TD6 : Algorithmes probabilistes - Dérandomisation

On considère le problème suivant.

Max Sat

INSTANCE – Une formule propositionnelle $\varphi = \wedge_{i=1}^{m} C_i$ sous forme normale conjonctive sur les variables $V = \{x_1, \ldots, x_n\}$ et une fonction de poids $\omega : \{1, \ldots, m\} \to \mathbb{N}$ associant à chaque clause un entier positif.

QUESTION – Trouver une valuation qui maximise la somme des poids des clauses satisfaites.

1. Stratégie probabiliste naïve et dérandomisation

1.1. Écrire un algorithme probabiliste naïf permettant de fournir une solution approchée au problème **Max Sat**.

1.2. On note W_i la variable aléatoire donnant le poids de la clause i lors de l'exécution de cet algorithme : $W_i = \omega(i)$ si la clause est satisfaite, et 0 sinon. Calculer $\mathbb{E}[W_i]$.

1.3. Soit maintenant W la variable aléatoire correspondant à la somme des W_i. On note OPT le poids associé à la solution optimale. Montrer que

$$\mathbb{E}[W] \geq \frac{OPT}{2}$$

Que peut-on dire de ce facteur d'approximation lorsque les clauses contiennent beaucoup de littéraux ?

Un algorithme probabiliste peut être représenté par un arbre de décision. Chaque branche de cet arbre décrit les choix probabilistes pouvant être effectués par l'algorithme. Les feuilles de cet arbre décrivent les sorties de l'algorithme. Partant de l'algorithme probabiliste précédent, nous souhaitons obtenir un algorithme déterministe dont la sortie est au moins aussi bonne que son espérance. Une stratégie est alors de parcourir l'arbre de décision en choisissant, à chaque étape, la branche maximisant le poids de la solution.

1.4. On note $\mathbb{E}[W \mid x_1, \ldots, x_k]$ l'espérance de la variable aléatoire W connaissant les valeurs de vérité des variables $x_1, \ldots, x_k$. Écrire un algorithme pour calculer cette espérance. Quelle est sa complexité ?

1.5. Montrer que l'on peut dérandomiser l'algorithme probabiliste précédent en suivant la stratégie décrite ci-dessus.

2. Dérandomisation et classe de complexité

On note **BPP** (*Bounded-error Probabilistic Polynomial-time*) la classe suivante.

Définition 1 **BPP** est la classe des langages L pour lesquels on dispose d'un algorithme probabiliste A s'exécutant au pire cas en temps polynomial tel que
- Si $x \in L$, alors $\mathrm{P}(A(x) \text{ accepte}) \geq \frac{2}{3}$
- Si $x \notin L$, alors $\mathrm{P}(A(x) \text{ accepte}) \leq \frac{1}{3}$

Si $L \in \mathbf{BPP}$, il existe un algorithme probabiliste A s'exécutant en temps $t(n)$ où t est un polynôme. On sait alors que A utilise au plus $t(n)$ bits aléatoires. Ainsi, on peut voir A comme un algorithme déterministe prenant deux arguments : son argument standard $x \in \{0,1\}^n$ et un vecteur $r \in \{0,1\}^{t(n)}$ décrivant les résultats des tirages aléatoires.

2.1. Montrer que si L dans **BPP** avec un algorithme probabiliste s'exécutant en temps polynomial $t(n)$ et utilisant $r(n)$ bits aléatoires, alors $L \in DTIME(t(n).2^{r(n)})$.

2.2. En conclure que $BPP \subseteq EXP$ et donner une condition suffisante pour qu'un langage $L \in \mathbf{BPP}$ soit dans **P**.

2.3. Soit $A(x,r)$ un algorithme probabiliste pour un langage L admettant une probabilité d'erreur strictement inférieure à 2^{-n} pour toute entrée x de taille n. Montrer qu'il existe une séquence aléatoire r_n telle que $A(x,r_n)$ est correct pour tout $x \in \{0,1\}^n$.

On considère la classe **P/poly** suivante.

Définition 2 $L \in \mathbf{P/poly}$ s'il existe $L' \in \mathbf{P}$, un polynôme t et $\alpha_1, \alpha_2, \cdots \in \{0,1\}^*$ vérifiant $|\alpha_n| \leq t(n)$, tels que :

$$x \in L \iff (x, \alpha_{|x|}) \in L'$$

2.4. Quel est l'intérêt de la question précédente ?