DS5 (2 heures)

Lisez tout le texte avant de commencer. La plus grande importance sera attachée à la clarté, à la précision et à la concision de la rédaction. Toute réponse non justifiée ne sera pas prise en compte. Si vous repérez ce qui vous semble être une erreur d'énoncé, signalez la sur votre copie et poursuivez votre composition en expliquant les raisons de vos éventuelles initiatives. L'usage de tout dispositif électronique est interdit.

Exercice 1

Une *formule propositionnelle* φ, sous *forme normale conjonctive* (CNF), est représentée par la liste de ses clauses, chaque clause étant une liste de littéraux. Les *variables propositionnelles* sont indexées à partir de 1. On représente le littéral x_i par l'entier i et le littéral $\neg x_i$ par l'entier $-i$. On adopte les types suivants.

```
type litteral = int
type clause = litteral list
type cnf = clause list
```

NP-complétude de Max-2Sat

Le problème MaxSat est défini de la manière suivante.

- **Instance** : une formule propositionnelle φ en CNF.
- **Solution** : une valuation v définie sur les variables de φ.
- **Mesure à maximiser** : le nombre de clauses de φ satisfaites par v.

Ce problème est NP-difficile. Sa restriction Max-kSat aux formules en CNF où chaque clause contient au plus k littéraux reste NP-difficile dès que $k \geqslant 2$. Max-kSat est donc *plus difficile à résoudre* que kSat, lui-même NP-difficile dès que $k \geqslant 3$ mais qu'on peut se résoudre en temps linéaire pour $k = 2$.

Considérons le *problème d'optimisation* Max-2Sat défini ci-dessous.

- **Instance** : une formule propositionnelle φ en CNF où chaque clause contient au plus 2 littéraux, un entier k.
- **Solution admissible** : une valuation v sur les variables de φ.
- **Mesure à maximiser** : nombre de clauses de φ satisfaites par v.

Sa version *problème de décision* associé, avec un *seuil*, est la suivante.

- **Instance** : une formule propositionnelle φ en CNF où chaque clause contient au plus 2 littéraux, un entier k.
- **Question** : existe-t-il une valuation qui satisfait au moins k clauses de φ ?

On va montrer que Max-2Sat est NP-difficile en utilisant une réduction polynomiale depuis 3Sat.

Question 1. Soient trois littéraux l_1, l_2 et l_3 et une variable x indépendante des trois littéraux. On note $\overline{l_1}$ la négation de l_1. Ainsi : $\overline{x_1} = \neg x_1$ et $\overline{\neg x_1} = x_1$. On considère alors le groupe de 10 clauses suivant.

$$l_1 \wedge l_2 \wedge l_3 \wedge x \wedge (\overline{l_1} \vee \overline{l_2}) \wedge (\overline{l_2} \vee \overline{l_3}) \wedge (\overline{l_1} \vee \overline{l_3}) \wedge (l_1 \vee \neg x) \wedge (l_2 \vee \neg x) \wedge (l_3 \vee \neg x)$$

□ **1.1.** Montrer que si $l_1 \vee l_2 \vee l_3$ est valide alors on peut donner à x une valeur telle que 7 clauses sont satisfaites, mais pas plus. Autrement dit, si on dispose d'une valuation v sur les variables propositionnelles de l_1, l_2 et l_3 qui satisfait $l_1 \vee l_2 \vee l_3$ alors on peut étendre v sur x de sorte à satisfaire 7 clauses, mais pas plus.
□ **1.2.** Montrer que si $l_1 \vee l_2 \vee l_3$ n'est pas valide alors on ne peut pas satisfaire plus de 6 clauses.

Question 2. Étant donnée une formule 3Sat φ avec m clauses, définir une formule Max-2Sat φ de taille polynomiale et un seuil k tels que φ est satisfiable si et seulement s'il existe une valuation pour φ satisfaisant au moins k clauses.

Algorithme probabiliste

Soit φ une formule propositionnelle en CNF avec m clauses et n variables. Soit v une valuation telle que les valeurs de vérité des n variables propositionnelles de φ sont données par n tirages aléatoires indépendants, chaque variable étant associée à V avec une probabilité $\frac{1}{2}$. On note X_i la variable aléatoire qui vaut 1 si $v(x_i) = \mathsf{V}$ et 0 sinon.

On se demande alors à quel point la valuation obtenue est une bonne solution du problème MaxSat, c'est-à-dire combien de clauses elle satisfait. On note $\text{sat}(v, \varphi)$ le nombre de clauses satisfaites par v dans φ. $\text{sat}(v, \varphi)$ est donc une variable aléatoire.

Question 3. Montrer que si chaque clause de φ contient au moins k littéraux indépendants, c'est-à-dire qui portent tous sur des variables propositionnelles différentes, alors l'espérance du nombre de clauses satisfaites par la valuation aléatoire v vérifie $\mathbb{E}(\text{sat}(v, \varphi)) \geqslant m\left(1 - \frac{1}{2^k}\right)$.

Question 4. En supposant que φ ne contient aucune clause vide, quitte à supprimer les clauses vides de φ, donner une minoration de l'espérance du nombre de clauses satisfaites.

Question 5. On considère l'algorithme suivant.

Algorithme 1 : MaxSat_Proba

```
fonction MaxSat_Proba(φ)
    pour chaque x_i variable aléatoire de φ faire
        tirer uniformément au hasard une valeur de vérité X_i pour x_i
    v ← valuation qui à chaque x_i associe X_i
    renvoyer v
```

□ **5.1.** Cet algorithme est-il une $\frac{1}{2}$-approximation pour le problème MAX2SAT ? Justifier votre réponse.
□ **5.2.** Écrire une fonction `maxSat_proba : cnf -> bool array` qui implémente l'algorithme et renvoie une valuation aléatoire.

Algorithme d'approximation pour MaxSat

Cette partie étudie un moyen de *dérandomiser* l'algorithme précédent, c'est-à-dire de supprimer le caractère aléatoire afin de garantir que le nombre de clauses satisfaites soit supérieur ou égal à l'espérance du tirage aléatoire. Pour chaque variable x_i, on attribue la valeur V ou F selon ce qui maximise l'espérance du nombre de clauses satisfaites en sachant les valeurs de vérité déjà données aux variables précédentes $x_j, j < i$.

Soient X une variable aléatoire réelle sur un univers Ω fini et A un événement de probabilité non nulle. On définit l'*espérance conditionnelle de X sachant A* par :

$$\mathbb{E}(X \mid A) = \sum_{x \in X(\Omega)} x \times \mathbb{P}(X = x \mid A)$$

Si on dispose d'une valuation partielle v_i sur les variables x_1 à x_i, l'espérance conditionnelle $\mathbb{E}\left(\text{sat}(v, \varphi) \mid v_i\right)$ du nombre de clauses satisfaites sachant v_i est donnée par :

$$\mathbb{E}\left(\text{sat}(v, \varphi) \mid v_i\right) = \sum_{C \text{ clause de } \varphi} \mathbb{P}\left(C \text{ est satisfaite } \mid v_i\right)$$

La probabilité conditionnelle $\mathbb{P}\left(C \text{ est satisfaite } \mid v_i\right)$ qu'une clause C soit satisfaite dépend des littéraux de C et des valeurs de vérité déjà affectées à certains de ces littéraux.

- Si l'un des littéraux est défini par v_i et vaut V, alors $\mathbb{P}\left(C \text{ est satisfaite } \mid v_i\right) = 1$.
- Si tous les littéraux sont définis par v_i et valent F, alors $\mathbb{P}\left(C \text{ est satisfaite } \mid v_i\right) = 0$
- S'il y a k littéraux non définis (et aucun défini à V), alors $\mathbb{P}\left(C \text{ est satisfaite } \mid v_i\right) = 1 - \frac{1}{2^k}$

On considère alors l'algorithme suivant.

Algorithme 2 : MaxSat_Approx

```
fonction MaxSat_Approx(φ)
    pour chaque x_i variable aléatoire de φ de x_1 à x_n faire
        E_F ← espérance du nombre de clauses satisfaites par un tirage aléatoire des variables x_{i+1} à x_n,
              connaissant les valeurs déjà fixées pour x_1 à x_{i-1} et en donnant à x_i la valeur F.
        E_V ← espérance du nombre de clauses satisfaites par un tirage aléatoire des variables x_{i+1} à x_n,
              connaissant les valeurs déjà fixées pour x_1 à x_{i-1} et en donnant à x_i la valeur V.
        X_i ← { V  si E_F ⩽ E_V
              { F  sinon
    v ← valuation qui à chaque x_i associe X_i
    renvoyer v
```

Question 6. Écrire une fonction `maxSat_approx : cnf -> bool array` qui prend en argument une formule propositionnelle φ et qui renvoie une valuation obtenue par l'algorithme précédent.
Attention aux calculs sur les flottants. Multiplier par $2^{k_{max}}$, avec k_{max} le nombre maximal de littéraux dans une clause de φ, pour ne manipuler que des entiers. Comparer alors de manière équivalente $k_{max} \times E_F$ et $k_{max} \times E_V$ plutôt que E_V et E_F.

Question 7. Montrer que la propriété suivante est un invariant de la boucle `Pour` de l'algorithme MaxSat_Approx.

$$\mathbb{E}(\text{sat}(v, \varphi)) \leqslant \mathbb{E}\left(\text{sat}(v, \varphi) \mid v_i\right)$$

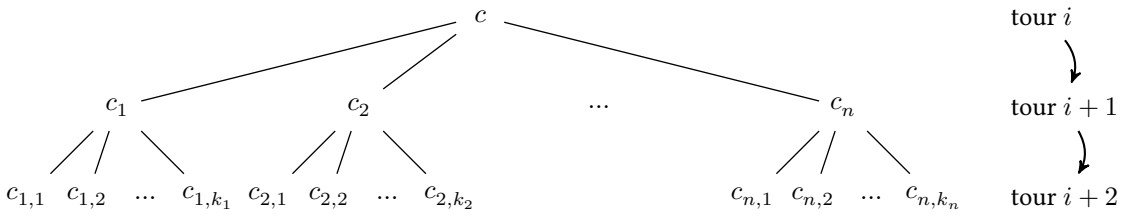
Question 8. En déduire que l'algorithme MaxSat_Approx calcule une valuation v satisfaisant dans φ un nombre de clauses supérieur ou égal à l'espérance $\mathbb{E}(\text{sat}(v, \varphi))$ du nombre de clauses satisfaites.

Exercice 2

On considère un *jeu à deux joueurs*, désignés par A et B. La configuration d'une partie est une donnée finie. À chaque tour, chaque joueur n'a qu'un nombre fini de coups possibles. On appelle *configuration* une description complète de l'état d'une partie. Par exemple, au puissance 4, une configuration est une description de la grille et l'information est : c'est à X de jouer. Aux échecs, une configuration est une description de l'échiquier et l'information est : c'est aux blancs de jouer (ou aux noirs), tels pions ont avancé de 2 cases initialement, mais aussi toutes les configurations précédentes (pour les répétitions de position), éventuellement le temps de jeu de chaque joueur, etc.

À toute configuration du jeu, une *fonction d'évaluation* associe une valeur réelle, avec des valeurs particulières pour la défaite, la victoire, et la partie nulle éventuellement. Comme son nom l'indique, cette fonction évalue la position d'un joueur : plus cette valeur est grande et plus le joueur est en bonne position. Chaque joueur essaie de maximiser sa fonction d'évaluation et de minimiser celle de l'autre joueur.

La *stratégie min-max* est basée sur cette idée. Supposons qu'on soit dans une configuration c et que le joueur A ait le trait. Soit f la fonction d'évaluation de A. L'arbre des coups possibles à partir de c est représenté ci-dessous.



Si A joue le coup i qui le mène en position c_i, alors B cherchera à minimiser la fonction d'évaluation de A au coup suivant, c'est-à-dire choisira la configuration $c_{i,j}$ qui minimise f. A a donc intérêt à choisir le coup i qui maximise ce minimum. C'est la stratégie *min-max*. A choisit donc de jouer le coup i_0 tel que :

$$\min_{1\leqslant j\leqslant k_{i_0}} f\left(c_{i_0,j}\right) = \max_{1\leqslant i\leqslant n} \min_{1\leqslant j\leqslant k_i} f\left(c_{i,j}\right)$$

On peut itérer plus loin cette méthode en prenant comme évaluation sur la configuration $c_{i,j}$ non pas la fonction f mais l'évaluation rendue par la méthode *min-max* elle-même. L'itérer n fois, c'est prévoir $2n$ coups à l'avance.

Question 1.

□ **1.1.** Écrire une fonction `minimize` qui prend une fonction de comparaison `cmp : 'a -> 'a -> bool`, une fonction d'évaluation `f : 'b -> 'a`, une liste `lst` d'objets de type `'b` et qui renvoie le couple `(x,v)` où `x` est un élément de `lst` minimisant la fonction `f` pour la relation `cmp` et `v` la valeur de `(f x)`.

□ **1.2.** En déduire deux fonctions `min_lst` et `max_lst` qui reçoivent une fonction `f : 'a -> float`, une liste `lst : 'a list` et qui renvoient un couple `(x,v)` qui minimise `f`.

Question 2. Soient `'a` le type des configurations du jeu et `eval : 'a -> float` la fonction d'évaluation. On suppose que cette fonction renvoie `0.0` pour une défaite, `1.0` pour une victoire, et un réel de $]0,1[$ sinon. On suppose également qu'il n'y a pas de partie nulle. Soit `suiv : 'a -> 'a list` une fonction qui renvoie, pour une configuration donnée, la liste des configurations suivantes possibles, c'est-à-dire la liste des coups possibles.

□ **2.1.** Écrire une fonction `minmax` qui reçoit en arguments `eval`, `suiv`, une configuration `c` et qui renvoie le coup à jouer par la stratégie *min-max*, pour un seul niveau d'analyse (c'est-à-dire 2 coups).

□ **2.2.** Écrire une fonction `minmax_n` qui reçoit les mêmes arguments, un entier `n` et qui renvoie le coup à jouer pour une analyse sur $2n$ coups.

Question 3. On applique la stratégie précédente au jeu des allumettes. Initialement, des allumettes sont alignées sur une table. À tour de rôle, chaque joueur prend 1, 2 ou 3 allumettes. Celui qui prend la dernière allumette a perdu. Il existe une stratégie gagnante pour ce jeu que la méthode *min-max* permet de trouver. On choisit comme type des configurations le couple d'un booléen, qui indique quel joueur doit jouer, et un entier qui indique combien il reste d'allumettes sur la table.

```
type configuration = bool * int
```

□ **3.1.** Écrire une fonction `affiche_config` affichant une configuration. Par exemple, la configuration `(true, 13)` pourrait être affichée comme suit.

```
au joueur 1: |||||||||||||
```

□ **3.2.** Écrire une fonction `suiv` qui, pour une configuration, renvoie la liste des configurations suivantes possibles.

□ **3.3.** Écrire une fonction `eval` d'évaluation d'une configuration.

□ **3.4.** Écrire une fonction `joue` qui associe à une configuration la configuration suivante, obtenue avec la stratégie *min-max* appliquée à une profondeur suffisante pour trouver la stratégie gagnante (si elle existe).

□ **3.5.** En déduire un programme où l'ordinateur fait jouer les deux joueurs et un programme où vous affrontez l'ordinateur.