Algorithmes d'approximation et Algorithmes randomisés

Nicolas.Schabanel@ens-lyon.fr

5-6 mai 2003 – CIRM

Table des matières

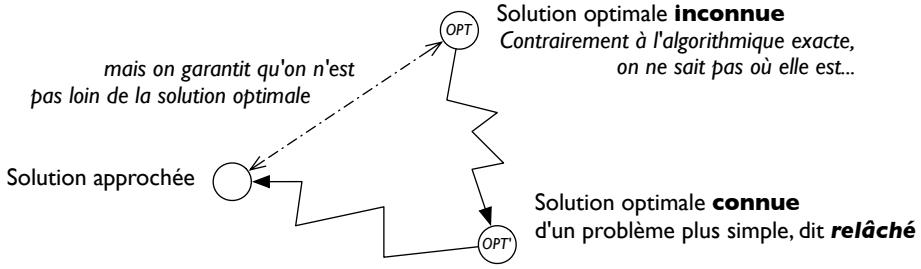
Première partie
Algorithmes d'approximation

1 Préliminaires et définitions

1.1 Algorithmes d'approximation vs algorithmes exacts

L'approche en algorithmique d'approximation diffère assez largement de celle de l'algorithmique exacte. Contrairement à la résolution d'un problème de la classe P, où la solution est connue et où il s'agit de l'obtenir le plus vite possible, l'approximation d'un problème d'optimisation NP-difficile (grosso modo, insoluble en temps polynomial, à moins que $P = NP$, ce qui serait une petite révolution) repose sur l'obtention d'informations sur une solution qu'on ne connait pas, et qu'on ne pourra jamais connaître (sauf par une irréalisable énumération exponentielle).

L'idée est, en général, de résoudre un autre problème que le problème original, puis de transformer la solution optimale du second problème en une solution du premier. Tout l'art de l'approximation est d'assurer qu'à chaque étape, on ne s'éloigne pas trop de l'optimum original. Voici une illustration du principe général de l'approche :



Le second problème est souvent qualifié de *relâché*, car il est souvent obtenu en relâchant des contraintes du problème original. Les sections suivantes illustrent ces principes sur des exemples.

1.2 Vocabulaire et définitions

Commençons par quelques définitions.

Définition 1 Nous cherchons à résoudre des *problèmes d'optimisation*, i.e., nous recherchons dans un ensemble fini $\mathcal{A}$, muni d'une *fonction de poids*[1] sur ses éléments, un élément $a \in \mathcal{A}$ de poids minimum, dans le cas d'un *problème de minimisation*, ou de poids maximum, dans le cas d'un *problème de maximisation*. Voici quelques exemples :

– Dans un graphe non-orienté $G = (V, E)$,[2] muni d'une fonction de poids

[1] ou *de coût*, les deux termes sont utilisés indifféremment.

[2] **Rappel :** On appelle *graphe* $G = (V, E)$ une relation binaire non-réflexive E sur un ensemble fini de *sommets* V. Les relations $(u, v) \in E$ sont appelées les *arêtes* du graphe. Lorsque la relation E est symétrique, on dit que le graphe est *non-orienté*, et chaque arête (u, v) est notée simplement uv. Lorsque la relation E n'est pas symétrique, on dit que le graphe est *orienté*, et les arêtes sont appelées des *arcs*. Un graphe non-orienté est habituellement représenté par des traits reliant des points : les points sont les sommets et les traits, les relations. Lorsque le graphe est orienté, chaque arc (u, v) est représenté par une flèche partant de u et pointant vers v. On note habituellement $n = |V|$ le nombre de sommets, et

sur ses arêtes, rechercher l'arbre couvrant de poids minimum. Il s'agit ici d'un problème de minimisation. $\mathcal{A}$ est l'ensemble des arbres couvrants de G et le poids d'un arbre est la somme des poids de ses arêtes.

– On se donne un graphe non-orienté $G = (V, E)$, muni d'une fonction de poids sur les arêtes. Une *coupe* de G est un ensemble C d'arêtes tel qu'il existe une partition des sommets en deux ensembles S et $V \setminus S$ telle que C soit l'ensemble des arêtes entre les deux composantes de cette partition, i.e., $C = \{uv : u \in S \text{ et } v \in V \setminus S\}$. On recherche une coupe de poids maximum. Il s'agit d'un problème de maximisation. Ici, $\mathcal{A}$ est l'ensemble des coupes de G, et le poids d'une coupe est la somme des poids des arêtes qui la composent.

Le premier de ces deux exemples est « facile » : l'arbre couvrant de poids minimum se trouve en temps $O(m \log n)$ (et même $O(m + n \log n)$) avec un algorithme glouton. Ce problème est dit *polynomial*, car il se résoud en un temps polynomial en la taille de l'entrée $n + m$. On note P la classe des problèmes polynomiaux.

Le second exemple est « difficile » : le problème de la coupe maximum est NP-difficile. La classe NP est l'ensemble des problèmes dont on peut *vérifier* une solution en temps polynomial. Ici, on peut vérifier en temps polynomial qu'une coupe C a bien un poids supérieur à une constante K. Bien entendu, $P \subset NP$. Dans la classe NP, certains problèmes sont considérés comme « maximaux », ce sont les problèmes *NP-difficiles*. Tous les autres problèmes de la classe s'y réduisent : si on sait résoudre en temps polynomial un problème NP-difficile, alors tous les problèmes de NP se résolvent en temps polynomial grâce à cette réduction. Le problème de la coupe maximum est NP-difficile. À moins que $P = NP$, il est impossible de répondre à la question "Existe-t-il une coupe de poids $\geq K$ dans G ?" en temps polynomial pour tout K et G ; il est donc, en particulier, impossible de trouver une coupe maximum. La conjecture $P \neq NP$ est l'un des grands problèmes ouverts en informatique (sa résolution a d'ailleurs été mise à prix pour 1 000 000 $).

Il est donc désespéré de trouver un algorithme efficace pour résoudre un problème d'optimisation NP-difficile. Or, la plupart le sont : en biologie, en réseau, en électronique,... une immense majorité de ces problèmes sont NP-difficiles. Une solution est de rechercher des heuristiques admettant des *garanties de performances*. Un exemple de telle garantie est, pour un problème de minimisation, que la solution construite par l'heuristique ait toujours un poids $\leq K \cdot \mathrm{OPT}$ pour une certaine constante $K > 1$. Nous sommes alors sûrs de ne jamais payer plus de K fois le poids optimal. Pour un problème de maximisation, une garantie équivalente est que l'heuristique produise toujours une solution de poids $\geq K \cdot \mathrm{OPT}$, pour une certaine constante $K < 1$.

Définition 2 On appelle *facteur d'approximation* d'une heuristique polynomiale A pour un problème de minimisation Π, la borne supérieure sur toutes les instances I du problème Π, du quotient du poids $A(I)$ de solution construite par A sur l'instance I, par le poids $\mathrm{OPT}(I)$ de la solution optimale pour I :

$$\sup\{A(I)/\,\mathrm{OPT}(I) : I \in \Pi\}.$$

Pour une heurisitique polynomiale, lorsque le facteur d'approximation est borné, majoré par une constante K ($K > 1$), l'heuristique est appelée *al-*

$m = |E|$ le nombre d'arêtes. Naturellement : $m \leq n(n-1)/2$ pour un graphe non-orienté. Le degré $d(u)$ d'un sommet u d'un graphe non-orienté G est le nombre d'arêtes qui lui sont incidentes : $d(u) = |\{e \in E : u \in e\}|$.

gorithme d'approximation, ou plus précisément *K-approximation*. Parfois, le facteur d'approximation dépend de la taille de l'instance du problème (v. section 3), i.e., il existe une suite K_n telle que $\sup\{A(I)/\operatorname{OPT}(I) : I \in \Pi \text{ et } |I| = n\} \leq K_n$, on parle alors de *$K_n$-approximation*.

Pour un problème de maximisation, le facteur d'approximation se définit symétriquement par :

$$\inf\{A(I)/\operatorname{OPT}(I) : I \in \Pi\}.$$

On parle de même de K-approximation ou K_n-approximation, lorsque le facteur d'approximation est minoré par K ou K_n (ici, $K < 1$ et $K_n < 1$).

Pour une K-approximation, K quantifie la qualité de l'approximation : plus K est proche de 1, meilleur est l'approximation. La constante K est souvent le fruit d'une analyse des performances de l'algorithme. Pour déterminer si cette analyse est bonne, c'est-à-dire précise, il est utile d'obtenir des *instances critiques*.

Définition 3 Une *instance critique* d'une K-approximation A pour un problème d'optimisation Π, est une instance I telle que $A(I)/\operatorname{OPT}(I) \approx K$. Une *famille $\mathcal{F}$ d'instances critiques* est un ensemble infini d'instances tel que :
- $\sup\{A(I)/\operatorname{OPT}(I) : I \in \mathcal{F}\} = K$, si Π est un problème de minimisation,
- $\inf\{A(I)/\operatorname{OPT}(I) : I \in \mathcal{F}\} = K$, si Π est un problème de maximisation.

La recherche d'instances critiques est une activité cruciale pour bien comprendre les faiblesses d'un algorithme ou de son analyse, et entreprendre leurs améliorations.

Exercice 1.1 Définissez "instance critique" et "famille d'instances critiques" pour une K_n-approximation :

$\Diamond$

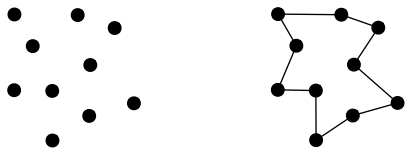
À l'instar des classes P et NP en algorithmique exacte, il existe différentes classes pour les problèmes d'optimisation, quantifiant leurs capacités à être « approximés » par des algorithmes polynomiaux. De même, il existe dans chacune de ces classes des problèmes « maximaux », tels que si on sait les approcher à un certain facteur, alors il en est de même pour tous les problèmes de la classe. Les définitions et études de ces classes ne font pas l'objet de cette leçon.

2 Un exemple : le problème du représentant de commerce (TSP)

Problème 1 (Représentant de commerce (TSP)) Étant donné un graphe complet G muni d'une fonction de poids positive sur les arêtes, trouver un cycle de poids minimum passant par chaque sommet exactement une fois. Un tel cycle est appelé *tour du représentant de commerce*.[3]

[3] *Traveling salesman problem*, en anglais.

Voici un ensemble de points du plan et son tour optimal. Sur cet exemple, le poids de chaque arête est sa longueur dans le plan Euclidien.



2.1 Le TSP non-métrique est non-approximable

Dans toute sa généralité, on ne peut garantir aucune approximation pour TSP, à moins que $P = NP$.

Théorème 1 *Quelque soit $\alpha(n)$, calculable en temps polynomial, il n'existe pas de $\alpha(n)$-approximation pour TSP, à moins que $P = NP$.*

Preuve. L'idée est de réduire polynomialement, le problème NP-difficile de l'existence d'un cycle Hamiltonien,[4] à celui de la recherche d'un cycle court ou long. Prenons $G = (V, E)$ une instance du cycle Hamiltonien. Construisons le graphe complet $G' = (V, E')$ sur le même ensemble de sommets V, et où le poids de l'arête e est 1 si $e \in E$, et $\alpha(n) \cdot n$ sinon.

Supposons par l'absurde qu'il existe une $\alpha(n)$-approximation pour le TSP.

□

2.2 Un minorant pour le TSP métrique

Problème 2 (Représentant de commerce métrique (TSP métrique)) Il s'agit de la restriction du problème du représentant de commerce aux graphes munis d'une *fonction de poids métrique* sur les arêtes, i.e., telle que pour tout $u, v, w \in V$, $\text{poids}(uw) \leq \text{poids}(uv) + \text{poids}(vw)$.

Lemme 2 *Si G est métrique, le poids minimum d'un arbre couvrant de G est un minorant du poids optimal d'un tour du représentant de commerce de G.*

Preuve.

[4] Un *cycle Hamiltonien* d'un graphe G est un cycle qui passe par tous les sommets du graphe, une fois et une seule. Déterminer si un graphe non-orienté admet un cycle Hamiltonien est un problème NP-difficile.

□

2.3 Une 2-approximation pour le TSP métrique

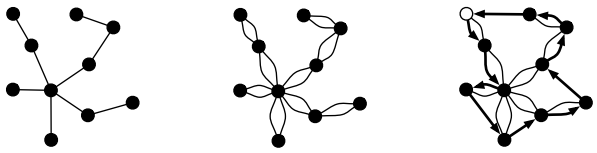
Définition 4 Un *graphe Eulérien* est un graphe, où tous les sommets sont de degré pair. Ces graphes sont exactement ceux qui admettent un cycle qui passe par toutes les arêtes, une fois et une seule. De plus, nous savons construire un tel cycle très facilement, par un simple parcours des arêtes en temps linéaire.

L'algorithme suivant pour le TSP métrique, repose sur la transformation d'un arbre couvrant de poids minimum de G en un graphe Eulérien.

Algorithme 1 (TSP métrique – 2-approximation)

1. Calculer un arbre couvrant de poids minimum T de G.
2. Dupliquer toutes les arêtes de T pour obtenir un graphe Eulérien.
3. Calculer un cycle Eulérien $\mathcal{T}$ de ce graphe.
4. Retourner le cycle $\mathcal{C}$ qui passe par tous les sommets de G dans l'ordre de leur première occurence dans $\mathcal{T}$.

Voici un exemple d'exécution de cet algorithme sur notre ensemble de points dans le plan Euclidien. Sont illustrés l'arbre de poids minimum, le graphe Eulérien, et le tour obtenu en partant du sommet dessiné en blanc.



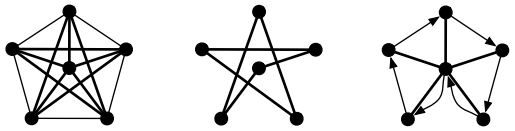
Théorème 3 *L'algorithme 1 est une* 2*-approximation pour le problème du représentant de commerce métrique (TSP métrique).*

Preuve.

□

Exercice 2.1 Prouvez que la famille de graphes complets, illustrée pour $n = 6$ ci-dessous, est une famille d'instances critiques pour cet algorithme. Les arêtes en gras coûtent 1, les autres coûtent 2. Sont représentés, le graphe, un tour

optimal, et un arbre couvrant de poids minimum avec un tour obtenu à partir de cet arbre couvrant :



◊

2.4 Exercice : Une 3/2-approximation pour le TSP métrique

L'idée de cet algorithme est de n'ajouter que le minimum d'arêtes (au sens du poids) nécessaires pour obtenir un graphe Eulérien à partir de l'arbre couvrant de poids minimum.

Lemme 4 *Tout graphe G admet un nombre pair de sommets de degré impair.*

Preuve.

□

Définition 5 Un *couplage* d'un graphe $G = (V, E)$ est un ensemble d'arêtes $M \subset E$ tel que pour toutes arêtes $e, e' \in M$, $e \neq e' \Rightarrow e \cap e' = \varnothing$. Un couplage M est dit *parfait*, s'il couvre tous les sommets de G, i.e., si pour tout $x \in V$, il existe $e \in M$, telle que $x \in e$. Il existe des algorithmes polynomiaux très simples pour calculer un couplage parfait de poids minimum dans un graphe ayant un nombre pair de sommets.

L'amélioration de notre algorithme pour le TSP repose sur un nouveau minorant du poids d'un tour optimal.

Lemme 5 *Soit M un couplage d'un sous-graphe induit par des sommets de G, alors le poids de M est inférieur à la moitié du poids du tour optimal.*

Preuve.

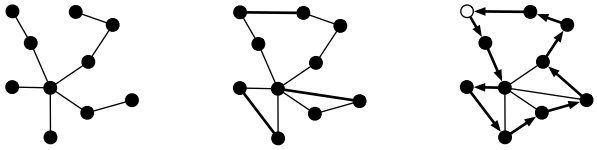
□

Voici l'algorithme :

Algorithme 2 (TSP métrique – 3/2-approximation)

1. Calculer un arbre couvrant de poids minimum T de G.
2. Calculer un couplage parfait de coût minimal M du sous-graphe induit par les sommets de degré impair de T. Ajouter M à T pour obtenir un graphe Eulérien.
3. Calculer un cycle Eulérien $\mathcal{T}$ de ce graphe.
4. Retourner le cycle $\mathcal{C}$ qui passe par les sommets de G dans l'ordre de leur première occurrence dans $\mathcal{T}$.

Les figures ci-dessous illustrent le déroulement de cet algorithme sur notre ensemble de points du plan Eulérien. Sont illustrés, l'arbre couvrant de poids minimum, l'arbre complété par un couplage de poids minimum, et le tour obtenu en partant du sommet dessiné en blanc.

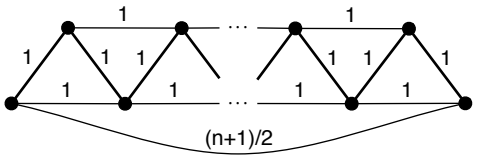


Théorème 6 (Christofides, 1976) *L'algorithme 2 est une* 3/2*-approximation pour le problème du représentant de commerce.*

Preuve.

□

Exercice 2.2 Démontrez que le graphe suivant définit une famille d'instance critique pour cet algorithme. Il a n sommets, avec n impair :



L'arbre couvrant trouvé à la première étape est dessiné en gras.

◊

3 2ème exercice : couverture par ensembles

Problème 3 (Couverture par ensembles) Étant donnés un univers U de n éléments, une collection de sous-ensembles de U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, et une fonction de coût $c : \mathcal{S} \to \mathbb{Q}^+$, on appelle *couverture par ensembles*[5] de U, tout sous-ensemble C de $\mathcal{S}$ qui couvre tous les éléments de U, i.e., tel que $\bigcup_{S \in C} S = U$. Le problème est de trouver une couverture par ensembles de coût minimum.

Voici un algorithme glouton : choisir l'ensemble de coût efficace minimum, éliminer les éléments couverts et itérer jusqu'à ce que tous les éléments soient couverts. Soit C l'ensemble des éléments déjà couverts au début d'une itération. Le *coût efficace* d'un ensemble S, durant cette itération, est le coût moyen auquel il couvre de nouveaux éléments, i.e., $c(S)/|S - C|$. Le *prix* d'un élément est le coût efficace moyen auquel il peut être couvert. De façon équivalente, on peut imaginer que, quand un ensemble S est choisi, son coût est réparti uniformément parmi les nouveaux éléments couverts.

Algorithme 3 (Couverture par ensembles)

1. $C \leftarrow \varnothing$.
2. Tant que $C \neq U$ faire
   Soit S, l'ensemble de coût efficace minimum de l'itération courante.
   Poser $\alpha = \frac{\text{coût}(S)}{|S-C|}$, le coût efficace de S.
   Sélectionner S et, pour chaque $u \in S - C$, poser $\text{prix}(u) = \alpha$.
   $C \leftarrow C \cup S$.
3. Retourner les ensembles sélectionnés.

L'analyse de cet algorithme n'utilise pas de minoration explicite du coût optimal, mais le coût optimal directement.

Numérotons $u_1, \ldots, u_n$, les éléments de U dans l'ordre dans lequel ils sont couverts par l'algorithme (les ex æquo sont classés arbitrairement).

Lemme 7 *Pour tout $k \in \{1, \ldots, n\}$, $\text{prix}(u_k) \leq \text{OPT}/(n - k + 1)$.*

Preuve. Remarquons qu'à la k-ème itération, les ensembles non-sélectionnés de la solution optimale couvrent les éléments restants pour un coût total inférieur à OPT. Or il reste $|\overline{C}|$ éléments à couvrir,

□

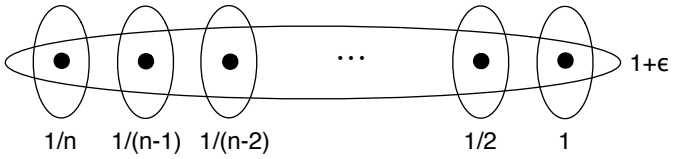
[5] *Set cover*, en anglais.

Théorème 8 *L'algorithme glouton 3 est une H_n-approximation pour le problème de la couverture par ensembles minimum, avec $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$.*

Preuve.

□

Exercice 3.1 Démontrez que l'instance suivante définit une famille d'instances critiques pour l'algorithme 3 (les poids sont indiqués à côté des ensembles) :



◊

De façon surprenante, on peut démontrer que l'algorithme naturel ci-dessus est essentiellement ce qu'on peut espérer de mieux pour le problème de la couverture par ensembles, car obtenir une « meilleure » approximation conduirait à une « catastrophe » similaire à $P = NP$.

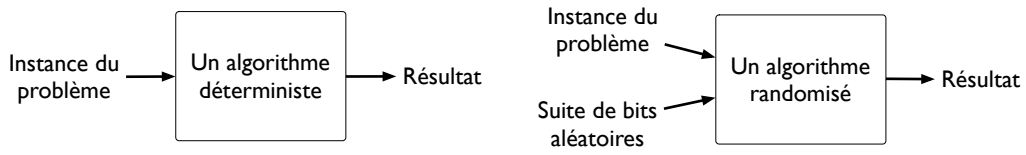
Deuxième partie

Algorithmes randomisés

4 Définitions

Un algorithme déterministe base ses décisions sur le résultat de tests, essentiellement des comparaisons d'entiers ou de rationnels. Un algorithme randomisé prend certaines décisions « au hasard ». On peut formaliser un algorithme randomisé ainsi.

Définition 6 Un *algorithme randomisé* est un algorithme (déterministe !) qui prend deux entrées : la suite de bits de l'instance du problème à résoudre ; et une suite de bits aléatoires, tirés uniformément et indépendamment dans $\{0, 1\}$. La suite de bits aléatoires est différentes pour chaque exécution de l'algorithme. Notez que l'algorithme randomisé s'exécutera de la même façon si on lui soumet la même suite de bits aléatoires pour la même instance.



Les performances d'un algorithme randomisé se mesurent selon différents paramètres :

- **Performance en temps** : L'espérance du temps de calcul sur une instance donnée, où la moyenne est prise sur la suite de bits aléatoires. Quel est le temps moyen d'exécution de l'algorithme en fonction de la taille de l'instance du problème ?
- **Performance en recours à l'aléatoire** : L'espérance du nombres de bits aléatoires utilisés, où la moyenne est prise sur la suite de bits aléatoires. Quel est le nombre de bits aléatoires requis par l'algorithme en fonction de la taille de l'instance du problème ? Les bits aléatoires doivent-ils être totalement indépendants, ou seulement 2-à-2 indépendants, k-à-k indépendants,... ?
- **Qualité de la solution :** Pour un problème d'optimisation, l'espérance du poids de la solution produite pour une instance donnée, où la moyenne est prise sur la suite de bits aléatoires. Quelle est la valeur du quotient de l'espérance du poids de la solution construite sur le poids optimal, en fonction de la taille de l'instance du problème ?

Les algorithmes randomisés sont plus puissants que les algorithmes déterministes, car tout algorithme déterministe est un algorithme randomisé, qui choisit d'ignorer la suite de bits aléatoires. On peut donc en espérer de meilleures performances. Les algorithmes randomisés sont souvent bien plus simples à écrire et à analyser que leurs équivalents déterministes.

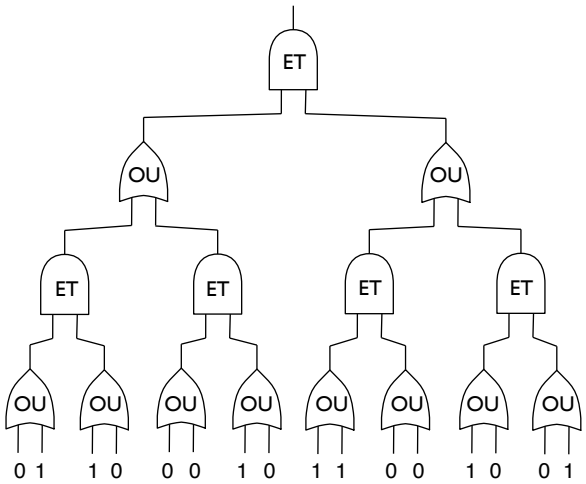
La section 5 présente un problème qu'un algorithme randomisé résout beaucoup vite que n'importe quel algorithme déterministe. La section 6 présente une 1/2-approximation randomisée pour le problème de la coupe maximum. Cette section aborde également un point important : la dérandomisation, i.e. une méthode pour éliminer le recours à l'aléatoire dans un algorithme randomisé, et obtenir une 1/2-approximation déterministe et non randomisée.

5 Évaluation d'un arbre de jeu Booléen

Définition 7 On appelle *arbre de jeu Booléen* T_k, l'arbre binaire complet de profondeur $2k$, dont les nœuds internes à distance paire de la racine sont étiquetés "ET", et les noeuds internes à distance impaire, "OU".

Problème 4 (Arbre de jeu Booléen) Une instance du problème est un arbre de jeu T_k muni de valeurs Booléennes sur ses $n = 2^{2k} = 4^k$ feuilles. Le but est d'évaluer T_k, c'est-à-dire de calculer la valeur de la racine, sachant que la valeur d'un nœud est la conjonction ou la disjonction (suivant l'étiquette du nœud) des valeurs de ses enfants.

Voici une instance du problème pour $k = 2$:



Exercice 5.1 Évaluez l'arbre ci-dessus.

Imaginons qu'obtenir la valeur d'une feuille soit une opération très coûteuse. Nous mesurerons donc les performances de chaque algorithme par le nombre de feuilles visitées sur la pire instance.

Théorème 9 *Pour tout algorithme déterministe évaluant les arbres de jeu Booléens T_k, il existe une instance de T_k telle que l'algorithme inspecte toutes les $n = 4^k$ feuilles.*

Exercice 5.2 Prouvez ce théorème.

$\Diamond$

Nous allons démontrer qu'il existe un algorithme randomisé élémentaire, qui visite beaucoup moins de feuilles *quelque soit* l'instance, i.e., *indépendamment* de l'instance. *Il ne s'agit pas d'analyse en moyenne sur les instances, mais*

d'une analyse en moyenne pour une instance donnée ; la moyenne est prise sur la suite de bits aléatoires donnée en entrée.

L'algorithme évalue l'arbre récursivement, mais choisit au hasard quel sous-arbre fils, il va évaluer en premier : si le premier sous-arbre s'évalue à 1 et que l'étiquette du nœud est "OU", alors il renvoie 1, sans évaluer le second ; si le premier sous-arbre s'évalue à 0 et que l'étiquette est "ET", alors il renvoie 0, sans évaluer le second ; dans tous les autres cas, il évalue le second et en revoie la valeur.

Algorithme 4 (Arbre de jeu Booléen)

1. **Procédure Évalue_nœud(x) :**
   Si x est une feuille, alors Retourner la valeur de x,
   sinon :

   Tirer uniformément $r \in \{0, 1\}$.

   Si $r = 1$,
   alors $v \leftarrow$ Évalue_nœud(fils gauche de x),
   sinon $v \leftarrow$ Évalue_nœud(fils droit de x).

   Si (étiquette(x)="OU" et $v = 1$) ou (étiquette(x)="ET" et $v = 0$),
   alors Retourner v,
   sinon si $r = 1$,
   alors Retourner Évalue_nœud(fils droit de x),
   sinon Retourner Évalue_nœud(fils gauche de x).
2. Évalue_nœud(racine).

Théorème 10 *Pour toute instance de T_k, l'espérance du nombre de feuilles inspectées par l'algorithme randomisé 4 est majorée par $3^k = n^{\log_3 2} = n^{\approx 0.793}$.*

Preuve. Notons Y_k l'espérance du nombre de feuilles visitées pour un arbre T_k. Procédons par récurrence sur k.

□

Notre algorithme visite donc en moyenne $n^{\log_3 2} = n^{\approx 0.793}$ feuilles, au lieu de n pour n'importe quel algorithme déterministe. On peut démontrer qu'aucun algorithme randomisé ne peut faire mieux. La preuve utilise une adaptation du théorème min-max de Von Neumann : Yao remarqua que le théorème de Von

Neumann peut être adapté au cadre des algorithmes randomisés, et dit alors que "l'espérance du temps de calcul sur la pire instance du meilleur algorithme randomisé est égale au temps de calcul du meilleur algorithme déterministe pour la pire des distributions d'instances". Ainsi en choisissant judicieusement une distribution pour les valeurs des feuilles des arbres T_k et en déterminant les performances du meilleur algorithme déterministe pour cette distribution, nous obtenons le théorème suivant :

Théorème 11 *Pour tout algorithme randomisé, il existe une instance T_k pour laquelle l'espérance du nombre de feuilles visitées est $\Omega(3^k)$.*

6 Le problème de la coupe maximum

Soit $G = (V, E)$ un graphe non-orienté, muni d'une fonction de poids sur ses arêtes.

Définition 8 Une *coupe* de G est un ensemble d'arêtes $C \subset E$, tel qu'il existe une partition des sommets en deux ensembles S et $V \setminus S$, telle que C soit l'ensemble des arêtes entre les deux composantes de la partition, i.e., $C = \{uv \in E : u \in S \text{ et } v \in V \setminus S\}$. Le poids d'une coupe est la somme des poids des arêtes qui la composent.

Problème 5 (Coupe maximum (Max-Cut)) Étant donné un graphe non-orienté G avec des poids sur les arêtes, trouver une coupe de G de poids maximum.[6]

Ce problème est NP-difficile. Nous recherchons donc des algorithmes d'approximation.

6.1 Un algorithme randomisé pour la coupe maximum

Commençons par définir la notion de K-approximation randomisée. Il existe essentiellement deux types d'algorithmes d'approximation randomisés.

Définition 9 Un algorithme A est dit *Las Vegas* si son temps d'exécution dépend de la suite de bits aléatoires, mais pas son résultat. Un algorithme A est dit *Monte-Carlo* si son résultat dépend de la suite de bits aléatoires, mais pas son temps de calcul. On dira qu'un algorithme randomisé est *polynomial* si l'espérance de son temps de calcul est polynomial en la taille de l'instance (cette définition s'applique aux deux types d'algorithmes).

Définition 10 Soit A un algorithme randomisé polynomial pour un problème de minimisation Π. On appelle *facteur d'approximation randomisé* de A, la borne supérieure sur toutes les instances I, du quotient de *l'espérance* du poids $\mathbb{E}[A(I)]$ de la solution construite par A pour I sur le poids optimal $\mathrm{OPT}(I)$ d'une solution pour I : $\sup\{\mathbb{E}[A(I)]/\,\mathrm{OPT}(I) : I \in \Pi\}$. Si ce facteur est majoré par une constante K, on dit que A est une *K-approximation randomisée*. Si la majoration K_n du facteur d'approximation dépend de la taille n de l'instance considérée, on dit que A est une *K_n-approximation randomisée*.

Exercice 6.1 Définissez les notions de K-approximation randomisée et K_n-approximation randomisée pour un problème de maximisation Π :

[6] *Maximum cut*, en anglais.

◊

Voici un algorithme randomisé élémentaire pour la coupe maximum.

Algorithme 5 (Coupe max – 1/2-approximation randomisée)

1. $S \leftarrow \varnothing$ et $S' \leftarrow \varnothing$.
2. Pour chaque sommet $u \in V$, placer à pile-ou-face u dans S ou dans S'.
3. Retourner la coupe C définie par la partition (S, S').

Théorème 12 *L'algorithme 5 est une* $1/2$*-approximation randomisée pour le problème de la coupe maximum.*

Preuve.

□

Exercice 6.2 Quel majorant de OPT avons-nous utilisé dans cette analyse ?

◊

Exercice 6.3 On appelle *graphe biparti* tout graphe $G = (X \cup Y, E)$ non-orienté, tel que $X \cap Y = \varnothing$ et tel que pour tout $e \in E$, $e = xy$ avec $x \in X$ et $y \in Y$. Montrez que l'ensemble des graphes bipartis est une famille d'instances critiques pour cet algorithme.

◊

6.2 Un paradigme de dérandomisation

Il existe des liens entre certains algorithmes randomisés et certains algorithmes déterministes. Ces liens peuvent s'établir en montrant que l'un est une version randomisée ou dérandomisée de l'autre.

Commençons par quelques rappels sur les probabilités.

Définition 11 La probabilité d'un événement B conditionnée par l'événement A est définie par :

$$\Pr\{B|A\} = \frac{\Pr\{A \cap B\}}{\Pr\{A\}}$$

L'espérance d'une variable aléatoire discrète X conditionnée par l'événement A est définie par :

$$\mathbb{E}[X|A] = \sum_x x \Pr\{X = x|A\}$$

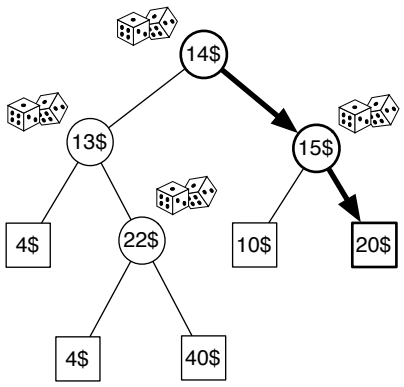
Propriété 13 *Soient X et Y deux variables aléatoires discrètes :*

$$\mathbb{E}[X] = \sum_y \Pr\{Y = y\}\mathbb{E}[X|Y = y] \leq \max_y E[X|Y = y]$$

Plaçons-nous dans la perspective d'un problème de maximisation. Une technique classique de dérandomisation est la *méthode de l'espérance conditionnelle*. Il s'agit de remplacer chaque décision aléatoire par « le meilleur choix aléatoire ». Représentons l'exécution d'un algorithme randomisé par un arbre de décision dont les nœuds sont les tirages aléatoires et les fils sont les différentes exécutions possibles suivant le résultat du tirage aléatoire. Étiquetons chaque nœud par l'espérance de la valeur du résultat, conditionnée par les résultats des tirages aléatoires précédents. Pour chaque nœud x de l'arbre, l'espérance conditionnelle en x est la moyenne des espérances conditionnelles de ses fils. Ainsi, le fils ayant la plus grande espérance conditionnelle a une espérance supérieure à celle de son père. C'est donc, dans la perspective d'un problème de maximisation, le « meilleur choix aléatoire » !

Ainsi, lorsqu'il est possible de calculer, *en temps polynomial*, l'espérance de la valeur du résultat, conditionnée par les décisions précedentes, on peut transformer un algorithme randomisé en un *algorithme glouton* qui remplace chaque tirage aléatoire par le choix maximisant l'espérance conditionnelle de la valeur du résultat. Par récurrence immédiate, la valeur du résultat de l'algorithme glouton sera supérieure à l'espérance de la valeur du résultat de l'algorithme randomisé.

La figure suivante illustre la méthode : en chaque sommet, l'algorithme randomisé va à droite ou à gauche à pile-ou-face ; l'espérance conditionnelle des nœuds est inscrite au centre de chaque nœud ; la valeur des feuilles est celle du résultat de l'exécution de cette branche. La branche dessinée en gras est celle suivie par l'algorithme glouton maximisant l'espérance conditionnelle, elle conduit à une solution valant 20$, qui n'est pas la solution optimale (40$) mais qui est supérieure à l'espérance de l'algorithme randomisé (14$).



Plusieurs remarques. Cette méthode ne permet pas de trouver la solution optimale : sur l'exemple ci-dessus, l'algorithme glouton trouve une solution valant 20$, alors que l'optimum est 40$. Ensuite, nous obtenons un algorithme

glouton polynomial, seulement si l'espérance conditionnelle se calcule en temps polynomial.

Exercice 6.4 Reprendre la méthode de l'espérance conditionnelle dans le cadre d'un problème de minimisation.

◊

Plutôt que de théoriser plus longuement, mettons cette méthode en pratique sur notre algorithme randomisé pour la coupe maximum.

6.3 Un algorithme glouton pour la coupe maximum

Pour $u \in V$ et $A \subset V$, notons $E(u, A)$ l'ensemble des arêtes entre u et A : $E(u, A) = \{e \in E : e = uv \text{ et } v \in A\}$.

Commençons par évaluer l'espérance du poids de la coupe finale conditionnées par les décisions précédentes.

Lemme 14 *Étant donnés $S, S' \subset V$ dans l'algorithme randomisé 5, pour tout $u \in V \setminus (S \cup S')$,*

$$\begin{aligned}&\mathbb{E}[\text{poids de la coupe}|u \in S, S, S'] \\ &\quad = \mathbb{E}[\text{poids de la coupe}|u \in S', S, S'] + \text{poids}(E(u, S')) - \text{poids}(E(u, S)).\end{aligned}$$

Preuve.

□

Nous en concluons que l'algorithme glouton naturel suivant est la version dérandomisée de l'algorithme randomisé 5 ! Ces deux algorithmes sont donc "les mêmes", ce qui n'était pas évident au premier regard.

Algorithme 6 (Coupe maximum – 1/2-approximation gloutonne)

1. $S \leftarrow \varnothing$ et $S' \leftarrow \varnothing$.
2. Pour chaque sommet $u \in V$, placer u dans la composante, S ou S', qui maximise le poids de la coupe courante entre S et S'.
3. Retourner la coupe C définie par la partition (S, S').

Par la méthode de l'espérance conditionnelle, la propriété suivante est un un corolaire direct du théorème 12.

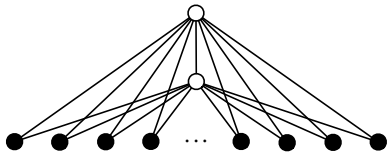
Corollaire 15 *L'algorithme glouton 6 est une $1/2$-approximation.*

Exercice 6.5 Proposez une preuve directe du facteur d'approximation de l'algorithme glouton 6.

◊

On pourrait penser que l'algorithme glouton dérandomisé est bien meilleur que l'algorithme randomisé. L'exercice suivant démontre qu'il n'en est rien.

Exercice 6.6 Montrez que le graphe suivant définit une famille d'instances critiques pour l'algorithme glouton 6, pour peu qu'il commence par les deux sommets blancs.



◊

Exercice 6.7 Pouvez-vous encore poursuivre la déterminisation de l'algorithme 6 ? Quel serait l'algorithme obtenu ? Analysez-le et exhibez une famille d'instances critiques.

◊

6.4 Exercice : Algorithme randomisé et dérandomisé pour Max-SAT

Définition 12 Étant données n variables Booléennes $x_1, \ldots, x_n$, on appelle *littéral*, une variable x_i ou sa négation $\overline{x}_i$. Une *clause* est une disjonction de littéraux : par exemple, $x_1 \vee x_2 \vee \overline{x}_3$. La taille d'une clause est son nombre de littéraux. On élimine les clauses triviales (e.g., $x_1 \vee x_2 \vee \overline{x}_1$) et les littéraux redondants (e.g., $x_1 \vee x_3 \vee x_1$).

Problème 6 (Satisfaction maximum (Max-SAT)) Étant donné un ensemble de m clauses pondérées sur n variables Booléennes $x_1, \ldots, x_n$, trouver une instanciation des x_i qui maximise le poids total des clauses satisfaites.[7]

On propose l'algorithme randomisé naïf suivant :

Algorithme 7 (Max-SAT – 1/2-approximation randomisée)

1. Pour chaque variable x_i, tirez la valeur de x_i uniformément dans $\{0, 1\}$.
2. Retourner l'instanciation calculée.

Lemme 16 *À la fin de l'algorithme 7, pour toute clause c, si* $\text{taille}(c) = k$, *alors* $\Pr\{c \text{ est satisfaite}\} = 1 - 2^{-k}$.

Exercice 6.8 Prouvez ce lemme.

◊

Théorème 17 *L'algorithme 7 est une* $1/2$*-approximation randomisée pour le problème de la satisfaction maximum.*

Exercice 6.9 Prouvez ce théorème.

◊

Propriété 18 *Si toutes les clauses sont de tailles supérieures à k, alors l'algorithme 7 est une* $(1 - 2^{-k})$*-approximation randomisée.*

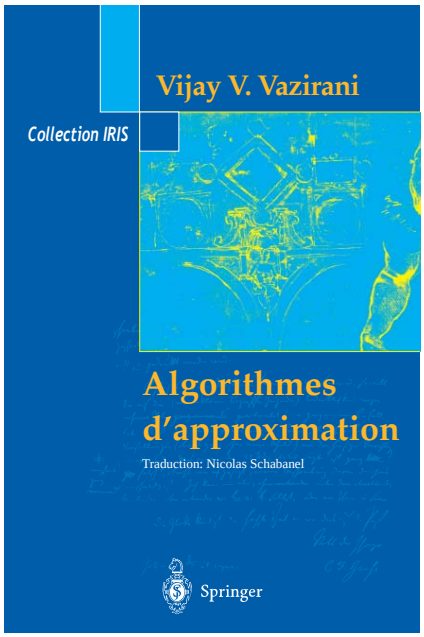
Exercice 6.10 Prouvez cette propriété.

◊

Exercice 6.11 Proposez un algorithme pour calculer, en temps polynomial, l'espérance du poids de l'instanciation conditionnée par les variables déjà instanciées. En déduire un algorithme glouton obtenu par dérandomisation de l'algorithme 7. Exhibez des familles d'instances critiques pour chacun de ces algorithmes.

[7] *Maximum satisfaction*, en anglais.

◊



Références

[1] T. Cormen, C. Leiserson, et C. Stein. *Introduction to algorithms*, 2ème édition. MIT press, 2001, ISBN 0-262-03293-7. Traduit en français : Dunod, 2002, ISBN 2-100-03922-9.

[2] R. Motwani et P. Raghavan. *Randomized algorithms*. Cambrige university press, 1995, ISBN 0-521-47465-5.

[3] V. V. Vazirani. *Approximation algorithms*, 2ème édition. Springer, 2003, ISBN 3-540-65367-8.

[4] V. V. Vazirani. *Algorithmes d'approximation*. Traduction de [3] par N. Schabanel. Springer, à paraître en décembre 2003.