

Rapport de projet Graphes et OpenData

Le problème des 4 cavaliers

Projet réalisé du 4 janvier 2021 au 20 mars 2021

Membres du groupe

MembreMembre

Table des matières

1	Introduction	3
2	Environnement de travail	4
3	Description du projet et objectifs	4
3.1	Exemples de problèmes d'échecs	4
3.2	Notre sujet	6
4	Bibliothèques, Outils et technologies	7
5	Travail réalisé	8
6	Difficultés rencontrées	9
7	Bilan	10
7.1	Conclusion	10
8	Bibliographie	12
9	Webographie	13
10	Annexes	14
A	Cahier des charges	14
B	Exemple d'exécution du projet	14
C	Manuel utilisateur	14

1 Introduction

Notre sujet concerne un jeu d'échecs et le déplacement d'une pièce particulièrement intéressante : le cavalier.

La naissance des échecs peut être estimée entre le troisième et sixième siècle en Asie. Ses origines sont un peu floutées mais nos échecs modernes se dérivent du Chatranj, l'ancêtre du jeu d'échecs.

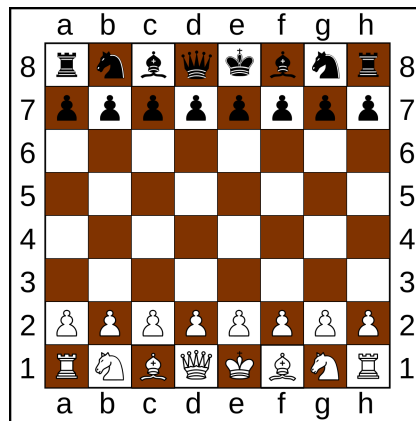


FIGURE 1 – Un échiquier moderne

Il est important de comprendre que chaque pièce dans un jeu d'échecs ne contient pas les mêmes propriétés de déplacement. Il y'a 6 pièces différentes : les pions, les fous, les tours, la dame, le roi et finalement, le cavalier. Nous allons aujourd'hui nous intéresser au cavalier car notre sujet de projet ne concerne que lui.

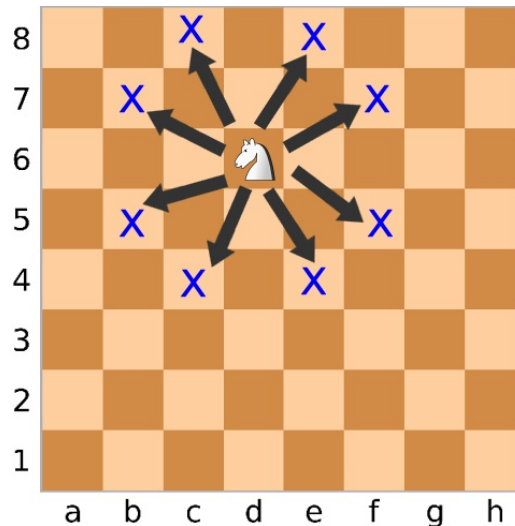


FIGURE 2 – Le déplacement d'un cavalier dans un jeu d'échecs.

Il est seulement permis de glisser un cavalier vers une case vide en utilisant les mouvements en forme de "L" (2 cases vers le haut ou le bas et une case vers la gauche ou la droite, 2 cases vers la gauche ou la droite et une case vers le haut ou le bas).

2 Environnement de travail

Le travail a été entièrement fait sur la plateforme Google Colaboratory et codée en Python car c'est le langage qui nous a été demandé d'utiliser.

Nous avons trouvé Colaboratory efficace car elle nous permettait de collaborer sur le code en direct, on pouvait travailler sur nos propres postes en mettant à jour notre code sur le moment présent.

Son intégration avec Python nous a permis d'importer les bibliothèques et d'afficher les graphes avec beaucoup de facilité.

3 Description du projet et objectifs

L'ensemble de sujets de recherche mathématiques que l'on peut traiter avec les échecs est abondant. En effet, les deux vont très bien ensemble, et nous allons vous montrer quelques exemples de sujets qui peuvent être traités.

3.1 Exemples de problèmes d'échecs

Le problème du cavalier C'est un problème où un cavalier est posé sur une case d'un échiquier et doit visiter toutes les cases sans repasser par la même case.

En 1883, le mathématicien Warnsdorff propose son algorithme pour résoudre le problème : *On déplace toujours le cavalier vers la case qui offre le moins de possibilités de mouvements ultérieurs vers les cases libres.*

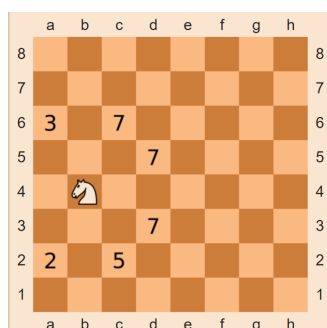


FIGURE 3 – L'application de l'algorithme de Warnsdorff. On voit que le cavalier doit se déplacer à A2 car c'est la case qui lui offre le moins de possibilités de mouvements vers les cases libres.

Il y'a beaucoup de solutions différentes proposées par une variété de mathématiciens, voici la première solution donnée pour ce problème par Al-Adli ar-Rumi au 9e siècle.

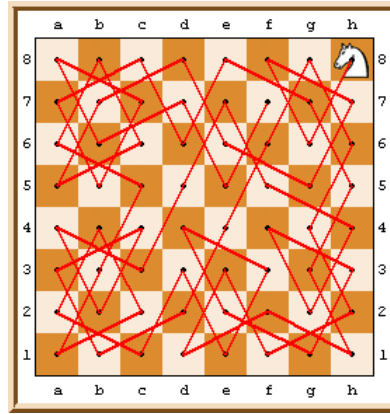


FIGURE 4 – Une solution donnée par Al-Adli ar-Rumi vers l'année 840.

Les 8 dames C'est un problème où on cherche à placer des reines de telle sorte qu'elles ne soient pas en position de se manger. On rappelle que la reine peut se déplacer tout droit vers l'avant, l'arrière, la gauche, la droite et ses 4 diagonales.

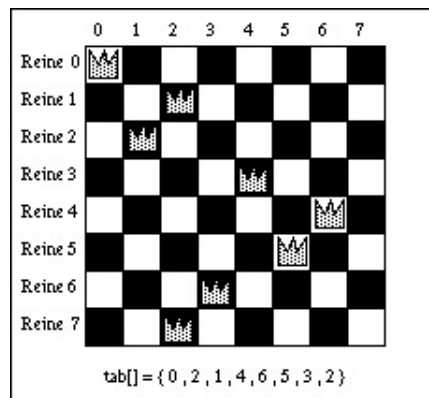


FIGURE 5 – Une des 92 solutions possibles pour un échiquier de taille 8x8

On a donc vu deux exemples de sujets mathématiques traitables par les échecs, mais nous allons maintenant voir le troisième sujet, qui est également notre sujet de projet : le problème des 4 cavaliers.

3.2 Notre sujet

Les 4 cavaliers : On a choisi de traiter le problème des 4 cavaliers pour notre projet Graphes et Open Data car on s'était intéressé au jeu peu de temps avant d'avoir commencé la matière.

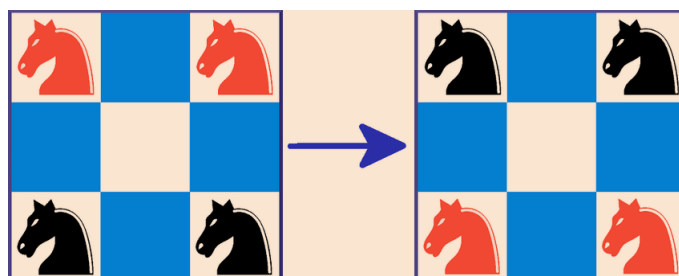


FIGURE 6 – Le problème des 4 cavaliers

Notre sujet ressemble un peu aux deux qu'on a précédemment vu, en effet, le problème des 4 cavaliers est un problème où on a 2 cavaliers blancs et 2 cavaliers noirs, face à face sur un échiquier 3x3, dans le but de s'échanger leurs positions sans se manger.

Il est seulement permis de glisser un cavalier vers une case vide en utilisant les mouvements du cavalier du jeu d'échecs, c'est à dire en forme de "L" (2 cases vers le haut ou le bas et une case vers la gauche ou la droite, 2 cases vers la gauche ou la droite et une case vers le haut ou le bas).

Le but du jeu est de permuter les positions initiales des cavaliers blancs et noirs avec le minimum de déplacements.

Il est important de noter que :

- Un cavalier a toujours maximum deux positions qu'il peut visiter.
- Les cavaliers n'ont pas le droit de se manger.
- Les cavaliers ne doivent pas forcément s'échanger face à face, ils peuvent s'échanger à la diagonale aussi.

Notre objectif sera donc de programmer un code python qui permet de trouver l'itinéraire de déplacement le plus court possible, peu importe la méthode d'échange des cavaliers.

4 Bibliothèques, Outils et technologies

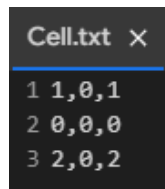
Pour commencer, nous avons utilisé Google Colaboratory, une plateforme qui permet à tout le monde d'exécuter du code sur son navigateur web. Cela nous a permis de collaborer sur le code lors des séances en présentiel mais également à distance.

Nous allons maintenant vous citer les bibliothèques utilisées pour faire fonctionner le code, en commençant par NumPy :

```
import numpy as np
```

Ce dernier nous a servi pour charger le contenu du fichier Cell.txt, un fichier contenant les cellules de l'échiquier ainsi que la position des cavaliers.

On note que 1 correspond aux cavaliers blancs, et 2 correspond aux cavaliers noirs.



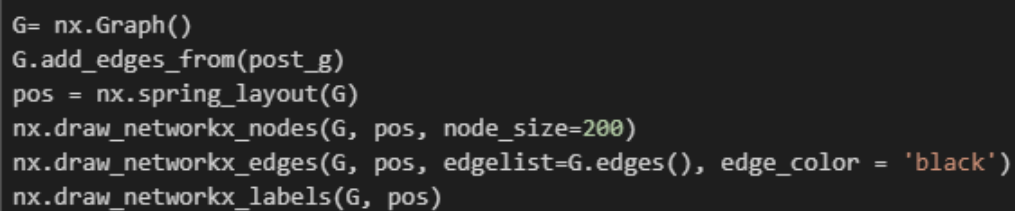
```
Cell.txt X
1 1,0,1
2 0,0,0
3 2,0,2
```

FIGURE 7 – Le fichier Cell.txt

Nous avons ensuite implémenté NetworkX :

```
import networkx as nx
```

Ce dernier nous a servi à représenter graphiquement notre solution à ce problème.



```
G= nx.Graph()
G.add_edges_from(post_g)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, node_size=200)
nx.draw_networkx_edges(G, pos, edgelist=G.edges(), edge_color = 'black')
nx.draw_networkx_labels(G, pos)
```

FIGURE 8 – Le bloc de code qui nous a servi à créer une représentation graphique du problème.

5 Travail réalisé

Nous avons créé 2 fonctions pour faire fonctionner le code :

- **parcourir()**
- **generate_graph()**

Nous avons aussi prévu d'utiliser **Dijkstra** et/ou l'algorithme **A*** pour trouver l'itinéraire le plus court mais ces algorithmes n'ont finalement pas été réalisés car nous avons réussi à trouver cet itinéraire sans devoir passer par ces algorithmes.

parcourir() La fonction `parcourir()` prend une case et, à partir de là, calcule les coups possibles sur l'échiquier. Comme on peut le constater dans le `if()`, il a été codé en dur pour un échiquier 3x3.

```
def parcourir(case):  
    mouv_possible = []  
    for m in mouv :  
        x = case[0] + m  
        if (x > 10 and x < 14) or (x > 20 and x < 24) or (x > 30 and x < 34):  
            mouv_possible.append(x)  
    return mouv_possible
```

FIGURE 9 – La fonction `parcourir()`

generate_graph() La fonction `generate_graph()` nous génère le graphe sous forme de liste d'arête, on calcule les coups possible pour chaque case grâce à la fonction `parcourir()`

```
def generate_graph(f):  
    g = []  
    for case in f:  
        for node in parcourir(case):  
            g.append((case[0],node))  
    return g
```

FIGURE 10 – La fonction `generate_graph()`

Talal s'est occupé de la création de ces deux fonctions, tandis que Ruben s'est occupé de la création du graphe. Nous avons tous les deux participé à la rédaction du LaTeX.

6 Difficultés rencontrées

Le premier problème qu'on a rencontré était avec le fichier **Cell.txt** dans Google Colaboratory. En effet, pour quelque raison, le fichier se supprimait automatiquement lorsqu'on fermait l'onglet et qu'il n'y avait plus aucune session active sur le projet. On a donc dû recréer ou importer le fichier à chaque session de travail.

Le deuxième problème était les tours. En effet, notre graphique fait bien l'échange des pièces mais il ne respecte pas l'ordre du jeu. On a voulu implémenter l'algorithme **A*** pour faire respecter les tours mais on n'a pas réussi à le faire fonctionner.

Le troisième problème était l'implémentation de l'algorithme de **Dijkstra** pour trouver le chemin le plus court. Le code source des deux algorithmes sont présents en commentaire sur le projet mais on n'a pas réussi à les faire fonctionner.

On a fini par trouver l'itinéraire le plus court, mais on ne s'est pas servi de ces deux algorithmes pour s'y faire.

7 Bilan

7.1 Conclusion

Le code nous donne donc le graphe si dessous :

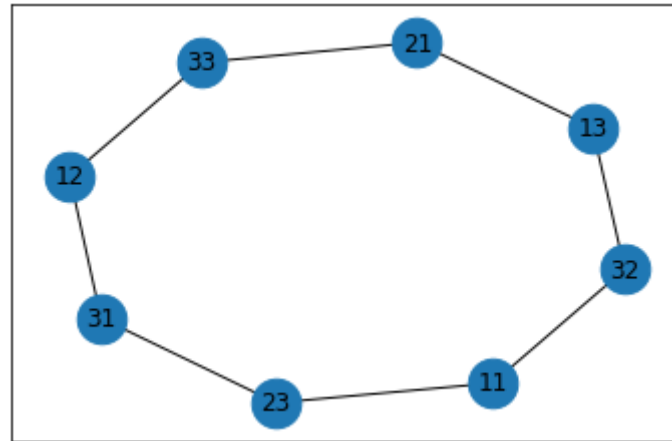


FIGURE 11 – La réponse au problème des 4 cavaliers, générée par notre code.

En effet, il correspond à une boucle, on voit qu'en répétant cette boucle 4 fois, on finit avec un inversement des positions.

On voit qu'il y'a 8 sommets sur le graphique or un déplacement correspond à 2 sommets, car on déplace le cavalier concerné d'un point à l'autre, et on avance ensuite vers le cavalier d'après. Cela fait réellement 4 déplacements itération, **soit 16 déplacements pour inverser les positions.**

Pour conclure, on a trouvé que le nombre de déplacements minimale pour déplacer la position des cavaliers était de 16.

L'inversement est donc possible, et voici son fonctionnement :

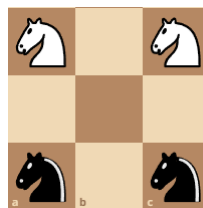


FIGURE 12 – Les cavaliers à leur initialisation, les cavaliers blancs en (11, 13) et les cavaliers noirs en (31,33)

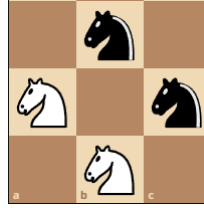


FIGURE 13 – La position des cavaliers après un tour

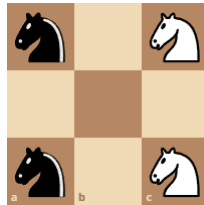


FIGURE 14 – La position des cavaliers après deux tours

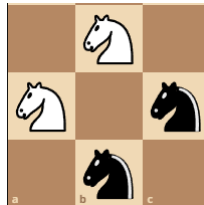


FIGURE 15 – La position des cavaliers après 3 tours

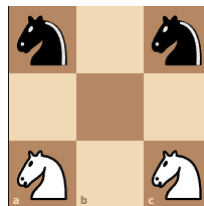


FIGURE 16 – La position des cavaliers après 4 tours : l'inversement a été effectué

8 Bibliographie

[label] Auteur, TITRE, editeur, annee

[LAM94] L. LAMPORT, *TEX : A Document preparation system*, Addison-Wesley, 1994

9 Webographie

[HISTOIRE DES ECHECS] https://fr.wikipedia.org/wiki/Histoire_du_jeu_d%27%C3%A9checs#:~:text=Le%20lieu%20pr%C3%A9cis%20est%20toujours,qui%20pr%C3%A9pare%20sa%20forme%20moderne.

[WARNSDORFF] <https://www.geeksforgeeks.org/warnsdorffs-algorithm-knights-tour-pr>

[8 DAMES] https://www.researchgate.net/figure/a-A-solution-to-the-non-attacking-8-fig1_278681097

[FOUR KNIGHTS] <https://mindyourdecisions.com/blog/2014/03/17/monday-puzzle-four-knights/#:~:text=In%20order%20for%20the%20knights,the%20minimum%20answer%20is%2016.>

10 Annexes

Annexe A : Cahier des charges

Annexe B : Exemple d'exécution du projet

Annexe C : Manuel utilisateur

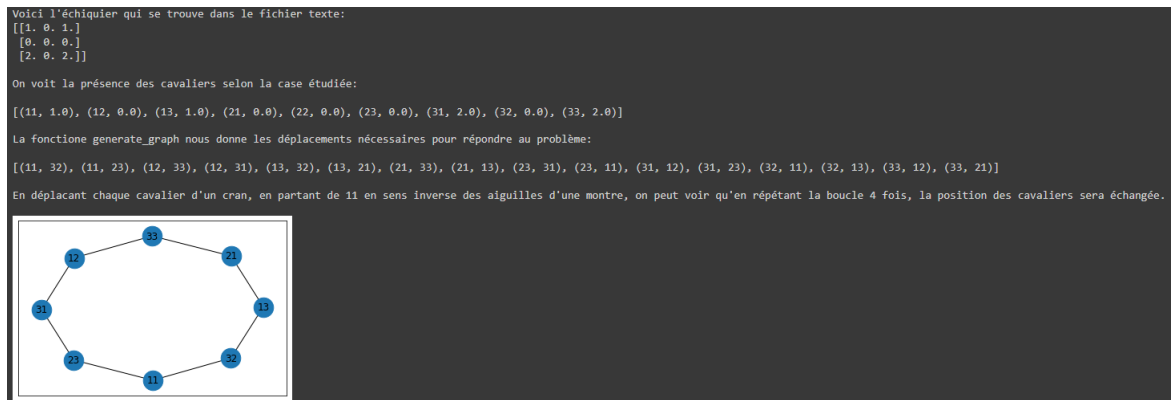


FIGURE 17 – L’affichage du code

Une fois le code exécuté, ceci est son affichage.

Vous trouverez le code entier ci-dessous. Vous trouverez ci-joint un fichier Cell.txt qu’il faudra garder dans le même dossier que le code source (qui est également fourni). Afin de faciliter l’exécution du code, nous vous conseillons d’utiliser Google Colaboratory.

Je vous remercie d’avoir lu notre rendu de projet, bonne journée.

```

import numpy as np
import networkx as nx
#from collections import deque

mouv = [21,12,19,8,-21,-12,-19,-8]

def parcourir(case):
    mouv_possible = []
    for m in mouv :
        x = case[0] + m
        if (x > 10 and x < 14) or (x > 20 and x < 24) or (x > 30 and x < 34):
            mouv_possible.append(x)
    return mouv_possible

def generate_graph(f):
    g = []
    for case in f:
        for node in parcourir(case):
            g.append((case[0], node))
    return g

print("Voici l'ichier qui se trouve dans le fichier texte:")
arr = np.loadtxt("Cell.txt", delimiter = ",")
print(arr)

print("\nOn voit la pr sence des cavaliers selon la case tudie :\n")
pre_g = []
i = 11
for line in arr:
    for c in line:
        pre_g.append((i, c))
        i += 1
    i = i + 7
print(pre_g)

print("\nLa fonctione generate_graph nous donne les d placements n cess")
post_g = generate_graph(pre_g)
print(post_g)

G= nx.Graph()
G.add_edges_from(post_g)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, node_size=600)
nx.draw_networkx_edges(G, pos, edgelist=G.edges(), edge_color = 'black')
nx.draw_networkx_labels(G, pos)

print("\nEn deplacant chaque cavalier d'un cran, en partant de 11 dans un

```