Laboratoire 1 - Calculabilité et complexité

Exercice 1 - Mon ami ment toujours

Un ami un peu bizarre vient vous voir et vous dit *"Je mens toujours"*. On considère le problème de décision suivant : *"Votre ami est-il un menteur ?"*. Expliquez pourquoi ce problème est incalculable.

Exercice 2 - Équation de Navier-Stokes

Les équations de Navier-Stokes sont des équations de mécanique des fluides qui décrivent le mouvement, notamment des masses d'air et permettent donc de prévoir des phénomènes météo. On ne sait pas grand chose de leurs solutions, même pas si elles existent, mais imaginons que la physique nous donne ces solutions et considérons le problème de décision suivant : pleuvra-t-il 29 juillet 5287 ? Ce problème est-il calculable ? Et une fois qu'on l'a dit, est-on plus avancé ?

Exercice 3 - Machine de Turing

Soit une machine de Turing déterministe dont le but est de décider si un nombre écrit en base 10 est pair ou non. Pour cela, la machine écrira 0 si le nombre est pair et 1 sinon sur la case juste après le nombre. Proposez un alphabet, un ensemble d'états et une fonction de transition pour construire cette machine. Pour simplifier votre machine, vous pouvez introduire le symbole $*$ ayant pour sémantique : *on ne change pas le symbole de la case.*

Exécutez votre machine sur les nombre 3729 et 6492. On décrira le processus en donnant les transitions successives.

Exercice 4 - Tri d'une liste

On considère le problème du tri d'une liste de nombres entiers. Répondez aux questions suivantes.

1. Soit le problème de décision suivant : *"On retourne OUI si une liste de nombre est triée de manière croissante"*. Est-ce que ce problème de décision est calculable ? Expliquez pourquoi.
2. Soit l'algorithme de tri suivant : *"On teste toutes les permutations possibles des nombres et on vérifie si la liste est triée"*. Quelle est la complexité de cet algorithme ? Qu'en déduire sur la classe de complexité du problème de tri d'une liste ?
3. Est-ce que le problème de trier une liste est dans NP ? Justifiez-le de deux façons différentes.
4. Est-ce que le problème de trier une liste est dans P ?

Exercice 5 - Le problème du sac à dos

Soit un ensemble d'objets $\mathcal{N} = \{i_1, i_2, ..., i_n\}$ ayant chacun une valeur u_i et un poids w_i. Le problème du sac-à-dos consiste à sélectionner un sous-ensemble des objets de telle sorte que le poids total ne dépasse

pas une capacité C, et que la valeur cumulée des objets sélectionnés soit maximale. Une formulation mathématique du problème est la suivante :

$$\begin{aligned} \text{maximize} \quad & \sum_{i=0}^{n-1} x_i u_i \\ \text{subject to} \quad & \sum_{i=0}^{n-1} x_i w_i \leq C \\ & x_i \in \{0,1\} \qquad \forall i \in \{0,...,N-1\} \end{aligned}$$

Pour cet exercice, il vous est demandé d'implémenter ce problème en utilisant des techniques standards (algorithme glouton et programmation dynamique simple). L'objectif est de vous sensibiliser aux limitations et difficultés qu'éprouvent ces méthodes. Une implémentation de ce problème est proposée dans le fichier `knapsack.py`. De plus, le fichier *main.py* vous permet d'exécuter votre méthode sur tout un jeu d'instances (`small` ou `medium`) et d'obtenir une visualisation des résultats.

Algorithme glouton

Le principe d'un algorithme glouton est le suivant : *à chaque étape, une décision qui paraît bonne sur le court terme est exécutée*. Dans le cas du problème du sac à dos, un algorithme glouton simple consiste à trier les objets en fonction un certain critère. L'algorithme passe ensuite chaque objet en revue en commençant par le meilleur objet selon le critère. Si l'objet peut encore rentrer dans le sac, alors il est ajouté, sinon, on passe à l'objet suivant. L'algorithme s'arrête une fois qu'aucun objet ne peut être ajouté.

Votre objectif est de compléter la méthode `greedy_knapsack.py` en utilisant votre propre critère de sélection. Vous pouvez exécuter votre algorithme comme suit.

```
python3 main.py --size=small --agent=greedy
```

De plus, répondez aux questions suivantes.

1. Quelle est la complexité de cet algorithme ?
2. Est-ce que vous avez la garantie que votre algorithme trouve la solution optimale ?

Algorithme de programmation dynamique

La programmation dynamique se base sur un principe simple : la décomposition du problème initial en sous-problèmes, plus simples à résoudre. Les sous-problèmes sont résolus pour ensuite construire la solution du problème initial. Généralement, la solution du problème est formulée sous la forme d'équations de récurrence.

Dans le cas du problème du sac à dos, on définit le sous-problème comme étant la solution d'un problème de sac à dos avec i objets ($i \leq N$) et une capacité j ($j \leq C$). On definit $V[i,j]$ comme étant la solution optimale du problème avec les i premiers objets et la capacité maximum j. Si le nombre d'objets ou la capacité est nulle, la valeur optimale est triviale ($V[0,j] = 0$ et $V[i,0] = 0$). Ces valeurs sont communément nommées les *conditions initiales* du problème.

Le cœur de la méthode consiste à construire les équations de récurrence. Supposons que nous cherchons la solution optimale pour un certain objet i et un certaine capacité j. Deux choix sont possibles :

1. L'objet i ne fait pas partie de la solution optimale du sous problème. La meilleur valeur sans cet objet est alors la valeur optimale du sous problème avec la même capacité mais un objet en moins. Mathématiquement, on a que $V[i, j] = V[i-1, j]$.
2. L'objet i fait partie de la solution optimale du sous-problème. La meilleure valeur avec cet objet est alors l'utilité de l'objet plus la valeur optimale du sous-problème ayant un objet en moins, et son poids retiré. Mathématiquement, on a que $V[i, j] = u_i + V[i-1, j-w_i]$.

Ainsi, la solution optimale du sous-problème est le meilleur de ces deux scénarios. De plus, notez que le scénario (2) n'est possible que si la capacité du sac-à-dos le permet (mathématiquement : $j - w_i \geq 0$). Si cette condition n'est pas respectée, seulement le premier scénario est possible.

Tenant compte de cela, la formulation du problème en programmation dynamique est la suivante, avec $V[i, j]$ étant la meilleure valeur qui peut être obtenue en utilisant les i premiers objets avec une capacité maximale de j.

$$
\begin{aligned}
V[i,j] = &\quad \max\Big(V[i-1,j], u_i + V[i-1, j-w_i]\Big) && \text{si } j - w_i \geq 0 \\
V[i,j] = &\quad V[i-1,j] && \text{sinon} \\
V[0,j] = &\quad 0 && \text{Première condition initiale} \\
V[i,0] = &\quad 0 && \text{Deuxième condition initiale}
\end{aligned}
$$

Finalement, la solution du problème initiale est la valeur $V[N, C]$, consistant à la valeur optimale d'un sac à dos d'une capacité C et ayant considéré les N objets.

En guise d'illustration, considérons le problème suivant.

i	c	u
1	1	2
2	3	3
3	4	5

Les valeurs successives de $V[i, j]$, avec $N = 3$ et $C = 6$, correspondant à la résolution des différents sous-problèmes sont les suivantes.

	i=0	i=1	i=2	i=3
c=0	0	0	0	0
c=1	0	2	2	2
c=2	0	2	2	2
c=3	0	2	3	3
c=4	0	2	5	5
c=5	0	2	5	7
c=6	0	2	5	7

Votre objectif est de compléter la méthode `dynamic_knapsack_recursive.py` en implémentant une résolution par programmation dynamique. Vous pouvez exécuter votre algorithme comme suit.

```
python3 main.py --size=small --agent=dynamic
```

De plus, répondez aux questions suivantes.

1. Quelle est la complexité de cet algorithme ?
2. Est-ce que vous avez la garantie que votre algorithme trouve la solution optimale ?

Comparaison des méthodes

Finalement, il vous est demandé de réfléchir à comment vos deux algorithmes peuvent être comparés. Pour cela, exécuter la méthode suivante, qui présentera les résultats des deux méthodes sur le jeu d'instance considéré.

```
python3 main.py --size=small --agent=experiments
```

Quelles sont les forces et faiblesses des deux méthodes ? Pour les futurs rapports, il est important de réaliser des graphes de vos résultats. Nous verront dans la suite du cours d'autres façons de visualiser des résultats expérimentaux.

Votre propre agent

S'il vous reste du temps, vous êtes invités à implémenter votre propre agent et à analyser son efficacité par rapport aux deux compétiteurs. Vous pouvez exécuter votre algorithme comme suit.

```
python3 main.py --size=small --agent=advanced
```