IA - Résolution de problèmes

Guillaume Roussin - SRI 2A

I - Carnet de bord

Séance 1 Mise en place de l'environnement de développement sous Python. Utilisation d'anaconda pour mettre en place d'un environnement cloisonné.

Séance 2 Réalisation de l'étape 1.

Séance 3 Nettoyage du code de l'étape 1 et début de l'étape 2.

Séance 4 Fin de l'étape 2 cas 1.

Séance 5 Absent suite à un accident.

Séance 6 Début de l'étape 2 cas 2.

Séance 7 Continuation de l'étape 2 cas 2.

Séance 8 & 9 Fin de l'étape 2 cas 2. Début de l'étape 3.

Séance 10 Fin de l'étape 3. Réalisation de l'étape 4.

Séance 11&12 Essai de l'étape 5. Finalisation du livrable.

II - Réalisation

Étape 1 - Solveur SAT

On encode un état par un entier signé à 16 bits :
- 8 premiers bits pour le sommet
- 8 derniers pour la couleur

00000001 00000001 → 129 (sommet 1 à la couleur 1)

Pour la génération des clauses, on le fait en deux étapes :

- Les clauses concernant le sommet (le sommet doit être d'une seule couleur)
- Les clauses concernant les voisins (les voisins ne peuvent pas être de la même couleur que le sommet)

Étape 2 Cas 1 - Solveur A*

Pour le cas 1, nos états contiennent 2 variables : un sommet courant et un sommet objectif. On peut ainsi programmer notre état de la manière suivante :

estSolution	Vérifie que le sommet courant soit égal au sommet objectif
successeurs	Prends les sommets adjacents au sommet courant et les convertit en états
h	Retourne la distance euclidienne entre l'état courant et l'état objectif
k	Retourne le poids de l'arête liant le sommet courant à celui de l'état passé en paramètre

Étape 2 Cas 2 - Solveur A*

Pour le cas 2, on modifie la structure de notre état. Il contient maintenant le sommet de départ, le sommet courant, ainsi qu'une liste des sommets visités.

estSolution	Vérifie que le sommet courant soit égal au sommet de départ et que tous les sommets du graphe aient été visités
successeurs	Prends les sommets adjacents au sommet courant qui n'ont pas encore étés visités (ou le sommet de départ tout a été visité) et les convertit en états
h	Retourne le nombre d'états à visiter restant multiplié par le poids minimal entre deux sommets
k	Retourne le poids de l'arête liant le sommet courant à celui de l'état passé en paramètre

Étape 2 Cas 3 - Solveur A*

La structure interne reste identique à celle du cas 2.

estSolution	Pareil que pour le cas 2
successeurs	Prends les sommets qui n'ont pas encore étés visités (ou le sommet de départ tout a été visité) et les convertit en états
h	Retourne le nombre d'états à visiter restant multiplié par la distance minimale à vol d'oiseau entre deux sommets
k	Retourne la distance euclidienne entre le sommet courant et celui de l'état passé en paramètre

Étape 3 - Hill Climbing et Solveur Tabou

On compare deux algorithmes dans cette étape. On reprend la même logique que l'étape 2 cas 3 en adaptant la logique à celle des solveurs utilisés (passage de l'espace d'état à l'espace des solutions).

nlleSolution	Mélange tous les sommets (hors départ) du graphe pour en faire un chemin aléatoire, puis rajoute le sommet de départ au début et à la fin du chemin
lesVoisins	Génère des solutions voisines qui ont chacune deux sommets voisins échangés (voir exemple plus bas)
unVoisin	Retourne un élément au hasard de la liste obtenue de **lesVoisins**
eval	Évalue le coût d'une solution en calculant la somme des distances euclidiennes entre les sommets du chemin

Générations des voisins :

A-B-C-D-E-A ⇒ A-C-B-D-E-A | A-B-D-C-E-A | A-B-C-E-D-A

Résultats L'algorithme Hill Climbing donne des résultats à la précision aléatoire. Le solveur Tabou en revanche donne des résultats plus fiables.

Étape 4 - Solveur CSP

Rien de particulier à détailler, appeler le solveur a suffi à résoudre cette étape.

III - Retour d'expérience

Dans l'ensemble, le TP s'est bien passé malgré l'étape 5 non complétée. Ce projet m'a permis de mieux comprendre le fonctionnement des solveurs (surtout lors des étapes 2 et 3), et m'a même permis de résoudre un défi de programmation en dehors des cours (https://adventofcode.com/2022/day/12).

Cependant, ce projet n'a pas été sans difficultés / regrets :

- La mise en place du projet sur la machine locale, prenant en compte les différentes bibliothèques et versions de python à respecter pour garantir que le livrable puisse être exécuté sur la machine d'évaluation, a pris beaucoup de temps. De même, l'installation des bibliothèques pour l'étape 5 n'a pas abouti sur machine locale à cause d'une erreur obscure concernant une librairie non trouvée, ce qui m'a découragé d'essayer de résoudre cette partie.
- J'ai trouvé l'étape 4 très sommaire. Elle ne demandait pas de réflexion pour la compléter, et je trouve dommage de ne pas avoir plus appris sur le solveur CSP.
- Je trouve que le projet était très abstrait, ce qui nuisait beaucoup à la compréhension des tâches à réaliser. On nous vend un robot R2D2 qui dépose des cubes colorés dans des villes, mais à aucun moment, on voit cela se réaliser concrètement (on s'arrête à la réponse binaire de si une carte peut être colorée, ou au chemin au poids minimal). Je pense qu'il aurait était sympathique de voir une représentation plus visuelle des résultats (en prenant la carte à 10 villes par exemple) afin de montrer qu'utiliser des solveurs peut être bien plus efficace pour résoudre certains problèmes que d'essayer de calculer une solution manuellement.