Travaux de recherche en informatique

Présentée par **:**

Hajar BOUZIANE

Soutenu le **: 22 janvier 2021**

Formation **: Master1 Informatique**

Établissement **: Université Sorbonne Paris Nord**

Calcul d'un itinéraire destiné à visité tous les monuments nationaux de France

Mots clé : Kruskal, Prim, Boruvka, arbre couvrant de poids minimum, itinéraire.

SOMMAIRE

RESUME

Contexte : lors de ce travail de recherche, mon activité principale consistait à créer un algorithme qui permettrait de calculer la distance du plus petit itinéraire reliant tous les monuments nationaux de France. Pour répondre à ce type de problème il a fallu utiliser la méthode d'arbre courant de poids minimum pour calculer le plus court chemin depuis un monument national vers tous les autres. Mon étude s'est donc portée sur la comparaison du temps de calcul de ma solution et des algorithmes de Kruskal, de Prim et de Boruvka qui génèrent des arbres couvrants de poids minimum. Notons que ces trois derniers comptent parmi ceux qui sont les plus utilisés de nos jours car ils répondent efficacement à ce type de problème. Mais il s'agit d'algorithmes relativement complexes et qui soulignent l'importance de l'utilisation d'une ou plusieurs structures de données adaptées. C'est pourquoi il serait intéressant de voir s'il existe un moyen de créer un autre algorithme plus performant en temps de calcul.

Objectif et méthode : L'objectif de ce travail était donc de m'inspirer des algorithmes précédemment énumérés pour implémenter une hybridation et où son temps de calcul sera le plus faible. J'ai donc mené une série d'expériences afin d'évaluer et de comparer les performances de ma solution à celles existantes. J'y ai observé le temps de calcul de chacun, la distance des itinéraires qui ont été générés ainsi que leur représentation graphique.

Résultats : Les résultats ont montré que qu'elles que soit les algorithmes implémentés, il existait une connexion entre deux monuments qui ne devrait pas exister sur l'arbre couvrant. En effet, lorsqu'on visualise l'itinéraire sur une carte, on peut voir que la distance entre ces deux monuments et trop grande et qu'il pourrait exister des chemins plus courts pour y aller. Cependant, je n'ai pas su résoudre ce problème. En revanche, malgré plusieurs essais, l'algorithme de Boruvka est le plus rapide en temps de calcul. Certes celui de mon algorithme est plus faible que celui des algorithmes de Kruskal et de Prim mais il est tout de même plus élevé que celui de Boruvka.

Conclusion : Les résultats suggèrent donc l'importance de poursuivre les recherches afin de créer un algorithme qui proposerait un plus court itinéraire et qui soit plus rapide que celui de Boruvka.

INTRODUCTION

Outre pour le plaisir des yeux, visiter des monuments historiques chargés d'histoire est l'occasion de découvrir la culture d'un pays. Ils ont traversé les âges pour qu'on puisse les contempler et font partie des lieux incontournable à visiter lors d'un voyage longuement préparé. Cependant, dans le cadre de la crise sanitaire liée à la COVID-19 qui dure depuis près d'un an, des restrictions sur les déplacements ont été mis en place. Des confinements ainsi que des couvre-feux ont été imposées et des frontières ont été fermées. Les voyages sont donc devenus un loisir difficile à réaliser surtout en ce qui concerne le choix d'un itinéraire dont nous voulons qu'il soit le plus court. Le problème est donc le suivant : dans le cadre d'un voyage qui consiste à visiter tous les monuments de France, qu'elle serait le plus court itinéraire ? Comment peut-on calculer la plus courte distance reliant tous ces monuments ?

Pour répondre à ce problème, il existe de nombreuses solutions qui sont mis à notre dispositions comme les GPS (Global Position System). Cependant, on doit y renseigner un lieu de départ et d'arriver. Dans ce cas, ils se peut que l'itinéraire qui nous soit proposé ne passe pas par tous les monuments de France. En revanche, il existe des algorithmes qui permettent d'y remédier comme l'algorithme Boruvka (1926), l'algorithme de Kruskal (1956) ou encore l'algorithme de Prim (1957). Il s'agit d'algorithme complexe mais très efficace et qui calcule un arbre couvrant de poids minimum et où les nœuds sont des monuments historiques, les arêtes sont les connexions entre ces monuments et où les poids sont associés aux distances (en kilomètre) de chaque connexion. Ainsi, les principaux objectifs de mon travail de recherche sont qu'à partir d'une base de données qui stocke les coordonnées GPS de tous les monuments nationaux de France et des algorithmes précédemment énuméré, créé un nouvel algorithme plus performant en temps de calcul.

Dans cet article, j'exposerais la démarche entamée qui m'a permis de mener à bien ce travail de recherche. Je débuterais par une analyse des revues de littérature. Ensuite, j'exposerais les différentes expériences que j'ai effectuée. Enfin, je présenterais mes résultats expérimentaux.

I. ETAT DE L'ART

a. Modélisation d'un problème sous forme de graphe

Dans sa thèse [1], Tristram GRÄBENER mentionne l'existence d'un article publié en 1736 par un mathématicien et physicien du XVIII^{ème} siècle nommé Léonhard EULER. Ce dernier a démontré qu'il était impossible de traverser qu'une seule fois les sept ponts de la ville Kaliningrad. Pour cela, il modélisa le plan de cette ville avec un graphe où les nœuds correspondaient à des parties de la ville et où les arêtes représentaient les sept ponts. Les expériences qu'il a menées sur ce problème ont conduit à la naissance de la théorie des graphes.

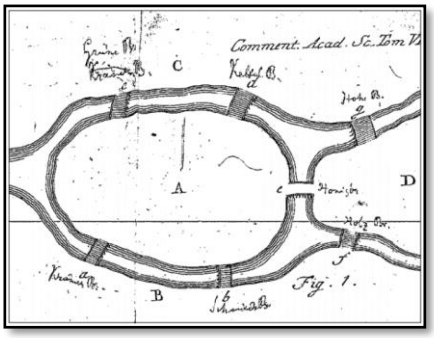

Figure 1 : Les sept ponts de Kaliningrad

En effet, un graphe $\mathcal{G}$ est une structure de données permettant de modéliser un réseau. Il est composé d'arêtes $\mathcal{A}$ correspondant à une relation entre deux objets (des nœuds $\mathcal{N}$) et qui sont étiquetées par un poids C. Ainsi un chemin est l'équivalent d'une suite d'arêtes et sa longueur est égale à la somme des poids de ces arêtes. Pour représenter informatiquement un tel graphe, on utilise une liste d'adjacence ou une matrice d'adjacence. On peut alors appliquer un ensemble d'algorithmes pour calculer un arbre couvrant de poids minimum sur ce graphe.

b. Arbres couvrants de poids minimum

Définition

Dans son mémoire [2], Edmond LA CHANCE explique en détails le principe d'un arbre couvrant de poids minimum. Ainsi, on en conclut qu'appliquer cette méthode sur un graphe $\mathcal{G}$ permet de générer un graphe qui relie tous ses nœuds sans produire de cycle et dont la somme des poids des arêtes serait la plus petite.

Caractéristiques des algorithmes

Suite à l'article publié par Léonhard EULER, de nombreux algorithme ont vu le jours avec chacun des caractéristiques qui leur sont spécifique :

- L'algorithme de Kruskal porte le nom de son créateur. Simple à implémenté, il est le plus utilisé pour calculer les arbres couvrant de poids minimum. Pour ce faire, l'algorithme trie les arêtes dans l'ordre croissant de leur poids. Elles sont ensuite traitées une par une, en commençant par celle de plus petit poids et en terminant par celle de plus grand poids. Le traitement d'une arête consiste à appliquer l'algorithme Union-Find sur ses extrémités. Si elles appartiennent au même ensemble, alors l'arête forme un cycle et ne sera donc pas ajouté à l'arbre couvrant.

  La complexité de cet algorithme est O(|A|*log|A|) où A est le nombre d'arêtes du graphe initial.
- L'algorithme de Prim. Dans cet algorithme, on crée un ensemble S de nœud initialement vide pour pouvoir y stocker des nœuds au fur et à mesure qu'ils soient visités. L'objectif de cet algorithme est de trouver l'arête de plus petits poids relient un nœud n_1 de l'ensemble S avec un nœud n_2 n'appartenant pas à cet ensemble, ajouté le n_2 à l'ensemble S et répété cette étape $|S|$-1 fois.

  Contrairement à l'algorithme de Kruskal, la complexité de cet algorithme est plus petite. En effet, elle est de O(|A| + |N|*log|N|) mais il est plus difficile à implémenter.
- L'algorithme de Boruvka dont le principe est le suivant : on sélectionne au hasard un nœud n et on ajoute à notre arbre couvrant l'arête de plus petit poids associé ce nœud. On répète cette étape autant de fois qu'il y a de nœud. L'algorithme de Boruvka est aussi compliqué à implémenté et ça complexité est identique à celle de l'algorithme de Prim. De plus, tout comme l'algorithme de Kruskal, Boruvka utilise les ensemble Union-Find pour détecter les arêtes qui forment un cycle.

  Finalement, la complexité de cet algorithme est identique à celle de Prim, mais Boruvka se révèle être le plus rapide.

Autres que ces algorithmes, il en existe un autre qui est très à implémenter mais dont la complexité est quasi linéaire. Il s'agit de l'algorithme de Bernard-Chazelle [3] qui se base sur la fonction inverse d'Ackermann.

II. METHODOLOGIE

a. Mise en place d'un environnement de travail

La création de mes scripts python et langage C ainsi que l'ensemble des expériences effectuées ont été réalisé sur une distribution linux (Ubuntu) installé dans une machine virtuelle (VirtualBox). Pour implémenter mes différents algorithmes, j'ai utilisé l'éditeur de texte Atom et pour visualiser mes graphes, j'ai importé les librairies matplotib, numpy et folium.

Notons que pour calculer un arbre couvrant de poids minimum sur l'ensemble des monuments nationaux de France, il a été nécessaire d'obtenir une base de données qui stocke les coordonnées GPS de ces monuments. J'ai pu obtenir ce type de données sur le site du data.gouv.fr [4] qui met à disposition un fichier téléchargeable qui liste le nom des monuments nationaux de France avec les valeurs de la latitude et de la longitude de leur coordonnées GPS. Ainsi, une fois téléchargé, j'ai pu débuter l'implémentation de mes algorithmes.

b. Application des algorithmes

Pour pouvoir manipuler les coordonnées GPS des monuments, il a fallu dans un premier temps implémenter des fonctions en langage C pour lire le fichier préalablement téléchargé pour pouvoir stocker les coordonnées dans une structure adaptée :

```c
typedef struct ListOfMonuments ListOfMonuments;

struct ListOfMonuments
{
  int nbMonument;
  char** name;
  float* lon;
  float* lat;
};
```

Figure 2 : Structure qui stocke le nombre de monuments, leur nom, leur longitude et leur latitude

De plus, comme on souhaite calculer un arbre couvrant de poids minimum, il a été nécessaire de créer un arbre complet à partir des cordonnées des monuments. Pour ce faire deux autres structures ont été ajouté. La première nommé *Edge* à servit à créer des arêtes : elles sont composées de deux monuments et de la distance qui les séparent. La seconde structure nommé *Graph* a été implémenté pour créer un graphe caractérisé par le nombre de monuments (nœuds), le nombre de connexions (arêtes) et enfin les arêtes la liste des arêtes.

```c
typedef struct Edge Edge;

struct Edge {
  int src, dest;
  float weight;
};
```

Figure 3 : Structure qui stocke des arêtes

```c
typedef struct Graph Graph;

struct Graph {
  int V, E;
  Edge* edge;
};
```

Figure 4 : Structure qui créé un graphe

La seconde étape achevée j'ai pu appliquer les algorithmes de Kuskal, Boruvka et Prim sur mon graphe complet, calculer la distance de l'arbre couvrant généré, le temps de calcul des algorithmes et enfin visualiser l'itinéraire sur une carte et sur une figure.

c. Implémentation de mon algorithme

Créer mon propre algorithme a été l'étape la plus difficile. J'ai dans un premier temps voulu en créer un qui proposait un plus petit itinéraire que celui proposé par les trois algorithmes. Mon objectif était donc de supprimer une connexion trop grande entre deux monuments que j'avais identifiés et qui apparaissaient dans les trois arbres couvrants. Pour ce faire, comme les arêtes sont créées dans l'ordre du fichier de coordonnées GSP, j'ai voulu créer une fonction qui mélangerait les monuments pour créer de nouvelles arêtes. Cependant, implémenté une telle fonction était difficile et n'ayant aucune garantit que cela fonctionnerait, j'ai utilisé les fonctionnalités de EXCEL pour mélanger les lignes du fichier (grâce à la fonction ALEA). J'ai répété l'expérience une vingtaine mais il n'y a eu aucun changement.

J'ai donc décidé d'améliorer les performances de l'algorithme de Kruskal car il est celui où la complexité est la plus élevée. J'ai identifié les points positifs de cet algorithme ainsi que ceux des deux autres pour pouvoir les fusionner et ainsi créer une hybridation :

- L'algorithme de Kuskal trie ces arêtes dans un premier temps. Cette étape permet de sélectionner d'abord les arêtes de plus petit poids et donc de créer plus rapidement un arbre couvrant.
- L'algorithme de Boruvka est le plus rapide grâce à sa méthode de traitement des arêtes.
- Les algorithmes de Kruskal et Boruvka utilisent tous les deux les ensembles Union-Find pour déterminer les arêtes qui forment des cycles.

En combinant ces caractéristiques, j'ai créé un nouvel algorithme qui commence par trier les arêtes d'un graphe complet. Puis au lieu de traité les arêtes une part une, il sélectionne un sommet et l'arête de plus petit poids qui lui est associée. Comme les arêtes sont préalablement triées, le recherchent de l'arête de plus petit se fait plus rapidement.

Une fois implémenté et exécuté, j'ai comparé avec les autres algorithmes son temps de calcule, la distance de l'arbre couvrant et la représentation graphique de ce dernier.

III. RESULTATS EXPERIMENTAUX

Temps de calcul et distance :

Après avoir répété l'expérience plusieurs fois, voici les résultats obtenus :

Expérience n°	Nom de l'algorithme	Temps de calcul (ns)	Distance (km)
1	Kruskal	1 340 751	11185.175781
	Boruvka	580 277	11185.175781
	Prim	2 436 476	11185.174805
	HbAlgo	1 087 275	11185.175781
2	Kruskal	767 414	11185.175781
	Boruvka	332 186	11185.175781
	Prim	1 268 559	11185.174805
	HbAlgo	501 156	11185.175781
3	Kruskal	2 400 452	11185.175781
	Boruvka	1 057 722	11185.175781
	Prim	4 156 137	11185.174805
	HbAlgo	1 562 760	11185.175781

Comme attendu, l'algorithme de Prim est le plus tandis que celui de Boruvka est plus rapide. En revanche, lorsqu'on observe le temps de calcul de mon algorithme, on remarque qu'il est plus faible que celui de Kruskal et Boruvka mais est supérieur à celui de Boruvka. Ainsi, l'algorithme que je propose et qui est une hybridation entre m'algorithme de Kruskal et de Boruvka est plus efficace que deux des algorithmes que nous avons l'habitude d'utiliser.

Visualisation de l'itinéraire :

En appliquant mon script python sur les arbres couvrant généré par les quatre algorithmes, on obtient les mêmes représentations graphiques :

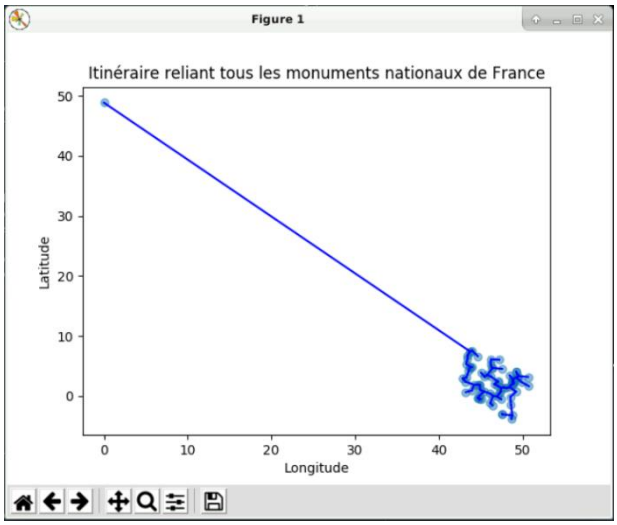

Figure 5 : Itinéraire tracé avec la librairie myplotlib

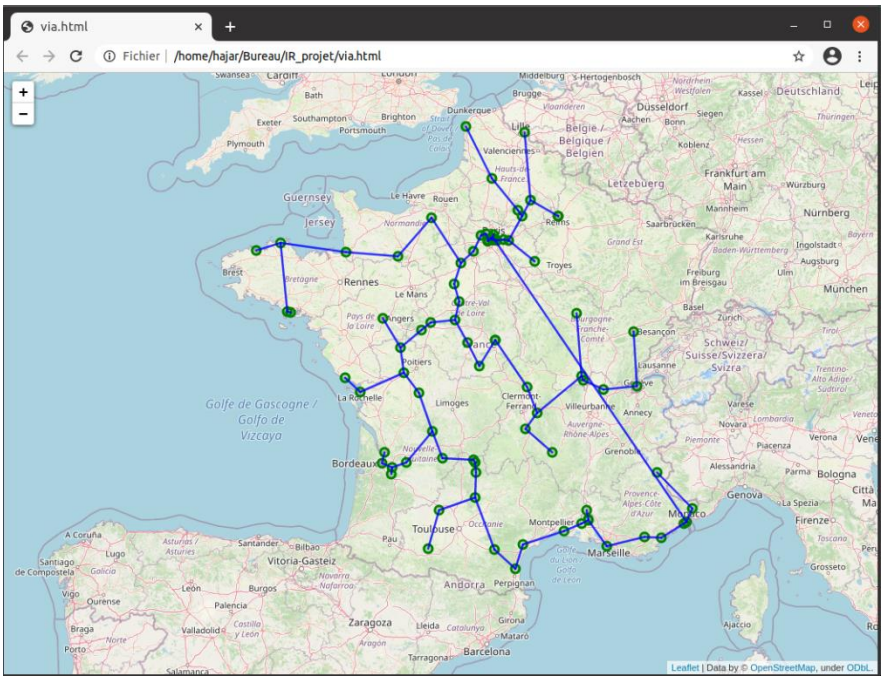

Figure 6 : Itinéraire tracé avec la librairie folium

On remarque sur ces graphes que la connexion entre deux monuments que je voulais supprimer existe toujours. Ceci peut s'expliquer par le fait que les arbres couvrants générés à partir d'un graphe complet sont unique saufs s'il existe des arêtes qui ont le même poids. Or comme ce n'est pas le cas dans cette situation, ça permet de justifier ces observations.

CONCLUSION

Contributions :

Calculer le plus court itinéraire reliant les monuments nationaux de France était le sujet principal de travail de recherche. L'objectif était de créer un algorithme dans le but de calculer un arbre couvrant de points minimum d'un graphe où les nœuds étaient des monuments (caractérisés par leurs coordonnées GPS) et où les arêtes représentaient les connexions entre ces nœuds.

Pour y parvenir, construire l'état de l'art a permis de déterminer trois algorithmes très utilisés de nos jours et qui peuvent répondre efficacement à ce problème : l'algorithme de Kruskal, de Boruvka et de Prim. Par la suite, un protocole expérimental a été mis en place. Il a permis d'analyser les points positifs de ces algorithmes pour pouvoir les fusionner. Ceci a permis d'implémenter une nouvelle solution qui s'avère être une hybridation de l'algorithme de Kruskal et Boruvka. Enfin, une série d'expériences a été effectuées pour comparer les performances de ces algorithmes et les résultats ont montré que l'algorithme de Boruvka était le plus performant, mais qu'il était tout de même suivi de très par mon algorithme.

Perspectives :

Dans le cas d'une poursuite des recherches sur ce sujet, il serait intéressant de calculer un itinéraire qui prendrait en compte plusieurs critères. En effet, il serait intéressant de proposer une solution où l'utilisateur pourrait choisir un mode de transport (à vélo, à pied, en voiture, en métro, …). Prendre en compte le trafic, la condition météorologique serait aussi des fonctionnalités qui seraient intéressant d'implémenter.

De plus, nous avons mentionné l'algorithme de Bernard-chazelle qui est très performant plus que sa complexité est quasi-linéraire. Ainsi, dans le cas où on souhaiterait optimiser les performances d'un algorithme, se pencher sur celui de Bernard-Chazelle pourrait augmenter significativement ces performances.

REFERENCES BIBLIOGRAPHIQUES

[1] T. GRÄBENER, « Calcul d'itinéraire multimodal et multiobjectif en milieu urbain », thèse de doctorat, Université de Toulouse, Toulouse, France, 2010. [En ligne]. Disponible : *https://tel.archives-ouvertes.fr/tel-00553335/document* .

[2] E. LA CHANCE, « Algorithmes pour le problème de l'arbre couvrant minimal », mémoire, Université du Québec, Chicoutimi, Québec, 2014. [En ligne]. Disponible : *https://constellation.uqac.ca/2899/1/LaChance_uqac_0862N_10016.pdf* .

[3] B. CHAZELLE, « A minimum spanning tree algorithme with inverse-Ackermann type complexity», Jounal of the ACM, Vol. 47, No. 6, publié en 2000. [En ligne]. Disponible : *https://www.cs.princeton.edu/courses/archive/fall03/cs528/handouts/Minimum%20Spanning%20Tree-Inverse.pdf*

[4] data.gouv.fr, « liste-coordonnées-gps-des-monuments.csv», fichier .csv, publié en 2019. [En ligne]. Disponible : *https://static.data.gouv.fr/resources/liste-des-coordonnees-gps-des-monuments-nationaux/20190729-144937/liste-coordonnees-gps-des-monuments.csv*