Optimisation des coûts de transport d'un voyage en Europe en utilisant la théorie des graphes

1 Introduction

Une des utilisations principales des mathématiques est d'étudier des phénomènes pour pouvoir optimiser le résultat au maximum. Ces optimisations peuvent être utilisées pour trouver la surface minimale pour stocker un tiers de litre de soda, ou bien pour trouver la capacité maximale d'une production avec un certain budget. Ce que j'ai trouvé intéressant d'optimiser, c'est le coût de transport pour aller en vacances dans des villes Européennes. Car après tout, si je connais comment optimiser un voyage, je pourrais personnaliser le trajet pris pour réduire considérablement le coût des transports et donc avoir plus d'argent pour faire du tourisme.

J'ai trouvé qu'il existe une branche dans les mathématiques discrètes, la théorie des graphes, qui me permet de modéliser mes destinations sous forme d'un graphe, ce qui me permettra de trouver le trajet le moins cher qui passe par toutes les villes que je veux visiter. En étant un élève qui étudie l'option analyse au BI, j'ai trouvé ce problème très intéressant à explorer. En plus, je suis intéressé par l'informatique et ce type de problème est très connue et étudié en informatique, car mon problème est un exemple d'un problème beaucoup plus intéressant et abstrait.

Je voudrais donc trouver le trajet le moins cher en avion passant par les 12 villes suivantes : **A**thènes, **Be**rlin, **Bu**dapest, **H**elsinki, **L**ondres, **M**adrid, **P**aris, **R**ome, **So**fia, **St**ockholm, **Va**rsovie et **Vi**enne. Je commencerai également à Athènes car c'est la destination la plus proche pour moi, un habitant du Caire.

2 Présentation du problème

2.1 Modélisation du problème

Cette situation peut être parfaitement modélisée avec la théorie des graphes, avec les villes comme sommets et le coût de transport entre eux comme les arêtes. Je modéliserai cette situation avec le graphe simple complet de 12 sommets mais pondéré. Un graphe complet est un graphe dans lequel il existe une arête entre chaque paire de sommets. Afin de rendre le problème plus simple, je considérerai que le graphe est non-orienté ; le prix d'aller de ville A à ville B est le même pour aller de ville B à ville A. Je prendrai donc la moyenne des deux prix, on parle alors d'un graphe

symétrique. Le prix pourra également varier de jour en jour donc je collecterai mes données pour les prix du ticket du 1er juillet 2020.

Voici le tableau des coûts de transports par avion [1] en Euros :

	A	Be	Bu	H	L	M	P	R	So	St	Va	Vi
A	0	93,5	125	125	92	99,5	88,5	49,5	59	107,5	80	81,5
Be	93,5	0	39,5	63	56	80	41,5	49,5	147	37,5	106,5	40
Bu	125	39,5	0	126,5	49,5	108,5	79	55	68	89,5	95,5	87,5
H	125	63	126,5	0	97	155,5	100,5	144,5	159,5	77	99	94,5
L	92	56	49,5	97	0	53	36	62	65,5	74,5	97,5	68
M	99,5	80	108,5	155,5	53	0	63	53,5	147,5	121	121,5	118,5
P	88,5	41,5	79	100,5	36	63	0	55	176,5	86	94	53
R	49,5	49,5	55	144,5	62	53,5	55	0	76	121,5	78	106
So	59	147	68	159,5	65,5	147,5	176,5	76	0	182,5	154	123,5
St	107,5	37,5	89,5	77	74,5	121	86	121,5	182,5	0	108,5	89,5
Va	80	106,5	95,5	99	97,5	121,5	94	78	154	108,5	0	84,5
Vi	81,5	40	87,5	94,5	68	118,5	53	106	123,5	89,5	84,5	0

Tableau 1 – Prix de transport entre chaque ville

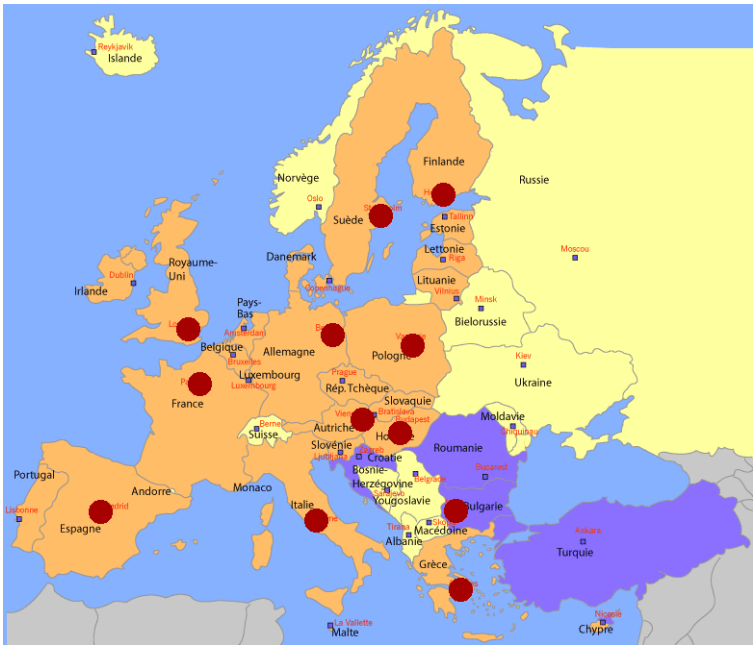

FIGURE 1 – Carte d'Europe avec les capitales que je voudrais visiter marquées en rouge [3]

1. COLLECTIF, « Expedia Travel ». -[Consulté le 11 janvier 2020]. Disponible sur : https://www.expedia.com/

3. COLLECTIF, « La Historia ». -[Consulté le 20 janvier 2020]. Disponible sur : https://www.lahistoriaconmapas.com/atlas/carte-capitale/carte-capitale-pays-union-europ%c3%a9enne.htm

2.2 Description du problème

Ce problème est en effet un exemple d'un problème très recherché en théorie des graphes et dans l'informatique, le problème du voyageur de commerce. Le but du problème est de trouver le cycle, une suite de sommets adjacentes qui ont le même sommet de départ et d'arrivée, qui coûte le moins et qui passe par tous les sommets une seule fois. Autrement dit, on cherche le cycle Hamiltonien le moins lourd dans un graphe. Ce cycle peut ne pas exister mais mon graphe est complet alors je suis sûr qu'il existe.

Avant de parler du problème du voyageur de commerce, il faut parler du nombre d'opérations maximum que l'ordinateur doit faire afin de résoudre le problème. Ce nombre est utilisé afin de comparer et évaluer les algorithmes. On utilise la notation $O(f(n))$ avec $f(n)$ comme variable discrète pour voir la croissance du nombre d'opérations maximum (et par la suite, le temps maximum pris pour résoudre le problème) quand $n \to \infty$. On peut également dire que c'est la complexité en temps. On prend en considération seulement du monôme le plus grand sans son coefficient. On prend également en considération les autres fonctions (exponentielles, logarithmiques, etc.) ; l'élément essentiel pour comprendre le comportement d'une fonction quand on augmente n.[4] On dit alors qu'un problème est résolu dans un temps polynomial quand $O(n^k)$ pour $k \in \mathbb{Z}^+$. Quand un problème est résolu dans un temps polynomial ou moins (logarithmique), il est considéré comme un problème qui peut être rapidement résolu.[5]

Ce qui intéressant dans ce problème est le fait que c'est classifié comme un problème NP-difficile ; il n'existe pas un algorithme qui permet de résoudre le problème dans un temps polynomial, ni un algorithme qui permet de vérifier la solution dans un temps polynomial. La méthode la plus directe de résoudre ce problème est de calculer le poids de tous les cycles Hamiltoniens. Pour un graphe complet de n sommets, il y a $\frac{1}{2}(n-1)!$ cycles différents et donc le problème croit d'une manière factorielle, $O(n!)$.

Pour un graphe qui est relativement petit, de 12 sommets, ce problème peut être résolu dans un temps raisonnable avec un ordinateur. Cependant, si je veux visiter plus de villes, à un point le temps de résolution ne sera plus raisonnable. A 12 sommets, il y a environ 20 millions de cycles Hamiltoniens mais à 20 sommets il y a environ $1{,}22 \times$

4. BELL Rob, « A beginner's guide to Big O notation ». -[Consulté le 26 janvier 2020]. Disponible sur : https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

5. David Terr, « Polynomial Time ». -[Consulté le 26 janvier 2020]. Disponible sur : http://mathworld.wolfram.com/PolynomialTime.html

10^{18}, qui est 10^{11} fois plus grand. En revanche, on peut encadrer le coût optimal entre 2 valeurs en trouvant une borne inférieure et une borne supérieure. Si cet encadrement est petit, alors j'aurais plus ou moins résolu le problème ; je serai sûr que je suis très proche du trajet optimal.

3 Résolution du problème

Pour avoir un encadrement pertinent, il doit être le plus petit possible. Autrement dit, il faut trouver la borne inférieure la plus grande et la borne supérieure la plus petite. Évidemment, les méthodes utilisées pour trouver ces bornes doivent avoir au maximum un temps polynomial, sinon il n'est pas pertinent.

3.1 Trouver la borne inférieure

Une borne inférieure peut être trouvée en trouvant l'arbre couvrant de poids minimal (ACM) du graphe sans prendre compte d'un des sommets, ensuite en incluant les arêtes les moins lourds qui relient avec le sommet ignoré au début. Un arbre est un graphe connexe, un graphe dans lequel on peut aller de n'importe quel sommet vers tous les autres sommets à travers des arêtes, où il n'y pas de cycle. Cette méthode donc trouve les n arêtes les plus légères pour n sommets ; l'ACM est l'arbre dans lequel la somme de ses arêtes est la plus petite possible. Si ce graphe forme un cycle Hamiltonien, alors le problème est résolu.

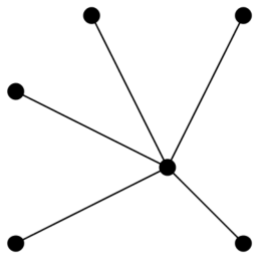

FIGURE 2 – Exemple d'un arbre à 6 sommets

Je commence donc par enlever un sommet quelconque, Athènes. L'étape suivante est de trouver l'ACM. Il existe plusieurs algorithmes pour trouver cet arbre. Il est essentiel également de prendre en considération qu'il peut y avoir plusieurs arbres mais ils ont tous le même poids quoi qu'il en soit. Les deux algorithmes que j'ai trouvés sont l'algorithme de Prim et de Kruskal. La différence entre les deux est que celui

de Kruskal crée l'arbre en cherchant les arêtes les plus légères ; il faudra donc utiliser un autre algorithme pour vérifier qu'un cycle ne se crée pas. Pourtant il est plus rapide pour faire à la main que celui de Prim. Quant à l'algorithme de Prim, il crée l'arbre en prenant compte déjà des sommets visités et donc il n'a pas besoin d'un algorithme supplémentaire pour vérifier qu'il n'y a pas de cycles créés. J'utiliserai donc l'algorithme de Prim, car mon graphe est plus dense ; c'est un graphe complet. [6]

L'algorithme de Prim commence par choisir un sommet quelconque que je considérerai comme le premier sommet de mon arbre, Berlin et je l'ajoute dans une liste des arêtes visitées. Ensuite je considère toutes les arêtes communes avec Be et je choisi le plus léger, BeSt et j'ajoute St dans la liste des sommets visités. Ensuite je considère toutes les arêtes communes avec les sommets qui sont dans ma liste et je choisi le plus léger, à condition que l'autre sommet ne soit pas dans ma liste des sommets visités. J'ajoute alors BeBu et j'ajoute Bu dans la liste. Je continue jusqu'à avoir visité tous les sommets, donc à $n-2$ arêtes de mon graphe (j'ai $n-1$ villes car j'ai déjà enlevé Athènes au début). [7]

J'arrive donc à ce graphe avec un poids de 503,5 € :

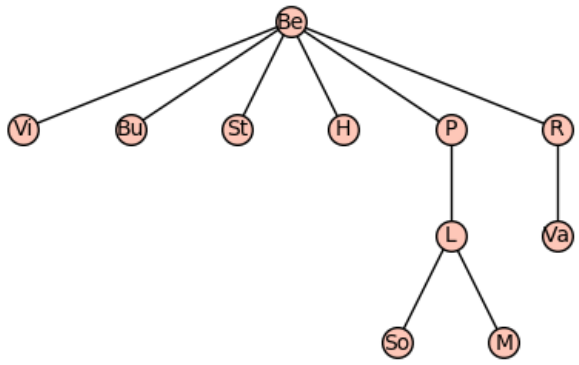

FIGURE 3 – ACM de mon graphe sans A

Il me reste seulement d'ajouter les deux arêtes les plus légères qui sont communes avec A, AR et ASo. Ceci me donne donc un prix total de 612 €. Je pourrais également faire ceci en ignorant d'autres sommets afin d'avoir plusieurs résultats et donc d'autres bornes inférieures.

La beauté de cette méthode (et les algorithmes en général) est que je peux

6. Akshay Singhal, « Difference Between Prim's and Kruskal's Algorithm » -[Consulté le 31 janvier 2020]. Disponible sur : https://www.gatevidyalay.com/prims-and-kruskal-algorithm-difference/

7. COLLECTIF, « Prim's Spanning Tree Algorithm ». -[Consulté le 29 janvier 2020]. Disponible sur : https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm

utiliser la technologie afin d'effectuer tous ces calculs. J'utiliserai donc « Sage Math », un logiciel avec lequel je peux manipuler les graphes (voir annexe 1).

Sommet ignoré	Borne inférieure	Sommet ignoré	Borne inférieure
A	612	P	596
Be	660	R	614
Bu	596	So	612
H	623,5	St	621
L	596,5	Va	626,5
M	600	Vi	599,5

Tableau 2 – Valeurs différentes des bornes inférieures

Ma borne inférieure la plus grande est alors 660. Je doit donc prévoir au moins 660 € comme coûts de transport.

3.2 Trouver la borne supérieure

Je peux également trouver une borne supérieur en utilisant un algorithme. Une borne supérieure possible est le double du poids de l'ACM du graphe. L'ACM a un poids de 546,5 donc une borne supérieure est 1093 €.

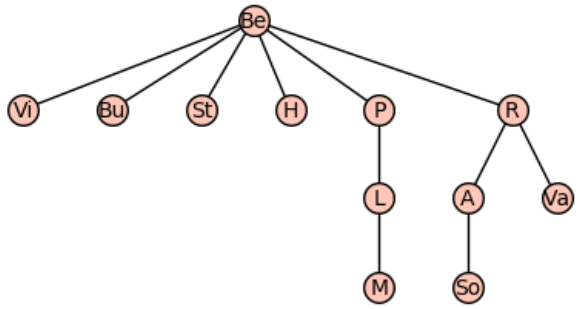

FIGURE 4 – ACM de mon graphe

Pourtant, cette borne n'est pas vraiment pertinente. Le sens réel de cette borne est que je reviens sur mes pas, ce qui n'est pas très pratique dans la vie réelle ; ceci implique que je prendrai $2n - 2$, 22 pour 12 villes, vols.

Il y a également une autre méthode pour trouver une meilleure borne supérieure. Cette méthode consiste à trouver des cycles Hamiltoniens en utilisant un algorithme glouton, un algorithme qui choisi le choix local le plus optimal. L'algorithme de Prim et celui de Kruskal sont également considérés comme des algorithmes gloutons.

L'algorithme pour trouver une borne supérieure est de choisir un sommet arbitrairement, A et l'ajouter à la liste des sommets visités. Ensuite je cherche le sommet le plus proche, R, je note l'arête AR et j'ajoute R dans la liste des sommets visités. Ensuite je cherche le sommet le plus proche de R qui n'est pas dans la liste des sommets visités, Be, je note l'arête RBe et j'ajoute Be dans la liste des sommets visités. Je continue ce processus jusqu'à avoir visité tous les sommets ensuite j'ajoute l'arête du dernier sommets à A, HA dans cette situation. Je trouve alors ce cycle Hamiltonien avec un poids de 976. [8]

$$A \to R \to Be \to St \to L \to P \to Vi \to Va \to Bu \to So \to M \to H \to A$$

Je peux commencer également avec des différents sommets pour avoir des bornes supérieurs différentes.

Sommet choisi	Borne supérieure	Sommet choisi	Borne supérieure
A	976	P	789
Be	878	R	809,5
Bu	873	So	962,5
H	859	St	962,5
L	899	Va	866,5
M	869	Vi	888

Tableau 3 – Valeurs différentes des bornes supérieures

Le cycle Hamiltonien le plus léger que j'ai trouvé avec un poids de 789 est

$$P \to L \to Bu \to Be \to St \to H \to Vi \to A \to R \to So \to Va \to P$$

Puisque j'ai trouvé un cycle qui coûte 789 €, je suis sûr que paierai en plus 789 €. Par la suite, je sais que le trajet le moins cher coûte entre 660 et 789 € (une différence de 129 €).

8. Paul Fannon,Vesna Kadelburg, Ben Woolley et Stephen Ward. *Mathematics Higher Level Topic 10-Option : Discrete Mathematics for the IB Diploma*. Cambridge : Cambridge University Press, 2013. Chap 7.G « Visiting all the vertices : Travelling salesman problem », p.118-122.

4 Évaluation

4.1 La valeur exacte

Car il y a seulement 12 sommets, j'ai pu trouver la solution en utilisant Sage Math (voir annexe 2), qui coûte 709€.

$$A \to So \to Bu \to Be \to St \to H \to Va \to Vi \to P \to L \to M \to R \to A$$

Je vois qu'il y a des arêtes communes avec les bornes supérieures que j'ai trouvées, notamment BeSt et PL. Je pourrai donc commencer à Athènes ensuite se diriger vers n'importe quelle direction dans le cycle. Je vois également que la solution exacte est en effet entre les bornes que j'ai choisi et elle n'est pas relativement loin d'eux.

Pourtant, Sage Math n'a pas résolu le problème en essayant tous les cycles Hamiltoniens possibles ; il y existe d'autres algorithmes qui résolvent le problème, même si la complexité en temps est supérieur à un temps polynomial. Le problème du voyageur du commerce peut être résolu avec des différents approches en informatique. Un approche très connu pour résoudre ce problème avec une complexité en temps de $O(n^2 2^n)$ est l'algorithme de Held-Karp, un algorithme qui fait partie d'une branche d'informatique nommée l'informatique dynamique. Cet algorithme nous permet de résoudre le problème avec un peu plus de sommets. [9]

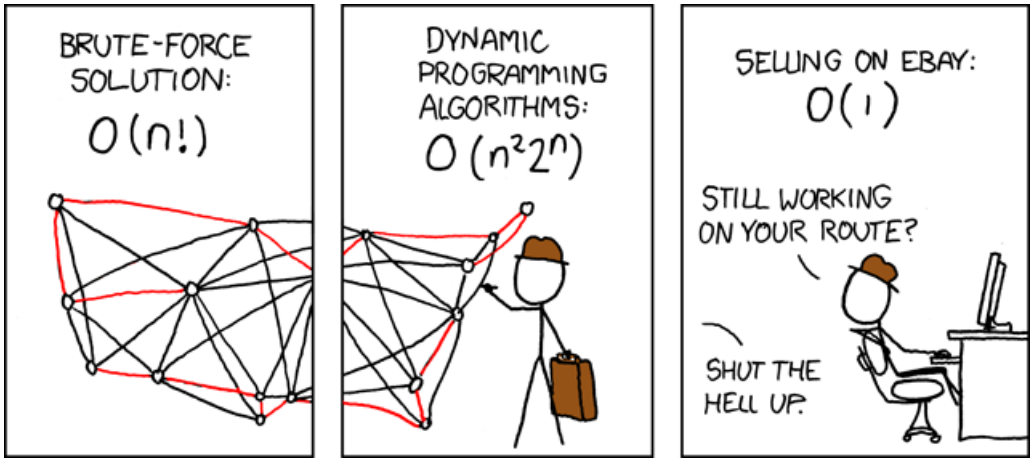

FIGURE 5 – Un opinion pertinent sur le problème du voyageur de commerce [10]

9. Richard Bellman. *Dynamic programming treatment of the travelling salesman problem* Journal of Assoc. Computing Mach. 9. 1962.

10. Randall Munroe, « xkcd ». -[Consulté le 31 janvier 2020]. Disponible sur https://www.xkcd.com/399/

4.2 Conclusion

J'ai donc trouvé le cycle Hamiltonien le plus léger pour mon trajet, et même si le nombre de villes n'étais pas grand, mon encadrement était pertinent. En effet, dans beaucoup de situations avec des milliers de sommets, les informaticiens cherchent seulement un encadrement pour lequel la différence entre les bornes est relativement petit pour être sûr que leur valeur supérieur est assez proche de la valeur optimale. Je pourrais entrer plus en détails en essayant de résoudre le problème de voyageur de commerce avec un graphe asymétrique pour savoir laquelle des directions coûte le moins et donc optimiser encore plus. Cependant, ceci prendra plus de temps et le temps de résolution de ce problème est encore plus grand. Cette méthode nécessitera également d'autres techniques pour résoudre le problème du voyageur de commerce.

5 Bibliographie

BELL Rob, « A beginner's guide to Big O notation ». -[Consulté le 26 janvier 2020]. Disponible sur : https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

BELLMAN Richard. *Dynamic programming treatment of the travelling salesman problem* Journal of Assoc. Computing Mach. 9. 1962.

COLLECTIF, « Expedia Travel ». -[Consulté le 11 janvier 2020]. Disponible sur : https://www.expedia.com/

COLLECTIF, « Prim's Spanning Tree Algorithm ». -[Consulté le 29 janvier 2020]. Disponible sur https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm

COLLECTIF, « La Historia ». -[Consulté le 20 janvier 2020]. Disponible sur : https://www.lahistoriaconmapas.com/atlas/carte-capitale/carte-capitale-pays-union-europ%c3%a9enne.htm

FANNON Paul, KADELBURG Vesna, WOOLLEY Ben et WARD Stephen. *Mathematics Higher Level Topic 10-Option : Discrete Mathematics for the IB Diploma*. Cambridge : Cambridge University Press, 2013. Chap 7.G « Visiting all the vertices : Travelling salesman problem », p.118-122.

MUNROE Randall, « xkcd ». -[Consulté le 31 janvier 2020]. Disponible sur https://www.xkcd.com/399/

TERR David, « Polynomial Time ». -[Consulté le 26 janvier 2020]. Disponible sur : http://mathworld.wolfram.com/PolynomialTime.html

6 Annexe 1

Code pour trouver la borne inférieure :

```
M = Matrix([(0,93.5,125,125,92,99.5,88.5,49.5,59,107.5,80,81.5),
...
(81.5,40,87.5,94.5,68,118.5,53,106,123.5,89.5,84.5,0)])
cities = ["A", "Be", "Bu", "H", "L", "M", "P", "R", "So", "St", "Va", "Vi"]
for x in cities:
    G = Graph(M, format='weighted_adjacency_matrix');
    G.relabel({0:'A', 1:'Be', 2:'Bu', 3:'H', 4:'L', 5:'M', 6:'P',
    7:'R', 8:'So', 9:'St', 10:'Va', 11:'Vi'})
    first_city= list()
    for y in G.edges_incident(x):
        first_city.append(y[2])
    first_city.sort()
    G.delete_vertex(x);
    G = Graph(G.min_spanning_tree(by_weight=True));
    print(sum(G.edge_labels())+first_city[0]+first_city[1])
```

7 Annexe 2

Code pour résoudre le problème de voyageur du commerce :

```
M = Matrix([(0,93.5,125,125,92,99.5,88.5,49.5,59,107.5,80,81.5),
...
(81.5,40,87.5,94.5,68,118.5,53,106,123.5,89.5,84.5,0)])
G = Graph(M, format='weighted_adjacency_matrix');
    G.relabel({0:'A', 1:'Be', 2:'Bu', 3:'H', 4:'L', 5:'M', 6:'P',
    7:'R', 8:'So', 9:'St', 10:'Va', 11:'Vi'})
tsp = G.traveling_salesman_problem(use_edge_labels=True)
print(sum(tsp.edge_labels()))
tsp.show();
```