Algorithmique et complexité Avancé

- on s'interesse aux ~~probl~~ problèmes difficiles
- plusieurs manières de mesurer l'approximation
- compromis espace / temps / précision
- deux approches complementaires
  - → positif : algorithmes d'approximation.
  - → négatif : résultat d'inapproximabilité

Révisions :

P : problèmes qu'on peut résoudre avec un algorithme polynomial en la taille de la donnée
→ décision

NP : • ∃ algo nondéterministe polynomial pour décider la solution (→ décision)
- ∃ algo vérification ~~poly~~ polynomial en la taille de la donnée qui vérifie si un certificat est valide pour le problème ~~vérifiés~~

Rq : suivant ce qu'on accorde comme capacité de calcul aux machines de Turing utilisées on défini de nouvelles classes.

Problème d'optimisation :

→ problèmes qu'on à déjà vu avec la programmation linéaire
→ problèmes cours de graphe : ex + court chemin, flot max, coupe min, arbre couvrant de poids minimum.
→ problème d'ordonancement, sac-à-dos, ~~d~~ → voir glouton et heuristiques en ACT.

Défini par :
- I_P = ensemble des instances du problème P
- S_P = fonction qui associe à une instance $x \in I_P$ l'ensemble des solutions réalisables pour cette donnée
- m_P = fonction de mesure : $m_P(x,y)$ = valeur de la solution y de l'instance x → fonction objectif
- g_P = min ou max

Differentes manière de mesurer la qualité de la solution

1. Erreur Absolue : Difference entre la valeur obtenue par l'algorithme et la valeur optimale

$$D(x,y) = | m^*(x) - m(x,y) |$$

où $m^*(x)$ = valeur optimale pour l'instance x
y = une solution réalisable

$D(x,y)$ = erreur absolue.

→ Absolute approximation Algorithm = algorithme pour lequel cette difference est bornée par une constante k (indépendante de la donnée)

$$D(x, A(x)) \leq k \quad \forall x \in I$$

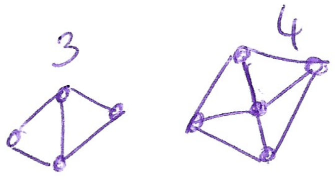
A : algo d'approx

exemple de problème qui a un tel algorithme :

Min Edge Coloring

Donnée : G : Graphe

Sortie : colorer le graphe avec un nombre minimum de couleur tq deux arêtes incidentes à un même sommet n'ont pas la même couleur.



borne inférieure : il faut au moins le degré maximum du graphe, noté Δ

$\exists$ un algorithme qui permet de trouver une coloration avec au plus $\Delta+1$ couleurs. (Misra et Gries)

Exemple du sac-à-dos :

Maximum knapsack :

Données : - n objets, $w_1, \ldots w_n$ = poids des objets
- c = capacité du sac
- $p_1 \cdots p_n$ = profits associés aux objets.

Sortie : valeur maximum que peut contenir le sac

$\Rightarrow \sum_{i=S} p_i$ maximum pour S = sous ensemble des objets, $\sum w_i \leq c$

Th : A moins que $P = NP$, il n'y a pas d'algorithme polynomial d'approximation avec une erreur absolue pour Max Knapsack.

Indice : utiliser l'algorithme qui aurait une erreur absolue k pour écrire un algorithme exact pour Max knapsack.

- on suppose qu'on a un algorithme avec une erreur absolue k.
  - → on crée une nouvelle instance de Max knapsack
    - ⇒ on multiplie tous les profits par $k+1$
    - ⇒ on garde les n objets et les poids w_i, et c la capacité du sac.
  - → l'ensemble des solutions réalisables pour les deux instances est le même.
  - → la valeur de la sol. opt pour la nouvelle instance $= (k+1)\,opt$ avec opt = valeur opt de l'instance de départ.
  - → ~~la~~ solution optimale est la seule à pouvoir être à distance au + k de ~~la~~ valeur optimale
  - ⇒ on peut résoudre de manière exacte le problème Max knapsack ~~xx~~ en temps polynomial ce qui est impossible sauf si $P = NP$

Même résultat : voyageur de commerce (TSP)
max independent set ...

2. <u>Erreur Relative</u> : erreur normalisée par rapport au problème.

$$E(x,y) = \frac{|m^*(x) - m(x,y)|}{\max\{m^*(x), m(x,y)\}}$$

un algorithme A pour un problème d'optimisation est une ε-approximation si $\forall x$ instance du problème l'erreur relative de la solution approchée $A(x)$ est bornée par ε : $E(x, A(x)) \leq \varepsilon$

3- Ratio de Performance : (≥ 1)

$$R(x,y) = \begin{cases} \dfrac{m(x,y)}{m^*(x)} & \text{(minimisation)} \\[2ex] \dfrac{m^*(x)}{m(x,y)} & \text{(maximisation)} \end{cases}$$

Un algorithme ~~A~~ d'approximation A pour un problème P est une r- approximation si $\forall x$ instance de P
$$R(x, A(x)) \leq r.$$
↳ facteur d'approximation

* pour un problème de minimisation :

$$\frac{m(x, A(x))}{m^*(x)} \leq r$$

$$m^*(x) \leq m(x, A(x)) \leq r \cdot m^*(x)$$

* maximisation

$$\frac{m^*(x)}{m(x, A(x))} \leq r$$

$$\frac{1}{r} m^*(x) \leq m(x, A(x)) \leq m^*(x)$$

Classe de Problème d'optimisation :

NPO : problèmes d'optimisation ~~définis par~~ tq
~~Ip : ensemble d'instan~~

- décision en tps polynomiale en $|x|$ si x est bien une instance du problème
- pour une instance $x \in I$, et y de taille polynomiale en $|x|$ (ie : $\exists$ polynôme q tq $|y| \leq q(|x|)$), on peut vérifier en temps polynomiale en $|x|$ que c'est bien une solution réalisable pour x $(y \in S(x))$
- pour toute solution réalisable $y \in S(x)$ $|y|$ est polynomiale en $|x|$
- la fonction ~~f~~ $m(x,y)$ est calculable en temps polynomiale en $|x|$

* Problème de décision associé à un problème d'optimisation (5)

Donnée : ~~Instance~~ Données du problème d'optimisation
• valeur k

sortie : oui si $\exists y \in S$ tq $m(x,y) \leq k$ (minimisation)
$m(x,y) \geq k$ (max)

th : $\forall \Pi \in NPO$, le problème de décision associé à Π est NP.

→ NPO se divise en 5 sous classes en fonction des propriétés des problèmes.

Exemple :
Min Set Cover
Données : U = ensemble d'éléments
$C \in 2^U$ = collection de sous ensemble de U

Sortie : $C' \subseteq C$ tq : - C' est une couverture de U
$\forall v \in U, \exists c \in C'$ tq $u \in c$
Tous les éléments de U apparaissent dans au moins un sous ensemble de C'.
- $|C'|$ est minimum.

Exemple : $U = \{a,b,c,d,e\}$
$C = \{\{a,b\}, \{b,c,e\}, \{a,d\}, \{b,e\}\}$
couverture minimum pour $U = \{\{b,c,e\}, \{a,d\}\} = C'$
$|C'| = 2$

(Rq : Pb de décision associé, $\exists$ couverture de taille $\leq k$)

Exercice : Algorithme d'approximation pour MSet Cover
* Idée de l'algorithme glouton : choisir à chaque itération l'ensemble de C qui permet de couvrir le plus d'éléments non encore couvert.
* Idée pour prouver un facteur d'approximation :
→ répartir le coût d'ajouter un nouvel ensemble à C' sur tous les éléments qu'il permet de couvrir.
→ $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \leq \ln(n) + 1$