Cours 3 — 18 janvier

Enseignant : Frédéric Magniez *Rédacteur : Frédéric Maillasson*

3.1 Problème 3-SAT

3.1.1 Première tentative : Application de WALKSAT

On peut, dans un premier temps, appliquer au problème 3-SAT le même algorithme que pour 2-SAT.

ALGORITHME (WALKSAT)

1. Choisir arbitrairement $a \in \{0,1\}^n$
2. Tant que $\phi(a) = 0$, répéter :

   Choisir une clause C_j telle que $C_j(a) = 0$

   Changer la valeur d'une variable de C_j au hasard

On suppose que s est une affectation telle que $\phi(s) = 1$, et on étudie l'algorithme modifié suivant :

ALGORITHME (WALKSAT MODIFIÉ)

1. Choisir arbitrairement $a \in \{0,1\}^n$
2. Tant que $\phi(a) = 0$, répéter :

   Choisir une clause C_j telle que $C_j(a) = 0$

   Changer la valeur d'une variable de C_j au hasard

   Si $\phi(a) = 1$ mais que $a \neq s$, on modifie un bit de a sur lequel s diffère.

On remarque que le nombre d'itérations de la boucle dans le second algorithme *majore* le nombre d'itérations de la boucle dans le premier.

Pour tout $i \in \mathbb{N}$, notons X_i le nombre de bits sur lesquels a et s diffèrent après i itérations. On a alors :

$$\mathbb{P}[X_{i+1} = n-1 | X_i = n] = 1$$

$$\mathbb{P}[X_{i+1} = 0 | X_i = 0] = 1$$

$$\forall 1 \leq j \leq n-1, \qquad \mathbb{P}[X_{i+1} = j-1 | X_i = j] \geq \frac{1}{3}$$

$$\forall 1 \leq j \leq n-1, \qquad \mathbb{P}[X_{i+1} = j+1 | X_i = j] \leq \frac{2}{3}$$

De même que dans l'analyse de l'algorithme pour 2-SAT, on considère donc le graphe linéaire dont l'ensemble des états est $\{0, 1, \ldots, n\}$, avec les probabilités de transition :

$$\forall (n, m), \qquad \mathbb{P}_{n \to m} = \mathbb{P}[X_{i+1} = m | X_i = n]$$

L'analyse du nombre d'itérations dans l'algorithme modifié se ramène au calcul du temps moyen pour atteindre 0 en effectuant une marche aléatoire sur le graphe précédent, en partant de l'état n, que l'on note h_n. Dans le pire des cas [1], ce temps moyen est donné par la relation de récurrence :

$$\forall 1 \leq j \leq n-1, \qquad h_j = 1 + \frac{1}{3} h_{j-1} + \frac{2}{3} h_{j+1}$$

avec les conditions aux limites $h_0 = 0$, $h_n = 1 + h_{n-1}$.

On obtient ainsi un temps moyen qui exponentiel :

$$h_n = \Theta(2^n)$$

Cette "explosion", par rapport à 2-SAT [2], du temps de calcul moyen, est liée au fait que la probabilité de s'éloigner de 0 est ici supérieure à la probabilité de s'en rapprocher. En parcourant le graphe aléatoirement, on aura naturellement tendance à s'éloigner de l'origine à long terme.

3.1.2 Une amélioration du temps de calcul moyen

Etant donné que cette tendance à s'éloigner de l'état 0, on peut penser qu'au lieu de pousser la marche aléatoire jusqu'à tomber sur l'état 0, on a intérêt à repartir, de manière régulière, d'une position aléatoire. D'où le nouvel algorithme :

ALGORITHME (WALKSAT MODIFIÉ 2)

1. $a \leftarrow (0, \ldots, 0)$
2. Tant que $\phi(a) = 0$, répeter
   (a) Choisir aléatoirement $a \in \{0, 1\}^n$
   (b) Tant que $\phi(a) = 0$, répéter **au plus** $3n$ **fois** :
   Choisir une clause C_j telle que $C_j(a) = 0$
   Changer la valeur d'une variable de C_j au hasard
   Si $\phi(a) = 1$ mais que $a \neq s$, on modifie un bit de a sur lequel s diffère.
3. Renvoyer a

Comme prévu, l'algorithme modifié est plus efficace que l'algorithme WALKSAT :

1. c'est-à-dire dans le cas où $\mathbb{P}_{n \to n+1} = \frac{2}{3}$ et où $\mathbb{P}_{n \to n-1} = \frac{1}{3}$
2. On rappelle que pour 2-SAT, on avait $h_n = n^2$

Théorème 3.1 (Schoening, 1999). *Dans l'algorithme précédent, le nombre moyen d'exécutions de la boucle 2 est majoré par :*

$$\Theta(\sqrt{n}(\frac{4}{3})^n)$$

Preuve: Soit :

$q = \mathbb{P}_{a_0}[\text{L'algorithme trouve une solution en } 3n \text{ itérations de la boucle (b), en partant de } a = a_0]$

Et :

$q_j = \mathbb{P}_{a_0}[\text{L'algorithme trouve une solution en } 3n \text{ itérations de la boucle (b), en partant de } a = a_0 | X_0 = j]$

On a :

$$q = \sum_j q_j \mathbb{P}[X_0 = j] = \sum_j q_j \frac{1}{2^n}\binom{n}{j}$$

Or :

$q_j \geq \mathbb{P}_{a_0}[\text{L'algorithme trouve en au plus } 3j \text{ itérations, en partant de } a = a_0 | X_0 = j]$

Or, partant de $X_0 = j$, et effectuant $2j$ transitions vers la gauche [3] et j transitions vers la droite, dans un ordre quelconque, on atteint forcément l'état 0 en au plus $3j$ itérations. Donc :

$q_j \geq \mathbb{P}_{a_0}[\text{L'algorithme effectue } 2j \text{ transitions vers la gauche et } j \text{ transitions vers la droite}]$

Cette dernière quantité se calcule facilement : la probabilité d'effectuer $2j$ transitions vers la gauche et j transitions vers la droite est $(\frac{1}{3})^{2j}(\frac{2}{3})^j$, et il y a $\binom{3j}{j}$ manières d'ordonner ces transitions. Donc :

$$q \geq \sum_j \frac{1}{2^n}\binom{n}{j}\frac{2^j}{3^{3j}}\binom{3j}{j}$$

Par la formule de Stirling, on a pour $j \neq 0$:

$$\binom{3j}{j} \sim \frac{1}{\sqrt{2\pi}}\sqrt{\frac{3}{2j}}\frac{3^{3j}}{2^{2j}}$$

Donc :

$$\begin{aligned} q &\geq \Theta(1)[\frac{1}{2^n} + \sum_{j=1}^{n} \frac{1}{2^n}\binom{n}{j}\frac{1}{\sqrt{j}}\frac{1}{2}] \\ &\geq \Theta(1)[\frac{1}{2^n} + \sum_{j=1}^{n} \frac{1}{2^n}\binom{n}{j}\frac{1}{\sqrt{n}}\frac{1}{2}] \\ &\geq \Theta(1)[\frac{1}{2^n} + \frac{1}{2^n}\frac{1}{\sqrt{n}}(1+\frac{1}{2})^n] \\ &\geq \Theta[\frac{1}{\sqrt{n}}(\frac{3}{4})^n] \end{aligned}$$

On conclut à l'aide du lemme suivant :

3. des transitions du type $k \to k-1$

Lemme 3.2. *Soit $(z_k)_{k\in\mathbb{N}}$ une suite de variables aléatoire booléenne indépendante, telle que :$\forall i \in \mathbb{N}, \mathbb{P}[z_i = 1] = p$. Alors :*

$$\mathbb{E}[\min\{t/z_t = 1\}] = \frac{1}{p}$$

□

L'algorithme reste donc exponentiel, mais la constante sous l'exposant est meilleure que pour l'algorithme WALKSAT initial. Le même raisonnement donne, pour tout $k \geq 3$, un algorithme de complexité moyenne $\sqrt{n}(2 - \frac{1}{k})^n$.

3.1.3 Application : 2-coloriage d'un graphe 3-coloriable

Soit $G = (X, E)$ un graphe non-orienté *3-coloriable*, c'est-à-dire qu'il existe une application $C : X \to [\![0; 2]\!]$ telle que

$$\forall\, (x, y) \in X^2,\ (x, y) \in E \Rightarrow c(x) \neq c(y)$$

On cherche un algorithme qui colorie G avec seulement 2 couleurs, de telle sorte qu'aucun triangle ne soit mono-chromatique. Ceci est toujours possible pour un graphe 3-coloriable : en effet, il suffit de "fusionner" deux couleurs pour obtenir un graphe bicolore dans lequel aucun triangle n'est monochromatique.

On remarque qu'à un triangle (u, v, w), on peut associer une clause $C_{u,v,w}$ vraie si, et seulement si, le triangle (u, v, w) n'est pas monochromatique. En effet, si $c_u \in \{0, 1\}$ est la couleur du sommet u, alors (u, v, w) n'est pas monochromatique si, et seulement si, $C_{u,v,w} = 1$ avec :

$$C_{u,v,w} = \neg(a_u = a_v = a_w) = (a_u \vee a_v \vee a_w) \wedge ((\neg a_u) \vee (\neg a_v) \vee (\neg a_w))$$

Ceci ramène la résolution du problème à un problème 3-SAT. Dans ce cas particulier, l'algorithme est cependant polynomial :

2-COLORATION D'UN GRAPHE 3-COLORIABLE

1. Choisir une coloration $a : G \to \{\text{Bleu, Vert}\}$ quelconque
2. Tant qu'il existe un triangle monochromatique dans G colorié par a :
   - Choisir un triangle monochromatique
   - Changer la couleur de l'un des sommets au hasard
3. Retourner a

Soit $b : G \to \{\text{R, B, V}\}$ une 3-coloration du graphe. Pour $i \in \mathbb{N}$, on note X_i le cardinal, après i itérations de la boucle, de l'ensemble :

$$\{u \in V\ /\ (b(u) = B \wedge a(u) = V) \vee (b(u) = V \wedge a(u) = B)\}$$

Soit (u, v, w) un triangle monochromatique à l'étape i, par exemple $a(u) = a(v) = a(w) = B$. On peut de plus supposer, sans perte de généralité, que $b(u) = R, b(v) = V, b(w) = B$. Alors :

– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de u : $a(u) = V$. On a alors : $X_{i+1} = X_i$;
– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de v : $a(v) = V$. On a alors : $X_{i+1} = X_i - 1$;
– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de w : $a(w) = V$. On a alors : $X_{i+1} = X_i + 1$.

En notant, comme dans la section précédente, h_n le temps moyen pour atteindre l'état 0 par une marche aléatoire sur le graphe linéaire $\{0, 1, \ldots, n\}$ muni des probabilités de transition $\mathbb{P}_{n\to m} = \mathbb{P}[X_{i+1} = m | X_i = n]$, on obtient :

$$\forall 1 \leq j \leq n, \qquad h_j = 1 + \frac{1}{3}(h_{j-1} + h_j + h_{j+1})$$

On obtient ainsi $h_n = \Theta(n^2)$: l'algorithme est bien polynomial.

3.2 ST-connectivité

3.2.1 Algorithme général

Soient $G = (V, E)$ un graphe non-orienté et $(s, t) \in V^2$ deux sommets fixés de G. On cherche à déterminer s'il existe un chemin de s à t dans G. On note $|V| = n$ et $|E| = m$

Il existe des algorithmes déterministes (programmation dynamique notamment) de complexité $\mathcal{O}(m + n)$ en temps, mais $\mathcal{O}(n)$ en espace, ce qui peut poser problème si le graphe est trop gros (graphe d'Internet par exemple). On cherche ici un algorithme plus lent, mais moins gourmand en mémoire.

Considérons l'algorithme suivant :

ST-Connectivité

1. $u \leftarrow s$
2. Tant que $u \neq s$, répéter :

   Choisir v aléatoirement dans $\{v' \in V / \{u, v'\} \in E\}$

   $u \leftarrow v$

On voit immédiatement que cet algorithme est moins gourmand en espace (complexité $\mathcal{O}(\log n)$: il suffit de garder en mémoire le sommet actuel u). Il reste assez performant en temps, comme le prouve le théorème suivant et son corollaire :

Théorème 3.3. *Si il existe un chemin de s à t dans G, alors l'algorithme termine en un temps borné par $4nm$.*

Corollaire 3.4. *L'algorithme de* ST-connectivité *randomisé est de complexité $\mathcal{O}(\log n)$ en mémoire et $\mathcal{O}(4mn)$ en temps.*

3.2.2 Preuve du théorème

On suppose donc qu'il existe un chemin de s à t. Quitte à ne s'intéresser qu'à la composante connexe de s et t, on peut supposer G connexe.

Marche aléatoire

Soit P la matrice définie par :

$$P_{ij} : \begin{cases} \deg(i)^{-1} & \text{si} \{i,j\} \in E \\ 0 & \text{sinon} \end{cases}$$

Avec les notations de l'algorithme, on a :

$$\mathbb{P}[v = j | u = i] = P_{ij}$$

On admet le résultat suivant :

Lemme 3.5. *Il existe une unique distribution stationnaire* Π*, i.e. une unique application* $\Pi : V \to \mathbb{R}$ *telle que :*

$$\begin{aligned} \forall i \in V, \quad \Pi(i) &\geq 0 \\ \sum_{i \in V} \Pi(i) &= n \\ \forall i \in V, \quad \Pi(i) &= \sum_{\{i,j\} \in E} \Pi(j) \frac{1}{\deg(j)} \end{aligned}$$

Elle est donnée par :

$$\Pi(i) = \frac{\deg(i)}{2m}$$

Hitting time

Définition 3.6 (Hitting time). *Pour deux sommets* u *et* v*, on définit le* hitting-time $h_{u,v}$ *comme le temps moyen pour atteindre* v *en partant de* u*, et en effectuant une marche aléatoire sur le graphe. Prouver le théorème revient à montrer :* $h_{s,t} \leq 4nm$*.*

On admet le résultat suivant :

Théorème 3.7. *Si* $u \in V$*, le temps moyen de retour à* u *est :*

$$h_{u,u} = \frac{1}{\Pi(u)}$$

Montrons alors le lemme suivant :

Lemme 3.8. *Soit* $(u,v) \in V^2$ tel que $\{u,v\} \in E$*. Alors :*

$$h_{u,v} \leq 2m - 1$$

Preuve:

$$\begin{aligned} h_{u,u} &= \frac{1}{\deg(u)} \sum_{\{u,w\}\in E} 1 + h_{w,u} \\ &\geq \frac{1}{\deg(u)}(1 + h_{v,u}) \end{aligned}$$

Donc $h_{v,u} \leq \deg(u) h_{u,u} - 1$, et d'après 3.5 et 3.7 :

$$h_{v,u} \leq 2m - 1$$

□

Cover time

Définition 3.9 (Cover time). *On définit le* temps de couverture du graphe $C(G)$ *comme le maximum, pour $u \in V$, du temps moyen nécessaire pour parcourir tous les sommets au moins une fois, en partant de u, et en effectuant une marche aléatoire sur le graphe.*

Pour tout $(u, v) \in V^2$, on a clairement $h_{u,v} \leq C(G)$. On aura donc montré le théorème si l'on prouve que $C(G) \leq 4nm$, ce qui découle du lemme suivant :

Lemme 3.10.

$$C(G) \leq 2(n-1)(2m-1) \leq 4nm$$

Preuve: Fixons $u \in V$. En effectuant un parcours en profondeur d'un arbre couvrant de G, en partant de u, on obtient un tour $T = (u_1 = u, u_2, \ldots, u_N = u)$ passant par tous les sommets de G, et tel que $\{u_i u_{i+1}\} \in E$. Ce tour passe au plus 2 fois par chaque arête de l'arbre couvrant, qui compte n sommets et donc $n - 1$ arêtes. Donc :

$$N \leq 2(n-1)$$

On remarque ensuite que $C(G)$ est inférieur au temps moyen de couverture selon un chemin partant de u, puis passant par u_2, puis par u_3, ..., puis par u_N, soit :

$$C(G) \leq h_{u_1,u_2} + \cdots + h_{u_{N-1},u_N}$$

Or pour tout i, $\{u_i u_{i+1}\} \in E$, donc d'après le lemme 3.8 : $h_{u_i,u_{i+1}} \leq 2m - 1$, et donc :

$$C(G) \leq N - 1 \cdot (2m-1) \leq 2(n-1)(2m-1)$$

□

3.2.3 Exemples

Graphe linéaire

Si $V = \{1, \ldots, n\}$ et $E = \{\{i, i+1\}/i \in \{1, \ldots, n-1\}\}$, on trouve d'après le lemme 3.10 :

$$C(G) \leq \Theta(n^2)$$

On remarque que cette borne est optimale (à un facteur multiplicatif près), car on avait calculé, dans l'analyse de l'algorithme WALKSAT pour 3-SAT, que le temps moyen pour atteindre l'état 0 en partant de l'état n était de l'ordre de n^2.

Graphe complet

Dans le cas d'un graphe complet, on trouve d'après le lemme 3.10 :

$$C(G) \leq 4n \cdot \frac{n(n-1)}{2} = \Theta(n^3)$$

Ce résultat n'est pas optimal ; on peut montrer que $C(G) \sim n \log n$.

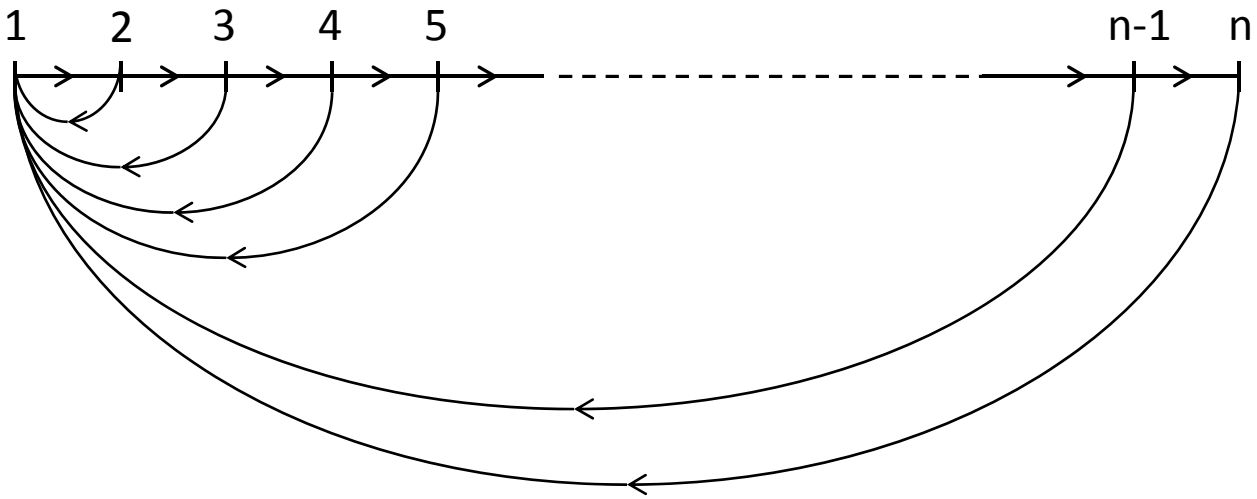
Lollipop

Un graphe en *lollipop*, obtenu en raccordant un graphe linéaire (comptant $\frac{n}{2}$ sommets) à un sommet d'un graphe complet (comptant aussi $\frac{n}{2}$ sommets), on trouve d'après le lemme 3.10 :

$$C(G) \leq \Theta(n^3)$$

Ce résultat est optimal.

Cas des graphes orientés



Pour le graphe orienté ci-dessus, on peut montrer que : $C(G) = \Theta(2^n)$. $C(G)$ n'est donc plus borné polynomialement par n dans le cas d'un graphe orienté.