TP - Optimisation et Graphes

Eliott Georges, Quentin Guerisse

4A ICY - 2024

Table des matières

Programme python à utiliser :

Le fichier `main.py` permet d'executer les différentes résolutions de problèmes

1 Plus court chemin

1.1 Construction du graphe

Voir les fichiers `graph.py` et `node.py` dans le code fournis.

1.2 Modélisation mathématique

1.2.1 Modèle théorique

L'objectif de cette sous-partie est de définir un modèle mathématique d'optimisation permettant de résoudre le problème du plus court chemin avec CPLEX.

On considère les données suivantes :

— N : ensemble des noeuds
— E : ensemble des arêtes, par exemple si le noeud 4 est lié à 5 on a $(4, 5) \in E$
— $\forall i \in N,\ V(i)$ est l'ensemble des voisins de i
— $\forall (i, j) \in E,\ d_{i,j}$ est la distance entre les noeud i et j
— $s, t \in N$ sont respectivement les noeuds de départ et d'arrivée

On définit alors la fonction objective suivante :

$$min \sum_{(i,j)\in E} d_{i,j} x_{i,j}$$

Contraintes :

La première contrainte permet de s'assurer que l'on a une unique arête partant de s :

$$\sum_{k\in V(s)} x_{s,k} + \sum_{k\in V(s)} x_{k,s} = 1$$

Ensuite, nous voulons nous assurer que l'on arrive bien en t avec une unique arête :

$$\sum_{k\in V(s)} x_{t,k} + \sum_{k\in V(s)} x_{k,t} = 1$$

Enfin, pour garantir une continuité du chemin parcourus on a :

$$\forall k \in E \setminus \{s, t\},\ \sum_{k\in V(i)} x_{i,k} - \sum_{k\in V(j)} x_{k,j} = 0$$

1.2.2 Résultats CPLEX

Les résultats sont affichés avec la librairie `networkx` en python.

`reseau_10_10_1.txt` :

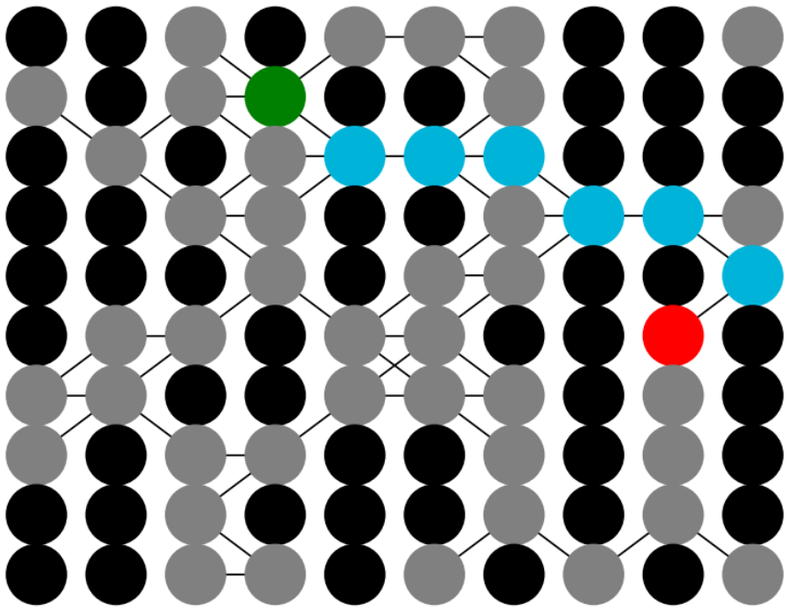

FIGURE 1 – Résultat CPLEX pour `reseau_10_10_1.txt`

`reseau_20_20_1.txt` :

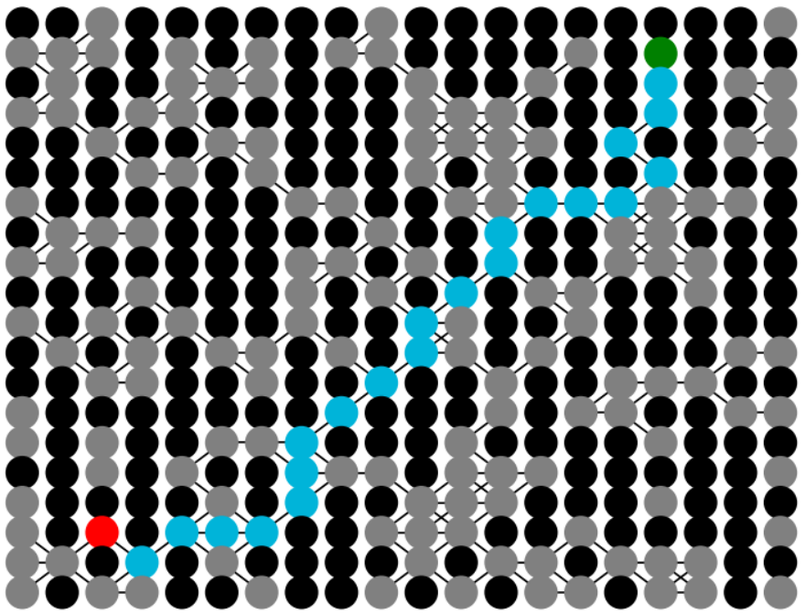

FIGURE 2 – Résultat CPLEX pour `reseau_20_20_1.txt`

`reseau_50_50_1.txt` :

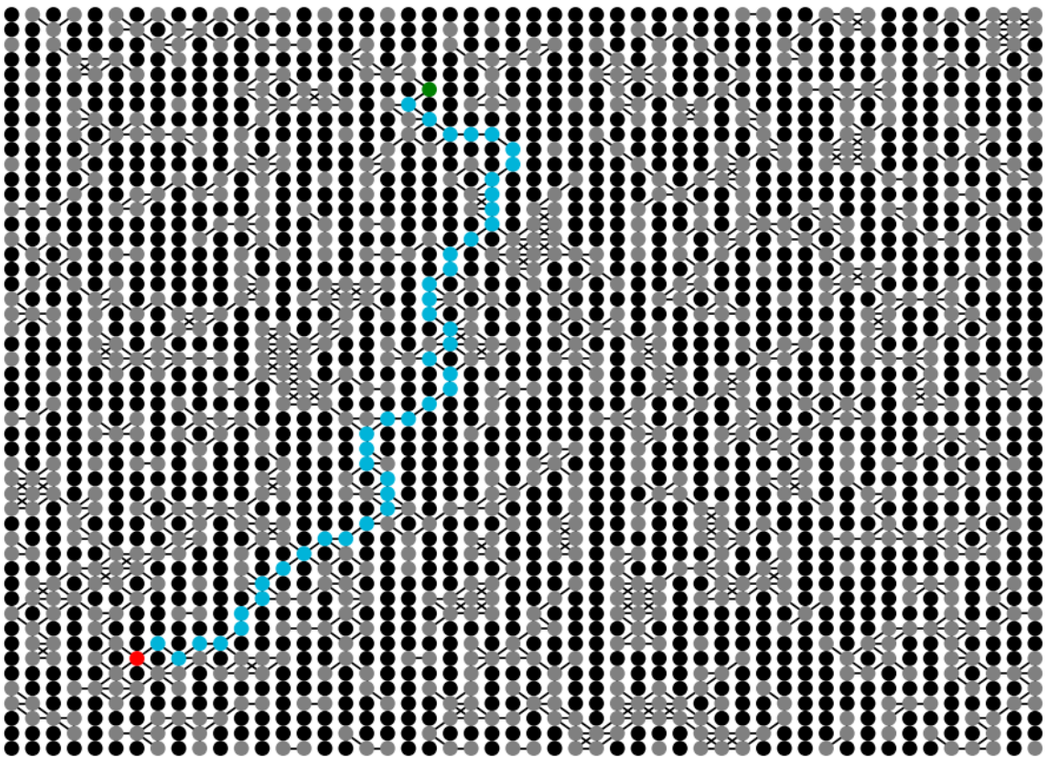

FIGURE 3 – Résultat CPLEX pour `reseau_50_50_1.txt`

1.3 Algorithme de cheminement

1.3.1 Implémentation de A*

En nous inspirant du code de la page wikipedia donnée dans le sujet, nous obtenons ces résultats :

`reseau_10_10_1.txt` :

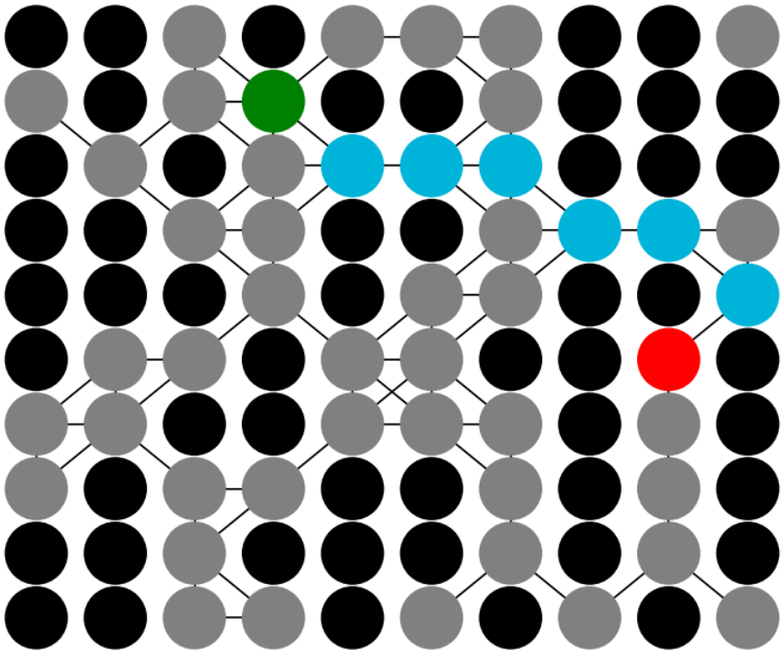

FIGURE 4 – Résultat A* pour `reseau_10_10_1.txt`

`reseau_20_20_1.txt` :

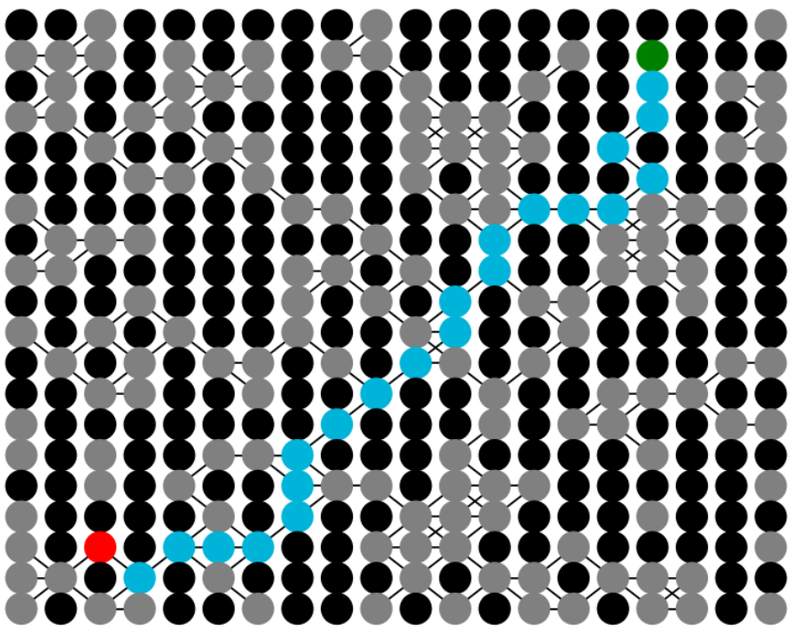

FIGURE 5 – Résultat A* pour `reseau_20_20_1.txt`

`reseau_50_50_1.txt` :

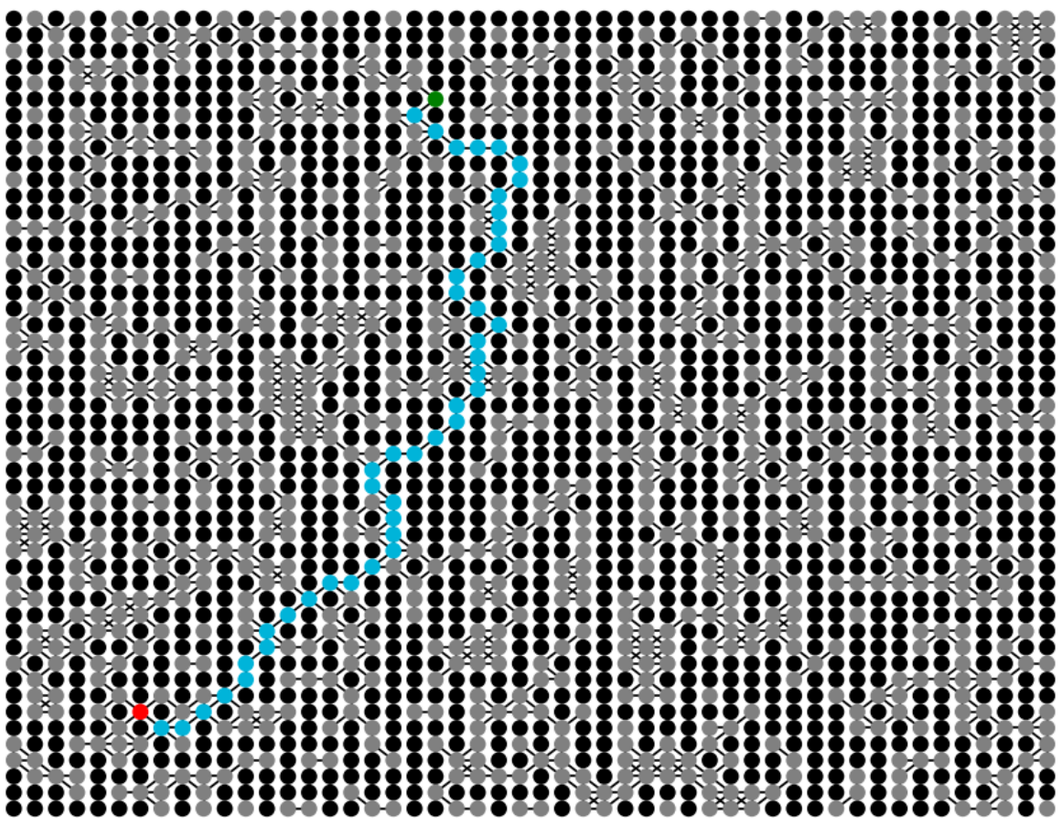

FIGURE 6 – Résultat A* pour `reseau_50_50_1.txt`

1.3.2 Cas $h(x) = 0$

Lorsque $h(x) = 0$, on tombe sur le cas particulier de l'algorithme de Dijkstra. Celui-ci parcourt le graphe en semi-profondeur semi-largeur en utilisant des sortes de coupures. En effet, lorsque le cout devient trop élevé, il se réoriente pour éviter de parcourir tout le graphe inutilement.

On remarque une différence de performances temporelle :

TABLE 1 – Temps d'execution (secondes) CPU en fonction de l'heuristique

	$h(x) = 0$	Distance euclidienne
`reseau_50_50_1.txt`	0.00442	0.00373
`reseau_20_20_1.txt`	0.00096	0.00076
`reseau_10_10_1.txt`	0.00019	0.00016

Cette différence est surrement due au fait que l'heuristique de distance euclidienne dirige la recherche dans le sens du parcours optimal, en évitant ainsi l'exploration d'un grand nombre de noeuds.

Essayons sur un graphe de taille 200×200 généré aléatoirement avec une densité d'obstacles faible :

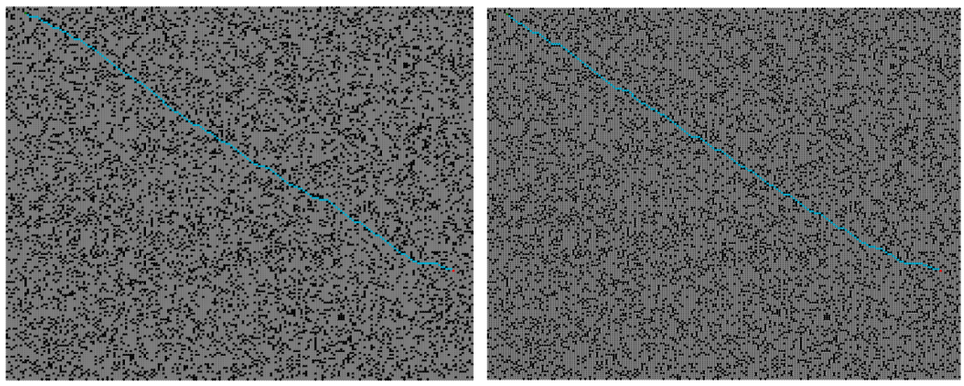

FIGURE 7 – Dijkstra (gauche) vs Distance euclidienne (droite) pour A*, sur `graph_0.8_200_200.txt`

TABLE 2 – Temps d'execution (secondes) CPU en fonction de l'heuristique

	$h(x) = 0$	Distance euclidienne
`graph_0.8_200_200.txt`	0.1764	0.0662

1.4 A* contre CPLEX

Nous avons vu deux méthodes de résolution du plus court chemin, A* et CPLEX. Ces deux méthodes sont efficaces et donnent des résultats proches :

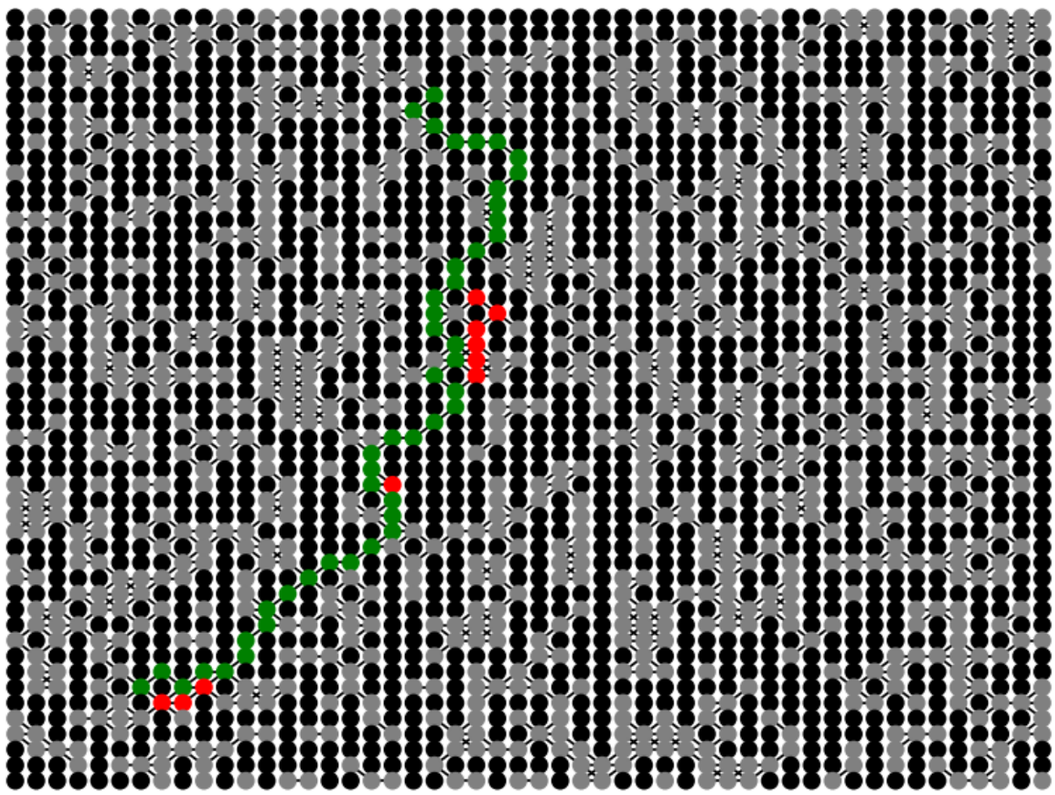

FIGURE 8 – Comparaison des résultats entre A*(rouge) et CPLEX (vert) sur `reseau_50_50_1.txt`

En revanche, niveau performances les différences se font ressentir :

TABLE 3 – Temps d'execution (secondes) CPU pour A* contre CPLEX

	CPLEX	A*
`reseau_50_50_1.txt`	0.16	0.00373
`reseau_20_20_1.txt`	0.11	0.00076
`reseau_10_10_1.txt`	0.05	0.00016

2 Problème du voyageur de commerce

2.1 Méthode d'énumération (Brute force)

Pour réaliser un algortihme de brute force, il a suffit de calculer dans un premier temps toutes les permutations possibles partant d'un noeud. Ppour éviter de les calculer absolument toutes, nous avons supprimé les cycles équivalents, ce qui est équivalent à calculer pour n noeuds, toutes les permutation commençant par le même noeud uniquement.

Exemple si l'on décide de partir de 0 :

$$
\begin{aligned}
&(0, 1, 2, 3, ...n)\\
&(0, 2, ...n, 1)\\
&\vdots\\
&(0, i, ...n, ...j)
\end{aligned}
$$

2.2 Modélisation mathématique

On considère des graphes non-orientés, complet ayant n noeuds. Ainsi $|E| = n(n-1)$.

Fonction objective :

$$min \sum_{(i,j)\in E} d_{i,j}x_{i,j}$$

Contraintes :

$$\forall k \in [\![1,n]\!]$$
$$\sum_{i=1,\ i\neq k}^{n} x_{i,k} = 1$$
$$\sum_{j=1,\ i\neq k}^{n} x_{k,j} = 1$$

Ajout des variables imaginaires de Miller–Tucker–Zemlin :

$$\forall i,j \in [\![2,n]\!] | i \neq j$$
$$u_i - u_j + 1 \leq (n-1)(1-x_{i,j})$$
$$2 \leq u_i \leq n$$

2.3 Résultats

L'affichage des résultats n'est pas très interessant ici, mais on remarque que l'on passe bien par tous les noeuds une unique fois :

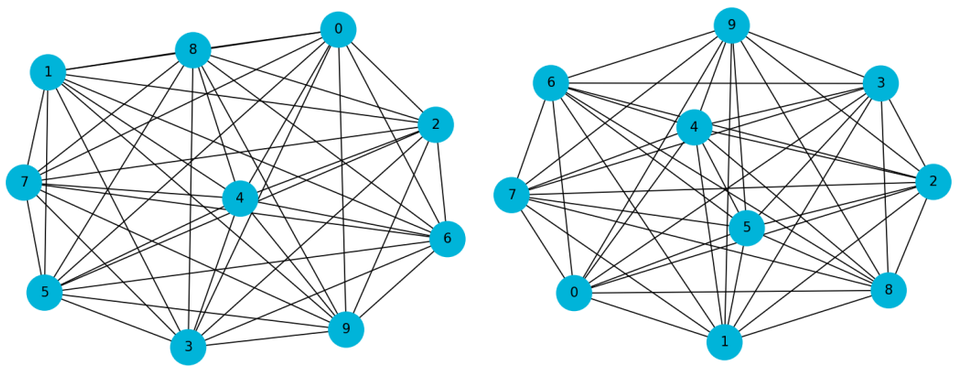

FIGURE 9 – Résultat CPLEX (gauche) et Brute force (droite) pour `graph_10.txt`

2.4 CPLEX contre Brute force

TABLE 4 – Temps d'execution (secondes) CPU pour CPLEX contre Brute force

	CPLEX	Brute force
`graph_50.txt`	0.03	$\sim \infty$
`graph_12.txt`	0.02	> 200
`graph_10.txt`	0.02	1.61
`graph_8.txt`	~ 0.0	0.015
`graph_5.txt`	~ 0.0	~ 0.0

CPLEX est plus rapide que la méthode brute force pour résoudre le problème du voyageur de commerce (TSP) grâce à son algorithme de branch and bound, qui exploite efficacement l'espace de

recherche pour identifier et exclure rapidement les solutions non viables. Cette approche est particulièrement avantageuse pour les grandes instances de TSP, où la méthode brute force serait impraticable en raison de sa complexité exponentielle. CPLEX ajoute des contraintes de sous-tours au problème et résout répétitivement avec les nouvelles contraintes, une méthode qui a été confirmée comme l'une des plus rapides pour les grandes instances de TSP. De plus, l'utilisation de formules telles que la formulation Miller-Tucker-Zemlin (MTZ) pour l'élimination des sous-tours contribue à l'efficacité de CPLEX dans la résolution de TSP, en permettant de gérer des contraintes complexes de manière efficace.