Le lemme local de Lovasz, version algorithmique

Alice Tosel, MPI, 2023-2024

6 juin 2024

Résumé

La méthode probabiliste est un procédé permettant de montrer non-constructivement l'existence d'un objet mathématique. Dès lors, une question se pose naturellement : existe t-il un algorithme qui exhibe en temps polynomial un objet dont l'existence a été montrée par cette méthode ?

Dans ce TIPE, nous verrons que sous certaines conditions, la réponse est positive. Nous étudierons l'algorithme élaboré par Moser et Tardos en 2009, qui exhibe efficacement un objet dont l'existence est connue par le lemme local de Lovasz, ainsi qu'une variante déterministe. Nous appliquerons ensuite cet algorithme à un problème de coloriage de graphe.

1 Énoncé et application du lemme local de Lovász

1.1 Lemme local de Lovász

Dans la suite, on note $(A_i)_{i\in[\![1,n]\!]}$ des évènements dans un espace probabilisé quelconque et $E \subset \mathscr{P}([\![1,n]\!])$ tel que pour i,j avec $\{i,j\} \notin E$, A_i est indépendant de A_j ou $i=j$.

Version générale : S'il existe des réels $x_1,...,x_n \in [0,1[$ tels que pour tout $i \in [\![1,n]\!]$,

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

alors :

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

(Preuve non-constructive en annexe)

Le corollaire suivant est plus simple à manipuler.

Version symétrique : Si pour tout $i \in [\![1,n]\!]$, $\mathbb{P}(A_i) \leq p$, que chaque évènement dépend d'au plus d autres évènements et $ep(d+1) \leq 1$, alors $\mathbb{P}(\bigcap_{i=1}^{n} \overline{A_i}) > 0$.

Démonstration. On pose $x_i = \frac{1}{d+1}$ pour tout i et on applique le lemme local en remarquant que la fonction $x \mapsto (1-\frac{1}{x+1})^x$ décroît vers $1/e$ en ∞. □

1.2 Une application

Coloriage de graphe

Soit $G = (V, E)$ un graphe non-orienté. A chaque sommet $v \in V$ est associé un ensemble $S(v)$ avec $|S(v)| = 8r$, de telle sorte que pour chaque couleur $c \in S(v)$, au plus r voisins u de v satisfont $c \in S(u)$. Alors, il existe un coloriage de ce graphe tel que pour tout v, la couleur affectée à v est dans $S(v)$.

Démonstration. Pour $e = \{x, y\} \in E$ et c une couleur telle que $c \in S(x), S(y)$, on pose $A_{e,c}$ l'évènement "x et y sont tous deux coloriés par c" ; on a $\mathbb{P}(A_{e,c}) \leq \frac{1}{(8r)^2} := p$. $A_{e,c}$ est mutuellement indépendant de tous les évènements à l'exception des $A_{f,d}$ où x (resp. y) est une extrémité de f et $d \in S(x)$ (resp. $S(y)$). Le degré de dépendance d est donc inférieur ou égal à $2 \cdot (r \cdot 8r) - 1 = 16r^2 - 1$ (on compte au moins une fois l'arête elle-même). On a alors $ep(d+1) \leq e \cdot \frac{1}{(8r)^2} \cdot 16r^2 = \frac{e}{4} \leq 1$, et on conclut grâce à la version symétrique du lemme local. □

2 L'algorithme Las Vegas de Moser-Tardos

2.1 L'algorithme

On se place dans les hypothèses suivantes, en pratique à peine plus contraignantes que celle du lemme général :

— on a $(X_j)_{j \in [\![1,m]\!]}$ des variables aléatoires et $vbl : [\![1, n]\!] \to \mathscr{P}([\![1, m]\!])$ telles que pour tout $i \in [\![1, n]\!]$, A_i dépend uniquement des valeurs de X_j, pour $j \in vbl(i)$;

— on a x_i telle que pour tout $i \in [\![1, n]\!]$, on a $\mathbb{P}(A_i) \leq x_i \cdot \prod_{j, \in N(i)} (1 - x_j)$ où $N(i) = \{j, vbl(i) \cap vbl(j) \neq \emptyset\}$

Pour la version symétrique, on supposera $|N(i)| \leq d$ pour tout i.

L'algorithme est le suivant :

Algorithme 1 : Algorithme de Moser-Tardos

Affecter à chaque X_j une valeur aléatoire ;
tant que *un des* A_l *est satisfait* **faire**
  Choisir l tel que A_l est satisfait ;
  Ré-affecter à chaque X_j pour $j \in vbl(l)$ une valeur aléatoire
fin

Manifestement, si l'algorithme termine, il renvoie une solution au problème. Le point délicat est de montrer qu'il termine presque sûrement et d'estimer l'espérance du temps d'exécution.

2.2 Encodage d'un log comme un arbre témoin

On notera $i \sim j$ lorsque $vbl(i) \cap vbl(j) \neq \emptyset$ ie quand A_i et A_j ne sont pas indépendants.

Définition (log). Un **log** L de cet algorithme après t itérations est la séquence $l_1, ..., l_t$ telle que l_k a été le l choisi à la k-ème itération.

Définition (arbre témoin). Un **arbre témoin** est la donnée d'un arbre enraciné non-vide T et d'une fonction d'étiquetage $\phi \colon V(T) \to [\![1, n]\!]$ telle que si u est enfant de v, alors $\phi(v) \sim \phi(u)$. Un arbre témoin est dit **propre** s'il vérifie de plus que si u et w partagent le même parent, alors $\phi(u) \neq \phi(w)$

.

Définition (arbre témoin associé à un log). Pour $L = l_1, ..., l_t$ un log, soit G le graphe orienté défini par

— $V(G) = \{1, ..., t\}$;

— $E(G) = \{(i, j), i < j, l_i \sim l_j\}$

0n pose V l'ensemble des i tels qu'il existe un chemin de i à t dans ce graphe. Alors $T(L)$ défini par

— $V(T(L)) = V$;

— $E(T(L)) = \{(i, j)$, la première arête du chemin minimal pour l'ordre lexicographique parmi ceux de longueur maximale de i à $t\}$

définit un arbre témoin propre lorsqu'enraciné en t et associé à l. Mieux : deux noeuds à la même profondeur n'ont pas la même étiquette. *(preuve en annexe)*

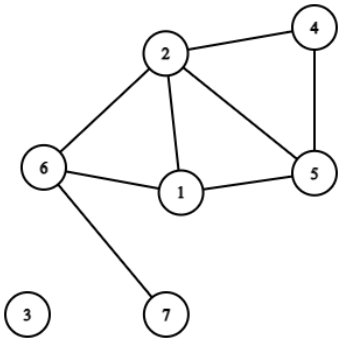

FIGURE 1 – Graphe des dépendances

Numéro	1	2	3	4	5	6	7
Évènement correspondant	1	5	7	6	2	3	4

FIGURE 2 – Log

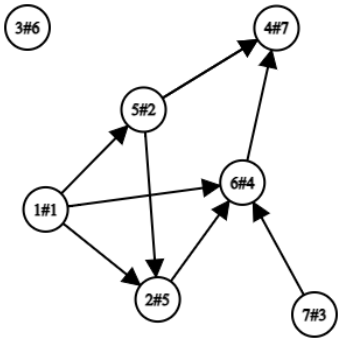

FIGURE 3 – Graphe correspondant à la construction

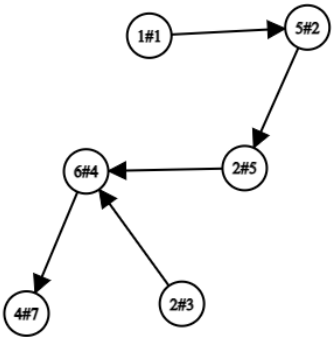

FIGURE 4 – Arbre témoin associé au log

On associe de même à un log une suite d'arbre témoins propres : il s'agit de $(T_n, \phi_n)_{n \leq t}$ où T_n est l'arbre associé au log $l_1, ..., l_n$ et ϕ_n la fonction d'étiquettage sur T_n pour tout n. Cette suite "encode" sous une forme appropriée l'exécution de l'algorithme.

Propriété. Pour $i \neq j$, on a $(T_i, \phi_i) \neq (T_j, \phi_j)$

Démonstration. Par l'absurde, soit $i < j$ avec $(T_i, \phi_i) = (T_j, \phi_j)$. Alors, on pose $f := \phi_i(i) = \phi_j(j)$; f n'étant pas indépendant de lui-même, le nombre de sommets étiquetés par f dans T_i est égal au

nombre d'apparitions de f dans le log avant l'indice i, qui est donc différent au nombre d'apparitions de f dans le log avant l'indice j, contradiction. □

Conséquence. On a $\mathbb{E}[longueur(Log)] = \sum_{(T,\phi)} \mathbb{P}((T,\phi) \in (T_n, \phi_n))$, où (T,ϕ) parcourt l'ensemble des arbres témoins propres.

Démonstration. En posant pour (T,ϕ) arbre témoin propre $N_{(T,\phi)}$ la variable aléatoire dénotant son nombre d'apparitions, on a vu que $N_{(T,\phi)}(\Omega) \subset \{0,1\}$, d'où le résultat par linéarité de l'espérance. □

2.3 Preuve de complexité

Définition ((T,ϕ)-vérification). Pour (T,ϕ) arbre témoin, on appelle (T,ϕ)-**vérification** la procédure qui consiste à visiter les sommets v de T dans un ordre décroissant pour la profondeur et pour chaque v prendre une évaluation aléatoire des $X_i, i \in vbl(v)$, pour vérifier que le résultat satisfait (ou non) $A_{\phi(v)}$. On dit que la (T,ϕ)-vérification **passe** si et seulement si tous les évènements sont satisfaits quand vérifiés.

Propriété. Si $(T,\phi) \in (T_n, \phi_n)_n$, alors la (T,ϕ)-vérification passe avec la même source aléatoire.

Démonstration. Supposons que $(T_t, \phi_t) = (T,\phi)$ et soit $v \in V(T)$. Pour $i \in vbl(v)$, le nombre de sommets w à profondeur strictement plus élevée que v tels que $i \in vbl(w)$ correspond au nombre de $w < v$ tels que $i \in vbl(w)$, par construction de l'arbre. Ainsi, quand on passe sur le sommet v, X_i a la valeur qu'elle avait au début de la v-ème itération de l'algorithme de Moser-Tardos ; par hypothèse, $A_{\phi(v)}$ était alors satisfait. □

La probabilité que (T,ϕ) apparaisse dans (T_n, ϕ_n) est donc inférieure à celle qu'une (T,ϕ)-vérification passe, qui est $\prod_{v \in V(T)} \mathbb{P}(A_{\phi(v)})$. Ainsi, il ne reste plus qu'à majorer $\sum_{(T,\phi)} \prod_{V \in V(T)} \mathbb{P}(A_{\phi(V)})$ pour (T,ϕ) parcourant l'ensemble des arbres témoins propres.

Proposition. Pour $w(i)$ cette somme pour (T,ϕ) parcourant l'ensemble des arbres témoins propres dont la racine est étiquettée par i, on a $w(i) \leq \frac{x_i}{1-x_i}$, quel que soit i.

Démonstration. On pose $w(D,i)$ la somme des $\prod_{v \in V(T)} \mathbb{P}(A_{\phi(v)})$ pour (T,ϕ) parcourant l'ensemble des arbres témoins propres de hauteur strictement inférieure à D dont la racine est étiquettée par i. On va montrer par récurrence sur D que $w(D,i) \leq \frac{x_i}{1-x_i}$ pour tout i, ce qui conclura étant donné que $w(i) = \lim_{D \to \infty} w(D,i)$.

Le cas de base est immédiat. Soit désormais $D \geq 1$ tel que le résultat est acquis au rang $D-1$, et $i \in [\![1,n]\!]$. (T,ϕ) est un arbre témoin propre de hauteur inférieure ou égale à D si et seulement si tous les sous-arbres enracinés en les enfants de la racine sont des arbres témoins propres lorsqu'associés à la restriction de ϕ sur les sommets du sous-arbre, dont les racines sont d'étiquettes distinctes, de hauteur inférieure ou égale à $D-1$. Ainsi, on a $w(D,i) = \mathbb{P}(A_i) \cdot \sum_{S \subset N(i)} \prod_{j \in S} w(D-1,j) = \mathbb{P}(A_i) \cdot \prod_{j \in N(i)} (1 + w(D-1,j)) \leq \mathbb{P}(A_i) \cdot \prod_{j \in N(i)} \frac{1}{1-x_j} \leq x_i \cdot \prod_{j \in N(i)} \frac{1-x_j}{1-x_j} = x_i \leq \frac{x_i}{1-x_i}$ et le résultat est acquis au rang D. □

Par ce qui précède, on a $\mathbb{E}[longueur(Log)] = \sum_{i=1}^{n} w(i) \leq \sum_{i=1}^{n} \frac{x_i}{1-x_i}$. En particulier, on a dans le cas symétrique $\mathbb{E}[longueur(Log)] \leq \frac{n}{d}$.

3 Une variante déterministe dans le cas symétrique

On supposera dans cette section que l'inégalité est stricte, ie qu'il existe $\epsilon > 0$ tel que pour tout i, $\mathbb{P}(A_i) \leq (1-\epsilon)x_i \cdot \prod_{j \in N(i)} (1-x_j)$, avec tous les x_i dans $[0,1[$.

3.1 Idées principales

Définition (arbre témoin cohérent). On dit qu'un arbre témoin (T, ϕ) est **cohérent** avec $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$ si, quand la j-ème affectation de X_i est $(v_i)^j$, la (T,ϕ)-vérification passe.

L'idée est de générer $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$ qui ne soit cohérente avec aucun arbre témoin suffisamment grand, et de remplacer la source aléatoire par cette suite. Alors, les arbres témoins qui apparaissent dans la suite (T_n, ϕ_n) sont de taille bornée, et il n'y en a qu'un nombre fini, ce qui garantit que le processus se termine. Mieux, on va même montrer qu'il se termine en temps polynomial en n et m, à d fixé, si l'on choisit judicieusement la suite.

On montre d'abord qu'il est possible de ne générer qu'un préfixe de $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$.

Proposition. $Q(k)$ l'espérance du nombre d'arbres témoins cohérents avec une évaluation aléatoire ayant plus de k noeuds est inférieure à $\sum_{i=1}^{n} \frac{x_i}{1-x_i}(1-\epsilon)^k$. On peut calculer en $O(1)$ c telle que la probabilité qu'un arbre témoin cohérent de taille supérieure à $c\log(n)$ existe est inférieure à $\frac{1}{2}$.

Démonstration. La probabilité que (T,ϕ) soit cohérent avec une évaluation est $\prod_{v \in V(T)} \mathbb{P}(A_{\phi(v)})$. La majoration voulue a été montrée plus haut, le $(1-\epsilon)^k$ découlant de l'inégalité plus forte et de la contrainte sur le nombre de noeuds.

Ainsi, pour a constante positive telle que $Q(k) \leq an(1-\epsilon)^k$, on a $an(1-\epsilon)^k \leq 1/2$ équivalent à $k \geq \frac{-ln(2)-ln(k)-ln(n)}{ln(1-\epsilon)}$ d'où l'existence de c qu'on peut calculer efficacement. □

Proposition. Pour $u \in \mathbb{N}$, s'il n'existe aucun arbre témoin de taille comprise entre u et $(d+1)u$ cohérent avec $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$, alors il n'existe pas d'arbre témoin de taille supérieure ou égale à u cohérent avec $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$.

Démonstration. Soit par l'absurde (T', ϕ') cohérent de taille minimale parmi ceux cohérents ayant une taille supérieure à u, qui serait alors de taille strictement supérieure à $(d+1)u$, et $w_1, ..., w_r$ les enfants de la racine dans T' (par contrainte sur les étiquettes, $r \leq d+1$). On pose $l_1, ..., l_n$ un log associé à T', et (T'_i, ϕ'_i) l'arbre témoin associé au log $l_1, ...l_{w_i}$, pour $i \in [\![1, r]\!]$. Chacun des (T'_i, ϕ'_i) est cohérent avec l'évaluation, et de taille supérieure à celle du sous-arbre enraciné en w_i dans T' ; il existe donc i tel que $|T'_i| \geq \frac{|T'|-1}{d+1} \geq u$, contradiction. □

La démarche est alors claire : on fixe petit à petit les $c\log n$ premières affectations de chaque variable de sorte à minimiser l'espérance du nombre d'arbres témoins de taille comprise entre $c\log n$ et $(d+1)c\log n$ cohérents avec une suite $(v_i^{(j)})_{1 \leq i \leq m, j \in \mathbb{N}}$ qui coïncide avec les valeurs déjà fixées. Quand toutes les valeurs auront été fixées, cette espérance correspondera au nombre d'arbres témoins réellement cohérents avec l'évaluation actuelle, qui est un entier positif strictement inférieur à 1, donc nul.

Le lemme qui précède montre qu'ainsi, aucun arbre témoin de taille supérieure à $c\log n$ ne sera cohérent avec ce début d'évaluation. De plus, toutes les espérances conditionnelles se calculent en temps polynomial car il a un nombre polynomial d'arbres témoins d'arbres témoins de taille comprise entre $c\log n$ et $(d+1)c\log n$, et cet algorithme est polynomial en n et m.

3.2 Majoration du nombre d'arbres témoins propres

On associe injectivement à chaque structure d'arbre enraciné de taille N un mot de Dyck de longueur $2(N-1)$ via un parcours un profondeur (un symbole (signifie que l'on descend une arête, un symbole) signifie qu'on remonte), ce qui justifie qu'il y en a moins de 4^N. De plus, à structure d'arbre fixé, il y a au plus $n(d+1)^{N-1}$ fonctions d'étiquettages qui en font un arbre témoin propre, grâce aux contraintes parent / enfant et frère / frère sur les étiquettes.

Finalement, le nombre d'arbres témoins de taille comprise entre $c\log n$ et $(d+1)c\log n$ est majoré par $\sum_{N=c\log n}^{(d+1)c\log n}(4(d+1))^N n \leq (d+1)c\log n(4(d+1))^{(d+1)c\log n}\cdot n = O(n^{(d+1)cln(4(d+1))+2})$, bien polynomial.

3.3 L'algorithme

Algorithme 2 : Algorithme déterministe

Générer la suite $(v_i^{(j)})_{1\leq i\leq m, j\in\mathbb{N}}$;
Appliquer l'algorithme probabiliste en remplaçant la source aléatoire par la suite calculée

Pour générer la suite, on commence par calculer c pour générer tous les arbres témoins de taille comprise entre $c\log n$ et $(d+1)c\log n$, par ordre croissant de hauteur. On calcule les espérances conditionnelles en parcourant l'ensemble des arbres témoins, ce qui peut être lourd mais reste polynomial.

Si cet algorithme déterministe a un intérêt théorique, il est inutilisable en pratique, sa complexité étant exponentielle en d.

4 Conclusion

En appliquant l'algorithme de Moser-Tardos à l'application donnée en première partie à des exemples obtenus grâce à un générateur aléatoire disponible en annexe, on obtient ces résultats :

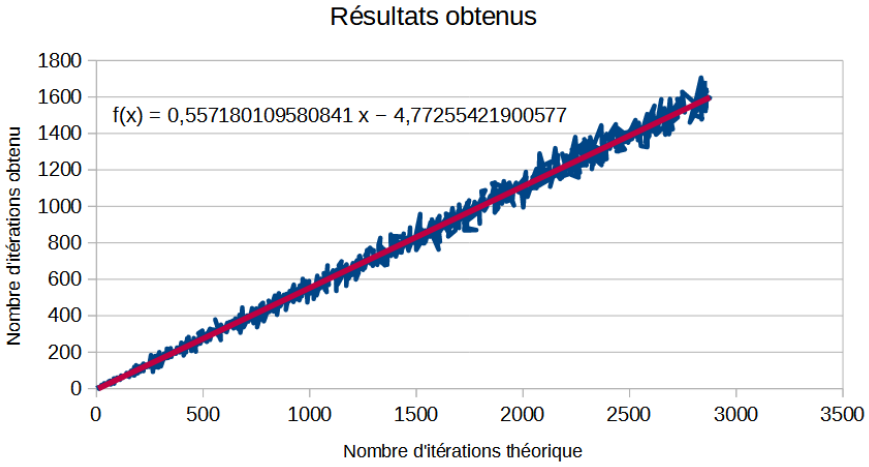

FIGURE 5 – Résultats

ce qui est en pratique presque deux fois meilleur que l'espérance théorique. Ce n'est pas étonnant en soi : on a grossièrement majoré la probabilité d'apparition d'un arbre lors de l'exécution. De plus,

la majoration tient pour toute distribution des x_i telle que les hypothèses du lemme général soient vérifiées ; il est possible qu'une distribution différente mène à une majoration plus proche de l'observation.

Si l'exemple d'application a été choisi en sorte qu'il soit facile et rapide de modifier les variables aléatoires dont dépendent les évènements considérés et de détecter les évènements satisfaits ($O(r)$ par itération), ces étapes constituent en général la limite de cet algorithme.

A Annexe

A.1 Preuves

Preuve : lemme local de Lovasz.

Démonstration. On commence par montrer par récurrence sur $|S|$ que pour tout $S \subset \{1, ..., n\}$ et $i \notin S$, $\mathbb{P}(A_i | \bigcap_{j \in S} \overline{A_j}) \leq x_i$.

— Pour $|S| = 0$, on a $S = \emptyset$ et le résultat est manifestement vrai ;

— On suppose le résultat acquis pour tout S' satisfaisant $|S'| < |S|$. On pose $S_1 = \{j \in S, \{i, j\} \in E\}$ et $S_2 = S \setminus S_1$. On a

$$\mathbb{P}(A_i | \bigcap_{j \in S} \overline{A_j}) = \frac{\mathbb{P}(A_i \cap \bigcap_{j \in S_1} \overline{A_j} | \bigcap_{l \in S_2} \overline{A_l})}{\mathbb{P}(\bigcap_{j \in S_1} \overline{A_j} | \bigcap_{l \in S_2} \overline{A_l})}$$

où :

$$\mathbb{P}(A_i \cap \bigcap_{j \in S_1} \overline{A_j} | \bigcap_{l \in S_2} \overline{A_l}) \leq \mathbb{P}(A_i | \bigcap_{l \in S_2} \overline{A_l}) = \mathbb{P}(A_i) \leq x_i \cdot \prod_{\{i,j\} \in E} (1 - x_j),$$

et, en écrivant $S_1 = \{j_1, ..., j_r\}$,

$\mathbb{P}(\bigcap_{j \in S_1} \overline{A_j} | \bigcap_{l \in S_2} \overline{A_l}) = (1 - \mathbb{P}(A_{j_1}, \bigcap_{l \in S_2} \overline{A_l})) \cdot (...) \cdot (1 - \mathbb{P}(A_{j_r} | \overline{A_{j_1}} \cap ... \cap \overline{A_{j_{r-1}}} \cap \bigcap_{l \in S_2} \overline{A_l})) \geq$
$(1 - x_{j_1}) \cdot ... \cdot (1 - x_{j_r}) \geq \prod_{\{i,j\} \in E} (1 - x_j)$, par hypothèse de récurrence et formule des probabilités composées ;
on en déduit que $\mathbb{P}(A_i | \bigcap_{j \in S} \overline{A_j}) \leq x_i$.

Alors,

$$\mathbb{P}(\bigcap_{i=1}^{n} \overline{A_i}) = (1 - \mathbb{P}(A_1)) \cdot (...) \cdot (1 - \mathbb{P}(A_n | \bigcap_{i=1}^{n-1} \overline{A_i}) \geq \prod_{i=1}^{n} (1 - x_i) > 0.$$ □

Preuve : arbre témoin associé à un log.

Démonstration. Si $i \in V$, $j \in V$ car il existe par hypothèse un chemin de j à t ; par ailleurs, le graphe ainsi défini est acyclique (les arcs sont dans le sens des i croissant), chaque sommet à l'exception de t a exactement un arc sortant et on a un chemin de tous les sommets à t en sorte qu'il s'agit d'un arbre enraciné en t. De plus :

— si u est enfant de v alors par construction de G, $l_u \sim l_v$;

— pour u et w de même profondeur, avec $u < v$, on a $l_u \nsim l_v$: autrement $(u, v) \in E(G)$ et le chemin de u à t passant par v puis prenant le plus long de chemin de v à t est plus long que le plus long chemin de u à t (en effet, la profondeur d'un sommet dans cet arbre est égale à la longueur du plus long de chemin de ce sommet à t dans G, car tout sommet faisant partie d'un chemin jusqu'à t dans G appartient à $V(T)$), absurde.

Il s'agit donc bien d'un arbre témoin propre. □

A.2 Code

Le générateur de graphe :

```cpp
#include <bits/stdc++.h>
using namespace std;

const int r = 5;

vector<vector<int>> genereGraphe(int nSommets, int nAretes) {
    vector<vector<int>> voisins(nSommets);
    for (int i = 0; i < nSommets; ++i)
        voisins[i] = {};

    set<pair<int, int>> dejaVues;
    for (int iArete = 0; iArete < nAretes; ++iArete) {
        int i = rand()%nSommets;

        int delta = rand()%(nSommets - 1);
        int j = (i + 1 + delta)%nSommets;

        if (dejaVues.find(make_pair(min(i, j), max(i, j))) != dejaVues.end()) continue;

        voisins[i].push_back(j);
        voisins[j].push_back(i);
        dejaVues.insert(make_pair(min(i, j), max(i, j)));
    }

    return voisins;
}

pair<vector<vector<int>>, vector<vector<vector<int>>>> genere(int nSommets, int nAretes) {
    vector<vector<int>> voisins = genereGraphe(nSommets, nAretes);

    vector<vector<int>> possibles(nSommets), nDep(nSommets);
    vector<vector<vector<int>>> voisinsFautifs(nSommets);

    for (int v = 0; v < nSommets; ++v) {
        possibles[v].resize(8 * r);
        nDep[v].resize(8 * r);
        voisinsFautifs[v].resize(8 * r);
        for (int i = 0; i < 8 * r; ++i) {
            possibles[v][i] = -1;
            nDep[v][i] = 0;
            voisinsFautifs[v][i] = {};
        }

        map<int, int> nOccs;
        set<int> interdit;

        for (int u : voisins[v]) {
            if (u > v) continue;

            for (int j = 0; j < 8 * r; ++j) {
                if (nDep[u][j] < r)
                    nOccs[possibles[u][j]]++;
```

```
                else
                    interdit.insert(possibles[u][j]);
            }
        }

        vector<pair<int, int>> dansGraphe;
        for (auto a : nOccs)
            dansGraphe.push_back(make_pair(a.second, a.first));

        sort(dansGraphe.begin(), dansGraphe.end());
        reverse(dansGraphe.begin(), dansGraphe.end());

        int val = 0, i = 0;
        while (i < 8 * r) {
            int trouve = -1;
            if (!dansGraphe.empty()) {
                if (dansGraphe.back().first >= r)
                    interdit.insert(dansGraphe.back().second);
                if (interdit.find(dansGraphe.back().second) == interdit.end())
                    trouve = dansGraphe.back().second;
                dansGraphe.pop_back();
            }
            else if (interdit.find(val) == interdit.end())
                trouve = val++;
            else
                val++;

            if (trouve != -1) {
                possibles[v][i] = trouve;
                voisinsFautifs[v][i] = {};

                for (int u : voisins[v]) {
                    if (u > v) continue;
                    for (int j = 0; j < 8 * r; ++j) {
                        if (possibles[u][j] == trouve) {
                            nDep[u][j]++;
                            nDep[v][i]++;
                            voisinsFautifs[u][j].push_back(v);
                            voisinsFautifs[v][i].push_back(u);
                        }
                    }
                }
                i++;
                interdit.insert(trouve);
            }
        }
    }

    return make_pair(possibles, voisinsFautifs);
}
```

Le code en lui-même :

```cpp
#include <bits/stdc++.h>
#include "generateur-graphe.h"
using namespace std;

const int C = 8 * r;

int MoserTardos(int N, int C, vector<vector<int>> possibles, vector<vector<vector<int>>> voisins) {
    vector<int> couleurs(N, -1);

    for (int i = 0; i < N; ++i)
        couleurs[i] = rand()%C;

    queue<pair<int, int>> enAttente;
    for (int i = 0; i < N; ++i) {
        for (int v : voisins[i][couleurs[i]]) {
            if (possibles[v][couleurs[v]] == possibles[i][couleurs[i]])
                enAttente.push(make_pair(i, v));
        }
    }

    int tailleLog = 0;
    while (!enAttente.empty() && tailleLog < 1e6) {
        ++tailleLog;
        int i = enAttente.front().first, j = enAttente.front().second;
        enAttente.pop();

        if (possibles[i][couleurs[i]] != possibles[j][couleurs[j]])
            continue;

        couleurs[i] = rand()%C;
        couleurs[j] = rand()%C;

        int modifs[2] = {i, j};
        for (int iModif = 0; iModif < 2; ++iModif) {
            int u = modifs[iModif];

            for (int v : voisins[u][couleurs[u]]) {
                if (possibles[u][couleurs[u]] == possibles[v][couleurs[v]])
                    enAttente.push(make_pair(u, v));
            }
        }
    }

    return tailleLog;
}

int total(vector<vector<vector<int>>> graphe) {
    int somme = 0;
    for (int i = 0; i < size(graphe); ++i)
        for (int j = 0; j < C; ++j)
            somme += size(graphe[i][j]);
    return somme/2;
```

```
}

int main() {
    srand(42);
    ofstream cout;
    cout.open("exemple.csv");

    for (int it = 0; it < 1000; ++it) {
        int n = 20 * (it + 1);
        int m = n * sqrt(n);
        pair<vector<vector<int>>, vector<vector<vector<int>>>> graphe = genere(n, m);

        cout << (double)(total(graphe.second))/(16 * r * r -1) << "," << MoserTardos(n, C,
graphe.first, graphe.second) << endl;
    }

    cout.close();
    return 0;
}
```

Références

[1] R.A. Moser et G.Tardos, *A constructive proof of the general Lovász Local Lemma*, 2009, https ://arxiv.org/pdf/0903.0544

[2] J. Spencer et N. Alon, *The probabilistic method*, 2000, John Wiley and Sons, inc.

[3] M. Mitzenmacher et E. Upfal, *Probability and computing*, 2005, Cambridge University Press

[4] Luke Postle, *Probabilistic Methods*, https ://www.youtube.com/watch ?v=$\mathrm{OP}_L Mr1Wd7klist = PL2BdWtDKMS6nRF72s3TOGyBqXwMVHYiLU, Youtube$.