Compléments autour du voyageur de commerce

Le problème du voyageur de commerce

Considérons un voyageur voulant planifier son voyage. Il connaît les villes qu'il doit visiter, mais ne sait pas encore dans quel ordre effectuer son trajet. Nous allons essayer de l'aider.

FIG. 0.1 – Exemple de solution au problème du voyageur de commerce

L'objectif est de créer un programme qui, quelque soit le nombre de villes, retourne un trajet optimal. Nous commencerons par effectuer quelques remarques sur une résolution naïve du problème. Ensuite, nous décrirons un algorithme aléatoire qui permet de trouver une solution qui est en général assez bonne.

1 Le nombre de chemins possibles

Une solution naïve au problème précédent est la suivante : on énumère tous les chemins possibles, pour chacun de ces chemins on calcule sa longueur, puis on choisit le chemin le plus court ! Cependant,...

Supposons que le voyageur doive visiter n villes dont les noms sont $1, \ldots, n$.
Un chemin est une succession de villes $(1, x_2, \ldots, x_{n-1}, 1)$. Ainsi, pour compter le nombre de chemins à tester, il suffit de compter le nombre de successions de villes puis de diviser par 2 (prendre un chemin à l'endroit ou à l'envers ne modifie pas sa longueur !)

Pour compter le nombre de trajets, on raisonne donc comme suit.

1. La ville de départ est fixée : 1 possibilité.
2. La deuxième ville est à choisir parmi les villes restantes : $(n-1)$ possibilités.
3. La troisième ville est à choisir parmi les villes restantes : $(n-2)$ possibilités.
4. $\vdots$

$n-1$. La $(n-1)$ème ville est celle qui nous reste à visiter : 1 possibilité.

Au total, on a

$$(n-1) \times (n-2) \times (n-3) \times \ldots \times 2 \times 1 = (n-1)!\ trajets,$$

soit

$$\frac{(n-1)!}{2} \text{ chemins possibles.}$$

Remarque. Pour $n = 10$, on trouve déjà 181 140 chemins possibles !!!

2 Les problèmes NP-complets

Le voyageur de commerce fait partie des problèmes qualifiés de NP-complets. Pour ces problèmes, le mieux que puisse faire un algorithme déterministe (c'est-à-dire n'utilisant pas de nombre aléatoire) est d'étudier toutes les possibilités et de tester chacune de ces réponses en un temps qui dépend polynomialement de la taille du problème.
Ces problèmes sont donc très couteux en temps de calcul. C'est pourquoi nous nous sommes intéressés à un algorithme probabilisé qui, même s'il ne donne pas toujours *le* bon résultat, donne toutefois un résultat acceptable *la plupart du temps*.

Pour plus d'informations :
`http://fr.wikipedia.org/wiki/Th%C3%A9orie_de_la_complexit%C3%A9#Classe_NP`

3 L'algorithme de Metropolis

3.1 Restons naïfs

La partie précédente suggère une idée intéressante. Supposons que l'on dispose d'un *super-dé* possédant autant de faces que de chemins qui peuvent être parcourus par notre voyageur, c'est à dire un dé à $n!/2$ faces. Supposons encore que le dé soit pipé, c'est à dire que chacune des faces n'ait pas la même probabilité d'apparition. On suppose ainsi que plus le chemin est court, plus la probabilité de tomber sur la face correspondante est grande.
Pour formaliser un peu plus, pour un chemin c, notons $l(c)$ la longueur de ce chemin. Notons également L la somme de l'inverse des longueurs de tous les chemins possibles. Nous supposerons que la probabilité que notre *super-dé* tombe sur le chemin c vaut $\frac{1}{l(c)L}$ (plus la longueur du chemin est courte, plus la probabilité de l'obtenir est grande).

Le problème est qu'on ne peut pas fabriquer un tel dé. En effet, pour appliquer la méthode précédente, il faut pouvoir calculer la probabilité d'apparition de chacune des faces du *super-dé*. Il faut donc pouvoir calculer L, donc la longueur de tous les chemins possibles, donc énumérer les $n!/2$ chemins... on a vu précédemment que cette méthode a ses limites ! !

On va utiliser un algorithme qui, étant donné un chemin initial, va donner, au bout d'un certain temps, un chemin qui est presque tiré par notre *super-dé*, le tout sans utiliser de *super-dé* !

3.2 La solution de Metropolis

Pour cela, étant donné un chemin c, on tire au hasard parmi les chemins voisins de c un chemin d (par exemple : on tire au hasard 2 villes et on échange leur position dans le voyage).

1. Si le chemin d est plus court que le chemin c (i.e. $l(d) \leq l(c)$), on le garde.
2. Si le chemin d est plus long que le chemin c (i.e. $l(d) > l(c)$), on le *garde* avec une probabilité dépendant de la différence de longueur entre c et d : si le chemin d est très long, la probabilité de le garder sera faible, s'il est un peu plus long que c, la probabilité de le garder sera plus grande. Plus précisément, on le garde avec une probabilité égale à $\frac{l(c)}{l(d)}$.

Dans cette méthode, toute l'astuce consiste à garder, de temps en temps, des chemins qui ne nous intéressent pas trop. Pourquoi ?

Imaginons une autre situation : vous êtes perdu(e) en montagne et vous voudriez bien redescendre boire un chocolat chaud dans la vallée. La première méthode consiste à chacun de vos pas, de choisir la direction dans laquelle vous allez diriger le suivant, regarder si ça monte ou si ça descend. Si ça monte, vous restez où vous êtes, si ça descend, vous faites un pas dans cette direction. Que risque-t-il de se passer ?
Vous risquez fort de vous trouver au bord d'une mare et de ne pas pouvoir sortir de la cuvette qui l'abrite. Si vous voulez en sortir, il vous faut vous autoriser à monter de temps en temps. C'est ce que propose l'algorithme de Metropolis.

3.3 Un peu plus de mathématiques

Faisons un pas vers la modélisation mathématique du problème. L'objet sous-jacent à cet algorithme est la *chaîne de Markov*. On appelle chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ un processus qui, connaissant *tout* le chemin qu'il a parcouru jusqu'à cet instant, regarde *seulement* l'endroit où il se trouve pour décider de sa prochaine étape, i.e.

$$Prob\left(X_{n+1} = \alpha \text{ connaissant tous le chemin parcouru jusqu'à l'instant } n\right) = Prob\left(X_{n+1} = \alpha \text{ connaissant } X_n\right).$$

Ici, notre chaîne de Markov, c'est la suite des chemins propsés par l'agorithme de Metropolis.

Nous constatons que notre chaîne de Markov est *irréductible* : étant donnés 2 chemins c et d, la chaîne de Markov doit toujours être capable de pouvoir passer du chemin c au chemin d en un nombre fini d'étapes. Ici, un résultat sur les permutations permet de montrer qu'en échangeant 2 villes, on peut atteindre tous les chemins possibles à partir de n'importe quel chemin. Nous noterons $P(c,d)$ la probabilité de passer du chemin c au chemin d.

Enfin, la probabilité de transition (i.e. la probabilité que sachant que l'algorithme ait obtenu le chemin c, il passe au chemin d à l'étape suivante) de la chaîne de Markov est notée $Q(c,d)$:

$$Q(c,d) = \begin{cases} P(c,d) & \text{si } l(d) < l(c) \\ P(c,d)\frac{l(c)}{l(d)} & \text{si } l(d) > l(c) \end{cases}$$

On remarque que la probabilité de transition Q est *réversible* par rapport à la probabilité *super-dé* π : si $l(y) > l(x)$,

$$\begin{aligned} \pi(x)Q(x,y) &= \frac{1}{Ll(x)}P(x,y)\frac{l(x)}{l(y)} \\ &= \frac{1}{Ll(y)}P(x,y) \\ &= \frac{1}{Ll(y)}P(y,x)\frac{l(y)}{l(x)} \\ &= \pi(y)Q(y,x). \end{aligned}$$

Un théorème sur les chaînes de Markov (cf. [MPB]) assure que la chaîne de Markov irréductible apériodique converge vers sa mesure invariante, i.e. au bout d'un certain temps, notre chemin aléatoire a été obtenu par un dé qui ressemble fortement à notre fameux *super-dé*. Notons π la loi de notre *super-dé*, i.e. pour tout chemin c,

$$\pi(c) = \frac{1}{l(c)L}.$$

Notons également $Q_n(c,d)$ la probabilité que l'algorithme, partant du chemin c, obtienne le chemin d au bout de n étapes.

Théorème 3.1. *Pour tout chemin c,*

$$\sum_d |Q_n(c,d) - \pi(d)| \underset{n\to\infty}{\longrightarrow} 0.$$

3.4 Un peu plus chaud

Pour favoriser encore plus les chemins courts, on peut rajouter un paramètre de température T. On considère alors la mesure de probabilités pour notre *super-dé*

$$e^{-\frac{1}{T}l(\omega)}Z.$$

Il existe un raffinement de la méthode de Metropolis appelé *algorithme de recuit simulé*. Dans cette variante, on fait évoluer le paramètre de température au cours de l'algorithme pour obtenir des températures de plus en plus faible. Ainsi, plus le temps passe, moins on s'autorise à sélectionner un chemin pire que celui qu'on possède déjà. Il est prouvé qu'un tel algorithme converge avec probabilité 1 vers le chemin de longueur minimale. Cependant, la manière de faire décroître la température dépend du problème étudié et n'est pas aisée à déterminer mathématiquement...

Références

[GS] Geoffrey R. Grimmett et David R. Stirzaker : *Probability and random processes. 3rd ed.*

[MPB] Laurent Mazliak, Pierre Priouret et Paolo Baldi : *Martingales et chaînes de Markov. (Martingales and Markov chains).*