Algorithmique avancée: Algorithmes probabilistes

Serge Haddad, LMF, ENS Paris-Saclay & CNRS

L3 et M2 FESUP Informatique

Plan

Algorithmes probabilistes

On considère une instruction probabiliste $\mathsf{Sample}(\mathsf{set}, \mathsf{dist})$ où :

- set est un ensemble fini ;
- dist est une distribution sur set ;
- $\mathsf{Sample}(\mathsf{set}, \mathsf{dist})$ renvoie un élément de set tiré aléatoirement selon dist.

La complexité de cette instruction est proportionnelle à $\log(|\mathsf{set}|)$.
(peut être améliorée en cas d'échantillonnages répétés)

Lorsqu'un algorithme contient des instructions probabilistes

- le résultat de l'algorithme est une variable aléatoire ;
- pour une entrée donnée, le temps d'exécution est une variable aléatoire.

$\boxed{\text{Comment définir la correction et la complexité ?}}$

Terminaison

Terminaison standard

Pour toute exécution, l'algorithme se termine.

Terminaison probabiliste

L'algorithme se termine presque sûrement (i.e. avec probabilité 1).

Correction : cas général

Correction standard

- Soit $out(in)$ le résultat attendu ;
- Soit $output(in)$ le résultat aléatoire de l'algorithme.

Pour tout in, $output(in) = out(in)$.

Correction p-probabiliste

- Soit $out(in)$ le résultat attendu ;
- Soit $output(in)$ le résultat aléatoire de l'algorithme.

Pour tout in, $\mathbf{Pr}(output(in) = out(in)) \geq p$.

Monte Carlo et Las Vegas

Un algorithme de Monte Carlo est :

- un algorithme qui se termine toujours ;
- et garantit une correction probabiliste.

Un algorithme de Las Vegas est :

- un algorithme qui se termine presque sûrement ;
- et garantit une correction standard.

Un algorithme de Monte Carlo qui renvoie soit le résultat correct soit « échec » peut-être transformé en algorithme de Las Vegas en l'itérant jusqu'à obtenir le résultat correct.

Correction : cas spécifiques

Problèmes d'optimisation

- Soit $Sol(in)$ un ensemble de solutions ;
- Soit f qui associe à tout $x \in Sol(in)$, une récompense positive ;
- Soit $out(in) = \sup(f(x) \mid x \in Sol(in))$;
- Soit $0 < c < 1$ une garantie de performance fixe.

Pour tout in, $output(in) \in Sol(in)$ et $\mathbf{E}(f(output(in))) \geq c \cdot out(in)$.

Problèmes d'approximation

- Soit $out(in)$ une valeur numérique strictement positive ;
- Soit $0 < \varepsilon, \delta < 1$ des garanties de performance, entrées de l'algorithme.

Pour tout in, $\mathbf{Pr}\left(|output(in) - out(in)| > \varepsilon \cdot out(in)\right) < \delta$.

Complexité

Soit $T(in)$ le temps d'exécution aléatoire.

Soit n une taille d'entrée, on note $T(n) = \max(T(in) \mid |in| = n)$.

Soit f une fonction de $\mathbb{N}$ dans $\mathbb{R}_+$.

Complexité en moyenne.

$$\mathbf{E}(T(n)) \in O(f(n))$$

Complexité au pire cas probabiliste.

$$\exists\, c > 0 \;\; \lim_{n\to\infty} \mathbf{Pr}(T(n) > c \cdot f(n)) = 0$$

Aucune des deux notions n'implique l'autre !

Illustration

Complexité au pire cas probabiliste $\not\Rightarrow$ Complexité en moyenne.

$\mathbf{Pr}(T(n) = n^2) = 1 - \mathbf{Pr}(T(n) = 0) = \frac{1}{n}$

$\mathbf{E}(T(n)) = n \notin O(1)$

$\mathbf{Pr}(T(n) > 0) = \frac{1}{n} = o(1)$

Complexité en moyenne $\not\Rightarrow$ Complexité au pire cas probabiliste.

Pour tout $1 \leq i$, $\mathbf{Pr}(T(n) = if(n)) = 2^{-i}$

$\mathbf{E}(T(n)) = f(n) \sum_{i \geq 1} i2^{-i} = 2f(n)$

Pour tout $i \geq 1$, $\mathbf{Pr}(T(n) > if(n)) = 2^{-i}$

Tri rapide probabiliste

TriRapide(T, ℓ, h)

If $h \leq \ell$ **then return**

$i \leftarrow$ Sample$(\ell \dots h, \text{uniform})$; Swap(i, h) ; $deb \leftarrow \ell - 1$

For j **from** ℓ **to** $h - 1$ **do**

  If $(T[j], j) < (T[h], i)$ **then**

    $deb \leftarrow deb + 1$

    Swap(deb, j)

$deb \leftarrow deb + 1$; Swap(deb, h)

TriRapide$(T, \ell, deb - 1)$

TriRapide$(T, deb + 1, h)$

Soit $Time(n)$, la complexité aléatoire du tri rapide d'un tableau de taille n.

- $Time(0) = Time(1) = 1$;
- pour $n \geq 2$, $\mathbf{E}(Time(n)) \leq Kn + \frac{1}{n} \sum_{0 \leq i \leq n-1} \mathbf{E}(Time(i)) + \mathbf{E}(Time(n-1-i))$.

Analyse de complexité

Soit $C(n)$ défini par :

- $C(0) = C(1) = 1$;
- pour $n \geq 2$, $C(n) = Kn + \frac{1}{n} \sum_{0 \leq i \leq n-1} C(i) + C(n-1-i)$.

Alors $C(n) = O(n \log(n))$.

Preuve.

$nC(n) = Kn^2 + 2\sum_{i=0}^{n-1} C(i)$ et $(n-1)C(n-1) = K(n-1)^2 + 2\sum_{i=0}^{n-2} C(i)$.

D'où $nC(n) - (n-1)C(n-1) = K(2n-1) + 2C(n-1)$

Puis $nC(n) = (n+1)C(n-1) + K(2n-1)$.

$$\begin{aligned}
\frac{C(n)}{n+1} &= \frac{C(n-1)}{n} + \frac{K(2n-1)}{n(n+1)} \leq \frac{C(n-1)}{n} + \frac{2K}{n+1} \\
&\leq \frac{C(n-2)}{n-1} + \frac{2K}{n} + \frac{2K}{n+1} \\
&\vdots \\
&\leq \frac{C(1)}{2} + \sum_{i=2}^{n} \frac{2K}{i+1} = O(\log(n))
\end{aligned}$$

Un algorithme « erroné »

```
TriRapide(T, ℓ, h)
If h ≤ ℓ then return
i ← Sample(ℓ ... h, uniform) ; Swap(i, h) ; deb ← ℓ − 1
For j from ℓ to h − 1 do
  If T[j] < T[h] then
    deb ← deb + 1
    Swap(deb, j)
deb ← deb + 1 ; Swap(deb, h)
TriRapide(T, ℓ, deb − 1)
TriRapide(T, deb + 1, h)
```

Cet algorithme trie le tableau ...

mais appliqué à un tableau de n valeurs identiques s'exécute en $\Omega(n^2)$!

Intérêt des algorithmes probabilistes

- Conception plus simple.
- Combiné avec la technique de dérandomisation fournit des algorithmes déterministes.
- Analyse de complexité indépendante d'hypothèses sur les données.
- Permet une analyse de complexité intermédiaire entre la complexité en moyenne et la complexité au pire cas.

Plan

Dictionnaire

Un dictionnaire est un ensemble d'enregistrements.

Chaque enregistrement est un couple (clé,valeur) où la clé est *unique*.

L'espace des clés est ordonné et de taille trop importante
pour indicer un tableau en mémoire.

Les opérations usuelles sont :

- Insérer(clé,valeur) qui insère un enregistrement si la clé n'est pas déjà présente ;
- Modifier(clé,valeur) qui modifie la valeur d'un enregistrement si la clé est présente ;
- Supprimer(clé) qui supprime un enregistrement si la clé est présente ;
- Chercher(clé) qui renvoie la valeur d'un enregistrement si la clé est présente.

La complexité de Modifier est essentiellement celle de Chercher.

Implémentation d'un dictionnaire

La structure de données doit être *dynamique*.

Les structures usuelles.

- Un tableau qu'on recopie dans un tableau plus grand (resp. plus petit) en cas de débordement (resp. sur-allocation) ;
- Une liste simplement ou doublement chaînée, linéaire ou circulaire, etc. ;
- Un arbre binaire de recherche *équilibré* ;
- Une table de hachage : un tableau de listes dont la fonction de hachage appliquée à la clé fournit l'entrée dans la table.
- Une structure à trous.

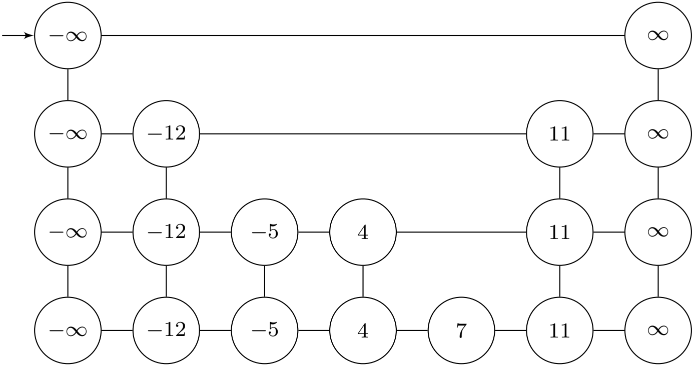
On étudiera la complexité des opérations en fonction de n,
le nombre courant d'enregistrements.

Structure à trous

Une structure à trous est constituée de listes doublement chaînées.

- Chaque liste contient un sous-ensemble des clés triées par ordre croissant et les bornes $-\infty$ et ∞ ;
- Les listes sont ordonnées verticalement ;
- La liste *basse* contient toutes les clés ;
- Chaque autre liste contient un sous-ensemble (non strict) des clés de la liste immédiatement en dessous ;
- Chaque clé de cette autre liste est doublement chainée à la même clé de la liste immédiatement en dessous ;
- La liste *haute* est l'unique liste qui ne contient que les bornes $-\infty$ et ∞ ;
- On *entre* dans la structure à trous par la clé $-\infty$ de la liste haute.

Illustration

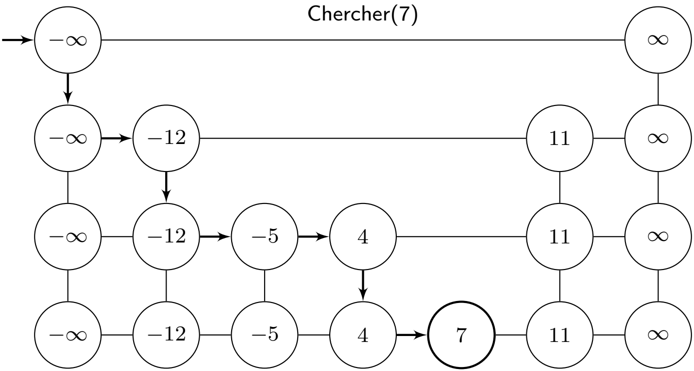


Recherche

On avance dans la liste courante sans dépasser la clé cherchée

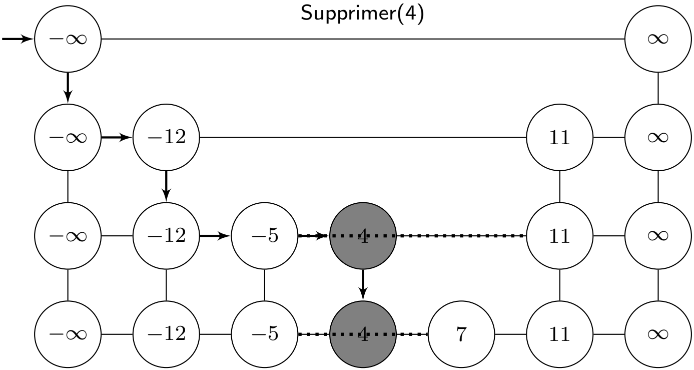
et on descend dans la liste suivante

jusqu'à ce qu'on atteigne la liste basse.



Suppression

On cherche la clé puis on supprime ses occurrences en descendant dans les listes et en supprimant chaque liste réduite à ses bornes (et éventuellement des listes $-\infty, \infty$ redondantes).

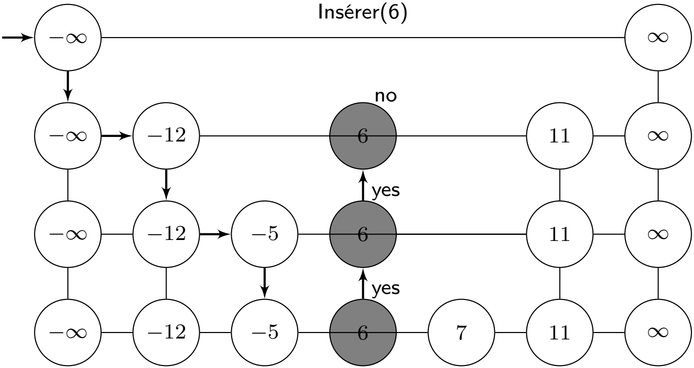


Insertion (probabiliste)

On cherche la clé puis on l'insère dans la liste basse

et itérativement avec probabilité $\frac{1}{2}$ on l'insère dans la liste au dessus.

en ajoutant une liste à chaque tirage positif lorsqu'on atteint la liste haute.



Nombre de listes

Rappels probabilistes.

- Inégalité 'union-somme' : $\mathbf{Pr}(\bigcup_{i=1}^n E_i) \leq \sum_{i=1}^n \mathbf{Pr}(E_i)$
- Inégalité 'au moins un' : $\mathbf{Pr}(\sum_{i=1}^n X_i \geq k) \leq \mathbf{Pr}(\bigvee_{i=1}^n X_i \geq \frac{k}{n}) \leq \sum_{i=1}^n \mathbf{Pr}(X_i \geq \frac{k}{n})$

Soit H la variable aléatoire associée au nombre de listes.

> Dans une structure à trous de $n > 0$ clés on a :
>
> $$\mathbf{Pr}(H \geq 3\log(n) + 2) \leq \frac{1}{n^2}$$

Preuve

Il y a au moins $h+2$ listes ssi une clé a été dupliquée au moins h fois.

Par l'inégalité 'union-somme', $\mathbf{Pr}(H \geq h+2) \leq n2^{-h}$.

Par conséquent, $\mathbf{Pr}(H \geq 3\log(n) + 2) \leq n2^{-3\log(n)} = \frac{1}{n^2}$. $\square$

On note R la variable aléatoire associée au nombre de parcours de pointeurs lors d'une recherche.

Pire cas probabiliste d'une recherche (1)

$$\boxed{\text{Si } n > 0 \text{ alors } \mathbf{Pr}(R > \log(n)(12\log(n)+8)) \le \frac{3\log(n)+3}{n^2}}$$

Preuve. Soit N_i le nombre de parcours de pointeurs horizontaux de la liste i.

$$
\begin{aligned}
& \mathbf{Pr}\left(R > \log(n)(12\log(n)+8)\right) \\
= {} & \mathbf{Pr}\left(\sum_{1\le i} \mathbf{1}_{H\ge i}(1+N_i) > \log(n)(12\log(n)+8)\right) \\
\le {} & \mathbf{Pr}\left(\sum_{1\le i\le 3\log(n)+2} (1+N_i) + \sum_{3\log(n)+2<i} \mathbf{1}_{H\ge i}(1+N_i) > \log(n)(12\log(n)+8)\right) \\
\le {} & \quad \mathbf{Pr}\left(\sum_{1\le i\le 3\log(n)+2} (1+N_i) > \log(n)(6\log(n)+4)\right) \\
& + \mathbf{Pr}\left(\sum_{3\log(n)+2<i} \mathbf{1}_{H\ge i}(1+N_i) > \log(n)(6\log(n)+4)\right) \\
\le {} & \mathbf{Pr}\left(\sum_{1\le i\le 3\log(n)+2} (1+N_i) > \log(n)(6\log(n)+4)\right) + \mathbf{Pr}\left(H > 3\log(n)+2\right) \\
\le {} & \sum_{1\le i\le 3\log(n)+2} \mathbf{Pr}\left(1+N_i > 2\log(n)\right) + \frac{1}{n^2}
\end{aligned}
$$

Pire cas probabiliste d'une recherche (2)

- Soit $e \neq -\infty$ de la liste i, un élément du parcours
  l'élément précédent du parcours est
  - soit un certain $e' < e$ dans la liste i ;
  - soit e dans la liste $i+1$.

Le deuxième cas intervient avec une probabilité $\frac{1}{2}$, d'après l'insertion.

- Si $e = -\infty$, ce cas a une probabilité égale à 1.

Par conséquent, N_i vérifie $\mathbf{Pr}(N_i \geq m) \leq 2^{-m}$. D'où :

$$\mathbf{Pr}\left(1 + N_i > 2\log(n)\right) \leq \frac{1}{2^{2\log(n)}} = \frac{1}{n^2}$$

et en reportant dans l'inéquation :

$$\mathbf{Pr}(R > \log(n)(12\log(n) + 8)) \leq \frac{3\log(n) + 2}{n^2} + \frac{1}{n^2}$$

Cas moyen d'une recherche

$$\boxed{\text{Si } n > 0 \text{ alors } \mathbf{E}(R) \leq 6\log(n) + 6}$$

Preuve. Le nombre de parcours de pointeurs horizontaux est $\leq n$.

$$\begin{aligned}
\mathbf{E}(R) &= \mathbf{E}\left(\sum_{1\leq i} \mathbf{1}_{H\geq i}(1+N_i)\right) \\
&\leq \sum_{1\leq i\leq 3\log(n)+2} \mathbf{E}(1+N_i) + n\mathbf{Pr}(H > 3\log(n)+2) + \sum_{3\log(n)+2<i} \mathbf{Pr}(H \geq i) \\
&\leq \sum_{1\leq i\leq 3\log(n)+2} \mathbf{E}(1+N_i) + 1 + \sum_{3\log(n)+2<i} \frac{1}{i^2} \\
&\leq \sum_{1\leq i\leq 3\log(n)+2} \mathbf{E}(1+N_i) + 2 \\
&\leq 6\log(n) + 6
\end{aligned}$$

car $\mathbf{E}(1+N_i) \leq 2$

Suppression et insertion

Le surcoût de l'insertion et de la suppression est lié

au tirage répété d'une pièce non biaisée et au nombre d'éléments ajoutés.

D'où :

Cas moyen.

Le nombre moyen d'éléments ajoutés est 2.

Pire cas probabiliste.

La probabilité que ce nombre soit supérieur ou égal à $\log(n)$

est inférieure ou égale à $\frac{1}{n}$.

Une grille pour des points

Soit un ensemble de n points du plan $\mathbf{P} = (p_i)_{i \leq n}$ avec $p_i = (x_i, y_i)$.

Soit $0 < r$ une largeur de grille.

Alors *l'indice* $id^r(p_i) = (\alpha_i^r, \beta_i^r) \in \mathbb{N}^2$ est défini par :

$$r\alpha_i^r \leq x_i < r(\alpha_i^r + 1) \text{ et } r\beta_i^r \leq y_i < r(\beta_i^r + 1)$$

La *cellule* de p_i est le carré $[r\alpha_i^r, r(\alpha_i^r + 1)[\times[r\beta_i^r, r(\beta_i^r + 1)[$.

Build$(r, \mathbf{P})$

- trouve en temps moyen $O(n)$ une fonction de hachage universelle h (avec complexité du choix en $0(1)$)
- sans collision pour $(id^r(p_i))_{i \leq n}$
- et stocke dans T les $(p_i)_{i \leq n}$ en fonction de leur indice.

Plus proche paire de points

$\sigma \leftarrow$ PermutationUniforme(n) ; $\mathbf{P} \leftarrow (p_{\sigma(i)})_{i \leq n}$

$r \leftarrow$ dist(p_1, p_2) ; $(T, h) \leftarrow$ Build$(r, \{p_1, p_2\})$

For i **from** 1 **to** 2 **do** Insert$(p_i, T[h(id^r(p_i))])$

For i **from** 3 **to** n **do**

  $(r = \min(\text{dist}(p_k, p_\ell) \mid k \neq \ell \wedge k, \ell \in [1, i-1])$

  $(T$ contient les points $(p_k)_{k<i})$

  $temp \leftarrow r$; $(\alpha, \beta) \leftarrow id^r(p_i)$

  For (k, ℓ) **in** $\{-1, 0, 1\}$ **do**

    $key = h(\alpha + k, \beta + \ell)$

    For p **in** $T[key]$ **do**

      If dist$(p, p_i) < temp$ **then** $temp \leftarrow$ dist(p, p_i)

  If $temp = r$ **then** Insert$(p_i, T[h(id^r(p_i))])$

  Else

    $r \leftarrow temp$; $(T, h) \leftarrow$ Build$(r, \{p_1, \ldots, p_i\})$

    For j **from** 1 **to** i **do** Insert$(p_j, T[h(id^r(p_j))])$

Correction

Il suffit de démontrer l'invariant de boucle pour établir la correction.

Lors de l'itération du point p_i

- soit p_i est inséré dans T ;
- soit T est reconstruit avec les points $\{p_1, \ldots, p_i\}$.

p_i améliore la distance minimale r s'il existe p_j avec $j < i$ tel que $\text{dist}(p_i, p_j) < r$.

Par conséquent la cellule de p_j doit être :

- soit la cellule de p_i ;
- soit une cellule adjacente.

Observation. L'absence de collision n'est pas nécessaire pour la correction.

Complexité (1)

Nous établissons qu'en moyenne l'algorithme s'effectue en $O(n)$ ou en $O(n \log(n))$ selon le modèle de calcul probabiliste adopté.

Itération du point p_i

- Puisque la fonction de hachage est sans collision, les points d'une liste $T[key]$ ont même identifiant et se situent donc dans un carré de côté r.
- Puisque la distance entre deux points de cette liste est au moins r, il y a au plus 4 points par liste.
- Par conséquent, le calcul de $temp$ opère en temps constant.
- L'éventuel appel à Build s'effectue en $O(i)$ en moyenne.

Complexité (2)

Soit :

- $\mathbf{P}_i = \{p_1, \ldots, p_i\}$;
- pour $\mathbf{Q} \subseteq \mathbf{P}$, $r(\mathbf{Q})$ la distance minimale entre points de $\mathbf{Q}$;
- Un point $p \in \mathbf{Q}$ est dit $\mathbf{Q}$-critique si $r(\mathbf{Q} \setminus \{p\}) > r(\mathbf{Q})$.

Examinons la possibilité de l'appel à Build :

- s'il n'y a pas de point $\mathbf{P}_i$-critique, $r(\mathbf{P}_{i-1}) = r(\mathbf{P}_i)$ et il n'y a pas d'appel à Build ;
- S'il y a un seul point $\mathbf{P}_i$-critique, alors il y a un appel à Build si ce point est p_i d'où une probabilité $\frac{1}{i}$.
- S'il y a deux points $\mathbf{P}_i$-critiques, alors il y a un appel à Build si l'un des points est p_i d'où une probabilité $\frac{2}{i}$.
- Il ne peut y avoir trois point $\mathbf{P}_i$-critiques p, p', p'' car (par exemple) si $r(\mathbf{P}_i) = \text{dist}(p, p')$, alors p'' n'est pas critique.

Par conséquent, la complexité moyenne cumulée des appels à Build est en :

$$\sum_{i \leq n} \frac{1}{i} O(i) = O(n)$$

Plan

Le problème de la coupe maximale

Etant donné un graphe $G = (V, E)$, le problème de la coupe maximale consiste à :

- renvoyer une partition $V = V_0 \uplus V_1$
- telle que $|\{\{u, v\} \in E \mid u \in V_0 \wedge v \in V_1\}|$ soit maximal.

Le problème de décision associé MAXCUT prend en entrée G et $K \leq |E|$
et renvoie vrai si :

- il existe une partition $V = V_0 \uplus V_1$
- telle que $|\{\{u, v\} \in E \mid u \in V_0 \wedge v \in V_1\}| \geq K$.

MAXCUT est NP-complet.

Preuve.

Par réduction de 3SAT à MAX2SAT puis de MAX2SAT à MAXCUT.

MAX2SAT **est** NP-**complet (1)**

Le problème MAX2SAT prend en entrée un entier k et des clauses d'au plus deux littéraux sur les variables $(x_i)_{i \leq n}$ et renvoie vrai si :

- il existe une valuation de $(x_i)_{i \leq n}$ dans $\{\bot, \top\}^n$
- telle qu'au moins k clauses soit satisfaites par cette valuation.

Réduction de 3SAT à MAX2SAT. Soit $\varphi = \bigwedge_{1 \leq i \leq p} a_i \vee b_i \vee c_i$.

L'instance de MAX2SAT **Ins** est définie par $k = 7p$ et pour $\bigcup_{i \leq p} Cl_i$ avec Cl_i l'ensemble des clauses suivantes :

- a_i, b_i, c_i, d_i ;
- $\neg a_i \vee \neg b_i, \neg a_i \vee \neg c_i, \neg b_i \vee \neg c_i$;
- $a_i \vee \neg d_i, b_i \vee \neg d_i, c_i \vee \neg d_i$.

où les d_i sont des nouvelles variables.

MAX2SAT **est** NP-**complet (2)**

Considérons une valuation $\mathbf{v}$ des variables de φ.

- Supposons que $\mathbf{v} \models a_i \vee b_i \vee c_i$. Examinons les différents cas de satisfaction :
  - $a_i = \top$ et $b_i = c_i = \bot$. En choisissant $d_i = \bot$, on satisfait 7 clauses de Cl_i. En choisissant $d_i = \top$, on satisfait 6 clauses de Cl_i.
  - $a_i = b_i = \top$ et $c_i = \bot$. En choisissant $d_i \in \{\bot, \top\}$, on satisfait 7 clauses de Cl_i.
  - $a_i = b_i = c_i = \top$. En choisissant $d_i = \bot$, on satisfait 6 clauses de Cl_i. En choisissant $d_i = \top$, on satisfait 7 clauses de Cl_i.
- Supposons que $\mathbf{v} \not\models a_i \vee b_i \vee c_i$. Si $d_i = \top$, on satisfait 4 clauses de Cl_i. Si $d_i = \bot$, on satisfait 6 clauses de Cl_i.
- Supposons que φ est satisfaisable. Par un bon choix de valuation pour les d_i, on satisfait $7p$ clauses de **Ins**.
- Supposons que φ n'est pas satisfaisable. Alors pour tout $\mathbf{v}$, il existe i tel que $\mathbf{v} \not\models a_i \vee b_i \vee c_i$. Il n'est donc pas possible satisfaire $7p$ clauses de **Ins**.

MAXCUT est NP-complet (1)

Réduction de MAX2SAT à MAXCUT. En préalable :

- Elimination des clauses unitaires.
  - Ajout d'une nouvelle variable y ;
  - Remplacement d'une clause a par $a \vee y$ et $a \vee \neg y$ puis incrémentation du seuil.
- Suppression des clauses redondantes $x \vee \neg x$ et décrémentation du seuil.

Soit une instance **Ins** de MAX2SAT définie par :
les variables $\{x_i\}_{i \leq n}$, les clauses $\{a_j \vee b_j\}_{1 \leq j \leq p}$ et $k \leq p$.

Le graphe $G_{\mathbf{Ins}} = (V, E_1 \uplus E_2)$ est défini par :

- $V = \{\perp\} \cup \{x_i^j, \overline{x}_i^j\}_{1 \leq i \leq n, 0 \leq j \leq 2p}$;
- $E_1 = \{\{x_i^j, \overline{x}_i^{j'}\}\}_{1 \leq i \leq n, 0 \leq j, j' \leq 2p}$ $\left(|E_1| = n(2p+1)^2\right)$;
- $E_2 = \bigcup_{1 \leq j \leq p} Tr_j$ avec $Tr_j = \{\{a_j^{2j-1}, b_j^{2j}\}, \{a_j^{2j-1}, \perp\}, \{b_j^{2j}, \perp\}\}$
  et $\neg x$ identifié à $\overline{x}$.

$K = |E_1| + 2k$.

MAXCUT est NP-complet (2)

Soit $V = V_0 \uplus V_1$ et $CUT = \{\{u, v\} \mid u \in V_0 \wedge v \in V_1\}$.

Observations.

- S'il existe $x_i^j \in V_0$ et $x_i^{j'} \in V_1$ alors $|E_1 \setminus CUT| \geq 2p + 1$;
- S'il existe $\overline{x}_i^j \in V_0$ et $\overline{x}_i^{j'} \in V_1$ alors $|E_1 \setminus CUT| \geq 2p + 1$;
- S'il existe m et i tel que $\bigcup_{j \leq 2p}\{x_i^j, \overline{x}_i^j\} \subseteq V_m$ alors $|E_1 \setminus CUT| \geq (2p + 1)^2$.

• Supposons que $|CUT| \geq K$. Alors :

- pour tout i, il existe m_i tel que $\{x_i^j\}_{j \leq 2p} \subseteq V_{m_i}$ et $\{\overline{x}_i^j\}_{j \leq 2p} \subseteq V_{1-m_i}$;
- $|\{j \mid |Tr_j \cap CUT| = 2\}| \geq k$.

On définit la valuation $\mathbf{v}$ par $\mathbf{v}(x_i) = \top$ ssi $\perp \notin V_{m_i}$.

• Supposons qu'une valuation $\mathbf{v}$ satisfasse au moins k clauses. Alors :
$V_1 = \{x_i^j \mid i \leq n \wedge j \leq 2p+1 \wedge \mathbf{v}(x_i) = \top\} \cup \{\overline{x}_i^j \mid i \leq n \wedge j \leq 2p+1 \wedge \mathbf{v}(x_i) = \perp\}$
et $V_0 = V \setminus V_1$.

Un algorithme probabiliste naïf

On note $V = \{v_i\}_{i \leq n}$.

$V_0 \leftarrow \emptyset$
$V_1 \leftarrow \emptyset$
$Cut \leftarrow \emptyset$
For i **from** 1 **do** n **do**
  $b \leftarrow \text{Sample}(\{\bot, \top\}, \text{uniform})$
  If b **then**
    $V_0 \leftarrow V_0 \cup \{v_i\}$
  Else
    $V_1 \leftarrow V_1 \cup \{v_i\}$
For $\{u, v\} \in E$ **do**
  If $\{u, v\} \cap V_0 \neq \emptyset$ **and** $\{u, v\} \cap V_1 \neq \emptyset$ **then**
    $Cut \leftarrow Cut \cup \{\{u, v\}\}$

Garantie probabiliste

Cut est une variable aléatoire.

Introduisons pour chaque $i \leq n$, X_i définie par : $X_i = m$ si $v_i \in V_m$.

Observons que $|Cut| = \sum_{\{v_i,v_j\} \in E} \mathbf{1}_{X_i \neq X_j}$. D'où :

$$\begin{aligned}
\mathbf{E}(|Cut|) &= \sum_{\{v_i,v_j\} \in E} \mathbf{Pr}(X_i \neq X_j) \\
&= \sum_{\{v_i,v_j\} \in E} \mathbf{Pr}(X_i = 0 \neq X_j) + \mathbf{Pr}(X_i = 1 \neq X_j) \\
&= \sum_{\{v_i,v_j\} \in E} \frac{1}{4} + \frac{1}{4}
\end{aligned}$$

D'où :

$$\mathbf{E}(|Cut|) = \frac{|E|}{2}$$

Dérandomisation de l'algorithme (1)

Abrégeons l'événement $\bigwedge_{i\leq k} X_i = x_i$ par $(x_i)_{i\leq k}$.

$$\begin{aligned}
\mathbf{E}(|Cut| \mid (x_i)_{i\leq k}) &= \sum_{j\in\mathbb{N}} j\mathbf{Pr}(|Cut| = j \mid (x_i)_{i\leq k}) \\
&= \sum_{j\in\mathbb{N}} \frac{j\mathbf{Pr}(|Cut| = j \wedge (x_i)_{i\leq k})}{\mathbf{Pr}((x_i)_{i\leq k})} \\
&= \sum_{j\in\mathbb{N}} \frac{j\mathbf{Pr}(|Cut| = j \wedge (x_i)_{i\leq k} \wedge X_{k+1} = 0) + j\mathbf{Pr}(|Cut| = j \wedge (x_i)_{i\leq k} \wedge X_{k+1} = 1)}{\mathbf{Pr}((x_i)_{i\leq k})} \\
&= \mathbf{Pr}(X_{k+1} = 0 \mid (x_i)_{i\leq k})\mathbf{E}(|Cut| \mid (x_i)_{i\leq k} \wedge X_{k+1} = 0) \\
&\quad + \mathbf{Pr}(X_{k+1} = 1 \mid (x_i)_{i\leq k})\mathbf{E}(|Cut| \mid (x_i)_{i\leq k} \wedge X_{k+1} = 1) \\
&= \frac{1}{2}\left(\mathbf{E}(|Cut| \mid (x_i)_{i\leq k}) \wedge X_{k+1} = 0) + \mathbf{E}(|Cut| \mid (x_i)_{i\leq k}) \wedge X_{k+1} = 1)\right)
\end{aligned}$$

Par conséquent, il existe une suite $(x_i)_{i\leq n}$ telle que :

$$\frac{1}{2}|E| = \mathbf{E}(|Cut|) = \mathbf{E}(|Cut| \mid (x_1)) \leq \mathbf{E}(|Cut| \mid (x_i)_{i\leq 2}) \leq \ldots \leq \mathbf{E}(|Cut| \mid (x_i)_{i\leq n})$$

$\boxed{\text{Comment la trouver (itérativement) ?}}$

Dérandomisation de l'algorithme (2)

$$\begin{aligned}
\mathbf{E}(|Cut| \mid (x_i)_{i\leq k} \wedge X_{k+1} = m) = & \sum_{\{v_i,v_j\}\in E \wedge i,j\leq k} \mathbf{1}_{x_i\neq x_j} \\
& + \sum_{\{v_i,v_{k+1}\wedge i\leq k\}\in E\wedge i,j\leq k} \mathbf{1}_{x_i\neq m} \\
& + \sum_{\{v_i,v_j\}\in E\wedge i\leq k\wedge j>k+1} \mathbf{Pr}(x_i \neq X_j) \\
& + \sum_{\{v_{k+1},v_j\}\in E\wedge j>k+1} \mathbf{Pr}(m \neq X_j) \\
& + \sum_{\{v_i,v_j\}\in E\wedge i,j>k+1} \mathbf{Pr}(X_i \neq X_j)
\end{aligned}$$

$$= \sum_{\{v_i,v_j\}\in E\wedge i,j\leq k} \mathbf{1}_{x_i\neq x_j} + \sum_{\{v_i,v_{k+1}\}\in E\wedge i\leq k} \mathbf{1}_{x_i\neq m} + \sum_{\{v_i,v_j\}\in E\wedge j>k+1} \frac{1}{2}$$

Par conséquent, $m = \arg\max(\sum_{\{v_i,v_{k+1}\wedge i\leq k\}\in E\wedge i,j\leq k} \mathbf{1}_{x_i\neq m})$.

Un algorithme glouton déterministe

$V_0 \leftarrow \{v_1\}$; $V_1 \leftarrow \emptyset$; $Cut \leftarrow \emptyset$

For i **from** 2 **do** n **do**

  $cnt_0 \leftarrow 0$; $cnt_1 \leftarrow 0$

  For j **from** 1 **do** $i-1$ **do**

    If $\{v_i, v_j\} \in E$ **then**

      If $v_j \in V_0$ **then** $cnt_0 \leftarrow cnt_0 + 1$ **else** $cnt_1 \leftarrow cnt_1 + 1$

  If $cnt_1 \geq cnt_0$ **then**

    $V_0 \leftarrow V_0 \cup \{v_i\}$

  Else

    $V_1 \leftarrow V_1 \cup \{v_i\}$

For $\{u, v\} \in E$ **do**

  If $\{u, v\} \cap V_0 \neq \emptyset$ **and** $\{u, v\} \cap V_1 \neq \emptyset$ **then**

    $Cut \leftarrow Cut \cup \{\{u, v\}\}$

Cet algorithme garantit $|Cut| \geq \frac{|E|}{2}$.

Plan

Bornes de Chernoff-Hoeffding

Soit $X_1, \ldots, X_n$ des variables aléatoires indépendantes et $\varepsilon \geq 0$.

On note : $X \stackrel{\text{def}}{=} \sum_{i \leq n} X_i$ et $\mu \stackrel{\text{def}}{=} \mathbf{E}(X)$.

- Bornes multiplicatives

  Supposons les X_i à valeurs dans $\{0, 1\}$ et $\varepsilon < 1$. Alors :

$$\mathbf{Pr}(X \geq (1+\varepsilon)\mu) \leq e^{-\frac{\mu\varepsilon^2}{3}} \text{ et } \mathbf{Pr}(X \leq (1-\varepsilon)\mu) \leq e^{-\frac{\mu\varepsilon^2}{2}}$$

Cas particulier. $X_1, \ldots, X_n$ i.i.d. avec $p = \mathbf{E}(X_i)$. Soit $M = \frac{X}{n}$. Alors :

$$\mathbf{Pr}(M \geq (1+\varepsilon)p) \leq e^{-\frac{np\varepsilon^2}{3}} \text{ et } \mathbf{Pr}(M \leq (1-\varepsilon)p) \leq e^{-\frac{np\varepsilon^2}{2}}$$

- Bornes additives

  Supposons que pour tout i, $a_i \leq X_i \leq b_i$ avec $a_i < b_i$. Alors :

$$\mathbf{Pr}(X \geq \mu + \varepsilon) \leq e^{-\frac{2\varepsilon^2}{\sum_{i \leq n}(b_i - a_i)^2}} \text{ et } \mathbf{Pr}(X \leq \mu - \varepsilon) \leq e^{-\frac{2\varepsilon^2}{\sum_{i \leq n}(b_i - a_i)^2}}$$

Cas particulier. $X_1, \ldots, X_n \in [0, 1]$ i.i.d. avec $p = \mathbf{E}(X_i)$. Soit $M = \frac{X}{n}$. Alors :

$$\mathbf{Pr}(M \geq p + \varepsilon) \leq e^{-2n\varepsilon^2} \text{ et } \mathbf{Pr}(M \leq p - \varepsilon) \leq e^{-2n\varepsilon^2}$$

Estimation probabiliste (1)

Soit X une variable aléatoire à valeurs dans $\{0,1\}$
telle que $0 < p = \mathbf{Pr}(X = 1)$ soit inconnue
mais qu'une borne inférieure $0 < p^* \leq p$ soit connue.

Soit $\varepsilon, \delta > 0$. Alors l'algorithme suivant via les bornes multiplicatives garantit que :

$$\begin{aligned} \mathbf{Pr}(|p - \frac{\ell}{k}| > p\varepsilon) = \quad & \mathbf{Pr}(|kp - \ell| > kp\varepsilon) < \quad & 2e^{-\frac{\left\lceil \frac{3\log(2/\delta)}{p^*\varepsilon^2}\right\rceil p\varepsilon^2}{3}} \\ \leq \quad & 2e^{-\frac{p\log(2/\delta)}{p^*}} \leq \quad & 2e^{-\log(2/\delta)} = \delta \end{aligned}$$

$k \leftarrow \left\lceil \frac{3\log(2/\delta)}{p^*\varepsilon^2} \right\rceil$; $\ell \leftarrow 0$
For i **from** 1 **do** k **do**
  $b \leftarrow$ Sample(X)
  If $b = 1$ **then** $\ell \leftarrow \ell + 1$
Return$(\frac{\ell}{k})$

Cet algorithme est polynomial en fonction de $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$ et $\frac{1}{p^*}$.

Estimation probabiliste (2)

Soit $cpt(in)$ un nombre à calculer en fonction d'une entrée in dont un calcul efficace n'est pas connu.

Méthode probabiliste. Définir deux ensembles $E(in) \subset F(in)$ tels que :

- $|E(in)| = cpt(in)$ et $|F(in)|$ peut être calculé efficacement ;
- un tirage uniforme de $e \in F(in)$ peut être effectué efficacement ;
- le test '$e \in E(in)$?' peut être décidé efficacement ;
- un minorant de $\frac{|E(in)|}{|F(in)|}$ peut être calculé efficacement.

Solution. On estime la probabilité p que $e \in F(in)$ appartienne à $E(in)$ et on renvoie $p|F(in)|$.

Comptage probabiliste d'interprétations

Soit $\varphi = \bigvee_{i \leq m} Cl_i$ avec $Cl_i = \bigwedge_{j \leq n_i} \ell_{i,j}$ où $\ell_{i,j} \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$.

(sans perte de généralité, pour tout k et i, $x_k, \neg x_k$ n'apparaissent pas simultanément dans Cl_i)

On veut calculer le nombre de valuations $\mathbf{v} \in \{\bot, \top\}^n$ telles que $\mathbf{v} \models \varphi$

Si P $\neq$ NP alors ce calcul ne peut pas s'effectuer en temps polynomial :
Soit ψ une formule CNF alors ψ n'est pas satisfaisable ssi 2^n interprétations satisfont $\neg\psi$ réécrite en temps linéaire en formule DNF.

On cherche alors une estimation probabiliste de ce nombre à l'aide de l'algorithme précédent :

- Soit la distribution uniforme sur l'ensemble des valuations ;
- Soit p la probabilité qu'une valuation aléatoire satisfasse φ ;
- Alors $2^n p$ est le nombre recherché.

Ici $F(in)$ est l'ensemble des valuations.

Problème : soit $p^* = \max_i(2^{-n_i})$ alors $\frac{1}{p^*}$ peut être exponentiel en n.

Un échantillonnage alternatif (1)

Soit $\mathbf{V}_i = \{\mathbf{v} \mid \mathbf{v} \models Cl_i\}$.

- $|\mathbf{V}_i| = 2^{n-n_i}$;
- on peut effectuer un tirage uniforme dans $\mathbf{V}_i$ en choisissant de manière équiprobable la valeur d'une variable absente de Cl_i.

Soit $\Omega = \{(i, \mathbf{v}) \mid i \leq m \wedge \mathbf{v} \in \mathbf{V}_i\}$.

- $|\Omega| = \sum_{i \leq m} |\mathbf{V}_i|$;
- on peut effectuer un tirage uniforme dans Ω en choisissant $i \leq m$ avec probabilité $\frac{|\mathbf{V}_i|}{|\Omega|}$ puis de manière équiprobable $\mathbf{v} \in \mathbf{V}_i$.

Un échantillonnage alternatif (2)

Soit $\Omega^* \subseteq \Omega$ défini par $\Omega^* = \{(i, \mathbf{v}) \in \Omega \mid \forall j < i \; (j, \mathbf{v}) \notin \Omega\}$.

Autrement dit, $(i, \mathbf{v}) \in \Omega^*$ si Cl_i est la *première* clause satisfaite par $\mathbf{v}$.

$|\Omega^*|$ est le nombre de valuations $\mathbf{v}$ telles que $\mathbf{v} \models \varphi$.

On cherche alors une estimation probabiliste de ce nombre
à l'aide de l'algorithme précédent :

- Soit la distribution uniforme sur Ω ;
- Soit p la probabilité qu'un couple aléatoire $(i, \mathbf{v})$ appartienne à Ω^* ;
- Alors $|\Omega|p$ est le nombre recherché.

Avantage : ici $p^* = \frac{1}{m}$ et $\frac{1}{p^*}$ est linéaire en $|\varphi|$.

Plan

MAX3SAT

Soit une formule $\varphi = \wedge_{j \leq m} a_j \vee b_j \vee c_j$
avec $\{a_j, b_j, c_j\} \subseteq \{x_i\}_{i \leq n} \cup \{\neg x_i\}_{i \leq n}$ et $|\{a_j, b_j, c_j\}| = 3$.

Le problème d'optimisation MAX3SAT consiste à renvoyer
le maximum de clauses de φ satisfaites par une valuation.

Observation. La restriction à exactement trois littéraux n'en est pas une
pour les algorithmes exacts.
Soit y, z deux nouvelles variables. La clause a_i est transformée en quatre clauses :

$$a_i \vee y \vee z,\ a_i \vee \neg y \vee z,\ a_i \vee y \vee \neg z \text{ et } a_i \vee \neg y \vee \neg z.$$

- Soit ν une valuation et ν' une valuation qui étend ν à y, z.
- Si $\nu \models a_i$ alors ν' satisfait les quatre clauses.
- Si $\nu \not\models a_i$ alors ν' satisfait exactement trois clauses.

De la même façon $a_i \vee b_i$ est transformée en deux clauses $a_i \vee b_i \vee y$ et $a_i \vee b_i \vee \neg y$.

Le problème de décision associé est NP-complet (déjà vrai pour MAX2SAT).

$\boxed{\text{D'où l'intérêt pour des algorithmes d'approximation.}}$

Un premier algorithme pour MAX3SAT

Considérons l'algorithme probabiliste suivant.

For i **from** 1 **do** n **do**
  $\nu(x_i) \leftarrow \text{Sample}(\{\bot, \top\}, \text{uniforme})$
$\textbf{Return}(\nu)$

Complexité. Cet algorithme opère en temps linéaire.

Correction.

- Soit N la variable aléatoire associée au nombre de clauses satisfaites par ν ;
- Soit X_j qui vaut 1 si $\nu \models a_j \vee b_j \vee c_j$ et 0 sinon. $\mathbf{E}(X_j) = \frac{7}{8}$;
- $N = \sum_{j \leq m} X_i$. D'où $\mathbf{E}(N) = \frac{7m}{8}$.

Observation. Si $m < 8$ alors φ est satisfaisable. Pourquoi ?

Un algorithme de Las Vegas

While True do
  For i **from** 1 **to** n **do**
    $\nu(x_i) \leftarrow$ Sample$(\{\perp, \top\},$ uniforme$)$
  $count \leftarrow 0$
  For j **from** 1 **to** m **do**
    If $\nu \models a_j \vee b_j \vee c_j$ **then** $count \leftarrow count + 1$
  If $count \geq \frac{7m}{8}$ **then** $\mathbf{Return}(\nu)$

Analyse. Soit $p_k = \mathbf{Pr}(count = k)$ et $p = \sum_{k \geq 7m/8} p_k$.

$$\frac{7m}{8} = \sum_{k < 7m/8} k p_k + \sum_{k \geq 7m/8} k p_k$$

Soit $m' = \max(k \mid k < \frac{7m}{8})$. Observons que $\frac{7m}{8} - m' \geq \frac{1}{8}$.
$\frac{7m}{8} \leq m'(1-p) + mp \leq m' + mp$. D'où $p \geq \frac{1}{m}(\frac{7m}{8} - m') \geq \frac{1}{8m}$.
Le nombre moyen d'itérations du **While** est donc majoré par $8m$.

MAXSAT

Soit une formule $\varphi = \wedge_{j \leq m} \vee_{k \leq n_j} \ell_{j,k}$ avec $\{\ell_{j,k}\} \in \{x_i\}_{i \leq n} \cup \{\neg x_i\}_{i \leq n}$.

Le problème d'optimisation MAXSAT consiste à renvoyer
le maximum de clauses de φ satisfaites par une valuation.

```
While True do
  For i from 1 to n do ν(x_i) ← Sample({⊥, ⊤}, uniforme)
  count ← 0
  For j from 1 to m do
    If ν ⊨ ∨_{k≤n_j} ℓ_{j,k} then count ← count + 1
  If count ≥ m/2 then Return(ν)
```

Analyse. La probabilité de satisfaire une clause est supérieure ou égale à $\frac{1}{2}$.
D'où $\mathbf{E}(N) \geq \frac{m}{2}$. Donc $\frac{m}{2} \leq \sum_{k<m/2} kp_k + \sum_{k \geq m/2} kp_k$
Soit $m' = \max(k \mid k < \frac{m}{2})$. Observons que $\frac{m}{2} - m' \geq \frac{1}{2}$.
$\frac{m}{2} \leq m'(1-p) + mp \leq m' + mp$. D'où $p \geq \frac{1}{m}(\frac{m}{2} - m') \geq \frac{1}{2m}$.
Le nombre moyen d'itérations du **While** est donc majoré par $2m$.

Un choix biaisé de la valuation

On note $Pos_j = \{i \mid \exists \ell_{j,k} = x_i\}$ et $Neg_j = \{i \mid \exists \ell_{j,k} = \neg x_i\}$.

Soit le programme linéaire $\max \sum_{j \leq m} z_j$ tel que :
$\forall i \leq n \; 0 \leq w_i \leq 1, \forall j \leq m \; 0 \leq z_j \leq 1 \wedge \sum_{i \in Pos_j} w_i + \sum_{i \in Neg_j} (1 - w_i) \geq z_j$

On note $(w_i^*)_{i \leq n}, (z_j^*)_{j \leq m}$, une solution optimale.

$\boxed{\text{Le choix biaisé de } \nu(x_i) = 1 \text{ se fait avec la probabilité } w_i^*.}$

Observation.

Soit $opt(\varphi)$ le nombre maximal de clauses de φ satisfaites par une valuation.

Etant donné une valuation ν, on définit $w_i^\nu = 1_{\nu(x_i) = \top}$ et $z_j^\nu = 1_{\nu \models \vee_{k \leq n_j} \ell_{j,k}}$.

$(w_i^\nu)_{i \leq n}, (z_j^\nu)_{j \leq m}$ est une solution de ce programme linéaire.

Par conséquent $opt(\varphi) \leq \sum_{j \leq m} z_j^*$.

Notation. Soit un littéral $\ell_{j,k}$.

Si $\ell_{j,k} = x_i$ alors $w_{j,k}^* = w_i^*$

Si $\ell_{j,k} = \neg x_i$ alors $w_{j,k}^* = 1 - w_i^*$

Interlude

$$\boxed{\text{Soit } (\alpha_i)_{i\leq h} \text{ des réels positifs et } \gamma \text{ leur moyenne. Alors } \prod_{i\leq h}\alpha_i \leq \gamma^h.}$$

Preuve.

Si l'un des α_i est nul, c'est immédiat.

Le logarithme est concave :

$\log(\gamma) = \log(\frac{1}{h}\sum_{i\leq h}\alpha_i) \geq \frac{1}{h}\sum_{i\leq h}\log(\alpha_i)$.

D'où : $\log(\gamma^h) \geq \log(\prod_{i\leq h}\alpha_i)$ équivalent à : $\prod_{i\leq h}\alpha_i \leq \gamma^h$.

Soit s un entier non nul et $f(x) = 1-(1-\frac{x}{s})^s$ pour $x\in[0,1]$.

$$\boxed{f(x) \geq x(1-(1-\tfrac{1}{s})^s) \text{ et } 1-(1-\tfrac{1}{s})^s \geq 1-\tfrac{1}{e}}$$

Preuve.

f est concave car $f''(x) = -\frac{s-1}{s}(1-\frac{x}{s})^{s-2} \leq 0$.

$f(0)=0$ et $f(1) = 1-(1-\frac{1}{s})^s$ d'où la première inégalité.

$1-x\leq e^{-x} \Rightarrow 1-\frac{1}{s} \leq e^{-\frac{1}{s}} \Rightarrow (1-\frac{1}{s})^s \leq e^{-1}$

d'où la deuxième inégalité.

Analyse (1)

On note $\beta_s = 1 - (1 - \frac{1}{s})^s$.

$$\boxed{\mathbf{Pr}(\nu \models \vee_{k \leq n_j} \ell_{j,k}) \geq \beta_{n_j} z_j^* \text{ et } \mathbf{E}(N) \geq (1 - \tfrac{1}{e}) \sum_{j \leq m} z_j^*.}$$

Preuve.

Par définition du programme linéaire, $\sum_{k \leq n_j} w_{j,k}^* \geq z_j^*$.

$$\mathbf{Pr}(\nu \models \vee_{k \leq n_j} \ell_{j,k}) = 1 - \prod_{k \leq n_j} (1 - w_{j,k}^*) \geq 1 - \left(1 - \frac{z_j^*}{n_j}\right)^{n_j} \geq \beta_{n_j} z_j^*.$$

car $\prod_{k \leq n_j} (1 - w_{j,k}^*) \leq \left(1 - \frac{\sum_{k \leq n_j} w_{j,k}^*}{n_j}\right)^{n_j} \leq \left(1 - \frac{z_j^*}{n_j}\right)^{n_j}$

$\mathbf{E}(N) \geq \sum_{j \leq m} \beta_{n_j} z_j^* \geq (1 - \frac{1}{e}) \sum_{j \leq m} z_j^*$.

Remarque. $1 - \frac{1}{e} \approx 0.632$.

Ceci conduit à un algorithme de Las Vegas

qui répète un tirage aléatoire biaisé

jusqu'à l'obtention d'un nombre de clauses supérieur ou égal à $(1 - \frac{1}{e}) \sum_{j \leq m} z_j^*$.

Analyse (2)

Soit N_1 le nombre de clauses satisfaites par le choix non biaisé

et N_2 le nombre de clauses satisfaites par le choix biaisé.

$$\boxed{\max(\mathbf{E}(N_1), \mathbf{E}(N_2)) \geq \frac{3}{4} \sum_{j \leq m} z_j^*.}$$

Preuve.

$\mathbf{E}(N_1) = \sum_s \sum_{j|n_j=s} 1 - 2^{-s} \geq \sum_s \sum_{j|n_j=s} (1 - 2^{-s}) z_j^*$

$\frac{1}{2}(\mathbf{E}(N_1) + \mathbf{E}(N_2)) \geq \sum_s \sum_{j|n_j=s} \frac{1-2^{-s}+\beta_s}{2} z_i^*$

- Pour $s \geq 3$, $1 - 2^{-s} + \beta_s \geq 2 - \frac{1}{e} - 2^{-s} \geq 2 - \frac{1}{e} - \frac{1}{8} \geq \frac{3}{2}$.
- Pour $s = 2$, $\beta_2 = 1 - \frac{1}{4} = \frac{3}{4}$, $1 - 2^{-2} + \frac{3}{4} = \frac{3}{2}$.
- Pour $s = 1$, $\beta_1 = 1$, $1 - 2^{-1} + 1 = \frac{3}{2}$.

Ceci conduit à un algorithme de Las Vegas

qui répète un tirage aléatoire biaisé et un tirage aléatoire non biaisé

jusqu'à l'obtention d'un nombre de clauses supérieur ou égal à $\frac{3}{4} \sum_{j \leq m} z_j^*$.

Calcul du médian : le principe

Soit T un tableau de n éléments tous distincts avec n impair.

On cherche le *médian* m défini par $|\{i \mid T[i] < T[m]\}| = \lfloor \frac{n}{2} \rfloor$.

Il existe un algorithme simple en $O(n \log(n))$ qui consiste à trier T

et renvoyer l'élément médian.

Schéma d'un algorithme de Monte Carlo.

- On sélectionne aléatoirement un sous-ensemble *significatif* R de valeurs de T ;
- On trie R et on définit un intervalle autour du médian de R ;
- On sélectionne le sous-ensemble C d'éléments de T compris dans cet intervalle en comptant le nombre d'éléments inférieurs et supérieurs à cet intervalle ;
- Il y a échec si le médian ne se trouve pas dans C ou si sa taille est trop grande ;
- On trie C et on retrouve le médian de T à partir du tri.

Calcul du médian : l'algorithme

R est un tableau de taille $\lceil n^{\frac{3}{4}} \rceil$.

C est un tableau de taille variable $\leq n$.

For i **from** 1 **to** $\lceil n^{\frac{3}{4}} \rceil$ **do** $R[i] \leftarrow T[\text{Sample}(1 \ldots n, \text{uniform})]$
Sort$(R, \lceil n^{\frac{3}{4}} \rceil)$
$d \leftarrow R[\lfloor \frac{1}{2} n^{\frac{3}{4}} - \sqrt{n} \rfloor]$; $f \leftarrow R[\lceil \frac{1}{2} n^{\frac{3}{4}} + \sqrt{n} \rceil]$
$\ell_d \leftarrow 0$; $\ell_f \leftarrow 0$; $j \leftarrow 0$
For i **from** 1 **to** n **do**
  If $T[i] < d$ **then** $\ell_d \leftarrow \ell_d + 1$
  Else If $T[i] > f$ **then** $\ell_f \leftarrow \ell_f + 1$
  Else $j \leftarrow j + 1$; $C[j] \leftarrow T[i]$
If $\ell_d > \frac{n}{2}$ **or** $\ell_f > \frac{n}{2}$ **or** $j > 4n^{\frac{3}{4}}$ **then** $\mathbf{Return}(\text{Fail})$
Sort(C, j) ; $\mathbf{Return}(C[\lceil \frac{n}{2} \rceil - \ell_d])$

Observation. Cet algorithme peut être transformé en algorithme de Las Vegas.

Complexité et correction (probabiliste)

Complexité en $O(n)$.

L'initialisation de R se fait en $O(n^{\frac{3}{4}} \log(n))$, donc en $O(n)$.

Le tri de R se fait en $O(n^{\frac{3}{4}} \log(n^{\frac{3}{4}}))$, donc en $O(n)$.

La boucle principale se fait en $O(n)$.

Le tri (éventuel) de C se fait en $O(n^{\frac{3}{4}} \log(n^{\frac{3}{4}}))$, donc en $O(n)$.

Correction.

d et f sont des valeurs de T et $C[1, j]$ contient le sous-ensemble trié des valeurs comprises entre d et f.

Si $\ell_d \leq \frac{n}{2}$ alors $d \leq m$.

Si $\ell_f \leq \frac{n}{2}$ alors $f \geq m$.

Par conséquent m est une des valeurs $C[1, j]$
et plus précisément la $\lceil \frac{n}{2} \rceil - \ell_d$ ième valeur.

Probabilité d'échec (1)

La probabilité d'échec est majorée

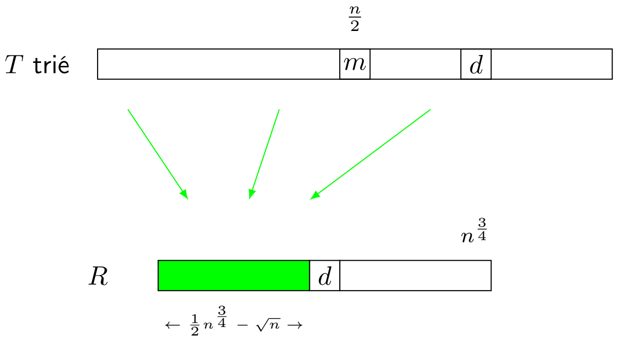
par la somme des probabilités des événements suivants :

E_1 Il y a moins de $\frac{1}{2}n^{\frac{3}{4}} - \sqrt{n}$ valeurs (répétées) de R inférieures ou égales à m.

E_2 Il y a moins de $\frac{1}{2}n^{\frac{3}{4}} - \sqrt{n}$ valeurs (répétées) de R supérieures ou égales à m.

E_3 Il y a plus de $4n^{\frac{3}{4}}$ valeurs de T entre d et f.

- $m < d$ **implique** E_1 **et** $f < m$ **implique** E_2**.**



Probabilité d'échec (2)

Rappel Chernoff-Hoeffding. Soit $(X_i)_{i \leq k}$ une famille de variables aléatoires indépendantes avec X_i à valeurs dans $[a_i, b_i]$.

Soit $X = \sum_{i \leq k} X_i$ et $\mu = \mathbf{E}(X)$. Alors : $\mathbf{Pr}(X \leq \mu - \varepsilon) \leq e^{-\frac{2\varepsilon^2}{\sum_{i \leq k}(b_i - a_i)^2}}$

Application.

Si la $i$ème valeur de R est inférieure ou égale à m alors $X_i = 1$ sinon $X_i = 0$.

Ici $[a_i, b_i] = [0, 1]$, $k = n^{\frac{3}{4}}$, $\mu = \frac{\lceil n/2 \rceil}{n} n^{\frac{3}{4}} \geq \frac{1}{2} n^{\frac{3}{4}}$ et $\varepsilon = \sqrt{n}$.

D'où $\mathbf{Pr}(E_1) \leq e^{-\frac{2n}{n^{3/4}}} = e^{-2n^{\frac{1}{4}}}$ et $\mathbf{Pr}(E_2) \leq e^{-2n^{\frac{1}{4}}}$.

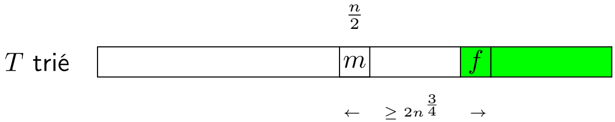
Probabilité d'échec (3)

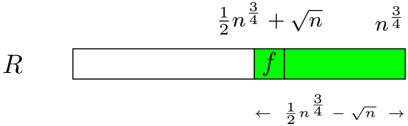
Si E_3 est réalisé alors :

$E_{3,1}$ soit il y a au moins $2n^{3/4}$ éléments supérieurs à m dans C ;

$E_{3,2}$ soit il y a au moins $2n^{3/4}$ éléments inférieurs à m dans C.

$E_{3,1}$ implique qu'il y a au moins $2n^{3/4}$ éléments compris entre m et f dans T.
D'où f est supérieure ou égale à $\frac{n}{2} + 2n^{3/4}$ valeurs de T
et R contient au moins $\frac{1}{2}n^{\frac{3}{4}} - \sqrt{n}$ valeurs supérieures ou égales à $\frac{n}{2} + 2n^{3/4}$ valeurs de T.





Probabilité d'échec (4)

Rappel Chernoff-Hoeffding. Soit $(X_i)_{i\leq k}$ une famille de variables aléatoires indépendantes avec X_i à valeurs dans $[a_i, b_i]$.

Soit $X = \sum_{i\leq k} X_i$ et $\mu = \mathbf{E}(X)$. Alors : $\mathbf{Pr}(X \geq \mu + \varepsilon) \leq e^{-\frac{2\varepsilon^2}{\sum_{i\leq n}(b_i - a_i)^2}}$

Application.

Si la $i$ème valeur de R plus grande que $\frac{n}{2} + 2n^{3/4}$ valeurs de T

(i.e., dans les $\frac{n}{2} - 2n^{3/4}$ plus grandes valeurs)

alors $X_i = 1$ sinon $X_i = 0$.

Ici $[a_i, b_i] = [0, 1]$, $k = n^{\frac{3}{4}}$, $\mu = n^{\frac{3}{4}}(\frac{1}{2} - 2n^{-\frac{1}{4}}) = \frac{1}{2}n^{\frac{3}{4}} - 2\sqrt{n}$ et $\varepsilon = \sqrt{n}$.

D'où $\mathbf{Pr}(E_{3,1}) \leq e^{-2n^{\frac{1}{4}}}$ et $\mathbf{Pr}(E_{3,2}) \leq e^{-2n^{\frac{1}{4}}}$.

Sélection du $k$ième plus grand élément

On applique l'algorithme récursif de séparation des valeurs avec deux différences :

- le séparateur est choisi uniformément parmi les valeurs du tableau ;
- Lors le tableau lié à un appel récursif est trop grand (supérieur à $\frac{3n}{4}$) on choisit à nouveau un séparateur plutôt que faire l'appel.

Select(T, n, k)

While True do

$med \leftarrow$ Sample$(1 \ldots n,$ uniforme$)$; $vmed \leftarrow T[med]$

$\ell_d \leftarrow 0$; $\ell_f \leftarrow 0$

For i **from** 1 **to** n **do**

If $T[i] < vmed$ **then** $\ell_d \leftarrow \ell_d + 1$; $T_d[\ell_d] \leftarrow T[i]$

Else If $T[i] > vmed$ **then** $\ell_f \leftarrow \ell_f + 1$; $T_f[\ell_f] \leftarrow T[i]$

If $k > \ell_d$ **and** $k \leq n - \ell_f$ **then return** $vmed$

If $k \leq \ell_d$ **and** $\ell_d \leq \frac{3n}{4}$ **then return** Select(T_d, ℓ_d, k)

If $k > n - \ell_f$ **and** $\ell_f \leq \frac{3n}{4}$ **then return** Select$(T_f, \ell_f, k - n + \ell_f)$

Analyse de complexité

- Soit $D_i = \{i \mid T[i] < vmed\}$. Par définition, $|D_i| = \ell_d$.
  med a été choisi parmi les $n - \ell_d$ autres indices.
  Si $\ell_d > \frac{3n}{4}$ alors med est choisi parmi moins de $\frac{n}{4}$ éléments.
  D'où $\mathbf{Pr}(\ell_d > \frac{3n}{4}) < \frac{1}{4}$.
- Soit $F_i = \{i \mid T[i] > vmed\}$. Par définition, $|F_i| = \ell_f$.
  med a été choisi parmi les $n - \ell_f$ autres indices.
  Si $\ell_f > \frac{3n}{4}$alors med est choisi parmi moins de $\frac{n}{4}$ éléments.
  D'où $\mathbf{Pr}(\ell_f > \frac{3n}{4}) < \frac{1}{4}$

Le nombre moyen d'itérations de la boucle **While** est donc majoré par 2.

Le temps local moyen d'un appel est donc majoré par cn

pour un certain c.

Le temps global moyen est majoré par $cn \sum_{i \in \mathbb{N}} \left(\frac{3}{4}\right)^i = 4cn = O(n)$.

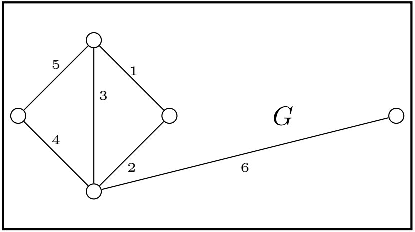
Contraction aléatoire d'un multi-graphe

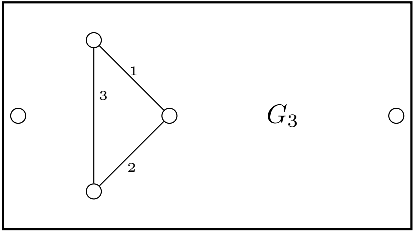
Soit un multi-graphe connexe $G = (V, E)$ avec $1 < n = |V|$ et $m = |E| \leq \frac{n(n-1)}{2}$.

On considère un ordonnancement aléatoire de $E = (e_i)_{i \leq m}$, uniformément choisi.

Pour tout $i \leq m$, $G_i = (V, (e_j)_{j \leq i})$.

Illustration.





Recherche de coupe par contraction

On cherche un i tel que G_i ait deux composantes connexes (CC) notées V_0 et V_1.

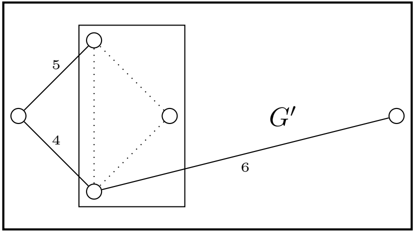
On calcule les CC de $G_{\lceil \frac{m}{2} \rceil}$: $\{W_k\}_{k \leq K}$.

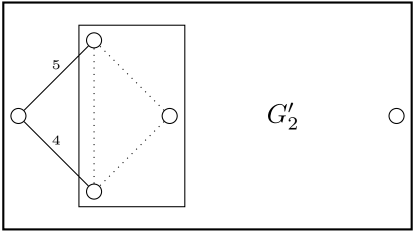
- Si $K = 2$ c'est gagné ;
- Si $K > 2$ alors on itère le procédé sur $G' = (\{W_k\}_{k \leq K}, (e_j)_{\frac{m}{2} < j \leq m})$;
- Si $K = 1$ alors on itère le procédé sur $G_{\lceil \frac{m}{2} \rceil}$.

Une itération s'effectue en temps linéaire par rapport au nombre des arêtes.

D'où une complexité en $O(m) + O(\frac{m}{2}) + \cdots = O(m)$.

Illustration.





Coupe minimale d'un multi-graphe

On cherche une partition $V = V_0 \uplus V_1$

telle que l'ensemble des arêtes joignant V_0 à V_1 soit de taille minimale.

Un algorithme de Monte Carlo pour la coupe minimale.

- On itère p fois le procédé précédent ;
- On renvoie la partition qui minimise la coupe.

Complexité de l'algorithme en $O(pm\log(n))$.

Quelle valeur choisir pour p ?

Observation.

Cet algorithme ne peut pas être transformé en un algorithme de Las Vegas.

Analyse de l'algorithme (1)

La probabilité qu'une coupe minimale soit trouvée
par une itération de l'algorithme est supérieure ou égale à $\frac{2}{n(n-1)}$.

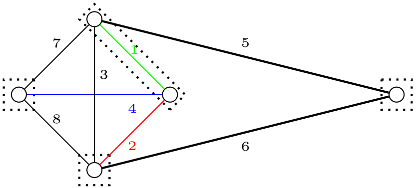
Preuve. Notons F les arêtes d'une coupe minimale $V = V_0 \uplus V_1$ et $k = |F|$.

Il suffit qu'à chaque diminution (jusqu'à deux) du nombre de CC

les arêtes de F ne soient pas choisies.

Invariant de boucle. Soit $\{W_\ell\}_{\ell \leq L}$ les composantes connexes de G_i.

Alors il existe $\{1, \ldots, L\} = I_0 \uplus I_1$ t.q. pour $s \in \{0, 1\}$, $V_s = \biguplus_{\ell \in I_s} W_\ell$.



Analyse de l'algorithme (2)

Preuve (suite). $m \geq \frac{kn}{2}$ car chaque sommet a au moins k arêtes.

Soit le multi-graphe des CC avant la $j+1$ème diminution de CC.

Puisque les arêtes de F n'ont pas été choisies, sa coupe minimale est "inchangée".

Il a $n-j$ sommets donc au moins $k(n-j)/2$ arêtes.

La probabilité de ne pas choisir une arête de F est minorée par

$$\frac{k(n-j)/2 - k}{k(n-j)/2} = \frac{n-2-j}{n-j}$$

La probabilité d'obtenir la coupe minimale est donc minorée par :

$$\prod_{j=0}^{n-3} \frac{n-2-j}{n-j} = \frac{2}{n(n-1)}$$

$\square$

D'où une probabilité d'erreur majorée par $(1 - \frac{2}{n(n-1)})^p \leq e^{-\frac{2p}{n(n-1)}}$.

Si $p = \frac{n(n-1)\log(n)}{2}$ alors la complexité de l'algorithme est $O(mn^2 \log^2(n))$

avec probabilité d'erreur inférieure ou égale à $\frac{1}{n}$.

Plan

Gestion de cache

La mémoire de l'ordinateur est constituée :

- d'une mémoire principale a priori infinie ;
- et d'une mémoire cache de n cellules.

On considère une séquence d'accès mémoire $\sigma \in \mathbb{N}^+$.

Une exécution mémoire $ex(\sigma) = op_1(i_i) \dots op_k(i_K)$ de σ est une suite d'opérations telles que pour tout $j \leq K$, $i_j \in \mathbb{N}$ et $op_j \in \{shortaccess, longaccess, unload\}$ vérifiant :

- la projection de $ex(\sigma)$ sur les accès est égale à σ ;
- la projection de $ex(\sigma)$ sur les opérations de $i \in \mathbb{N}$ est un préfixe de $(longaccess(i) shortaccess(i)^* unload(i))^\omega$;
- Pour tout préfixe $\rho = op_1(i_i) \dots op_k(i_k)$ de $ex(\sigma)$, $|\{j \leq k \mid op_j = longaccess\}| - |\{j \leq k \mid op_j = unload\}| \leq n$.

Dans la suite, on abrège shortaccess en sa, longaccess en la et unload en ul.

On définit $miss(ex(\sigma)) = |\{j \leq K \mid op_j = la\}|$ et $opt(\sigma) = \min(miss(ex(\sigma)))$.

Une famille d'algorithmes en ligne

La suite des opérations d'un algorithme « en ligne » générées par $\sigma[1, i]$ ne dépend pas de $\sigma[i+1, |\sigma|]$.

$Cache \leftarrow \emptyset$; $Mark \leftarrow \emptyset$ ($round \leftarrow 1$)
For i **from** 1 **to** $|\sigma|$ **do**
  If $\sigma(i) \in Cache$ **then** $\rho = \rho \cdot sa(\sigma[i])$
  Else
    If $|Cache| = n$ **then**
      If $Cache \setminus Mark = \emptyset$ **then** $Mark \leftarrow \emptyset$ ($round \leftarrow round + 1$)
      $j \leftarrow$ some item of $Cache \setminus Mark$
      $\rho \leftarrow \rho \cdot ul(j)$; $Cache \leftarrow Cache \setminus \{j\}$
    $\rho \leftarrow \rho \cdot la(\sigma[i])$; $Cache \leftarrow Cache \cup \{\sigma[i]\}$
  $Mark \leftarrow Mark \cup \{\sigma[i]\}$

Il s'agit d'une famille car le choix de j n'est pas précisé.

$Mark$ est l'ensemble des cellules accédées durant le tour courant.

L'algorithme LRU

$Cache \leftarrow \emptyset$ $(round \leftarrow 1\,;\ hround \leftarrow 1)$

For i **from** 1 **to** $|\sigma|$ **do**

  If $\exists(\sigma(i), k) \in Cache$ **then**

    $Cache \leftarrow Cache \setminus \{(\sigma(i), k)\} \cup \{(\sigma(i), i)\}\,;\ \rho = \rho \cdot sa(\sigma[i])$

  Else

    If $|Cache| = n$ **then**

      $j \leftarrow \arg\min(k \mid (j, k) \in Cache)$ $(old \leftarrow \min(k \mid (j, k) \in Cache)$

      (**If** $old \geq hround$ **then** $round \leftarrow round + 1\,;\ hround \leftarrow i$)

      $\rho \leftarrow \rho \cdot unload(j)\,;\ Cache \leftarrow Cache \setminus \{j\}$

    $\rho \leftarrow \rho \cdot la(\sigma[i])\,;$

  $Cache \leftarrow Cache \cup \{(\sigma[i], i)\}$

En définissant $Mark = \{j \mid (j, k) \in Cache \wedge k \geq hround\}$,

on vérifie que l'algorithme LRU (least recently use) appartient à la famille.

Analyse : une borne inférieure

$$\boxed{\sigma = (12\dots n+1)^{rn+1}}$$

L'exécution optimale de σ est :

$la(1)\dots la(n)ul(n)la(n+1)sa(1)\dots sa(n-1)ul(n-1)la(n)$

$sa(n+1)sa(1)\dots sa(n-2)ul(n-2)la(n-1)\dots la(1)sa(2)\dots sa(n-1)ul(n)la(n+1)$

avec $opt(n) = (r+1)(n+1)$

L'exécution ρ générée par LRU est :

$la(1)\dots la(n)ul(1)la(n+1)ul(2)la(1)\dots ul(n+1)la(n)ul(1)la(n+1)\dots$

avec $miss(\rho) = (rn+1)(n+1)$.

D'où un ratio $\frac{miss(\rho)}{opt(n)} = \frac{rn+1}{r+1} \sim n$ quand r est grand.

Analyse : une borne supérieure (1)

On décompose les accès d'une séquence arbitraire σ

selon les tours de l'exécution ρ d'un algorithme en ligne de la famille.

Observation. Cette décomposition ne dépend pas du choix de l'algorithme.

Sans perte de généralité, on suppose que σ effectue $r \geq 2$ tours

dont le dernier partiellement.

Pourquoi ?

Chaque tour débute avec l'accès qui a provoqué le « démarquage » du cache.

$$\sigma = \ldots i_1 \cdot (i_1^* || i_2^+ || \ldots || i_n^+) \cdot i_1' \ldots$$

avec $i_1, i_2, \ldots, i_n, i_1'$ tous distincts.

Analyse : une borne supérieure (2)

- L'algorithme effectue au plus un accès long par i_k lors du tour pour $k \leq n$.
  D'où : $miss(\rho) \leq rn$.

- Soit ρ^*, une exécution optimale σ.

  Lors du premier tour ρ^* effectue n accès longs.

  Soit $\sigma'(i_1^* || i_2^+ || \ldots || i_n^+) \cdot i_1' \ldots$ la sous-séquence de σ

  Au début de l'exécution de σ' par ρ^*, i_1 est en mémoire.

  Donc l'un des (premiers) accès à $i_2, \ldots, i_n, i_1'$ sera un accès long.

  Par conséquent $miss(\rho') \geq n + r - 2$.

D'où :

$$\frac{miss(\rho)}{opt(n)} \leq \frac{rn}{n + r - 2} \sim n$$

quand r est grand.

Un algorithme en ligne probabiliste

$Cache \leftarrow \emptyset\,;\ Mark \leftarrow \emptyset\ (round \leftarrow 1)$

For i **from** 1 **to** $|\sigma|$ **do**

  If $\sigma(i) \in Cache$ **then** $\rho = \rho \cdot sa(\sigma[i])$

  Else

    If $|Cache| = n$ **then**

      If $Cache \setminus Mark = \emptyset$ **then** $Mark \leftarrow \emptyset\ (round \leftarrow round + 1)$

      $j \leftarrow \text{Random}(Cache \setminus Mark, \text{uniforme})$

      $\rho \leftarrow \rho \cdot ul(j)\,;\ Cache \leftarrow Cache \setminus \{j\}$

    $\rho \leftarrow \rho \cdot la(\sigma[i])\,;\ Cache \leftarrow Cache \cup \{\sigma[i]\}$

  $Mark \leftarrow Mark \cup \{\sigma[i]\}$

Analyse (1)

Soit σ une séquence d'accès mémoire avec r tours et une exécution optimale.

On dit qu'une cellule mémoire i est *fraiche* au tour $j+1$ si :

- i n'appartenait pas à $Mark$ à la fin du tour j ;
- i appartient à $Mark$ à la fin du tour $j+1$.

On note c_j le nombre de cellules fraiches du tour j.

Observation. Il y a $n + c_j$ cellules accédées durant les tours j et $j+1$.

Il y a donc au moins c_j accès longs durant les tours j et $j+1$.

Soit o_j le nombre d'accès longs durant le tour j avec $o(0) = 0$. Alors :

$$2opt(\sigma) \geq \sum_{j=1}^{r-1} o_j + o_{j+1} \geq \sum_{j=0}^{r-1} c_{j+1} = \sum_{j=1}^{r} c_j$$

D'où : $opt(\sigma) \geq \frac{1}{2} \sum_{j=1}^{r} c_j$

Analyse (2)

- On considère ρ l'exécution aléatoire de l'algorithme probabiliste sur σ.
- Il y a au moins c_j accès à des cellules fraiches durant le tour j dont exactement c_j accès longs.
- Soit $NF_j = \{i_1, \ldots, i_{n-c_j}\}$ l'ensemble des cellules non fraiches du tour j ordonnées selon leur premier accès durant le tour j.
- Soit $i_k \in NF_j$, la variable aléatoire L_{i_k} vaut :
  - 1 si le premier accès à i_k durant le tour j est un accès long ;
  - 0 sinon.
- m_j, le nombre moyen d'accès longs durant le tour j, est égal à :

$$c_j + \sum_{k \leq n-c_j} \mathbf{E}(L_{i_k})$$

Analyse (3)

Si $c_j = n$ alors $m_j = c_j$ sinon $j > 1$.

Soit J les cellules dans le cache à la fin du tour $j-1$. On a $NF_j \subseteq J$.

Considérons le premier accès à i_k et notons $c \leq c_j$, le nombre de cellules fraiches actuellement dans le cache. Le cache contient alors :

- $\{i_1, \ldots, i_{k-1}\}$, les cellules non fraiches déjà accédées ;
- c cellules fraiches ;
- $n - c - k - 1$ cellules de $J \setminus \{i_1, \ldots, i_{k-1}\}$.

D'autre part, c cellules de J ont quitté le cache.

i_k appartient soit aux $n - c - k - 1$ cellules de $J \setminus \{i_1, \ldots, i_{k-1}\}$ dans le cache, soit aux c cellules de J qui ont quitté le cache (de manière uniforme).

La probabilité que i_k ne soit plus dans le cache est donc égale à $\frac{c}{n-k+1} \leq \frac{c_j}{n-k+1}$.

$m_j \leq c_j(1 + \sum_{k \leq n-c_j} \frac{1}{n-k+1}) = c_j(1 + \sum_{c_j+1 \leq \ell \leq n} \frac{1}{\ell}) \leq c_j \sum_{\ell \leq n} \frac{1}{\ell} \leq c_j(1 + \log(n))$

D'où :

$$\boxed{\frac{\mathbf{E}(miss(\rho))}{opt(\sigma)} \leq 1 + \log(n)}$$

Développements possibles

- Tri rapide probabiliste : présentation et analyse de complexité

- Structure à trous : présentation et analyse (partielle) de complexité

- Dérandomisation de l'algorithme probabiliste de recherche de coupe maximale

- Algorithme probabiliste de recherche de coupe minimale