > \frac{\mu(X)}{2}$. La partition $R' = \{(C_R \setminus A_n), (X \setminus (C_R \setminus A_n))\}$ vérifie alors $R' \in \mathcal{P}$ et $R \prec R'$, ce qui contredit la maximalité de R. □

Proposition A.4. *Soit $(X, \mathcal{F}, \mu)$ un espace mesuré avec $\mu(X) < \infty$. S'il n'admet pas d'atomes, alors l'image de μ est $[0, \mu(X)]$.*

Démonstration. On va construire des parties disjointes de X $(A_n)_{n\in\mathbb{N}^*}$ de mesures respectives $\frac{\mu(X)}{2^n}$. On le fait par récurrence sur n :

– D'après le lemme A.3, il existe une partie mesurable A_1 de X de mesure $\frac{\mu(X)}{2}$.
– On suppose construits $A_1, \ldots, A_n$. On sait alors que $B_n = X \setminus (\bigcup_{i=1}^{n} A_i)$ est de mesure $\frac{\mu(X)}{2^n}$. D'après le lemme A.3, il existe un ensemble mesurable $A_{n+1} \subset B_n$ tel que $\mu(A_{n+1}) = \frac{\mu(B_n)}{2} = \frac{\mu(X)}{2^{n+1}}$.

Soit $x \in [0, 1[$. On écrit la décomposition en base 2 de x : $x = \sum_{n=1}^{\infty} \frac{x_n}{2^n}$, avec $\forall n \in \mathbb{N}^*\ x_n \in \{0, 1\}$.

On pose alors $B_x = \bigcup_{n\in\mathbb{N}, x_n=1} A_n$. On a alors $\mu(B_x) = \sum_{n\in\mathbb{N}^*, x_n=1} \mu(A_n) = \sum_{n=1}^{\infty} \frac{x_n \mu(X)}{2^n} = x\mu(X)$.

Comme tout $x\mu(X)$, pour $x \in [0, 1[$ est atteint par μ, et que $\mu(X)$ est clairement atteint, l'image de μ est $[0, \mu(X)]$. □

On peut donc supposer que $(\Omega, \mathcal{F}, \mathbb{P})$ ne contient pas d'atomes. On est alors en mesure de construire les X_i.

Proposition A.5. *Soient $A_1, \ldots, A_n$ des événements dans un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$ ne contenant pas d'atomes. Alors il existe un nombre fini de variables aléatoires indépendantes $X_1, \ldots, X_m$ à valeurs dans $\{0, 1\}$ telles que les A_j soient déterminés par les X_i p.s.*

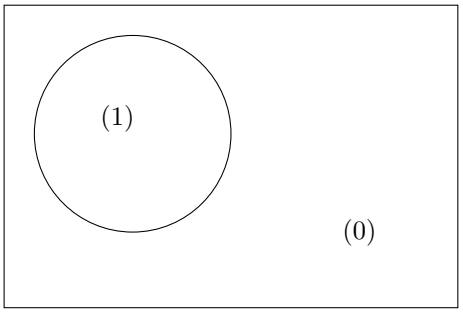
Démonstration. On commence par décomposer l'espace de probabilité. Pour $J \subseteq [\![1, n]\!]$, on note $B_J = \bigcap_{j\in J} A_j \cap \bigcap_{k\in[\![1,n]\!]\setminus J} \overline{A_k}$. Ainsi, $\Omega = \bigcup_{J\in\mathcal{P}([\![1,n]\!])} B_J$ et les B_J sont deux à deux disjoints. Soient $B_0, \ldots B_m$ les B_J de probabilité non nulle.

On raisonne par récurrence sur m avec l'hypothèse suivante : Pour toute partition $B_0, \ldots, B_m$ de Ω, il existe $X_1, \ldots, X_m$ des variables aléatoires indépendantes à valeurs dans $\{0, 1\}$ telles que, pour presque tout $\omega \in \Omega$:

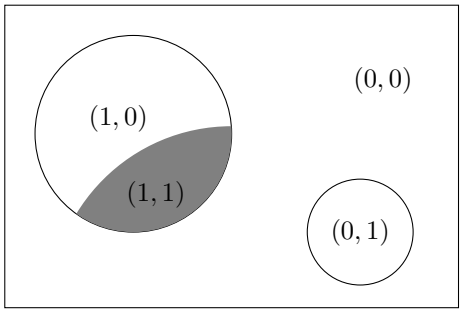
1. $\omega \in B_0 \Leftrightarrow \forall 1 \leq i \leq m\ X_i(\omega) = 0$
2. Si $j \geq 1$, $\omega \in B_j \Leftrightarrow \forall 1 \leq i \leq j-1\ X_i(\omega) = 0$ et $X_j = 1$.

– Si $m = 0$, alors, pour $\omega \in \Omega$, $\omega \in B_0$ p.s. On n'a besoin d'aucune variable X_i.
– Si $m = 1$, on pose $X_1 = \mathbb{1}_{B_1}$. Ainsi, pour preque tout $\omega \in \Omega$, $\omega \in B_0 \Leftrightarrow X_1(\omega) = 0$ et $\omega \in B_1 \Leftrightarrow X_1(\omega) = 1$.

– Soit $m \geq 2$ tel que le résultat soit vrai pour $m-1$. On l'applique à la décomposition $(B_0 \cup B_m), B_1, \ldots, B_{m-1}$. On dispose alors de $m-1$ variables aléatoires $X_1, \ldots, X_{m-1}$ indépendantes et à valeurs dans $\{0,1\}$.
Il reste à définir une variable X_m. On note $p = \frac{\mathbb{P}(B_m)}{\mathbb{P}(B_0)+\mathbb{P}(B_m)}$. Pour $\epsilon \in \{0,1\}^{m-1} \setminus \{0\}$, on note $C_\epsilon = \{\omega \in \Omega \mid X_1(\omega) = \epsilon_1, \ldots, X_{m-1}(\omega) = \epsilon_{m-1}\}$. Comme $\mathbb{P}$ n'admet pas d'atomes, d'après la proposition A.4, pour tout $\epsilon \in \{0,1\}^{m-1} \setminus \{0\}$ il existe $C_{\epsilon,1} \subset C_\epsilon$ mesurable tel que $\mathbb{P}(C_{\epsilon,1}) = p\mathbb{P}(C_\epsilon)$. On pose aussi $C_{\epsilon,0} = C_\epsilon \setminus C_{\epsilon,1}$. On définit alors une variable aléatoire X_m qui vaut 1 sur B_m et les $C_{\epsilon,1}$ et 0 sur B_0 et les $C_{\epsilon,0}$.



Cas où $m = 1$



Passage de $m = 1$ à $m = 2$

On vérifie les propriétés suivantes :

1.
$$\begin{aligned}\mathbb{P}(X_m = 1) &= \mathbb{P}(B_m) + \sum_{\epsilon \in \{0,1\}^{m-1}\setminus\{0\}} \mathbb{P}(C_{\epsilon,1}) \\ &= p(\mathbb{P}(B_0) + \mathbb{P}(B_m)) + \sum_{\epsilon \in \{0,1\}^{m-1}\setminus\{0\}} p\mathbb{P}(C_\epsilon) \\ &= p\end{aligned}$$

2. Si $1 \leq j \leq m-1$,
$$\begin{aligned}\mathbb{P}(X_j = 1 \text{ et } X_m = 1) &= \sum_{\epsilon \in \{0,1\}^{m-1}, \epsilon_j = 1} \mathbb{P}(C_{\epsilon,1}) \\ &= \sum_{\epsilon \in \{0,1\}^{m-1}, \epsilon_j = 1} p\mathbb{P}(C_\epsilon) \\ &= p\mathbb{P}(X_j = 1) \\ &= \mathbb{P}(X_j = 1)\mathbb{P}(X_m = 1)\end{aligned}$$
Donc X_j et X_m sont indépendantes.

3. Pour presque tout $\omega \in \Omega$, $\omega \in B_0 \Leftrightarrow \forall 1 \leq i \leq m\ X_i(\omega) = 0$

4. Si $1 \leq j \leq m$, pour presque tout $\omega \in \Omega$, $\omega \in B_j \Leftrightarrow \forall 1 \leq i \leq j-1\ X_i(\omega) = 0$ et $X_j = 1$ (vrai pour les $1 \leq j \leq m-1$ par hypothèse de récurrence et pour m par construction).

□

B Code Caml

On donne ici le code en Caml d'une implémentation de l'algorithme aléatoire de Moser pour le problème du cycle de longueur multiple de k dans un graphe orienté, traité en 2.5. L'adaptation de l'algorithme a été faite avec les idées de la preuve de 2.5, en coloriant avec k couleurs les sommets du graphe et en cherchant à ce que chaque sommet ait un successeur de la couleur suivante. On a besoin, comme entrées, de la taille du graphe n, l'entier k et des arêtes du graphe, données sous forme de liste de couples.

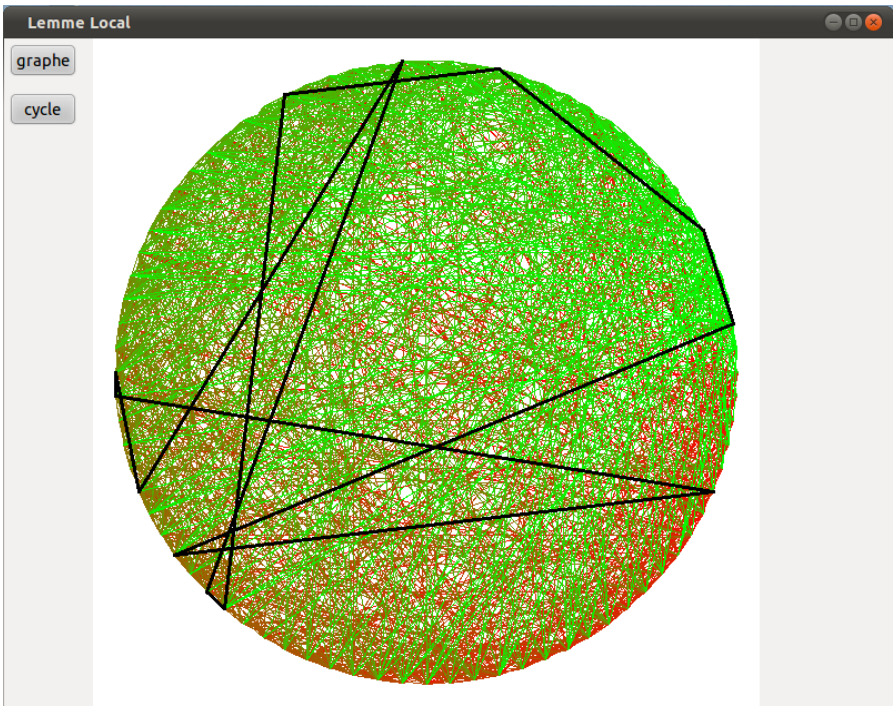
```
(*On définit le type "sommet": chaque sommet a un numéro entre 1 et n, une liste de voisins, une couleur et une variable pour vérifier s'il a un voisin de la couleur suivante *)

type sommet = {numero: int ; mutable  voisins: int list ; mutable  couleur: int ; mutable ok: bool};;

type bon_graphe = sommet_vect ;;

(*La fonction mod de Caml ne convient pas pour chercher l'élément suivant. Il faut donc définir une fonction modulo (qui sera appelée au plus deux fois dans les appels récursifs)*)
let rec modulo k m =
 if 0 > m then modulo k (m+k)
 else if m >= k then modulo k (m-k)
 else m;;

(*Initialisation: on crée un graphe de n sommets pour chercher un cycle de longueur divisible par k. On colorie aléatoirement chaque sommet quand il est créé.*)
 let graphe k n =
  let f k n = {numero = n ; voisins = [] ; couleur = random__int (k) ; ok = false } in
   init_vect n (f k) ;;

(*Rentree de donnees*)
(*add: fonction auxiliaire pour ajouter une arête et actualiser la variable "ok" du sommet.
  remplit: en lui donnant la liste des arêtes, elle actualise le graphe crée avec la fonction "graphe" *)
 let add k v n m =
 (*k: nombre de couleurs, v graphe, n->m arete *)
  let li = v.(n).voisins in v.(n).voisins <- m::li ;
  if modulo k (v.(m).couleur - v.(n).couleur) = 1 then v.(n).ok <- true ;;

 let rec remplit k v = function
   |[] -> ()
   |h::t -> let (n,m)= h in
        add k v n m ; remplit k v t ;;


(*Algorithme de recoloriage*)

(*test: fonction auxiliaire pour tester si les variables "ok" sont toutes vraies. Sinon, on renvoie l'indice minimal où "ok" est fausse *)
let rec test v n =
  let i = ref (-1) in
  for j=0 to (n-1) do
   if not v.(j).ok then i:= j done;
  !i ;;

(*cherche voisin: on teste si le sommet s a un voisin de la couleur suivante (l'entrée étant la liste des voisins de s)*)
let rec cherche_voisin k v s = function
(*k: nombre de couleurs, v graphe, s sommet *)
 |[]-> false
 |h::t -> if modulo k (v.(h).couleur - v.(s).couleur) = 1 then true
          else cherche_voisin k v s t ;;

(*recolorie_succ: on recolorie les sommets d'une liste donnée en entrée*)
let rec recolorie_succ k v j= function
(*k: nombre de couleurs, v graphe *)
 |[]-> ()
```

```
 |h::t -> let c = random__int k in v.(h).couleur <- c ; v.(j).ok <- ( c = modulo k (v.(j).↩
      couleur +1)); recolorie_succ k v j t ;;

(*recoloriage: c'est le coeur de l'algorithme: tant qu'il y a un sommet sans voisin de la ↩
    couleur suivante (test), on en coisit un (renvoyé par test) et on recolorie ses ↩
    succeseurs (recolorie_succ)*)
let recoloriage k v n =
(*k: nombre de couleurs, v graphe, n taille du graphe *)
 while (test v n) >= 0 do
  let j = test v n in
   v.(j).couleur <- random__int k ;
   recolorie_succ k v j v.(j).voisins
 done ;;

(*cherche_suivant: pour un sommet j, on cherche un de ses voisins ayant la couleur suivante↩
    *)
let rec cherche_suivant k v j = function
(*k nombre de couleurs, v graphe, j sommet*)
 |[]-> -1
 |h::t -> if modulo k (v.(h).couleur - v.(j).couleur) = 1 then h
          else cherche_suivant k v j t ;;

(*On suppose qu'on a un coloriage tel que tout sommet a un voisin de la couleur suivante. "↩
    cherche_cycle" donne un cycle en partant de 0 sous forme de liste. Le vecteur "chemin" ↩
    sert à dire quel est le sommet suivant à visiter, et a la valeur -1 aux indice non ↩
    encore visités. La deuxième partie du programme transforme le tableau en une liste ne ↩
    contenenant que le cycle cherché.*)
let cherche_cycle k v n =
(*k nombre de couleurs, v graphe, n taille du graphe*)
 let chemin = make_vect n (-1) and re = ref 0 in
 recoloriage k v n ;
 while chemin.(!re) < 0 do
  let s = cherche_suivant k v (!re) (v.(!re).voisins) in
  chemin.(!re) <- s ; re := s done ;

 let t= ref chemin.(!re) and cycle = ref ([(!re)]) in
 while !t <> (!re) do
  cycle := (!t)::(!cycle) ;
  t := chemin.(!t) done ;
 !cycle ;;

(*Finalement, la fonction "trouve_cycle" met ensemble le recoloriage et la recherche du ↩
    cycle*)
let trouve_cycle k v n =
(*k nombre de couleurs, v graphe, n taille du graphe*)
 recoloriage k v n;
 cherche_cycle k v n ;;
```

C Code C

On donne ici le code en C d'une implémentation de l'algorithme aléatoire de Moser pour le problème du cycle de longueur multiple de k dans un graphe orienté, et un générateur aléatoire de graphes véerifiant la condition de l'énoncé de 2.5. De même que pour le codage en Caml, on colorie les sommets du graphe avec k couleurs, et on cherche à ce que tous les sommets aient un successeur de la couleur suivante. On remarque, néanmoins, que lorsqu'on lance cet algorithme il n'y a que deux ou trois recoloriages qui sont effectués pour avoir cette propriété sur un graphe de 1000 sommets, degré sortant minimal 10 et degré entrant maximal 12, avec k valant 2.

Voici la représentation graphique lors de l'exécution du code, avec la donnée $(n, \delta, \Delta, k) = (800, 14, 20, 3)$. Les arêtes ayant la même couleur sont issues d'un même sommet et le chemin tracé en noir correspond à un cycle de longueur multiple de 3.




```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <gtk/gtk.h>
#define min(a,b) (a<b?a:b)

int show_image = 0;

struct node {
  // numéro du nud
  int n;
  // successeur et antécédent
  struct node **suc, **ant;
  // nombres de successeurs et d'antécédents
  int ns, na;
```

```
  // couleur du nud
  int color;
};

struct graph {
  // nombre de nuds, degré sortant minimal et degré entrant maximal
  int n, doutmin, dinmax;
  // poiteur vers le premier nud
  struct node *nodes;
  // nombre de couleurs
  int k;
  //
  int *viol;
  GtkWidget **widget;
  // liste des sommets
  GdkPoint *points;
  // liste des segments
  GdkSegment *segments;
};

// fonction qui génère un graphe aléatoire
void gen(struct graph *graph) {
  int i, j, next;
  int *linked;
  struct node *nodei;
  srand(time(NULL));

  graph->nodes = (struct node *) malloc(sizeof(struct node) * graph->n);
  graph->viol = (int *) malloc(sizeof(int) * graph->n);

  for (i=0; i<graph->n; i++) {
    (graph->nodes+i)->n = i;
    (graph->nodes+i)->suc = (struct node **) malloc(sizeof(struct node *) * graph->doutmin)↩
        ;
    (graph->nodes+i)->ant = (struct node **) malloc(sizeof(struct node *) * graph->dinmax);
    (graph->nodes+i)->na = (graph->nodes+i)->ns = 0;
  }

  linked = (int *) malloc(sizeof(int) * graph->n * graph->n);

  for (i=0; i<graph->n; i++) *(linked+i) = 0;

  for (i=0; i<graph->n; i++) {
    j = graph->doutmin;
    nodei = graph->nodes+i;
    *(linked+i*graph->n+i) = 1;
    while (j > 0) {
      next = rand() % graph->n;
      //printf("  %d %d %d\n", next, *(nodei->linked+next), (graph->nodes+next)->din);
      if (*(linked+i*graph->n+next) == 1 || (graph->nodes+next)->na == graph->dinmax) ↩
          continue;
      else {
        //printf("%d\n",next);
        *(linked+i*graph->n+next) = 1;
        *((graph->nodes+i)->suc+(graph->nodes+i)->ns) = graph->nodes+next;
        *((graph->nodes+next)->ant+(graph->nodes+next)->na) = graph->nodes+i;
        (graph->nodes+next)->na++;
        (graph->nodes+i)->ns++;
        //printf("%d %d\n", i, next);
        j--;
      }
    }
  }
}

// fonction de (re)coloriage
void rand_color(struct node *node, int k) {
  node->color = rand() % k;
}

// fonction qui teste si un nud vérifie la propriété
int violate(struct node *node, int k) {
  int i;
```

```
  for (i=0; i<node->ns; i++) {
    if ((node->color + 1) % k == (*(node->suc+i))->color) return 0;
  }
  return 1;
}

// fonction qui revoie le nud de plus petit numéro qui est vérifié
int extract_violate(int *viol, int n) {
  int i;
  int start;

  start = rand() % n;
  i = 0;
  while (i<n) {
    if (*(viol+(i+start)%n) == 1) return (i+start)%n;
    i++;
  }
  return -1;
}

// fonction qui recolorie le nud dans l'argument ainsi que ceux dont il dépend
void recolor(struct node *node, int k, int *viol) {
  int i;
  // on recolorie le nud
  rand_color(node, k);
  // on recolorie tous ses successeurs
  for (i=0; i<node->ns; i++) rand_color(*(node->suc+i), k);
  *(viol+node->n) = violate(node, k);
  // on renouvelle la liste de violation
  for (i=0; i<node->ns; i++) *(viol+ (*(node->suc+i))->n) = violate(*(node->suc+i), k);
  for (i=0; i<node->na; i++) *(viol+ (*(node->ant+i))->n) = violate(*(node->ant+i), k);
}

// fonction qui cherche un successeur possédant la couleur suivante
int search_succ(struct node *node, int k) {
  int i;
  int start;
  start = rand() % node->ns;

  for (i=0; i<node->ns; i++) {
    if ((node->color + 1) % k == (*(node->suc+(i+start)%node->ns))->color) return (*(node->↩
        suc+(i+start)%node->ns))->n;
  }
  return -1;
}

// fonction qui dessine le graphe
void construct_graph_data(struct graph *graph) {
  GdkPoint *points;
  GdkSegment *segments, *seg;

  int i, j;
  double R = 280, pi = acos(-1);

  graph->points = (GdkPoint *) malloc(sizeof(GdkPoint) * graph->n);
  graph->segments = (GdkSegment *) malloc(sizeof(GdkSegment) * graph->n * graph->doutmin);

  points = graph->points;
  segments = graph->segments;

  // les points sont placés sur le cercle de centre (300,300)
  for (i=0; i<graph->n; i++) {
    (points+i)->x = (gint) (300 + R*cos(2*pi*i/graph->n));
    (points+i)->y = (gint) (300 + R*sin(2*pi*i/graph->n));
  }

  // les deux extrémités des segments
  for (i=0; i<graph->n; i++) {
    for (j=0; j<(graph->nodes+i)->ns; j++) {
      seg = segments + i*graph->doutmin + j;
      seg->x1 = (points+i)->x;
      seg->y1 = (points+i)->y;
      seg->x2 = (points+ (*((graph->nodes+i)->suc+j))->n)->x;
```

```
      seg->y2 = (points+ (*((graph->nodes+i)->suc+j))->n)->y;
    }
  }
}

// fonction qui affiche le résultat
void output_result(struct graph *graph) {
  int i, succ, end;
  int *visited, *list;
  GtkWidget *widget = *graph->widget;
  GdkDrawable *drawable = widget->window;
  GdkPoint *point1, *point2;
  GdkGC *gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];
  GdkColor color;

  color.green = 0;
  color.blue = 0;
  color.red = 0;
  gdk_gc_set_rgb_fg_color(gc, &color);
  gdk_gc_set_line_attributes(gc, 3, GDK_LINE_SOLID, GDK_CAP_ROUND, GDK_JOIN_ROUND);

  visited = (int *) malloc(sizeof(int) * graph->n);
  list = (int *) malloc(sizeof(int) * graph->n);

  for (i=0; i<graph->n; i++) *(visited+i) = 0;
  *list = 0;
  *visited = 1;
  i = 0;
  succ = rand() % graph->n;

  // on part du sommet 0, on s'arrête dès qu'on revisite un sommet déjà visité
  while (i < graph->n) {
    succ = search_succ(graph->nodes+succ, graph->k);
    i++;
    *(list+i) = succ;
    if (*(visited+succ) == 1) break;
    else *(visited+succ) = 1;
  }

  end = i;
  i = 0;
  while (*(list+i) != succ) i++;
  // on retrace le chemin qui nous intéresse et dessine les arêtes en noir
  while (i < end) {
    printf("%d ", *(list+i));
    point1 = graph->points+*(list+i);
    point2 = graph->points+*(list+i+1);
    gdk_draw_line(drawable, gc, point1->x, point1->y, point2->x, point2->y);
    i++;
  }
  printf("%d\n", succ);

}

// initialisation de l'image
void image_init(struct graph *graph) {
  GtkWidget *widget = *graph->widget;
  GdkDrawable *drawable = widget->window;
  GdkGC *gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];
  GdkColor color;
  GdkPoint *points;
  GdkSegment *segments, *seg;
  int i, j;

  segments = graph->segments;
  points = graph->points;

  gdk_gc_set_line_attributes(gc, 1, GDK_LINE_SOLID, GDK_CAP_ROUND, GDK_JOIN_ROUND);

  if (show_image == 1) gdk_draw_points(drawable, gc, points, graph->n);

  if (show_image == 1) {
    for (i=0; i<graph->n; i++) {
```

```
      for (j=0; j<(graph->nodes+i)->ns; j++) {
        seg = segments + i*graph->doutmin + j;
        color.green = 65535*i/graph->n;
        color.blue = 65535*0/graph->n;
        color.red = 65535*(graph->n-i)/graph->n;
        gdk_gc_set_rgb_fg_color(gc, &color);
        gdk_draw_line(drawable, gc, seg->x1, seg->y1, seg->x2, seg->y2);
      }
    }
  }
}

// fonction qui dessine le graphe puis donne les couleurs aux sommets et vérifie les ↩
    propriétés
void paint_graph(GtkWidget *button, gpointer data) {

  int i, j;
  int recolor_n;
  struct graph *graph = (struct graph *) data;
  GtkWidget *widget = *graph->widget;
  GdkDrawable *drawable = widget->window;
  GdkGC *gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];
  GdkColor color;

  gen(graph);

  color.green = 65535;
  color.blue = 65535;
  color.red = 65535;
  gdk_gc_set_rgb_fg_color(gc, &color);
  gdk_draw_rectangle(drawable, gc, TRUE, 0, 0, 600, 600);

  construct_graph_data(graph);
  image_init(graph);

  for (i=0; i<graph->n; i++) rand_color(graph->nodes+i, graph->k);

  for (i=0; i<graph->n; i++) *(graph->viol+i) = violate(graph->nodes+i, graph->k);

}

// fonction qui trace le cycle en question
void paint_cycle(GtkWidget *button, gpointer data) {
  struct graph *graph = (struct graph *) data;
  GtkWidget *widget = *graph->widget;
  GdkGC *gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];
  GdkDrawable *drawable = widget->window;
  GdkColor color;
  int recolor_n;

  while ((recolor_n = extract_violate(graph->viol, graph->n)) != -1) {
    printf("recolor_n = %d\n", recolor_n);
    recolor(graph->nodes+recolor_n, graph->k, graph->viol);
  }
  output_result(graph);
}

int main(int argc, char *argv[]) {

  GtkWidget *window;
  GtkWidget *drawing_area;
  GtkWidget *button1, *button2;
  GtkWidget *hbox, *vbox, *gtkFrame;
  GdkColor color, *gc;

  int i;
  int n, doutmin, dinmax, k;
  int n1, n2;
  int *ns, *na;
  int recolor_n;
  struct graph *graph;
  double temp;
```

```
306   srand(time(NULL));

      graph = (struct graph *) malloc(sizeof(struct graph));

      // analyse de l'entrée
311   graph->n = atoi(argv[1]);
      graph->doutmin = atoi(argv[2]);
      graph->dinmax = atoi(argv[3]);
      graph->k = atoi(argv[4]);
      if (argc > 5 && strcmp(argv[5], "show_image") == 0) show_image = 1;
316   else show_image = 0;

      temp = exp(1);
      temp *= pow(1. - 1./(double) graph->k, graph->doutmin);
      temp *= min(graph->n, graph->doutmin * graph->dinmax) + 1;
321
      printf("%lf\n", temp);

      gtk_init(&argc, &argv);
      window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
326   gtk_window_set_title(GTK_WINDOW(window), "Lemme Local");
      gtk_window_set_default_size(GTK_WINDOW(window), 800, 600);

      drawing_area = gtk_drawing_area_new();
      gtk_widget_set_size_request(drawing_area, 600, 600);
331
      hbox = gtk_hbox_new(FALSE, 5);
      vbox = gtk_vbox_new(FALSE, 5);

      button1 = gtk_button_new_with_label("graphe");
336   button2 = gtk_button_new_with_label("cycle");

      gdk_color_parse("red", &color);
      gtk_widget_modify_fg(GTK_WIDGET(button1), GTK_STATE_NORMAL, &color);

341   gtk_box_pack_start(GTK_BOX(vbox), button1, FALSE, FALSE, 5);
      gtk_box_pack_start(GTK_BOX(vbox), button2, FALSE, FALSE, 5);
      gtk_box_pack_start(GTK_BOX(hbox), vbox, FALSE, FALSE, 5);
      gtk_box_pack_start(GTK_BOX(hbox), drawing_area, TRUE, TRUE, 5);
      gtk_container_add(GTK_CONTAINER(window), hbox);
346
      graph->widget = &drawing_area;

      g_signal_connect(GTK_OBJECT(button1), "clicked", G_CALLBACK(paint_graph), graph);
      g_signal_connect(GTK_OBJECT(button2), "clicked", G_CALLBACK(paint_cycle), graph);
351   g_signal_connect(GTK_OBJECT(window), "destroy", G_CALLBACK(gtk_main_quit), NULL);

      gtk_widget_show_all(window);
      gtk_main();

356   return 0;
}
```