Marches aléatoires – WalkSat

Bruno Grenet

Septembre 2017

L'objet de ce chapitre est de comprendre comment utiliser les propriétés des marches aléatoires pour concevoir un algorithme. L'exemple choisi est l'algorithme WalkSat qui permet de résoudre le problème k-sat en temps $O((2(k-1)/k)^n n^c)$ où n est le nombre de variables et c une constante. Par exemple pour 3-sat, la complexité est $O((4/3)^n n^c)$, à comparer avec l'algorithme de recherche exhaustive en $O(2^n n^{c'})$.

1 Marches aléatoires

Une marche aléatoire sur un ensemble discret E consiste à se déplacer aléatoirement sur les éléments de E (qu'on appelle *états*), en suivant certaines probabilités. On peut voir E comme un graphe orienté (potentiellement infini) avec des poids sur les arcs : un arc de poids p entre deux sommets (ou états) s et t de E signifie que lorsqu'on est sur le sommet s, on passe à l'étape suivante sur le sommet t avec probabilité $p_{s,t}$. On impose donc en particulier que pour un sommet s donné, la somme des probabilités des arcs issus de s soit égale à 1 : $\sum_{t\in E} p_{s,t} = 1$ pour tout s, avec la convention que $p_{s,t} = 0$ s'il n'y a pas d'arc de s à t. On s'intéresse à la suite des positions occupées au départ d'un sommet fixé et qu'on effectue la marche aléatoire. Ce modèle est très lié aux chaînes de Markov.

1.1 Une marche aléatoire particulière

On va s'intéresser à une marche aléatoire particulière. L'ensemble des états est l'ensemble $\mathbb{N}$ des entiers naturels. Depuis un état $i > 0$, on peut passer dans l'état $(i-1)$ avec probabilité p et dans l'état $(i+1)$ avec probabilité $(1-p)$. Depuis l'état 0, on ne peut que rester dans le même état, avec probabilité 1.

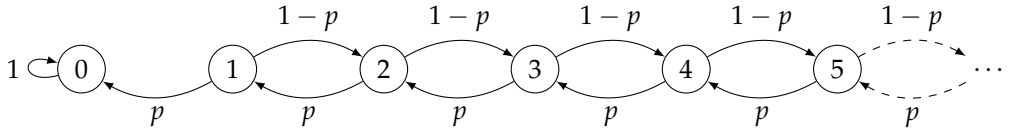

Figure 1 – Graphe de la marche aléatoire

La question qui nous intéresse est la suivante : étant donné un point de départ d, quelle est la probabilité P_d d'atteindre l'origine en un nombre fini d'étapes ? Puisque de l'état d, on va soit en

$d-1$ soit en $d+1$, on obtient la récurrence

$$P_d = p \times P_{d-1} + (1-p) \times P_{d+1}.$$

Le seul cas de base dont on dispose est *a priori* $P_0 = 1$ (car on reste infiniment dans l'état 0). On peut en fait obtenir un deuxième « cas de base » : $\lim_{d\to+\infty} P_d = 0$ car si on se trouve dans un état « infiniment éloigné » de l'origine, on ne pourra l'atteindre en temps fini. On suppose que $0 < p < 1$ car les cas extrémaux $p = 0$ et $p = 1$ sont triviaux : si $p = 0$ on s'éloigne toujours de l'origine et $P_d = 0$ pour $d > 0$ et c'est le contraire pour $p = 1$.

1.2 Résolution de la récurrence

La récurrence que l'on doit résoudre est de la forme $u_{n+2} = au_{n+1} + bu_n$. Remarquons en premier lieu que pour tout couple de valeur (u_0, u_1), il existe une unique solution u_n puisque la valeur de u_n pour $n > 1$ se déduit des valeurs initiales par récurrence.

Pour trouver la forme des solutions, on considère le *polynôme caractéristique* de l'équation de récurrence : $F(X) = X^2 - aX - b$. On distingue deux cas, selon que F admet deux racines distinctes ou une racine double.

Théorème 1.1. *Soit $(u_n)_{n\in\mathbb{N}}$ définie par $u_{n+2} = au_{n+1} + bu_n$. Alors il existe deux constantes λ et μ telles que pour tout n,*

— *$u_n = \lambda r_0^n + \mu r_1^n$ si le polynôme caractéristique admet deux racines distinces r_0 et r_1 ;*

— *$u_n = (\lambda + \mu n) r^n$ si le polynôme caractéristique admet une racine double.*

Démonstration. Chacun des deux cas se démontre aisément par récurrence. Les deux constantes sont fixées par les valeurs de u_0 et u_1.

Dans le premier cas, si $u_n = \lambda r_0^n + \mu r_1^n$\$ et $u_{n+1} = \lambda r_0^{n+1} + \mu r_1^{n+1}$\$, alors

$$u_{n+2} = au_{n+1} + bu_n = \lambda r_0^n (ar_0 + b) + \mu r_1^n (ar_1 + b).$$

Mais puisque r_0 et r_1 sont racines de F, $ar_0 + b = r_0^2$ et de même pour r_1. Donc $u_{n+2} = \lambda r_0^{n+2} + \mu r_1^{n+2}$, ce qu'il fallait démontrer.

Le cas d'une racine double r se démontre de manière similaire, en utilisant de plus le fait que $r = a/2$.

□

On peut appliquer ce résultat à la récurrence sur P_d. On peut la réécrire sous la forme

$$P_{d+2} = \frac{1}{1-p} P_{d+1} + \frac{p}{1-p} P_d.$$

Par un calcul facile, on vérifie que les racines du polynôme caractéristique sont 1 et $p/(1-p)$. Comme on a supposé $0 < p < 1$, $p/(1-p)$ est bien défini. On distingue trois cas.

— Si $p < 1/2$, $p/(1-p) < 1$. On est dans le premier cas du théorème et

$$P_d = \lambda 1^d + \mu \left(\frac{p}{1-p}\right)^d$$

pour deux constantes λ et μ. Les conditions initiales $P_0 = 1$ et $\lim_{d\to\infty} P_d = 0$ impliquent $\lambda = 0$ et $\mu = 1$. Autrement dit,

$$P_d = \left(\frac{p}{1-p}\right)^d \quad \text{pour tout } d \geq 0.$$

— Si $p > 1/2$, $p/(1-p) > 1$. On est à nouveau dans le premier cas, et les conditions initiales imposent cette fois-ci $\lambda = 1$ et $\mu = 0$, c'est-à-dire $P_d = 1$ pour tout d.
— Enfin si $p = 1/2$, le polynôme caractéristique a 1 comme racine double, et $P_d = (\lambda + \mu d)1^d$. Les conditions initiales imposent $\lambda = 1$ et $\mu = 0$, d'où $P_d = 1$ ici également.

On en déduit donc que lorsque $p \geq 1/2$, une marche aléatoire partant de l'état d repasse avec probabilité 1 à son point de départ. Lorsque $p < 1/2$, la probabilité devient $(p/(1-p))^d$.

Remarque. Le troisième cas aurait pu se déduire *par continuité*. En effet, quand $p > 1/2$, $P_d = 1$. Et quand $p < 1/2$, $P_d = (p/(1-p))^d$, or cette quantité tend vers 1 lorsque p tend vers $1/2$. Pour rendre cet argument formellement correct, il ne reste plus qu'à démontrer que P_d est une fonction continue de p.

1.3 Probabilité de succès en temps borné

On cherche maintenant à estimer la probabilité que la marche aléatoire atteigne l'origine en un temps borné. Pour simplifier les calculs, on se place dans le cas qui nous intéresse ensuite : on calcule la probabilité d'atteindre l'origine depuis l'état d après au plus $N = 3d$ pas quand $p = 1/3$. On rappelle que dans ce cas, $P_d = 2^{-d}$.

Considérons la marche aléatoire sur les entiers (positifs et négatifs) telle qu'on passe de l'état i à l'état $i-1$ avec probabilité $1/3$ et à l'état $i+1$ avec probabilité $2/3$. Alors la probabilité d'aller de l'état $d > 0$ à l'origine avec k pas vers la droite (à l'opposé de l'origine) et $k+d$ pas vers la gauche est $\binom{2k+d}{k}(1/3)^{k+d}(2/3)^k$ (le coefficient binomial est le nombre de tels chemins, et chacun arrive avec la même probabilité). Ceci est une borne inférieure pour la probabilité, dans notre marche aléatoire, d'être à l'origine après $2k+d$ pas dont k vers la droite puisque dans notre marche aléatoire, on reste en l'état 0 dès qu'on l'atteint.

On peut atteindre l'origine depuis d en $3d$ pas avec k pas vers la droite pour k entre 0 et d. La probabilité q_d d'atteindre 0 en $3d$ pas vérifie donc $q_d \geq \max_{0\leq k\leq d} \binom{2k+d}{k}(2/3)^{k+d}(1/3)^k$. En particulier pour $k = d$, $q_d \geq \binom{3d}{d}(1/3)^{2d}(2/3)^d$.

On peut simplifier cette formule en utilisant la formule de Stirling $n! \sim (n/e)^n\sqrt{2\pi n}$. On obtient après calculs $\binom{3d}{d} \sim \frac{c}{\sqrt{d}}\frac{3^{3d}}{2^{2d}}$ pour un constante c. Ainsi

$$q_d \geq \frac{c}{\sqrt{d}}\frac{3^{3d}}{2^{2d}}(1/3)^{2d}(2/3)^d = \frac{c}{\sqrt{d}}2^{-d}.$$

Autrement dit, la probabilité d'atteindre l'origine en $3d$ étapes est à peine moins grande que la probabilité de l'atteindre en un nombre quelconque d'étape. Cela motive l'idée algorithmique suivante pour atteindre l'origine avec une marche aléatoire : si au bout d'un nombre linéaire d'étapes on n'atteint pas l'origine, autant recommencer à un emplacement aléatoire.

2 L'algorithme WalkSat

L'algorithme WalkSat permet de résoudre, de manière probabiliste, le problème k-sat. L'entrée du problème est une formule booléenne sous forme normale conjonctive avec k littéraux par clause (formule k-CNF). Un exemple pour $k = 3$ est la formule

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4).$$

Cette formule est constituée de quatre clauses qui ont chacune trois littéraux, avec au total quatre variables. Une affectation des variables est la donnée d'une valeur $\top$ (vrai) ou $\bot$ (faux) à chaque variable. Par exemple, l'affectation $x_1 = \top$, $x_2 = \bot$, $x_3 = \bot$ et $x_4 = \top$ ne satisfait que les clauses 1, 3 et 4 et ne satisfait pas la formule. On peut cependant la satisfaire avec l'affectation $x_1 = \bot$, $x_2 = \top$, $x_3 = \top$, $x_4 = \top$ par exemple. Le problème k-sat consiste à décider, étant donnée une formule k-CNF, s'il existe une affectation des variables qui la satisfait. Ce problème est NP-difficile pour $k \geq 3$, et polynomial pour $k = 1$ et $k = 2$. Dans la suite, on suppose que $k \geq 3$.

L'idée générale de l'algorithme WalkSat consiste à partir d'une affectation aléatoire des variables, puis d'effectuer une marche aléatoire depuis cette affectation. Pour passer d'une affectation à la suivante, on change la valeur d'une unique variable, prise aléatoirement de telle sorte qu'une des clauses non satisfaite par l'affectation précédente soit satisfaite par l'affectation suivante. L'algorithme général est constitué de deux boucles imbriquées. On commence par étudier la boucle interne.

2.1 Boucle interne

La boucle interne est l'algorithme Walk(ϕ, p) ci-dessous qui prend en entrée une formule k-CNF ϕ et un paramètre p :

1. $A \leftarrow$ affectation aléatoire des variables ;
2. Répéter p fois :
3. Si $\phi(A) = \top$: renvoyer A.
4. Choisir aléatoirement une clause C de ϕ non satisfaite par A ;
5. Choisir une variable v aléatoirement dans C ;
6. Mettre à jour A, en changeant l'affectation de v.
7. Renvoyer « Probablement insatisfiable »

Remarque. Si l'agorithme renvoie A, alors A est bien une affectation valide. Si l'algorithme répond « probablement insatisfiable » il se peut qu'il existe une affectation. Autrement dit, si ϕ n'est pas satisfiable, l'algorithme répond toujours « probablement insatisfiable », et l'objectif est de montrer que si ϕ est satisfiable, la probabilité que l'algorithme ne trouve pas d'affectation n'est *pas trop grande*.

Lemme 2.1. *La complexité de l'algorithme* Walk(ϕ, p) *est* $p \times \mathsf{poly}(n)$ *où* n *est le nombre de variables de* ϕ *et* poly *un polynôme.*

Démonstration. On vérifie aisément qu'on peut tester si une affectation de variables satisfait une formule donnée en temps polynomial en la taille de la formule : pour chaque clause, on vérifie en temps linéaire en k si la clause est satisfaite. Pour conclure, il suffit de montrer que la taille de ϕ est bornée par un polynôme en n : si les clauses sont deux à deux distinctes, il existe au plus $(2n)^k$ clause de k littéraux faisant intervenir n variables.

□

Remarque. La preuve ci-dessus n'est pas entièrement formelle. On peut très bien imaginer avoir en entrée une formule arbitrairement longue avec des clauses répétées. Rien ne garantit alors que la taille de l'entrée soit polynomiale en le nombre de variables. Une hypothèse implicite dans ce qui précède (et dans la suite) est donc que la formule en entrée est *raisonnable*.

On peut voir l'algorithme WALK comme une marche aléatoire dont les états sont les affectations de variables. Cependant, cette marche aléatoire est très complexe à étudier. En particulier, il n'est pas évident de calculer les probabilités de passer d'une affectation à une autre car cela dépend de la formule en entrée.

Au lieu de cela, on suppose que ϕ est satisfiable (car on connaît le comportement de l'algorithme quand ϕ n'est pas satisfiable) et on fixe une affectation $A^\star$ telle que $\phi(A^\star) = \top$ [1]. L'objectif est de trouver la probabilité que partant d'une affectation aléatoire, l'algorithme finisse par arriver à une affectation valide de la formule. On va chercher quelque chose de plus restrictif : la probabilité que l'algorithme arrive à l'affectation $A^\star$ qu'on a fixé en avance.

On définit pour cela la distance $\delta(A_1, A_2)$ entre deux affectations A_1 et A_2 comme le nombre de variables pour lesquelles A_1 et A_2 sont en désaccord, autrement dit δ est la distance de Hamming. Pour analyser l'algorithme WALK, on va *suivre* l'évolution de $\delta(A^\star, A)$ au cours de l'exécution. On remarque qu'à l'étape 6., $\delta(A^\star, A)$ est soit augmentée de 1, soit diminuée de 1 : si les affectations $A^\star$ et A étaient en accord sur la valeur de la variable v choisie, $\delta(A^\star, A)$ augmente de 1, et sinon elle diminue de 1.

Lemme 2.2. *Soit A une affectation telle que $\phi(A) = \bot$ et A' l'affectation obtenue à partir de A à l'étape 6. de l'algorithme. Alors $\mathbb{P}[\delta(A^\star, A') = \delta(A^\star, A) - 1] \geq 1/k$.*

Démonstration. Puisque $A^\star$ est une affectation satisfaisant ϕ, $A^\star$ est en accord avec au moins une variable de la clause C choisie (c'est-à-dire qu'il existe soit un littéral positif x_i tel que $A^\star$ affecte $\top$ à x_i, soit un littéral négatif $\neg x_i$ tel que $A^\star$ affecte $\bot$ à x_i). D'autre part, puisque A ne satisfait pas la clause C choisie, A est en désaccord avec toutes les variables de la clause. Autrement dit, il existe au moins une variable à propos de laquelle $A^\star$ et A sont en désaccord. Choisir une telle variable fait diminuer la distance entre $A^\star$ et A. En choisir une à propos de laquelle $A^\star$ et A sont en accord fait augmenter la distance. Puisqu'on prend une variable aléatoire de C, la probabilité d'avoir une *bonne* variable est au moins $1/k$.

□

On sait également que la distance entre deux affectations ne peut être plus grande que n, le nombre de variables. Pour simplifier l'analyse, on va *oublier* cette information, et prétendre que $\delta(A^\star, A)$ peut aller jusqu'à $+\infty$. La valeur de $\delta(A^\star, A)$ suit alors la marche aléatoire de la partie 1.

Corollaire 2.3. *Soit ϕ une formule k-CNF satisfiable ayant n variables. Alors la probabilité de succès $P_{succès}$ de l'algorithme* WALK$(\phi, +\infty)$ *est* $\geq \left(\frac{2(k-1)}{k}\right)^{-n}$.

Démonstration. On fixe une affectation $A^\star$ satisfaisant ϕ. Si l'affectation A choisie au départ est à distance $d = \delta(A^\star, A)$ de l'affectation $A^\star$, les résultats de la partie 1 montrent que la probabilité de succès est au moins $1/(k-1)^d$ (en appliquant les résultats avec $p = 1/k$).

Pour trouver la probabilité globale de succès, on considère les 2^n affectations initiales possibles, toutes équiprobables. Il y en a, pour tout d, $\binom{n}{d}$ à distance d de $A^\star$ et pour chacune la probabilité

1. On a utilisé la même technique pour l'analyse de RANDMINCUT.

de succès est au moins $1/(k-1)^d$. Donc

$$P_{\text{succès}} \geq 2^{-n} \sum_{d=0}^{n} \binom{n}{d} \frac{1}{k-1}^{d} = \left(2\frac{k-1}{k}\right)^{-n}$$

en utilisant la formule du binôme $\sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} = (x+y)^n$.

□

Ce résultat ne suffit pas ! En effet, utiliser une marche aléatoire infinie est peu pratique... Si on utilise le résultat démontré à la section 1.3, on obtient le résultat suivant [2] :

Théorème 2.4. *Soit ϕ une formule k-CNF satisfiable ayant n variables. Alors la probabilité de succès $P_{succès}$ de l'algorithme* WALK$(\phi, 3n)$ *est $\geq \frac{c}{\sqrt{n}} \left(\frac{2(k-1)}{k}\right)^{-n}$ pour une certaine constante c.*

2.2 Boucle externe

Pour finir la description et l'analyse de l'algorithme WALKSAT, on utilise la remarque effectuée en fin de section 1.3 : si au bout d'un certain temps, on n'a pas trouvé l'origine, autant recommencer d'un autre point de départ. C'est exactement ce que fait l'algorithme WALKSAT(ϕ, m) décrit ci-dessous, où ϕ est une formule k-CNF à n variables, et m est un paramètre :

1. Répéter m fois :
2. Exécuter WALK$(\phi, 3n)$
3. En cas de succès, renvoyer « Satisfiable »
4. Renvoyer « Non satisfiable »

La complexité de cet algorithme est m fois celui de WALK, c'est-à-dire $mn\text{poly}(n) = m\text{poly}(n)$.

Théorème 2.5. *Si $m \geq \sqrt{n}(2(k-1)/k)^n$,* WALKSAT$(\phi, m)$ *renvoie le bon résultat avec probabilité $\geq \alpha$ pour une certaine constante α.*

Démonstration. Si ϕ n'est pas satisfiable, le résultat est bon à coup sûr. Dans le cas inverse, la probabilité que WALK ne trouve pas d'affectation satisfaisant ϕ en $3n$ étapes est au plus $1 - P_{\text{succès}}$, calculée dans la partie précédente. Donc toutes les invocations de WALK échouent avec probabilité $(1 - P_{\text{succès}})^m$. Autrement dit, la probabilité d'échec est

$$(1 - \frac{c}{\sqrt{n}}(\frac{k}{2(k-1)})^n)^m \leq \exp(-\frac{c}{\sqrt{n}}(\frac{k}{2(k-1)})^n \times m).$$

Mais d'après la borne sur m, on obtient que la probabilité d'échec est au plus $\exp(-c)$. Autrement dit, avec probabilité au moins $\alpha = 1 - exp(-c)$, le résultat obtenu est le bon.

□

En faisant une boucle un peu plus longue (en remplaçant par exemple $\sqrt{n}$ par $n\sqrt{n}$ dans l'énoncé du théorème), on obtient une probabilité de succès exponentiellement proche de 1.

On finit par énoncé le résultat dans le cas particulier de 3-SAT. La borne $1 - 2^{-n}$ est arbitraire : toute valeur exponentiellement proche de 1 est valable.

Corollaire 2.6. *Le problème* 3-SAT *peut être résolu en temps $(\frac{4}{3})^n \text{poly}(n)$ par un algorithme probabiliste dont la probabilité de succès est $\geq 1 - 2^{-n}$.*

2. On l'énonce ici pour k-SAT bien qu'on l'ait démontré pour 3-SAT uniquement.