TP - Optimisation et Graphes

Eliott Georges, Quentin Guerisse

4A ICY - 2024

Table des matières

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 arrêtes, 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 arrête 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 arrête :

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

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` :

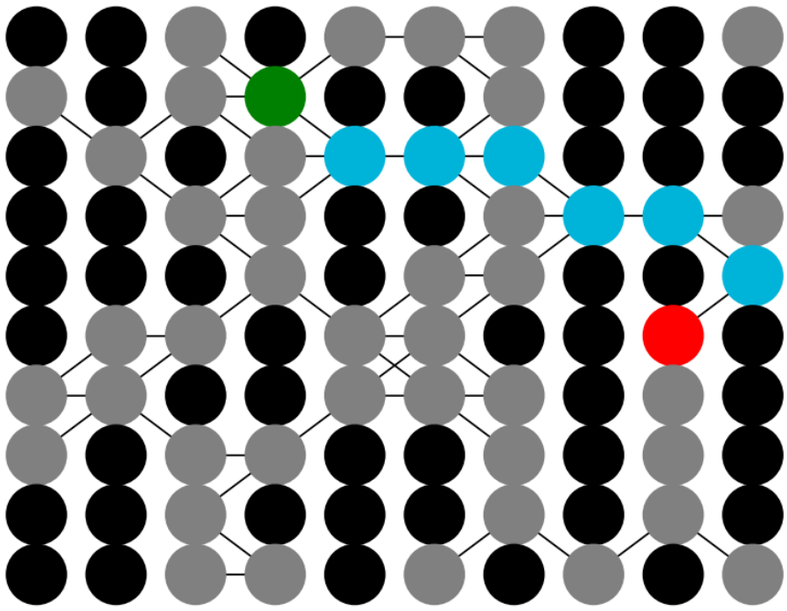

FIGURE 1 – Résultat CPLEX pour `reseau_10_10_1.txt`

`reseau_20_20_1.txt` :

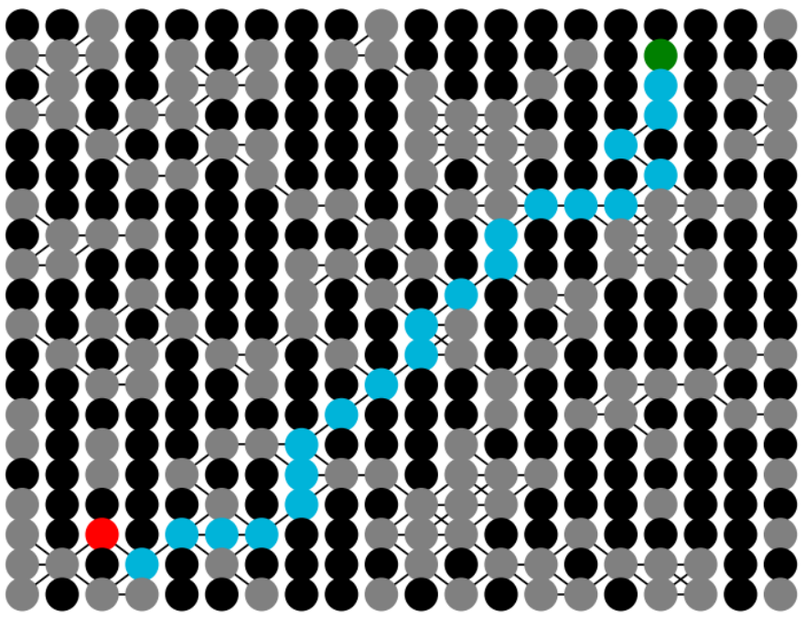

FIGURE 2 – Résultat CPLEX pour `reseau_20_20_1.txt`

`reseau_50_50_1.txt` :

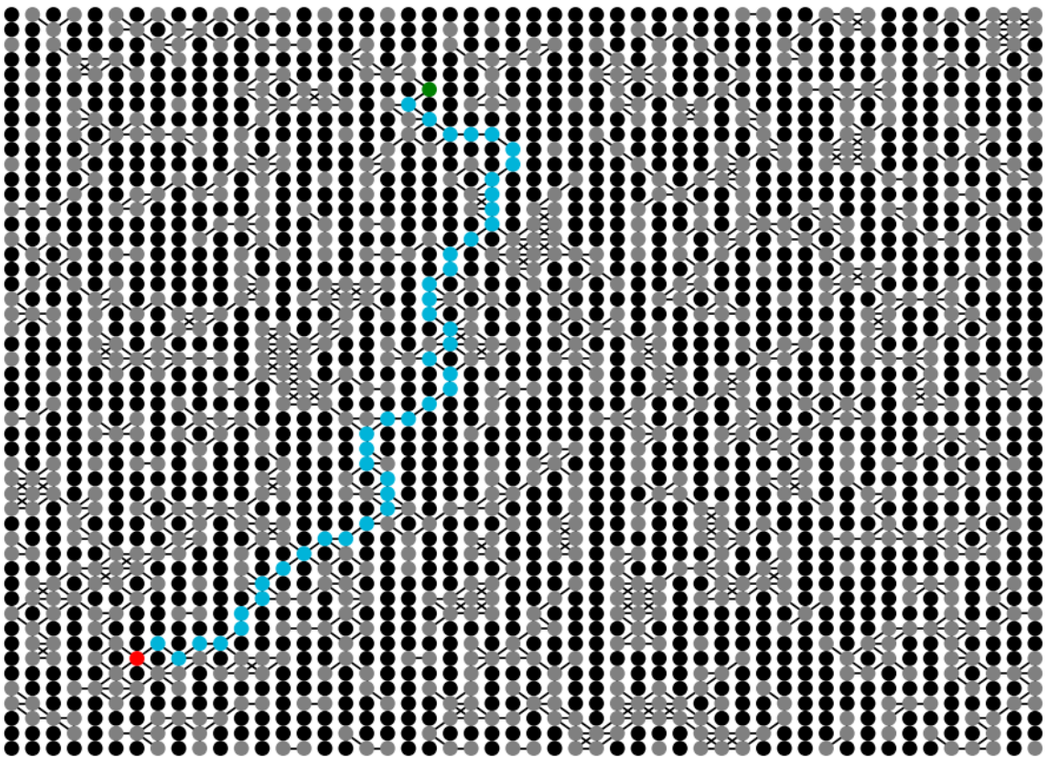

FIGURE 3 – Résultat CPLEX pour `reseau_50_50_1.txt`

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` :

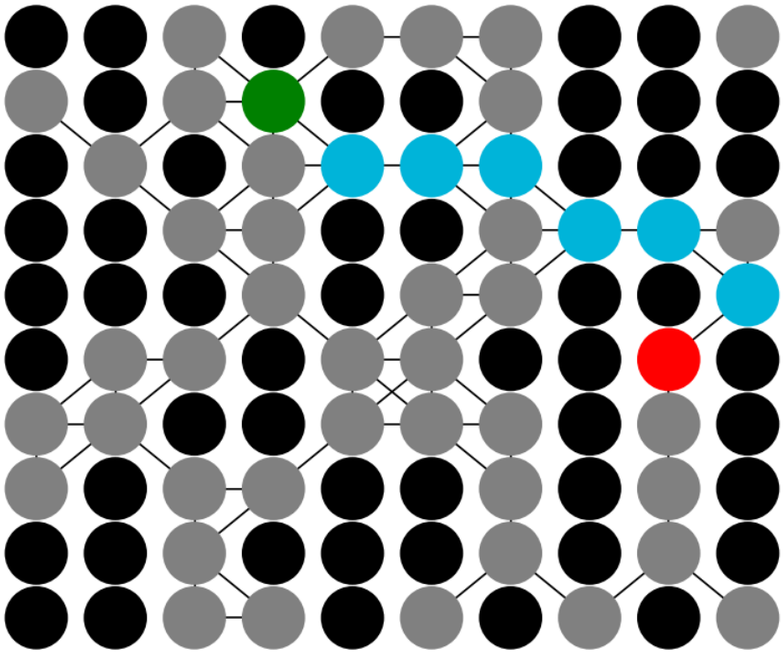

FIGURE 4 – Résultat A* pour `reseau_10_10_1.txt`

`reseau_20_20_1.txt` :

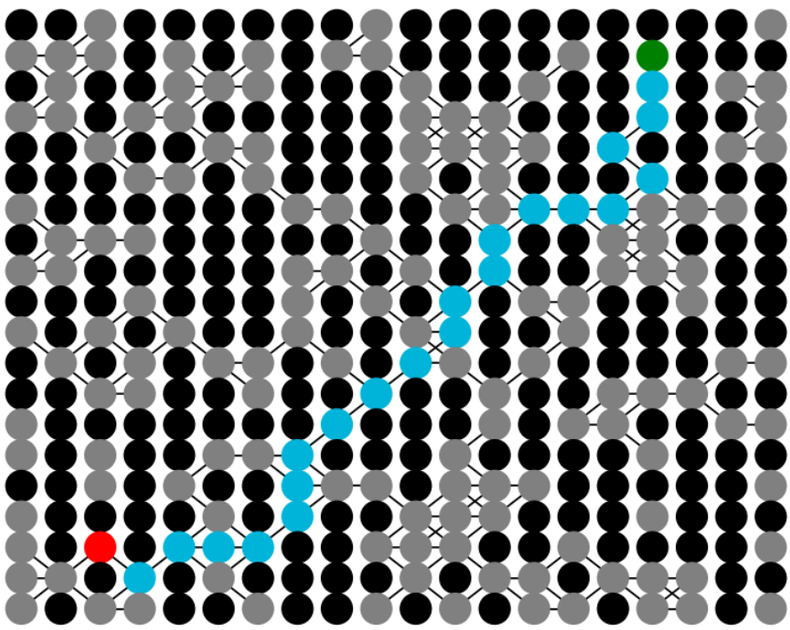

FIGURE 5 – Résultat A* pour `reseau_20_20_1.txt`

`reseau_50_50_1.txt` :

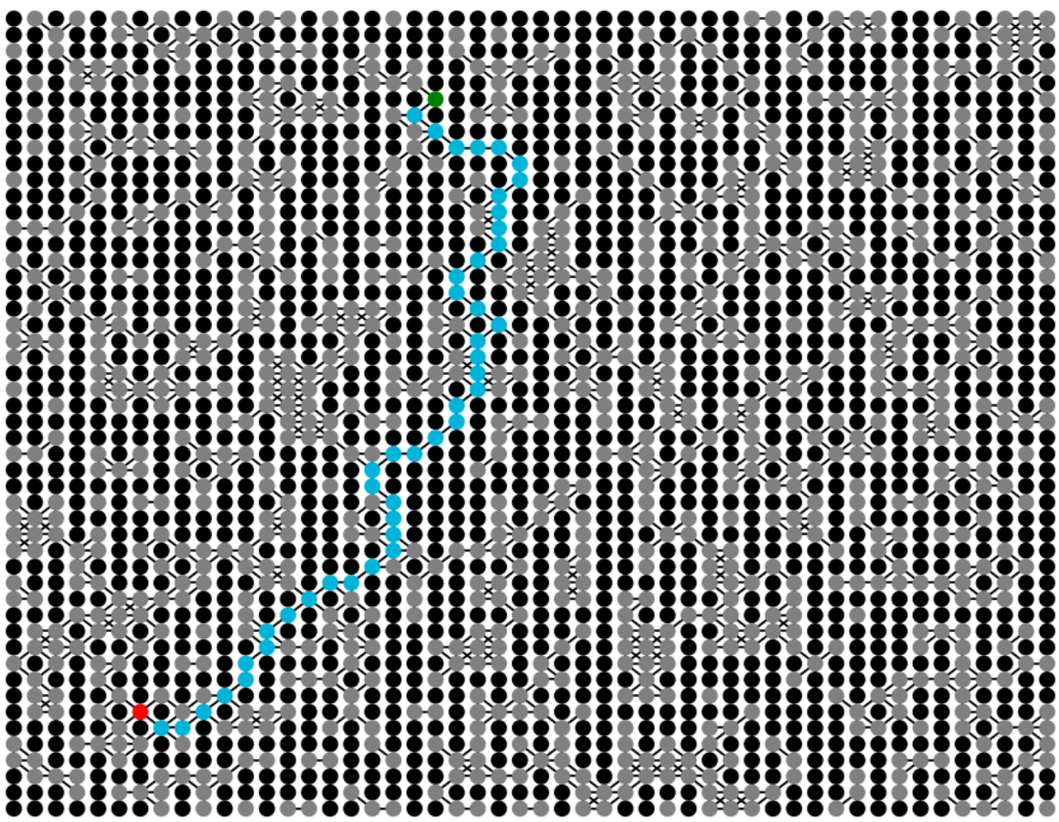

FIGURE 6 – Résultat A* pour `reseau_50_50_1.txt`

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 différnce 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`	0.0181	0.04713
`reseau_20_20_1.txt`	0.0012	0.0059
`reseau_10_10_1.txt`	0.00015	0.0024

1.3.3 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 :

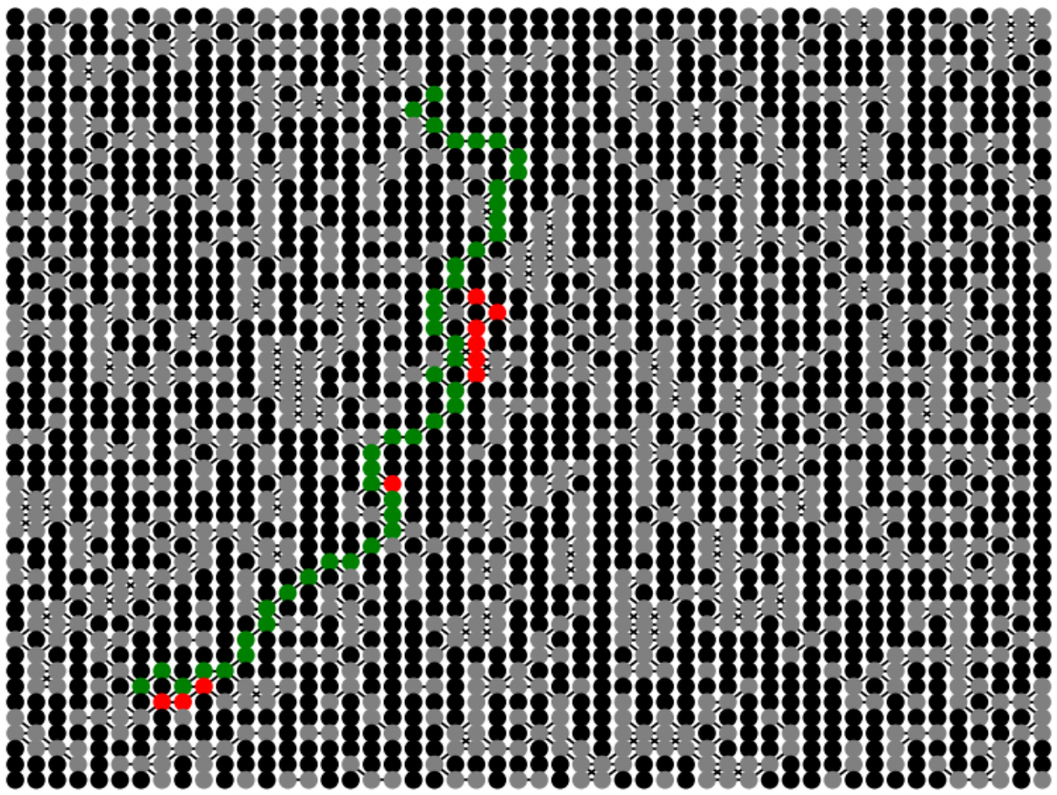

FIGURE 7 – Comparaison des résultats entre A*(rouge) et CPLEX (vert) sur `reseau_50_50_1.txt`

Pour visualiser ce résultat, il suffit d'utiliser le fichier `plotter.py` :

```
python plotter.py <nom fichier du graphe>
```

puis entrer "0", ce qui permet de comparer les 2 résultats.

En revanche, niveau performances les différences se font ressentir :

TABLE 2 – Temps d'execution (secondes) CPU pour A* contre CPLEX

	CPLEX	A*
`reseau_50_50_1.txt`	0.16	0.04713
`reseau_20_20_1.txt`	0.11	0.0059
`reseau_10_10_1.txt`	0.05	0.0024

2 Problème du voyageur de commerce