École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2014 - 2015

Rapport Final de PFE

Tournées de véhicules électriques : une approche basée sur les graphes de proximité relative

Encadrants
Ameur SOUKHAL
ameur.soukhal@univ-tours.fr

Etudiants
Nasreddine FERGANI
nasreddine.fergani@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2014 - 2015

Version du 22 avril 2015

Table des matières

Table des figures

Liste des tableaux

Introduction

Le monde connait aujourd'hui une renaissance énergétique où les gouvernements admettent la nécessité d'adopter des plans d'exploitation des énergies renouvelables. L'impact et l'étendu de ces plans varient en fonction des types de l'économie du pays et/ou de la culture des structures et services concernées. Le premier service touché par cette transition est la distribution postale qui intègre des véhicules hybrides, ou à combustion électrique entièrement. En outre, la technologie actuelle ne permet pas une bonne rentabilité des véhicules électriques comparés aux véhicules à énergies fossiles (essence, diesel, gaz). D'où l'importance d'optimiser et de restructurer la chaîne logistique des véhicules électriques.

La logistique des véhicules électriques ajoute une difficulté supplémentaire aux problèmes classiques, et ce en imposant de nouvelles contraintes et des objectifs nouveaux (comme la minimisation de la consommation énergétique au lieu de la distance parcourue). Un certain nombre de problématiques de véhicules électriques ont reçu de l'attention de la part des académiques comme des industriels. Des problèmes comme l'acheminement et les tournées de livraison en minimisant la consommation électrique totale, localisation de stations de rechargement ...etc. Ces travaux exploitent souvent des outils de l'optimisation combinatoire, combinés parfois à des méthodes géométriques ou statistiques.

Dans ce travail, nous adaptons une approche heuristique pour le problème de tournées de véhicules électriques à notre schéma de groupement et de clustering en utilisant les graphes de proximité relative. L'idée consiste à grouper les sites de livraison ayant des propriétés communes (même zone géographique ou autres propriétés) en utilisant une méthode à deux phase ; une première phase construit un graphe de proximité relative capable d'exprimer la proximité entre ces sites, et puis d'appliquer un algorithme de détection de communautés dans ce graphes. L'acheminement dans chaque groupe sera calculé par une heuristique de a littérature.

Ce document est divisé en deux parties. Une première partie de revue de la littérature, et qui porte sur les problèmes de la logistique des tournées de véhicules électriques au premier chapitre. Le deuxième chapitre expose quelques notions de base sur notre structure de clustering support : les graphes de proximité relative. Le troisième chapitre est un survol de quelques algorithmes et techniques de détection de communautés dans les graphes (et qui peuvent être appliqués sur le graphe de proximité relative). La deuxième partie présente notre travail. Le quatrième chapitre est une contribution au problèmes de construction et de mise à jour des graphes de proximité, où nous améliorons l'approche de l'état de l'art avec des ordres de grandeurs. Le cinquième chapitre expose notre approche de clustering et les résultats expérimentaux. Le sixième chapitre propose quelques tests et conclusion sur une heuristique de la littérature pour le problème des tournées de véhicules électriques.

Tournées de véhicules électriques : État de l'art

Dans ce chapitre, nous allons recenser un certain nombre de problématiques des véhicules électriques et les solutions qui y sont proposées. Nous commençons par le problème de cheminement dans des réseaux routiers avec comme objectif la minimisation de la consommation énergétique, on traite par la suite les tournées de véhicules électriques et dans laquelle il est possible de faire intervenir des approches de clustering.

1.1 Problématiques de cheminement

La problématique de cheminement des véhicules électriques dans un réseau doit prendre en compte un certain nombre de propriétés comme la récupération, et les contraintes liés à la capacité de la batterie (déchargement+capacité maximale). De surcroit, l'objectif de cette problématique est de minimiser la consommation énergétique afin de maximiser la portée de croisière (cruising range, la distance pouvant être parcourue avec une certaine quantité de charge). Cela impose deux défis non conventionnels à considérer : la récupération permet de recharger la batterie (quand le véhicule est en descente), et la capacité de la batterie qui contraint la quantité d'énergie récupérable. On en conclut que la conception d'un modèle de consommation est primordiale. Le modèle de consommation le plus simple et le plus complet -à notre connaissance -sur un réseau est celui proposé par Einsuer et al.[1] et repris par Moritz et al.[2].

1.1.1 Modèle de la consommation énergétique dans un réseau

Considérons un réseau orienté $G = (V, E)$. Les intersections sont représentés par V et les segments de tronçons par les arcs $(u, v) \in E$. Il est possible de spécifier n'importe quel réseau en ajoutant des sommets intermédiaires sur les tronçons afin de s'assurer qu'à une arête correspond une ligne monotone (montée ou descente), ce qui permet de mesurer plus efficacement le coût de consommation correspondant à chaque arc. A chaque arc $a = (u, v)$ nous associons une fonction f_a qui en fonction de l'état actuel de la batterie $b(u)$ à l'extrémité u, retourne une valeur de consommation énergétique (mesurée en mWh) nécessaire pour traverser l'arc a. La fonction f_a prend en compte les contraintes liées à la batterie comme le niveau de charge maximum, ainsi que la quantité minimale d'énergie pour traverser a.

La famille de fonctions $\mathbb{F}$ est telle que $f : \mathbb{R} \cup \{-\infty\} \to \mathbb{R} \cup \{\infty\}$, où ∞ et $-\infty$ sont des valeurs spéciales pour traiter le cas d'insuffisance de charge. La fonction f doit respecter la propriété qu'un niveau inférieur de charge ne peut jamais améliorer la consommation, ie $x \leq y \Rightarrow x - f(x) \leq y - f(y)$.
Nous définissons les deux opérations suivantes sur l'espace $\mathbb{F}$:

- $link(f, g) = f + g \circ (\mathsf{Id} - f)$
- $merge(f, g) = \min(f, g)$

Il est a noter que le coût d'un chemin $P = (v_1, \ldots, v_k)$ est obtenu par l'application itérative de l'opération $link$ sur les fonctions f de consommation sur les arcs constituant ce chemin. Donc le coût d'un chemin $P = (v_1 = s, \ldots, v_k = t)$ avec comme charge initiale $b(s)$ est donné par $f_P(b(s))$. Finalement, P est dit optimal à charge initiale $b(s)$ si $f_P(b(s))$ est minimale sur tous les chemins d'extrémités $s - t$ avec

$b(s)$ comme charge de départ (l'opération $merge$).

Les fonctions f sont linéaires par morceaux. Soit l'arc $a = (u, v)$, et c_a le coût de consommation de l'arc a. Alors on distingue les cas suivants :

- Si c_a est positif, alors $f_a(b) = \infty$ si $b < c_a$ et $f_a(b) = c_a$ sinon.
- Si c_a négatif (récupération), alors $f_a(b) = \min(b - M, c_a)$ avec M la charge maximale de la batterie.

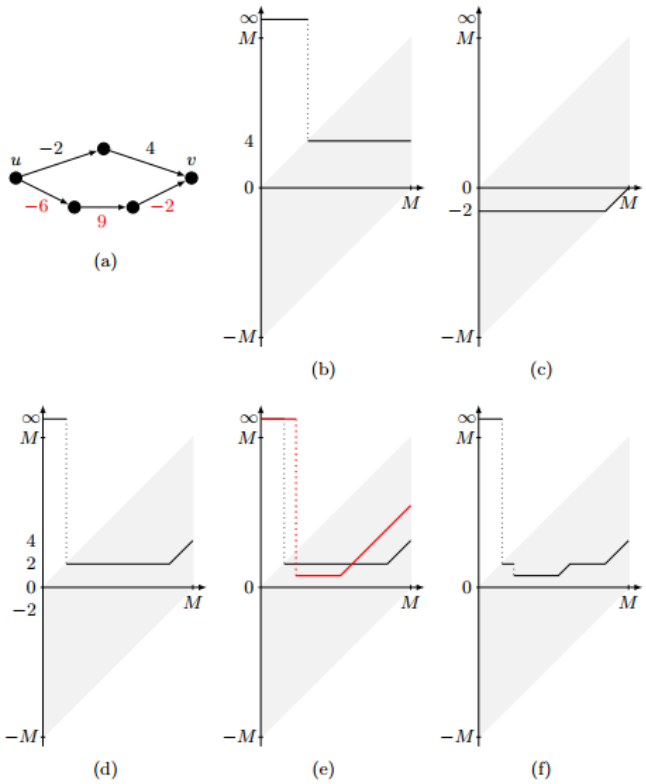


FIGURE 1.1 – Ref [2]. Illustration de la fonction de consommation f sur un petit graphe

La figure ci-dessus illustre le modèle de la consommation énergétique sur un petit graphe. (a) est le graphe support ; (b) est la fonction de consommation de l'arc de coût de consommation 4, la région grisée est la région de valeurs respectant les contraintes de charge de batterie ; (c) est la fonction d'un arc de coût -2. (d) est la fonction de coût du chemin noir supérieur (link sur le chemin) ; (e) est le coût des deux chemins du graphes ; (f) est le résultat de l'opération merge.

1.1.2 Détermination d'un chemin optimal

Nous présentons ici les grandes idée de la technique pour la détermination d'un chemin optimal. Le lecteur est prié de consulter l'article de Moritz at al [2] pour plus de détails sur l'algorithme et surtout sur les optimisations possibles. L'algorithme est une variante de l'algorithme de Dijkstra (*label correcting Dijkstra*). Comme dans l'algorithme de Dijkstra[3], chaque sommet u est étiqueté d'une valeur $lbl(u)$ décrivant le coût de consommation minimum courant pour visiter le sommet u. $lbl(v)$ est initialisé à ∞ pour tout v sauf la source s pour laquelle $lbl(s) = 0$ (coût de consommation nul au départ). Soit $L = V$ une liste. A chaque itération, nous choisissons le sommet $u \in L$ ayant $lbl(u)$ minimum, et puis pour tout arc $a = (u, v)$, on évalue sa fonction f_a pour $b(u) = b(s) - lbl(u)$.

$$\text{Si } \; lbl(u) + f_a(b(u)) < lbl(v) \; \text{ alors } \; lbl(v) \leftarrow lbl(u) + f_a(b(u))$$

Le sommet u est retiré de L et l'algorithme s'arrête lorsque L est vide.

Le problème qui se pose lors de la présence d'arcs de *récupération* de coûts négatives est que l'algorithme ne fonctionne pas correctement, il peut fonctionner si on omet la liste L et que nous choisissons à chaque itération d'étiquette minimum et attendre jusqu'à ce que l'algorithme converge ; L'algorithme peut ainsi effectuer des plusieurs scans (en nombre exponentiels) sur un même sommet, ce qui rend l'algorithme très couteux. La solution réside dans une technique de correction d'étiquette *label correcting*. La technique ressemble en gros à l'algorithme de Johnson[4] pour la détermination des chemins minimaux *All to All* en utilisant une fonction potentiel. Une fonction potentiel $\Pi : V \to \mathbb{R}$ est dite réalisable si $\forall a = (u, v), \min(f_a) + \Pi(u) - \Pi(v) \geq 0$. En imposant $c_a = \min(f_a) + \Pi(u) - \Pi(v)$, les coûts sur les arcs sont positives, et un chemin optimal dans G est toujours optimal dans le nouveau graphe. L'objectif est donc de trouver une évaluation d'une fonction potentiel dans une étape de pre-processing afin de pouvoir répondre plus rapidement aux requêtes de chemins optimaux par l'algorithme de Dijkstra.

1.2 Problème de tournées de véhicules

Notre objectif n'étant pas d'exposer un état de l'art complet de la problématique des tournées de véhicules, nous nous concentrons sur une approche de la littérature qui exploite le clustering et qui été reprise et adaptée plusieurs fois sur différents types d'instances.
Nous rappeleons la problématique des tournées de véhicules et des torunées de vehicules pour les véhicules électriques, et puis nous exposons l'approche de résolution basée sur les schémas cluster first-route second.

1.2.1 Définition

Le problème de tournées de véhicules [1] (Vehicle Routing Problem - VRP) se formule dans sa forme la plus simple de la manière suivante : ayant un réseau orienté $G = (V, E)$ tel que $D \subset V$ est un ensemble de dépôts (ça peut être un seul), à chaque arc (v_i, v_j) de E est associée une distance. L'objectif est de trouver m circuits hamiltoniens $(C_1, \ldots, C_m)$ de longueur totale minimale tel que : $\forall v \in V/D, \quad \exists! i \backslash v \in C_i$ et tel que $\exists d \in D, d \in C_i$. Autrement, planifier m tours de véhicules afin de satisfaire la livraison des dépôts D vers chaque client de V/D en minimisant la longueur totale des tournées. Les problématiques pratiques sont en effet beaucoup plus complexes que la version présentée ci-dessous. En effet, chaque véhicule peut disposer d'une capacité limité Q, et chaque client d'une demande q_i (Capacitated Vehicle Routing Problem - CVRP). Des contraintes supplémentaires peuvent encore s'ajouter, par exemple la livraison pour un client v_i doit se faire dans une fenêtre de temps $[a_i, b_i]$. On peut aussi avoir un parc de véhicules hétérogène, et/ou avec des véhicules spécialisés (des véhicules encombrants ne pouvant pas atteindre tous les clients, ainsi on achemine les produits au plus proche des clients possibles d'abord, et puis les produits sont déchargés et rechargés sur de petits véhicules).

Approche Cluster First - Route Second : Les heuristiques Cluster First - Route Second pour le VRP sont nombreuses, néanmoins leur principe est le même et se décompose en deux phases :

- La phase Cluster : établir un certain nombre de clusters (k clusters lorsque k véhicules sont disponibles).
- La phase route : Nous relions tous les clients d'un même cluster par un seul tour, autrement, on résout un problème TSP sur chaque cluster.

Différentes approches ont été proposées pour la première phase, comme l'algorithme de balayage [Wren & Holliday (1971)] (*The sweep algorithm*), The Generalized-assignment-based algorithm [Fisher & Jaikumur (1981)], ou Location based heuristic [Bramel & Simchi-Levi (1995)]. Dans ce dernier, les auteurs considèrent

1. Le lecteur est invité à consulter l'excellent ouvrage "Vehicle Routing : Problems, Methods, and Applications, Second Edition" sur le sujet par Paolo Toth, Daniele Vigo. *SIAM - Society for Industrial and Applied Mathematics ; 2nd Revised edition edition*

de résoudre un problème $k-$médians en prenant en compte la position géographique des clients, et puis d'assigner chaque client au médian le plus proche pour créer les clusters. Ces approches ont l'avantage de prendre en considération de la répartition et la distribution géographique des clients. Pour le problème des tournées de véhicules électriques, une approche différente a été proposée par Erdogan et Miller-Hooks et qui repose sur l'algorithme DBCA.

1.3 Tournées de véhicules électriques

La formulation de la problématique des tournées de véhicules électriques que nous adoptons est appelée *Green VRP* et a été proposée par Erdogan et Miller-Hooks [5]. Dans leur article, ils proposent une formulation adaptée, une modélisation mathématique, et quelques heuristiques ; une adaptaion de l'heuristique de Clarke and Wright, une deuxième selon le schéma Cluster First - Route Second[6] basée sur l'algorithme Density Based Clustering Algorithm (DBCA). Les auteurs proposent aussi une heuristique d'amélioration. Conrad et Figliozzi[7] proposent indépendamment la même problématique et qu'ils appelent *The Recharging Vehicle Routing Problem - RVRP*. Ils proposent aussi une modélisation mathématique pour la variante avec des clients ayant des demandes précises et une capacité limitée sur les véhicules (*The Capacitated Recharging Vehicle Routing Problem - CRVRP*). Schneider et al[8] ont traité le cas en présence de fenêtres de temps. Ils proposent aussi une hybridation VNS/TS (Variable Neighborhood Search) pour cette variante.

1.3.1 Formulation du GVRP

Le problème de tournées de véhicules éléctriques se formule de la manière suivante :
Soit $G = (V, E)$ un graphe complet non orienté. $V = I \cup \{v_0\} \cup F$, avec v_0 un dépôt unique, $I = \{v_1, \ldots, v_n\}$ est l'ensemble des clients. $F = \{v_{n+1}, \ldots, v_{n+s}\}$ est un ensemble de stations de rechargement. Nous admettons en plus que le dépôt permet le rechargement de la batterie. Toute station de rechargement a une capacité illimitée. A chaque arête $E = \{(v_i, v_j); v_i, v_j \in V, i < j\}$ est associé un temps de parcours t_{ij}, un coût de consommation c_{ij} et une distance d_{ij}. Les temps de parcours sont donc supposés constants sur les arcs. En plus, il n'y a pas de limite sur le nombre de stations à lesquelles un véhicule peut s'arrêter dans son trajet. Si un véhicule s'arrête dans une station, il se charge à plein.

L'objectif de ce problème est de visiter chaque client au moins une fois en utilisant m tours au minimum, en minimisant la distance totale parcourue tout en respectant les contraintes de consommation, et qu'une tournée ne peut durer plus qu'une durée T_{max}.

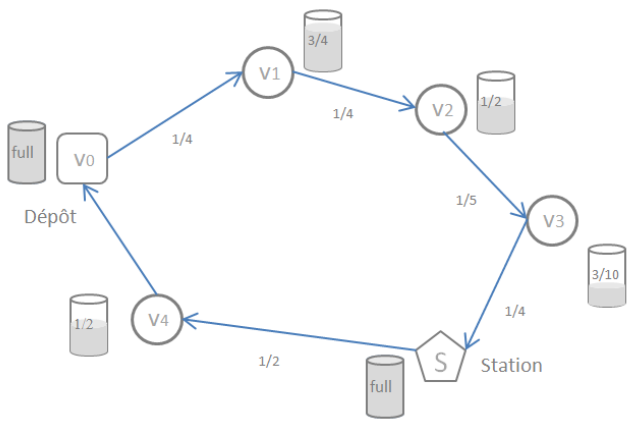


FIGURE 1.2 – Exemple d'une tournée avec rechargement

1.3.2 Approches de résolution

Le problème GVRP est $\mathcal{NP}$ difficile. Cela induit que les méthodes exactes connues ne sont pas adéquates pour la résolution de ce problème. Quelques méthodes heuristiques existent. Nous allons reprendre ces heuristiques et les adapter selon notre méthode de groupement de sites. Ainsi, nous avons jugé qu'il est nécessaire de les exposer dans cette section.

L'heuristique MCWS

L'heuristique MCWS (Modified Clarke and Wright Savings algorithm) pour le problème GVRP a été proposée par Erdogan et Miller-Hooks[5], et s'inspire largement de l'heuristique de Clarke and Wright (1964). Cependant, l'heuristique effectue un certain nombre d'étapes supplémentaires afin d'affiner la solution pour prendre compte des contraintes de consommation et de respect de la durée maximale T_{max} permie pour un tour. Un pseudo code de l'heuristique est donné ci-dessous.

Algorithme 1 : L'heuristique MCWS pour le GVRP

Données : $G = (V, E), c_{ij}, d_{ij}, t_{ij}, T_{max}$
Résultat : Un certain nombre de tournées de livraison

début
  Soit R l'ensemble des n tours $(v_0 - v_i - v_0)$
  Soit $R^+ \subset R$ la liste des tours réalisant les contraintes de consommation et de T_{max}
  $R^- = R/R^+$
  pour tous les $C = (v_0, v_i, v_0) \in R^-$ **faire**
    `/* Insérer une station si possible pour rendre les tours de` R^- `réalisables (si possible) */`
    Soit $v_f = \arg\min_{v_j \in F} c(v_i, v_0) = d(v_i, v_j) + d(v_j, v_0) - d(v_i, v_0)$
    si $C' = (v_0, v_f, v_i, v_f, v_0)$ *est réalisable* **alors**
      $R^+ = R^+ \cup C'$
    fin
  fin
  Soit $SPL = \{v_i; v_i$ extrémités de deux tours différents de $R^+ ou \in F\}$
  $L = \{(v_i, v_j)_{i \neq j} \in SPL \times SPL\}$
  pour tous les $(v_i, v_j) \in L$ **faire**
    Calculer les coûts d'insertion $s(v_i, v_j) = d(v_0, v_i) + d(v_0, v_j) - d(v_i, v_j)$
  fin
  Trier les éléments de L selon l'ordre décroissant de s
  tant que L *n'est pas vide* **faire**
    Soit $(v_i, v_j) \in L$ ayant s maximum
    si v_i, v_j *sont extrémité d'un tours* **alors**
      si *Les contraintes de consommation et de* T_{max} *sont réalisées* **alors**
        Fusionner les deux tours correspondants
      sinon
        si *La contrainte* T_{max} *est vérifiée* **alors**
          $v_f = \arg\min_{v_f \in F} c(v_i, v_j) = d(v_i, v_f) + d(v_f, v_j) - d(v_i, v_0) - d(v_j, v_0)$
          Fusionner les deux tours correspodnants en un tours C
          Insérer v_f entre (v_i, v_j)
        fin
        tant que $\exists v_f \in C \setminus C$ *est toujours réalisable en enlèvant* v_f **faire**
          Enlever v_f de C
        fin
      fin
    fin
    $L = L/(v_i, v_j)$
  fin
fin

Heuristique basée DBCA

Cette heuristique suit le schéma Cluster First - Route Second. La phase de clustering utilise l'algorithme Density Based Clustering Algorithm (DBCA)[2]. Cet algorithme a la particularité de la prise en compte de la distribution des sites clients.

Density Based Clustering Algorithm

Cet algorithme de clustering est l'un des algorithmes les plus connus dans la littérature (en plus du k-means). L'algorithme connait des applications aussi lorsque les données initiales contiennent du bruit. Afin d'illustrer l'idée derrière l'algorithme, il est nécessaire de donner un certain nombre de définitions.

Notions préliminaires

Soit un ensemble de points P dans un espace ($\mathbb{R}^2$ ou n'importe quel espace métrique), nous appelons ε-voisinage $N_\varepsilon(p)$ d'un point p l'ensemble des points de P à l'intérieur de la sphère (ou hyper-sphère) centrée sur p de rayon ε.
Les points de P sont divisés en trois types :

- Les points *cœur* (*core vertex*), un point p est dit cœur si et seulement si $|N_\varepsilon(p)| \geq min_{pts}$, avec min_{pts} un paramètre d'entrée de l'algorithme (un certain nombre de points). Si p est cœur, les points de P qui sont à l'intérieur de $N_\varepsilon(p)$ sont dits *directement accessible* de p. Aucun point n'est directement accessible d'un point qui n'est pas cœur.
- Un point q est dit accessible d'un point p s'il existe un chemin $(p_1 = p, p_2, \ldots, p_m = q)$ et tel que tout p_{i+1} est directement accessible de p_i (tous les points sont des points cœurs à l'exception de q éventuellement).
- Les points qui ne sont accessibles d'aucun autre point sont considérés comme du bruit.

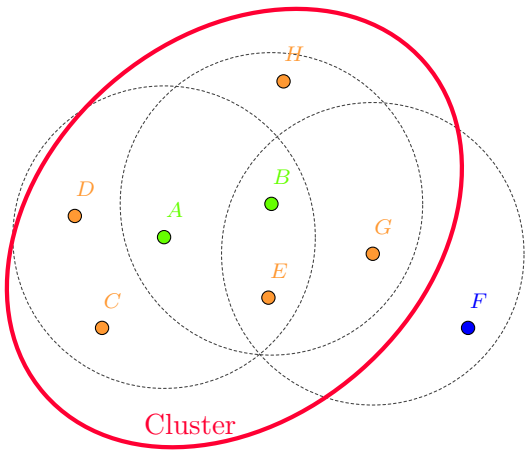


FIGURE 1.3 – Exemple d'un cluster avec $min_{pts} = 4$ et $\varepsilon = 2$. Les points A et B sont des points cœur, Les autres points appartenant au même cluster sont des frontières. Le point F est du bruit

Si un point est un cœur, alors il forme un cluster avec les points qui y sont accessibles. Donc, tout cluster contient au moins un point cœur et les points non cœur du cluster sont appelés frontière du cluster, car une fois arrivé à ceux là, on ne peut plus atteindre d'autres points.
La notion d'accessibilité n'est pas symétrique, car par définition, un point cœur ne peut être accessible d'un

2. Appelé originellement DBSCAN : Density-based spatial clustering of applications with noise

point non cœur. Ainsi, une notion de connectivité est nécessaire afin de définir les clusters tel que nous le fassions ci-dessus. Donc, deux point p, q sont dit connectés (*density connected*) s'il existe un point o tel que p et q sont tous les deux accessibles de o. La connexité (ou connectivité) est symétriquement.

Définition 1. *Un ensemble de point est un cluster si et seulement si tout couple de points y appartenant sont connectés (density connected), et si un point est accessible d'un point du cluster, alors il en fait partie.*

Selon la définition énoncée en haut et pour déterminer les clusters, il suffit de prendre un point cœur et d'effectuer une recherche en largeur sur les ε-voisinages, jusqu'à épuiser tous les points du cluster. Cette opérations est répétée jusqu'à ce que tous les points soient visités.

Algorithme 2 : DBCA

```
Données : Ensemble de points P dans R^d, ε, min_pts
Résultat : Un ensemble de clusters
1  début
1      ClusterNumber = 0 ;
2      pour tous les sommet p non marqué de P faire
3          marquer p comme visité
4          si |N_ε(p)| < min_pts alors
5              marquer p comme bruit
6          sinon
7              cluster(p)=ClusterNumber
8              RechercheLargeur(p, ε, min_pts, ClusterNumber)
9              ClusterNumber++
10         fin
11     fin
12 fin
```

Algorithme 3 : RechercheLargeur

```
Données : p point cœur, P, ε, min_pts, ClusterNumber
Résultat : Le cluster numéro ClusterNumber
1  début
1      Soit F une liste FIFO (Queue)
2      F.push(p)
3      tant que F non vide faire
4          p = F.pop()
5          si |N_ε(p)| ≥ min_pts alors
6              pour tous les q ∈ N_ε(p) faire
7                  cluster(q) = ClusterNumber ;
8                  si q n'est pas marqué visité ou marqué frontière alors
                       /* q peut être marqué bruit                    */
9                      marquer q comme visité
10                     F.push(q)
11                 fin
12             fin
13         sinon
14             marquer p comme frontière du cluster
15         fin
16     fin
17 fin
```

Notons que l'exploration en largeur se fait en temps linéaire $O(n)$, par contre la détermination d'un ε-voisinage se fait en $O(n)$ elle aussi, en total l'algorithme est en $O(n^2)$ (Des implémentations plus efficaces et des approximations sont envisageables avec une grosse masse de donnée dans de grosses dimensions).

Cet algorithme a différentes caractéristiques intéressantes en pratique, il permet de traiter des données bruitées. Le nombre de clusters est identifié automatiquement en fonction des paramètres ε et min_{pts}. Le

DBCA est capable d'identifier des clusters qui ne sont pas linéairement séparables contrairement à certains algorithmes. L'algorithme a une complexité relativement réduite.

L'heuristique GVRP basée sur le DBCA

Le principe est de détecter les clusters grâce à l'algorithme DBCA sur V. Supposant que DBCA retourne m clusters $\mathcal{C}_1, \ldots, \mathcal{C}_m$, l'idée consiste à augmenter chaque cluster de v_0 et F.

$$\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_0\} \cup F \tag{1.1}$$

Puis l'heuristique MCWS est appliquée sur chaque cluster augmenté. Ce procédé est réitéré pour différents paramétrages de ε et de min_{pts}. La meilleure solution de tous les paramétrages est gardée.

Les approches par clustering ont été aussi exploitées dans une problématique liée à la localisation des stations de rechargement afin de satisfaire les demandes dans les agglomérations en fonction de la densité de la demande [9].

1.4 Conclusion

Dans ce chapitre, nous avons recensé quelques travaux de la littérature qui visait à traiter des problèmes de cheminement des véhicules électriques. Au prochain chapitre, nous exposons un modèle de graphe que nous utilisons par la suite afin de remplacer l'heuristique DBCA par une autre approche de clustering. Cette approche de clustering se base sur les graphes de proximité relative.

Structure de voisinage support : Les graphes de proximité relative

Dans ce chapitre, nous introduisons les graphes de proximité relative (dits aussi de voisins relatifs) ou *Relative Neighborhood Graphs ou RNG*, nous montrerons leur potentiel à exprimer les similarités. Par la suite, nous présentons l'état de l'art sur les algorithmes de construction et de mise à jour existants.

2.1 Définitions

Soit $G = (V, E)$ un graphe non orienté, ayant V pour ensemble de sommets, et E l'ensemble de ses arêtes. Nous notons le nombre de ses sommets par $|V| = n$.
Soit $V = \{v_1, ..., v_n\} \subset \mathbb{R}^m$ un ensemble de n points (objets) d'un hyperplan de dimension m muni d'une mesure de distance $d(.,.)$. Nous admettons selon [12] que deux points v_i, v_j sont deux voisins relatifs, relativement proche ou "relatively close" si

$$d(v_i, v_j) \leq \max[d(v_i, v_k), d(v_j, v_k)], \quad \forall k = 1, \ldots, n, k \neq i, j. \tag{2.1}$$

Un graphe de voisinage $G = (V, E)$ (ou a Relative Neighborhood Graph-RNG) est un graphe dont l'ensemble des sommets est V, et tel qu'une arête existe entre tout couple de voisins relatifs de V.

L'interprétation géométrique de la proximité relative de deux points dans un plan est caractérisée principalement par la notion d'une lune. Une Lune des deux points A et B est l'intersection des deux cercles ayant pour centres respectifs A, B, et dont le rayon est égal à $d(A, B)$.

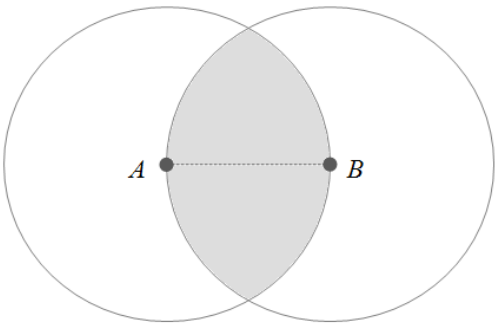


FIGURE 2.1 – Lune de deux points A et B dans le plan

Comme montré sur la figure 2.1, si la lune des deux points A,B est vide (ne contient pas de points à l'intérieur), alors A et B sont voisins. Il est possible d'adopter une notion similaire dans des dimensions supérieures en généralisant les cercles dans la définition précédente à des hypersphères.
Dans ce travail, nous proposons une approche efficace pour la construction incrémentale ainsi que la mise à jour des RNG. Ensuite, nous proposons l'exploitation de cette classe de graphes lors du clustering (groupage) pour le problème du VRP.

La définition ci-dessus sur plusieurs dimensions permet de donner un plus large éventail d'applications potentielles pour le RNG ; une modélisation de nos objets par des points dans un espace euclidien permet

de s'adresser à une large gamme de problématiques. En effet, chaque axe quantifie une caractéristique (variable) des objets, ce qui permet - dans le cadre du problème des tournées - la prise en compte de certains critères lors du groupement et de l'affectation. Une bonne modélisation peut intégrer (en plus des cordonnées géographiques) des caractéristiques comme :

– Les critères opérationnels, sociaux ou réglementaires : par exemple, il est possible d'ajouter une variables pour imposer qu'une certaine région (district) soit couverte exclusivement par un même ensemble de véhicules ; cela se fait facilement en ajoutant une caractéristique permettant de s'assurer que les points de cette région appartiendront à un même groupement. La même idée pourrait bien s'appliquer pour affecter un même ensemble de véhicules à des régions ayant même élévation (il est préférable d'éviter qu'un véhicule fait plusieurs montée et descentes dans son parcours, car une montée implique plus de consommation électrique).

2.2 Propriétés et Revue de la littérature

Les graphes RNG ont été proposés par Toussaint en 1980. Toussaint s'est adressé Typiquement à deux cas d'usage pour ces graphes : Les Problèmes de clustering ou détection de groupements ou communautés, et un deuxième cas relévant du domaine de l'imagerie où il est demandé de trouver une forme visuelle ayant un certain sens en reliant un ensemble de points par des arêtes. G.Toussaint a d'abord montré que le RNG est un supergraphe de l'arbre couvrant minimum *MST*[1] - permettant de conclure qu'il est connexe - et qu'il est un sous graphe de la triangulation de Delaunay[12].

Plusieurs algorithmes ont été proposés dans la littérature pour la construction, ou la mise à jour d'un graphe RNG. Le premier algorithme appelé parfois l'algorithme *naïf* [12] ou *standard*[11] découle directement de la définition des voisins relatifs.

Algorithme 4 : L'algorithme standard RNG

Données : Ensemble de points P dans $\mathcal{R}^d$
Résultat : Un graphe de voisinage $G = (V, E)$ associé à P

```
début
    E = φ
    pour tout couple (v_i, v_j) faire
        k = 1
        tant que k ≤ n faire
            si max[d(v_i, v_k), d(v_j, v_k)] < d(v_i, v_j) et k ≠ i, j alors
                break
            fin
            k++ ;
        fin
        si k > n alors
            E = E ∪ (v_i, v_j)
        fin
    fin
fin
```

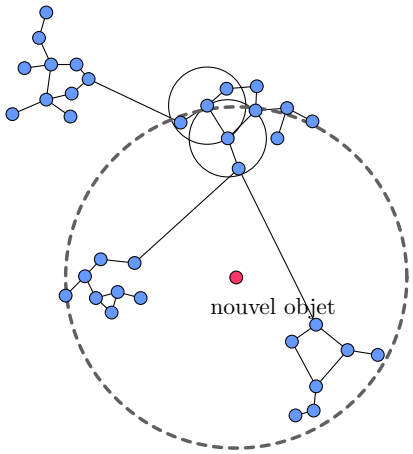
L'algorithme est en $O(n^3)$. Lorsque la dimension m est grande, il est préférable de calculer la matrice des distances au lieu de refaire les calculs de distance de manière répétitive sur la boucle interne. Ainsi le gain temporelle peut être très important pour de grandes dimension. Lorsque n est très grand aussi, il est impossible de sauvegarder la distance en mémoire, et l'implémentation devient en $O(mn^3)$.

On peut construire le RNG différemment, si un supergraphe du RNG (comme la triangulation de Delaunay) peut se construire efficacement, il suffit par la suite de parcourir les arêtes appartenant à ce graphe, et de vérifier si l'inégalité 2.1 est valide. Cette approche est plus efficace ssi le supergraphe n'est

1. minimum spanning tree

pas dense, et qu'il est possible de construire ceci en un temps inférieur à $O(n^3)$. Cette idée a été utilisée par Toussaint pour proposer un algorithme en $O(n^2)$ pour le cas $m = 2$ en se basant sur la triangulation de Delaunay.

Comme mentionné par Hacid et Yoshida [11], ces algorithmes ne prennent pas compte de l'aspect évolutif d'un RNG, et deviennent très rapidement difficiles à mettre en place. En d'autres terme, si un nouveau point est ajouté à l'ensemble P_n, alors il est nécessaire de reconstruire tout le graphe à nouveau. En réponse à ces remarques, Hacid et Yoshida [11] ont proposé une approche de mise à jour et de construction *Incrémentale* qui se base sur la reconstruction locale du RNG sur une hypersphère *optimale*[2]. Supposons $V' = V \cup \{v_{n+1}\}$, l'objectif est donc de construire $G' = (V', E')$ efficacement à partir de G. Soit $\mathcal{H}_{n+1}$ l'hypersphère centré à v_{n+1}, et ayant pour rayon l_{n+1}. Nous notons par $\mathcal{L}_{n+1}$ l'ensemble des points situés à l'intérieur de $\mathcal{H}_{n+1}$. Si l_{n+1} est choisi *convenablement*, nous pouvons construire G' en appelant l'algorithme standard sur l'ensemble $\mathcal{L}_{n+1}$; c'est une *procédure d'insertion locale*[10][11]. Dans cette approche, $l_{n+1} = (1+\varepsilon)(d' + d'')$, avec $d' = d(v_{n+1}, v_{i\star}) = \min\{d(v_{n+1}, v_i)\}, i = 1, \ldots, n$, et $d'' = \max\{d(v_{i\star}, v_i)\}, \forall i = 1, \ldots, n/(v_{i\star}, v_i) \in E$. Autrement, d' est la distance entre v_{n+1} et son plus proche voisin $v_{i\star}$ de V, et d'' la distance entre $v_{i\star}$ et son plus loin voisin. Le paramètre ε est un réel positif choisi empiriquement[3].

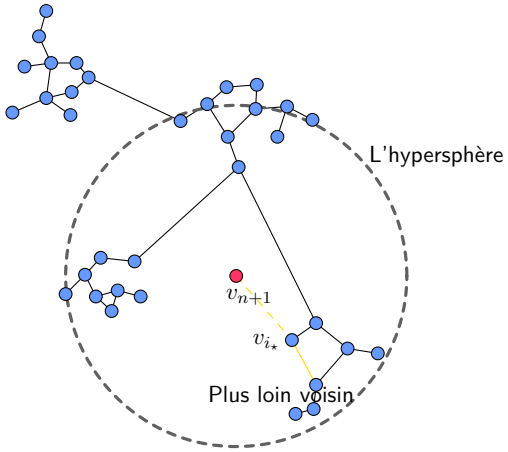


L'hypersphère

Les auteurs [11] proposent de construire tout un graphe RNG de manière *incrémentale* grâce à des appels successifs à cette méthode, afin d'insérer tous les sommets de façon itérative.

Cette approche de mise à jour a une complexité de $O(|\mathcal{L}_{n+1}|^3)$. Par contre, le paramétrage de l'approche conduit à compromettre l'exactitude au profit des performances ou inversement. De plus, il est facile de remarquer que nous ne disposons d'aucune garantie sur l'exactitude du graphe obtenu (des tests expérimentaux ont validé cette hypothèse), à moins que le rayon de l'hypersphère est très grand, et ce qui équivaut à reconstruire le graphe à nouveau à chaque mise à jour. En conclusion, l'approche est heuristique, et la mise à jour se fait en $O(n^3)$ (La construction entière du graphe en $O(n^4)$). Dans la suite, nous proposons une première procédure de mise à jour exacte pour le RNG, et nous améliorons l'heuristique d'insertion locale.

2. *Les auteurs l'ont appelé optimale, mais aucun sens d'optimalité selon un critère n'est précisé dans le papier*
3. Dans la version originale, les auteurs de la méthode ont considéré que ε est compris entre 0 et 1



L'algorithme de construction incrémentale d'un graphe RNG en utilisant l'approche décrite en haut est donné ci-dessous.

Algorithme 5 : Algorithme Incrémentale pour RNG

Données : Ensemble de points P dans $\mathcal{R}^d$, ε
Résultat : Un graphe de voisinage $G = (V, E)$ associé à P

```
début
    V = P, E = φ
    si |P| ≥ 2 alors
        E ⟵ E ∪ (p1, p2)
        pour i = 3 à n faire
            InsertionLocale (V, E, x_n, ε)
        fin
    fin
fin
```

Quant à la suppression d'un sommet de la collection, l'approche est une procédure inverse consistant à déterminer l'hypersphère entourant le point à supprimer, et puis de reconstruire le RNG localement sans considérer le point à supprimer.

2.3 Conclusion

Ce chapitre constitue un bref état de l'art sur la problématique de la construction et la mise à jour des graphe de proximité relative. Ce chapitre montre quelques algorithmes et idées importantes qui seront repris dans le chapitre 4 qui a pour but d'améliorer ces approches.

Détection de communautés dans les graphes

Dans ce chapitre, nous allons exposer quelques algorithmes de détection de communautés dans les graphes. Nous survolons quelques algorithmes par bissection suivis de quelques approches modernes qui sont plus efficaces et plus flexibles. Par la suite, l'idée consiste à proposer une approche de clustering basée sur les graphes de proximité relative. Cela se fait en deux étapes ; la première est une construction du RNG d'un ensemble de points, et la deuxième étape est d'appliquer une approche de détection de communautés sur le RNG, de cette manière nous arriverons à déterminer les clusters.

3.1 Définition de la problématique

Supposons un graphe simple $G = (V, E)$. La détection de k communautés dans G, connu aussi sous le nom de partitionnement de graphes, consiste à chercher à décomposer les sommets du graphes en k sous ensembles $V_1, V_2, \ldots, V_k$ tel que les arêtes de E à l'intérieur des ensembles V_i sont relativement *denses*, et qu'il n'y ait que *peu* d'arêtes entre deux ensembles V_i, V_j [14]. Remarquant que cette définition n'est pas très précise ni très complète non plus (le relativement dense...peu d'arêtes). D'autres définitions relavant des partitionnement des graphes peuvent être énoncées : L'objectif est de trouver une décomposition en k clusters de taille plus ou moins égale, tout en minimisant le nombre d'arêtes inter-groupes. Cette définition n'est pas très juste non plus car en pratique, des clusters peuvent avoir un nombre d'arêtes largement différents.

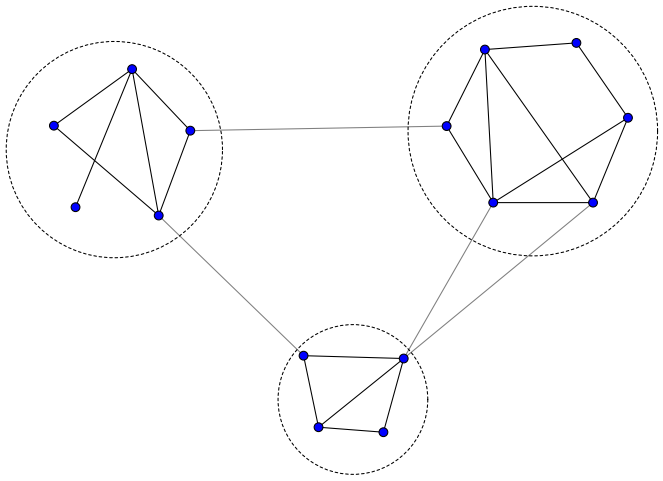


FIGURE 3.1 – Exemple de décomposition en trois clusters

3.2 Quelques algorithmes de clustering dans des graphes

En pratique, bon nombre de méthodes de clustering et de partitionnement fonctionnent en bissection (dichotomie). Le principe consiste à décomposer le graphe en deux sous graphes et puis de réitérer ce procédé sur l'un des sous graphes produits (ou les deux) jusqu'à aboutir à k clusters. Nous présentons ici quelques approches de détection de clusters dans des graphes.

3.2.1 Approches par bissection

Dans une approche par bissection ou par dichotomie. Le graphe est divisé en deux communautés (deux grands clusters), et puis ce même processus est réitéré clusters résultants jusqu'à atteindre le nombre nécessaire de clusters.

L'analyse spectrale est une approche par dichotomie et dont le principe repose sur le calcul des valeurs propres du Laplacien du graphe. Le Laplacien du graphe G est une $n \times n$ matrice symétrique L telle que $L_{ii} = d_i$, et $L_{ij} = -1$ si $(v_i, v_j) \in E, 0$ sinon. Autrement, on peut définir L comme $D - A$ avec D la matrice diagonale des degrés, et A la matrice d'adjacence de G. remarquons que chaque ligne ou colonne de L a une somme valant 0, car $D_{ii} = \sum_j A_{ij}$. On en conclut que le vecteur $\mathbf{1}= (1, \ldots, 1)$ est un vecteur propre de L associé à la valeur 0.

Imaginons que notre graphe peut être décomposé parfaitement en k sous graphes $G_1, \ldots, G_k$ (aucune arête entre deux sous graphes), alors le laplacien est diagonal par blocs. Les vecteurs v^k tels que $v_i^k = 1$ si $v_i \in G_k$ et 0 sinon sont aussi des vecteurs propres de L de valeur propre 0.
Si le graphe ne peut être décomposé parfaitement, mais qu'il puisse être décomposé en k clusters, avec peu d'arêtes inter-clusters, on trouve le vecteur propre $\mathbf{1}$ et k associé à la valeur propre 0, et $k-1$ vecteurs propres ayant des valeurs propres proches de 0 (positives car L est symétrique). ces vecteurs propres sont approximativement une combinaison linéaire des vecteurs v^k. Il est donc intéressant d'identifier cette structure.

Le cas particulier avec deux clusters seulement est plus simple à traiter ; le vecteur propre correspondant à la plus petite valeur propre positive supérieure à 0 a des valeurs positives sur les composantes d'une communauté (idéalement proche de 1), et des valeurs négatives ou nulles pour le deuxième cluster. Donc, on cherche le vecteur ayant la plus petite valeur propre supérieure à zéro, et puis on divise sur ce vecteur selon la propriété énoncée.

Une deuxième approche dichotomique est l'algorithme de Kernighan -Lin. L'idée consiste à assigner une fonction coût Q à toute décomposition du graphe en deux clusters, et d'optimiser cette fonction par une recherche locale. Soit V_1, V_2 une décomposition de V en deux clusters. $Q(V_1, V_2) = |E_1| + |E_2| - |V_1 \odot V_2|$, tel que $E_i = \{(v_j, v_k) \backslash v_j, v_k \in V_i\}$ et $V_1 \odot V_2 = \{(v_j, v_k) \backslash v_j \in V_1 et v_k \in V_2\}$. L'utilisateur doit spécifier

la taille de chaque cluster V_1, V_2. Cette contrainte limite l'aspect pratique de la méthode.

Algorithme 6 : L'algorithme Kernighan-Lin

Données : $G = (V, E), |V_1| = k_1, |V_2| = k_2$
Résultat : Deux clusters V_1, V_2

début
  Décomposer V en deux ensembles arbitraires V_1, V_2 de cardinales k_1, k_2
  $Q^* = Q(V_1, V_2)$
  répéter
    $Q = Q^*$
    pour tous les $(v, v') \in V_1 \times V_2$ **faire**
      $V_1' = V_1 \cup \{v'\} \backslash \{v\}$
      $V_2' = V_2 \cup \{v\} \backslash \{v'\}$
      si $Q(V_1', V_2') > Q$ **alors**
        $Q = Q(V_1', V_2'), V_1^* = V_1', V_2^* = V_2'$
      fin
    fin
    si $Q > Q^*$ **alors**
      $Q^* = Q, V_1 = V_1^*, V_2 = V_2^*$
    fin
  jusqu'à $Q^* = Q$
fin

Les méthodes traditionnelles par bissection présentent différentes propriétés limitant leur mise en pratique et leur exploitation[14]. Ces mécanismes ne permettent pas de trouver les divisions naturelles, et aucun indice ne peut être déduit pour choisir le prochain sous-ensemble à diviser en deux communautés. En plus, l'utilisateur doit spécifier le nombre de communautés présentes dans le réseau, et parfois (dans le cas de Kernighan et Lin) spécifier la taille de chaque communautés ; des informations dont nous ne disposons souvent sauf si on connaissait le bon découpage en communautés. Ces paramétrages rendent ces méthodes inexploitables dans plusieurs contextes.

Une solution au paramétrage consiste à essayer différentes combinaisons de paramétrages et d'en prendre la meilleure (selon une mesure comme la modularité que nous exposons dans la prochaine section). Cette solution est néanmoins couteuse en temps d'exécution.

3.2.2 Approches modernes

Un algorithme de détection de communautés est intéressant lorsqu'il présente les propriétés suivantes[14] :
- L'algorithme ne dépend d'aucun paramétrage particulier.
- Il doit être capable de détecter seul le nombre de communautés présentes dans le graphe (et dans le cas contraire, fournir une classification hiérarchique des meilleurs découpages pour n'importe quel nombre de communautés (de 1 à n).

Le premier algorithme satisfaisant ces conditions était *l'algorithme de Girvan and Newman*. Cet algorithme est basé sur la notion de Edge Betweenness sur les arêtes. Avant de définir cette notion, examinons les arêtes inter-communautés et imaginons un flux ou un trafic de type quelconque circulant dans le graphe. Les arêtes inter-communautés reçoivent un trafic élevé car celles ci sont moins nombreuses et connectent les communautés (une sorte de points d'engorgement).

L'idée est de détecter les arêtes inter-communauté *recevant le plus de trafic* et de les éliminer de manière itérative. Les composantes connexes résultantes sont les communautés cherchées.

Définition 2. *Edge Betweenness d'une arête est le nombre de plus courtes chaînes dans ce graphe passant par cette arête (en cas d'en avoir plusieurs, un chemin est choisi).*

Ce procédé permet d'avoir une séquence de découpages possibles, mais ne permet pas de déduire la meilleure division. Pour cela, une mesure appelée la modularité a été proposée pour choisir la meilleure division. Pour calculer la modularité Q d'une division en k groupes, nous définissons une $k \times k$ matrice $\mathbf{e}$ telle que $\mathbf{e}_{ij}$ est la fraction des arêtes dans le graphe original connectant les sommets du groupe i à ceux du groupe j. Q vaut donc :

$$Q = \sum_i \mathbf{e}_{ii} - \sum_{ijk} \mathbf{e}_{ij}\mathbf{e}_{ki} = \operatorname{Tr} \mathbf{e} - ||\mathbf{e}^2||$$

Cette mesure permet non seulement de juger la qualité d'une des divisions données par l'algorithme de Girvan and Newman, mais peut plutôt s'appliquer à n'importe quel division possible. L'algorithme de Girvan and Newman est en $O(m^2n)$ et en $O(n^3)$ pour un graphe creux, l'algorithme est quand même couteux. Un algorithme plus rapide reposant entièrement sur l'optimisation de modularité est proposé par Newman.

Un algorithme plus simple inspiré de l'algorithme précédent est l'algorithme glouton d'optimisation de modularité (*Fast greedy Modularity Optimization Community Detection*). L'algorithme [16] a été proposé par Newman et possède une complexité réduite comparé à l'algorithme précédent. Le principe est simple, on commence par n groupes, et à chaque itération, on choisit de fusionner les deux groupes dont la fusion permet la plus grande augmentation de modularité (ou la plus petite dégradation). L'algorithme finit lorsqu'on aboutit à une seule communauté constituée par tous les sommets du graphe. On effectue un backtrack par la suite afin de choisir la division ayant Q maximum. Il est à noter que l'approche est hiérarchique. L'algorithme peut être implémenté en $O((m+n)n)$ et en $O(n^2)$ sur les graphes creux.

Radicchi et al.[14] proposent un algorithme qui repose sur le même principe d'élimination d'arête de Girvan et Newman. par contre, au lieu d'utiliser la mesure du Edge betweenness, ils utilisent une autre mesure qui se calcule plus rapidement. Radicchi et al calculent en effet le nombre de triangles passant par chaque arête, en se basant sur le constat que les arête inter-communautaires ont moins de chances d'appartenir à un grand nombre de communautés ; pour avoir un triangle qui contient une arête liant deux communauté, nous devons avoir une deuxième arête entre le même couple de communauté et cela est rare sur de grands graphe.

Supposant une arête (v_i, v_j) tel que d_i, d_j sont les degrés respectifs de v_i, v_j. Le nombre maximum de triangles pouvant passer par (v_i, v_j) est égal à $\min(d_i - 1, d_j - 1)$. Soit z_{ij} le nombre de triangles dans le graphe G passant par l'arc (v_i, v_j). Radicchi et al définissent ce qu'ils appellent *The Edge Clustering Coefficient* valant :

$$C_{ij} = \frac{z_{ij} + 1}{\min(d_i - 1, d_j - 1)}$$

Ce coefficient diffère peu du ratio des triangles passant par (v_i, v_j). Le $+1$ sur le numérateur est ajouté à la fraction pour éviter de trop pénaliser les arêtes qui n'appartiennent à aucun triangle mais qui relient des sommets de degrés très petits. Il a été montré que la valeur C_{ij} est anti-corrélée avec la mesure du Edge Betweenness, et donc des arêtes ayant une petite valeur C_{ij} sont des arêtes inter-communautaires. Cependant, ce phénomène n'est pas toujours vrai, il a été montré que l'approche de Radicchi et al. fonctionne bien sur des graphes de liaisons dans des réseaux sociaux, mais pas sur d'autres graphes.

3.3 Conclusion

Dans ce chapitre, nous avons fait le tour de quelques techniques de détection de communautés dans les graphes. Nous avons présenté quelques approches traditionnelles pour cela, et puis des approches plus modernes ont été proposées aussi.

Nouvelle approche de construction pour les graphes de proximité relative

Ce chapitre a pour but de présenter une nouvelle approche plus efficace pour la mise à jour et la construction des graphes RNG. Des résultats de tests viennent en compléments des preuves théoriques sur l'efficacité de l'approche.

4.1 Notre approche

Notre approche proposée peut être utilisée pour effectuer une mise à jour (ou insertion) exacte ou approchée suite à l'ajout de nouveaux points ou données. Cela permet donc d'appliquer cette approche de manière incrémentale afin de construire un graphe de proximité relative. Nous commençons par présenter un premier algorithme d'insertion et de construction exacte efficace, que nous adaptons ensuite pour effectuer une insertion approchée en se basant sur l'idée de l'insertion locale[**?**] décrite au deuxième chapitre.

4.1.1 Construction exacte

Notre approche exacte d'insertion du nouveau point v_{n+1} se fait en deux phases :

- Procédure *suppression* : éliminer de G les arêtes selon la propriété 1.
- Procédure *reconstruction* : introduire les arêtes de type $(v_{n+1}, v_i) \in E'$.

En effet, nous pouvons montrer la propriété suivante :

Propriété 1. *Pour toute arête* $(v_i, v_j) \in E$, $(v_i, v_j) \notin E'$ *ssi* $\max\{d(v_{n+1}, v_i), d(v_{n+1}, v_j)\} < d(v_i, v_j)$.

Algorithme 7 : Procédure *suppression*

Données : Le RNG $G = (V, E), v_{n+1}$
Résultat : E modifié

```
1 début
2     pour tous les (v_i, v_j) ∈ E faire
3         si max{d(v_{n+1}, v_i), d(v_{n+1}, v_j)} < d(v_i, v_j) alors
4             E = E\{(v_i, v_j)}
5         fin
6     fin
7     retourner E
8 fin
```

La procédure de *suppression* nécessite un temps proportionnel à $|E|$, borné par $O(n^2)$. L'analyse faite sur des instances réelles proposées dans Liu et al. [13], montre que le nombre d'arêtes du RNG est borné par $O(n)$. Ainsi, la procédure *suppression* nécessite un temps de calcul de l'ordre de $O(n)$.

Par la seconde phase, on reconstruit G' en reliant le nouveau sommet v_{n+1} à ses voisins relatifs. Contrairement à l'approche connue qui consiste à itérer sur tout sommet v_i de V puis insérer (v_i, v_{n+1}) respectant l'équation 2.1, nous ne considérons que les sommets v_j pour lesquels $d(v_{n+1}, v_j) < d(v_{n+1}, v_i)$.

Cette technique permet d'accélérer l'approche de 30% à 300% sur les cas pratiques.

Algorithme 8 : Procédure *reconstruction*

Données : $E' = suppression(G, v_{n+1})$
Résultat : RNG $G' = (V', E')$

début
  $V' = V \cup \{v_{n+1}\}$; $\sigma = V$
  renuméroter les sommets de σ selon l'ordre croissant de $d(v_{n+1}, \sigma_i)$
  pour $i = 1$ *à* n **faire**
    $j = 1$
    tant que $(d(v_{n+1}, \sigma_j) < d(v_{n+1}, \sigma_i)$ *et* $d(\sigma_j, \sigma_i) \geq d(v_{n+1}, \sigma_i))$ **faire**
      $j = j + 1$
    fin
    si $d(v_{n+1}, \sigma_j) \geq d(v_{n+1}, \sigma_i)$ **alors**
      $E' = E' \cup \{(v_{n+1}, \sigma_i)\}$
    fin
  fin
  retourner *G'*
fin

L'insertion du sommet v_{n+1} nécessite donc un temps de calcul borné par $O(n^2)$. On constate aussi qu'une construction incrémentale d'un graphe RNG à l'aide de notre approche incrémentale est bornée par $O(n^3)$.

4.1.2 Construction heuristique

Noter que si nous nous intéressons à une insertion locale en appellant notre méthode selon les critères définis dans [11] et présentés dans la section précédente, nous transformons notre méthode exacte en une heuristique de complexité $O(|\mathcal{L}_{n+1}|^2)$. Il est clair à ce stade que notre approche heuristique est meilleure que celle proposée dans [11] avec des ordres de grandeur.

4.2 Extension

Nous avons montré dans des travaux antérieurs -l'année passée à travers un mini projet des sciences de la décision- une propriété intéressante de l'approche incrémentale de Hacid et Yoshida. Cela consiste à remarquer qu'une taille d'hypersphère plus grande (en fixant ε à des valeurs plus grandes) ne garantie pas un rapport d'exactitude meilleur.

Il est possible d'optimiser les étapes de la détermination de la sphère optimale, en accélérant la détermination du plus proche voisin, et puis la détermination des points intérieurs à la sphère. Nous montrerons dans la section suivante les résultats de tests

4.2.1 Locality Sensitive Hashing (LSH)

Une optimisation potentielle serait de réfléchir à des moyens pour accélérer la détermination du paramètres d' dans l'heuristique d'insertion. Ce problème équivaut à la recherche d'un plus proche voisin dans un espace euclidien. La méthode que nous avons étudié et implémenté est appelé Locality Sensitive hashing (LSH)[17]. Dans cette partie, nous allons présenter le cadre général de la méthode, et un exemple d'utilisation dans $\mathbb{R}^m$ avec la distance euclidienne comme mesure de similarité.

Problème du plus proche voisin - Nearest Neighbor (NN)

On suppose travailler avec une base d'objets $X = \{x_1, \ldots, x_n\}$ modélisés par des points dans un espace multi-dimensionnel (dimension m) muni d'une mesure de distance $d(.,.)$. Le problème du plus proche voisin

a pour objectif d'effectuer un pre-processing de la base des objets -n objets- afin de trouver rapidement x_i le plus proche à un objet requête x. Soit $NN(x)$ le plus proche voisin de x. La solution la plus triviale à ce problème consiste à effectuer un scan linéaire de toute la base, cela se fait en $O(n)$. Sauf qu'en pratique, lorsque n et m sont très grands, le calcul devient coûteux. Des approches arborescentes existent comme les Kd-Trees, et qui permettent d'obtenir une complexité de recherche de l'ordre de $O(\log n)$ lorsque m est petit. Malheureusement, ces approches ne sont pas au rendez-vous lorsque m est grand (malédiction de la dimension).

Présentation de la méthode Locality Sensitive Hashing (LSH)

Souvent, en pratique il est suffisant de trouver le NN de façon approchée. LSH est une telle approche. Son principe est simple : si on trouve une fonction de hashage $h(.)$ tel que deux éléments *proches* ait même valeur de h, alors le $NN(x)$ se trouve dans le bucket $h(x)$. Donc, le principe est de trouver une fonction de hashage qui permet de distribuer les objets de sorte que deux éléments proches soient dans un même bucket ou dans des bucket proches. La version originale de la méthode propose d'avoir plutôt k fonctions de hashages aléatoires, et suppose que si les fonctions $h_1, \ldots, h_k$ sont choisies convenablement, alors il y' aurait de fortes chances que $NN(x)$ soit dans l'un des buckets $h_1(x), \ldots, h_k(x)$.

Pour mieux comprendre l'idée, on donne l'exemple ci-dessous[18] d'un ensemble de points dans $\mathbb{R}^3$. La figure montre deux projections différentes en dimension deux de la sphère et de quatre points à l'intérieur. Les deux cercles (qui sont véritablement proches comparés aux carrés) sont proches sur les deux projections, et donc prenant des projections aléatoires (sur $\mathbb{R}^2$), il est bien probable que les deux cercles sont proches selon cette projection. Chacune des deux projections permet de préserver la proximité des objets en réduisant la dimension des données. Dans cet exemple, une projection peut être vue comme un hashage.

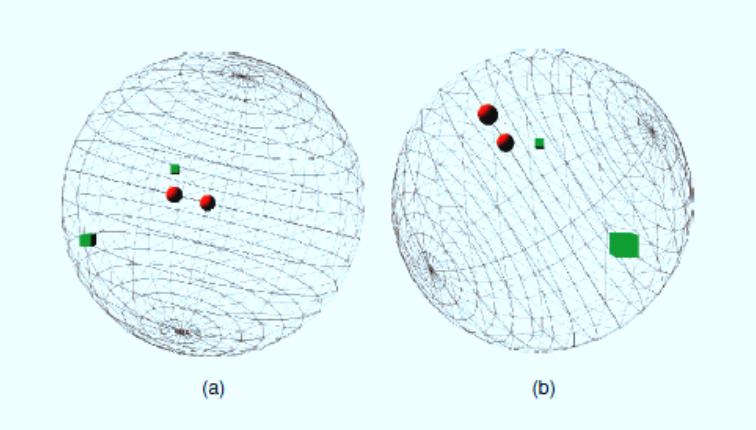


FIGURE 4.1 – Ref.[18]. Exemple montrant des projections de deux cercles proches et deux cubes moins proche

Une fonction de hashage pour la distance euclidienne

La famille de fonctions de hashage à choisir dépend de la mesure de distance considérée. Considérons un espace euclidien $\mathbb{R}^m$ avec une distance euclidienne, ie

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{m} (x_{ik} - x_{jk})^2} \tag{4.1}$$

Un exemple de fonction de hashage est d'effectuer une projection de x sur une ligne $\mathscr{L}$ ayant pour vecteur directeur le vecteur aléatoire $\overrightarrow{v}$. On divise la ligne portant ce vecteur en des sections de longueur w[1]. Et $h(x)$ est la partie entière de la longueur de la projection du vecteur $\overrightarrow{x}$ sur $\overrightarrow{v}$. La fonction de hashage est le résultat $h(x) = \lfloor \frac{\overrightarrow{x}\cdot\overrightarrow{v}}{w} \rfloor$. Cette famille retourne le numéro du bucket à lequel on assigne l'élément x. Il est à remarquer que si le paramètre w est choisi très grand, les vrais négatifs seront très peu car la taille d'un bucket est très grande, sauf que le nombre des faux positifs sera grand. Cela implique plus de calcul. D'autre part, si w est très petit, il y'a de moins en moins de chances de trouver un plus proche voisin approché de bonne qualité.

Ci dessous, nous présentons les procédures de pre-processing et puis de la résolution approchée du NN, selon le schéma général proposé dans le papier original [17].

Algorithme 9 : Preprocessing

Données : Base de données $X = \{x_1, \ldots, x_n\}$, k le nombre de fonctions de hashage
Résultat : Tables de hashage $T_i, i = 1, \ldots, k$

```
début
    pour i de 1 à k faire
        Initialiser une table de hashage T_i
        Générer une fonction de hashage aléatoire h_i(.)
    fin
    pour i de 1 à k faire
        pour j de 1 à n faire
            Insérer le point x_j dans le bucket h_i(x_j) de la table T_i
        fin
    fin
fin
```

Algorithme 10 : requête NN approximé

Données : un point requête x, accès au tables de hashage T_i
Résultat : NN(x) approché

```
début
    S = φ
    pour i de 1 à k faire
        S = S ∪ {Points du bucket h_i(x) de la table T_i}
    fin
    retourner Le plus proche voisin de x dans S
fin
```

L'avantage des pseudo-codes en haut c'est qu'ils permettent de modéliser l'approche LSH indépendamment du type de données (la base X), et de la distance aussi. Nous avons tiré plein avantage de cette définition pour implémenter un code générique permettant d'adapter les fonctionnalités du LSH à n'importe quel type de données qu'on souhaite utiliser.

Notre implémentation du LSH

Le code a été implémenté en C++ (comme tout le reste du code d'ailleurs). Nous avons pris soin de déclarer une classe template LSH a deux paramètres :

- `Point` qui est un point de données (dans $\mathbb{R}^m$, texte, type de données quelconque).
- `HashFunction` définit la famille de fonctions de hashage compatibles avec le type de données Point.

Un code minimal des fonctionnalités essentielles d'une classe LSH est donné ci-dessous :

```
#pragma once
#include <vector>
#include <set>

template <class Point, class HashFunction>
```

1. w est un paramètre que nous choisissons, et qui peut affecter à la fois l'exactitude, et la vitesse de l'approche

```
class LSH
{
public :
  HashFunction* hashSet;  // table of hashtables
  int n; // number of hash functions to use
public:

  /** Constructor : define the number of hash functions to use */
    LSH (int hashesNumber=15) : n(hashesNumber){
    // default constructor of HashFunction which should be defined
      hashSet = new HashFunction[n];
    }

  /** This procedure indexes a new Point p in the Locaity Hash tables */
    void push (Point p){
      for (int i=0; i<n; i++){
        hashSet[i].push(p);
      }
    }

  /** This procedure returns in myset the IDs of possible Nearest Neighbors
         of data point p according to all Locality hash functions
  */
    void KNN(Point p, std::set<int>& myset){
      for (int i=0; i<n; i++){
        hashSet[i].NearestBucket(p, myset);
      }
    }
};
```

Le type `HashFunction` doit impérativement implémenter un constructeur par défaut qui a pour but de générer une fonction de hashage aléatoire et d'initialiser une table de hashage qui contiendra les ID (entiers) des points de données correspondant au type que nous choisissons. En outre, elle devra implémenter les deux fonctions suivantes :

- `push(Point p)` : permet d'indexer un point en l'insérant dans les tables de hashage LSH.
- `NearestBucket(Point p, std::set<int>& nearestPoints)` : voyant dans notre algorithme de requête NN approximé ; `nearestPoints` est l'ensemble S. Nous laissons à l'utilisateur de chercher le plus proche voisin via une recherche linéaire sur `nearestPoints`.

Nous fournissons l'exemple d'une fonction de hashage pour la distance euclidienne sur le modèle de données support MovieLens denos tests empiriques.

```
#pragma once
#include <stdlib.h>
#include "MyPoint.h"
#include <set>
#include "MyHashTable.h"
#include <random>

/**
  Parameters depend on MovieLens Data points properties
  */

static const int DIMENSION = 90000;
static const int min_coordinate = 0;
static const int max_coordinate = 10;
static const double bucket_width = 10;
static const double projector_width = 100000;

class Sorted_Eucleadian_Hash
{
```

```
20 private:
21   // dimension of data space
22   int dimension;
23   // a random double vector
24   double* projector;
25 public :
26   MyHashTable<int> hashTab;
27
28 public:
29   Sorted_Eucleadian_Hash() : hashTab((int)(projector_width/bucket_width)){
30     dimension = DIMENSION;
31     projector = new double [dimension];
32     std::random_device rd;
33     std::mt19937 gen(rd());
34     std::normal_distribution<> d(1,1);
35     // create a 2-stable projector by generating the components
36     // with the gaussian distribution
37     for (int i=0; i<dimension; i++)
38       projector[i]=abs(d(gen));
39
40   }
41
42   int hash(MyPoint p){
43     // project the point on the projector
44     double t = 0;
45     for (int i=0; i<p.coordinates.size(); i++){
46       t+=p.coordinates[i].value*projector[p.coordinates[i].index];
47     }
48     return MAX(floor((t)/bucket_width-((double)rand()/RAND_MAX)),0);
49   }
50
51   void push (MyPoint p){
52     int bucket_number = hash (p);
53     hashTab.pushBack(bucket_number, p.id);
54   }
55
56   void NearestBucket(MyPoint p, std::set<int>& bucket){
57     int bucket_number = hash (p);
58     MyHashTable<int>::Iterator it = hashTab.iterator(bucket_number);
59     while (it.hasNext()){
60       bucket.insert(it.next());
61     }
62   }
63   ~Sorted_Eucleadian_Hash(void){}
64 };
```

L'avantage majeure de l'approche LSH est sa relative simplicité de mise en œuvre. Le code simple et générique fourni en est meilleur exemple. Par contre, pour obtenir des performances optimales, il est nécessaire d'effectuer un grand nombre de paramétrages, qui sont souvent difficiles à prédire sans un grand effort de tests empiriques. Nos tests confirment cette remarque.

4.3 Résultats expérimentaux

Afin de mesurer les performances de l'approche heuristique, nous avons proposé d'effectuer un test de notre méthode sur un modèle réel issu des instances publiques fournies par le groupe GroupLens. GroupLens est un moteur de recherche et de recommandation de films à but non lucratif mise en place et maintenu par l'université de Minnesota (USA).

4.3.1 Modèle de données

Dans un système de recommendation à base de contenu (tout comme sur une plateforme de vente en ligne telle Amazon.com), un utilisateur a la possibilité de donner un score à un produit (ou un film). Ce score est souvent une valeur entière entre 1 et 10 à renseigner (ou 1 à 5 étoile). Il est donc commode de modéliser chaque utilisateur par un point dans un espace multi-dimensionnel, tel que chaque dimesnion (axe ou variable) représente un produit (un film dans le cas de GroupLens). Chaque coordonnés a une valeur entière comprise entre 1 et 10. Notre instance support est la GropuLens 10M. Environ 10 millions de préférences sont exprimés par 70000 utilisateurs sur l'ensemble de 10000 films. Nous remarquons d'abord que les instances réelles sont très creuses, cela fait une moyenne d'environ 143 films regardé par utilisateur.

4.3.2 Démarche expérimentale

Nous validons notre approche heuristique moyennant deux critères : le temps d'exécution (performances) et ratio d'exactitude mesurant la fidelité du graphe obtenu par rapport au RNG exact. Nous avons aussi choisi de fixer le paramètre ε à 0.

Temps d'exécution

Afin de mesurer les performances de notre algorithme, on suit la même démarche que celle utilisée dans [11] pour valider l'approche incrémentale originale. Ainsi, on construit le graphe RNG correspondant à notre instance support pour tous les utilisateurs de manière incrémentale. On part d'un graphe vide, et on met à jour ce graphe en appelant itérativement notre algorithme de mise à jour en mesurant le temps nécessaire pour cette opération. Le schéma ci-dessous est un comparatif des deux approches incrémentale ; l'approche de l'état de l'art [11] en vert et notre méthode incrémentale heuristique (que nous avons appelé H1) en rouge. Sur l'intervalle [0,8000], il est clair que notre méthode est beaucoup plus rapide que l'approche de Hacid et Yoshida. Au delà des 8000 utilisateurs, l'approche de Hacid et Yoshida prenanit énormément beaucoup de temps expliquant pourquoi nous avons arrêté les calculs. En effet, le temps nécessaire pour construire le graphe des 8000 premiers utilisateurs avec leur méthode, est largement suffisant pour construire tout le graphe avec notre méthode.

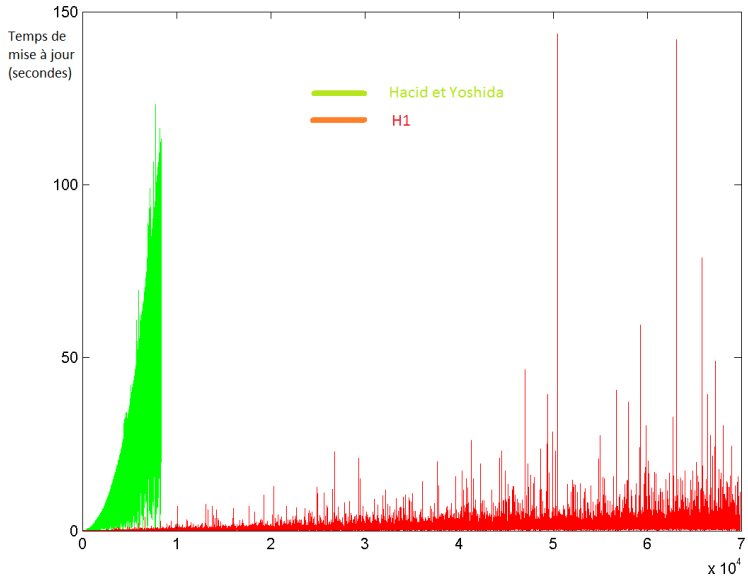


FIGURE 4.2 – Comparatif des temps de mise à jour des deux méthodes incrémentales

Exactitude

Nous avons mesuré l'exactitude du graphe construit grâce à notre approche incrémentale pour les n premiers utilisateurs ($n \in \{1, \ldots, 50000\}$). Pour mesurer la distance entre le graphe RNG exact et le RNG approché de notre méthode, nous utilisons la distance de Jaccard : $ratio = \frac{E_{exact} \cup E_{approche}}{E_{exact} \cap E_{approche}}$. Le résultat est très satisfaisant, avec un ratio d'exactitude de l'ordre de 99%.

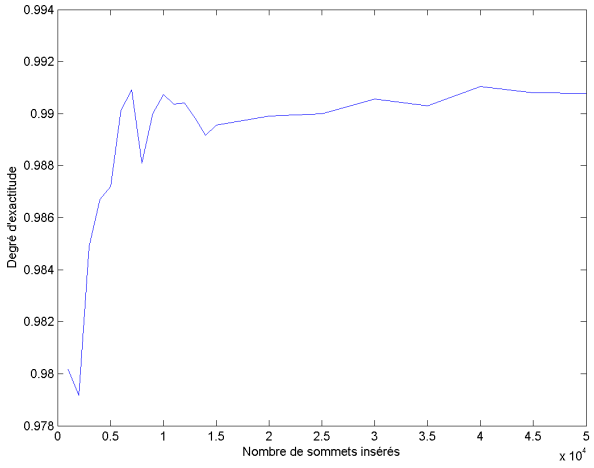


FIGURE 4.3 – Ratio d'exactitude de notre méthode de construction incrémentale

Tests de l'optimisation LSH sur notre instance support et impact sur la méthode

Lors de nos tests avec l'approche LSH implémentée, nous avons retrouvé des difficultés particulières avec le paramétrage. D'abord, la dimension est très grandes et les données semblent avoir un nombre limité de concentrations ; souvent, les utilisateurs préférant un genre particuliers sont beaucoup plus concentrés sur de bons scores sur des films de ce genre. Puisque les données ne sont pas réparties de façon *uniforme*, la largeur w d'un bucket joue un rôle crucial : lorsque celle-ci est grande, le bucket contient une grande proportion des données initiales, et le gain perçu lors du scan linéaire est perdu dans la recherche de l'ensemble des candidats positifs. Lorsque w est petit, le plus proche voisin n'est pas d'une qualité satisfaisante, et parfois, tous les buckets sont vides, donc on se retrouve obligé d'augmenter le nombre de fonction de hashage k, ce qui ralentit les calculs. Nous croyons que la nature des données -grande dimension, et nombre de points $= O(\text{dimension})$ - ne permet pas un gain substantiel. Nous croyons que si le nombre d'utilisateurs dans la base MovieLens était égal à quelques millions, LSH se comporterait beaucoup mieux.

4.4 Conclusion

Nous avons présenté à travers ce chapitre une nouvelle approche pour la construction incrémentale des graphes de proximité. Nous avons montré l'efficacité de notre approche par rapport à l'algorithme incrémental de l'état de l'art. On avait eu aussi l'opportunité d'effectuer quelques tests d'optimisations qui n'ont pas abouti à des améliorations sur notre jeu de données support, mais ça nous a permis de maîtriser une technique d'un grand intérêt pratique (LSH).

Nouvelle approche à deux phases de résolution du problème de tournées de véhicules électriques

Notre problématique est le groupement de sites proches ayant certaines propriétés permettant de minimiser les coûts de transport et de consommation énergétique. Cela soit dans le but de résoudre des problématiques de tournées type Cluster first - Route second, ou l'allocation d'un certain nombre de transporteurs afin de satisfaire des acheminement de types divers (plus courts chemins). Pour cela, nous allons exploiter la structure RNG combinée aux algorithmes de détection de communautés traditionnelles dans les graphes.

5.1 Notre approche à deux phases

Notre approche se décompose en deux phases pour la détection de communautés :

- La première étape consiste à construire le graphe RNG de la collection des points, cela permet de tirer profit des propriétés structurelles des graphes de proximité relative.
- La seconde étape est la détection de communautés directement du graphe RNG construit.

L'avantage d'une telle approche est de tirer profit de la structure du graphe de proximité d'un côté, et des avantages des méthodes modernes de détection de communautés qui permettent d'éviter tout paramétrage.

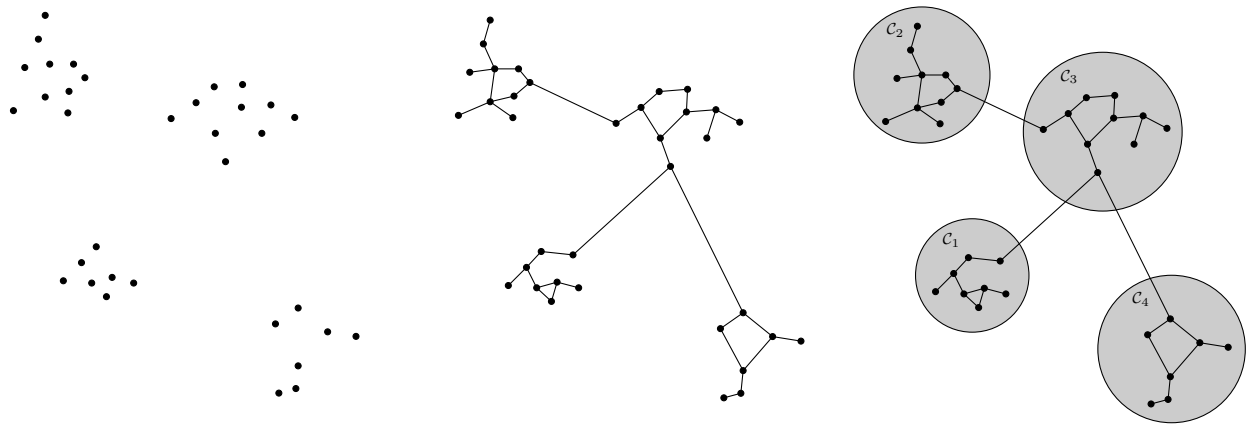


FIGURE 5.1 – Exemple illustrant l'application de l'approche sur un petit graphe

Remarque

La majorité des algorithmes de détection de communautés sont applicables à notre approche, mais pas tous, prenant par exemple l'algorithme de Radicchi et al. qui est incapable de trouver les communautés

dans un graphe RNG. Cela est due au fait que les triangles sont très rares dans un graphe RNG. Si un triangle appartient à un graphe RNG, alors celui là est équilatéral. Il est très improbable d'avoir un triangle équilatéral dans un ensemble de positions géographiques.

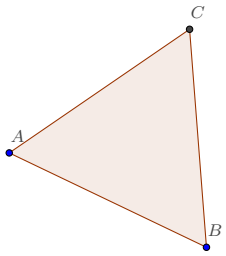
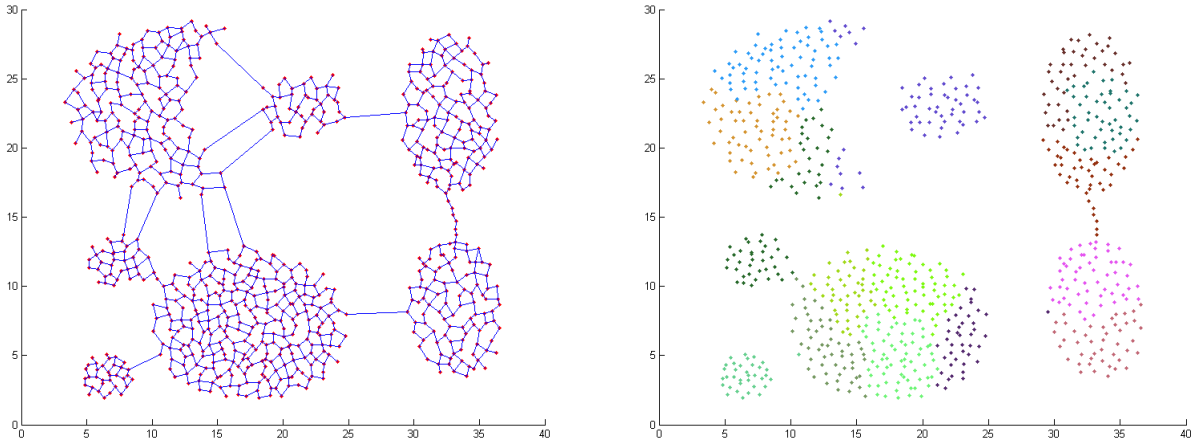


FIGURE 5.2 – Seul un triangle équilatéral peut faire partie d'un RNG

5.2 Résultats expérimentaux

Nous avons testé notre approche à deux phases en combinant le RNG avec l'algorithme glouton d'optimisation de modularité de Newman. Les instances de tests sont des instances connues dans la littérature, et sont largement utilisées pour évaluer différentes approches de clustering dans des espaces de 2 dimensions. Les instances de tests peuvent être trouvées sur : http://cs.joensuu.fi/sipu/datasets/. Pour l'évaluation, nous n'allons pas utiliser une métrique particulière, mais nous nous appuyons sur le résultat visuel en coloriant chaque cluster identifié d'une couleur différente. Cela nous permettra d'analyser le comportement de l'approche, et avoir une évaluation instinctif global.

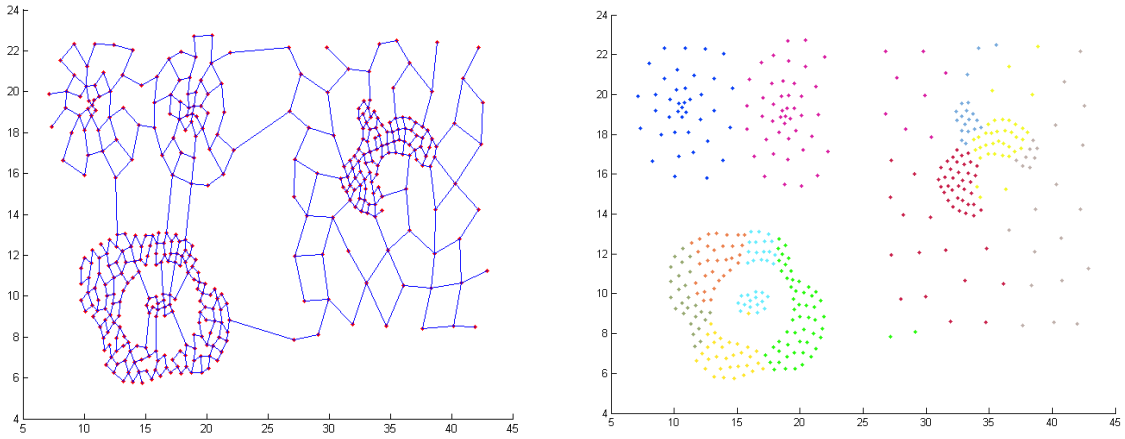
Instance Aggregation [1] : $n = 788$, nombre de clusters réels $= 7$, nombre de clusters retournés par notre approche $= 15$.



Sur cette instance, nous remarquons que notre approche trouve des difficultés à retrouver tous les clusters, et surtout lorsque ceux là sont formés par un grand ensemble de points. Le nombre de clusters fourni est égal au double du nombre de clusters réels. Néamoins, les clusters retorunés sont bien condensés.

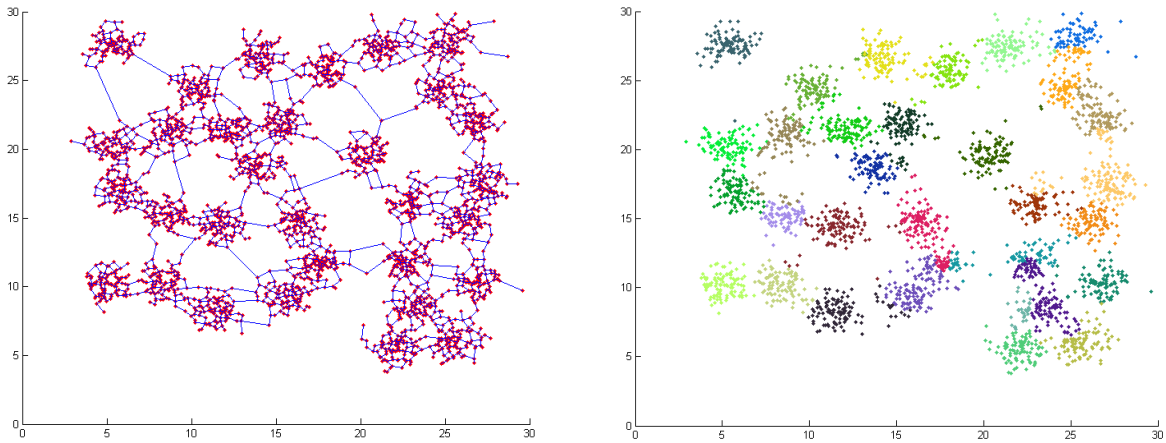
1. Gionis, A., H. Mannila, and P. Tsaparas, Clustering aggregation. ACM Transactions on Knowledge Discovery from Data (TKDD), 2007. 1(1) : p. 1-30

Instance Compound [2] : $n = 399$, nombre de clusters réels $= 6$, nombre de clusters retournés par notre approche $= 11$.



Le nombre de clusters détectés est toujours supérieur à celui des clusters réels. Cette instance nous permet d'apporter une véritable analyse du résultat. D'abord, notre approche trouve plutôt des clusters ayant des points à distance égale (sorte de grille), ce phénomène peut être remarqué sur les contours du trou en bas à gauche, où le clusters a été découpé en 5 autres clusters, alors que le clusters du milieu à été joint au cluster du haut gauche. On remarque que les différences de densité créent des contours pour de nouveaux clusters, les deux clusters en haut à gauche ont été parfaitement identifiés.

Instance D31 [3] : $n = 3100$, nombre de clusters réels $= 31$, nombre de clusters retournés par notre approche $= 31$.

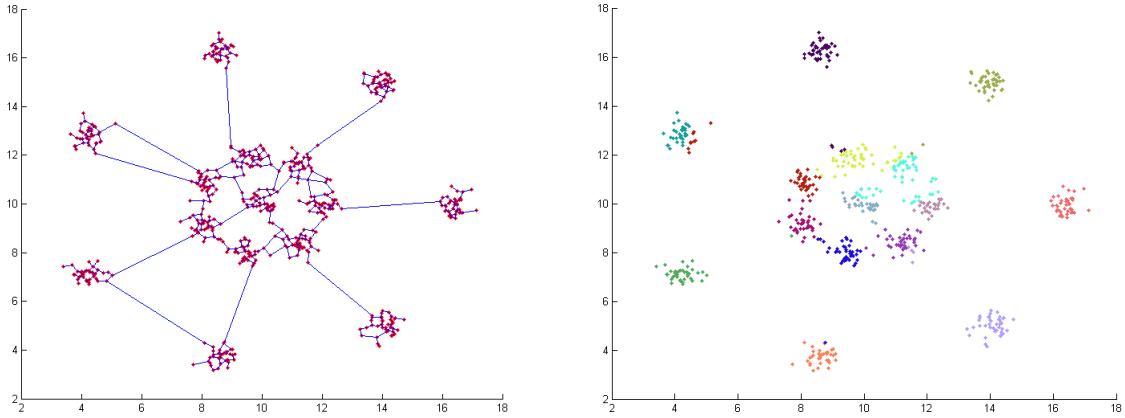


Cette instance ne contient pas de formes particulières comme sur les instances précédentes. Notre approche arrive à identifier 31 clusters (nombre correct de clusters). En plus, la majorité des points ont été bien identifiés, et la figure que les groupements trouvés restent corrects.

2. Zahn, C.T., Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on Computers, 1971. 100(1) : p. 68-86.

3. Veenman, C.J., M.J.T. Reinders, and E. Backer, A maximum variance cluster algorithm. IEEE Trans. Pattern Analysis and Machine Intelligence 2002. 24(9) : p. 1273-1280

Instance R15 [4] : $n = 600$, nombre de clusters réels $= 15$, nombre de clusters retournés par notre approche $= 15$.



Le nombre de clusters trouvés est correct. Les clusters extérieurs sont bien identifiés, on remarque plus d'erreur sur les frontières des clusters intérieurs. Souvent, quelques éléments de la frontières d'un cluster seront assignées à un autre cluster plus proche dans la même direction.

De façon générale, on peut considérer que les résultats de détection de communautés sont acceptables, à condition que l'instance support ne comporte de formes artificielles très particulières (comme sur les deux premières instances de tests). On remarque que les résultats de l'approche sont meilleurs lorsque le nombre de points est élevé, et lorsque les clusters sont denses. Nous remarquons aussi que dans ces derniers cas les erreurs sont souvent commises sur les frontières des clusters, mais ces erreurs ne sont pas très nombreuses.

5.3 Conclusion

Dans ce chapitre, nous avons décrit brièvement l'approche de détection de communautés basée sur les graphes de proximité relative. Par la suite, des tests ont été effectué sur des instances de la littérature. L'approche a l'avantage de ne dépendre d'aucun paramétrage. L'approche peut être utilisé si nous connaissant que les clusters n'ont pas de formes très artificielles et particulières. Dans le cas contraire, d'autres méthodes sont un meilleur choix.

4. Veenman, C.J., M.J.T. Reinders, and E. Backer, A maximum variance cluster algorithm. IEEE Trans. Pattern Analysis and Machine Intelligence 2002. 24(9) : p. 1273-1280

Résultats Expérimentaux

Une fois que les clusters déterminés via une méthode de clustering, l'heuristique MCWS est appliquée sur chacun des clusters. Dans ce bref chapitre, nous allons effectuer une expérimentation sur l'heuristique MCWS afin de mieux comprendre son comportement, et si une certaine dépendance existe entre la nature d'une instance et les résultats retournés. L'objectif de ce bref chapite est de montrer l'importance de l'approche par tests numérique afin de s'assurer qu'un réseau de distrribution à véhicules électrique est capable de servir tous les clients.

6.1 Complément sur l'heuristique MCWS

D'abord, nous pouvons faire un certain nombre de remarques concernant l'heuristique MCWS présentée au premier chapitre et proposée par Erdogan et al.[5]. L'heuristique ne peut pas garantir de couvrir tous les sites clients par les véhicules électriques. Cette limite est justifiée par la complexité de la problématique. Les auteurs ne considèrent pas un site qui ne pouvait pas être couvert par une tournée réalisable [1] de longueur 2 ou trois maximum (lignes 1 à 9 de l'agorithme 1. Dans notre implémentation nous ajoutons une procédure qui après la fin de MCWS, essayera d'inclure un site non couvert dans une tournée à condition qu'elle reste réalisable.

Algorithme 11 : Inssertion Réalisable

Données : $G = (V, E), c_{ij}, d_{ij}, t_{ij}, T_{max}$. Des tournées réalisables $\mathcal{C} = \mathcal{C}_1 \cup \ldots, \cup \mathcal{C}_k$
Résultat : k tournées augmentées si possible

1 **début**
2 **pour tous les** $v_i \notin \mathcal{C}$ **faire**
3 Soit $\mathcal{T} = \{v_j \in \mathcal{C} \backslash$L'insertion de$v_i$après$v_j$donne une tournée réalisable$\}$
4 **pour tous les** v_j *de* $\mathcal{T}$ **faire**
   `/* Calculer le coût d'insertion après` v_j `*/`
5
6 Soit v_h Le successeur de v_j dans $\mathcal{C}$
7 Calculer le coût d'insertion $s(v_j, v_i) = d(v_j, v_i) + d(v_i, v_h) - d(v_j, v_h)$
8 **fin**
9 Insérer v_i après v_j ayant cout d'insertion minimum
10 **fin**
11 **retourner** *Des tournées augmentées si possible*
12 **fin**

Cette procédure est un dernier passage sur les sommets non encore couvert par aucun véhicule, et qui peuvent éventuellement être couvert suite à une insertion dans une des tournées construites par MCWS.

6.2 Démarche expérimentale et Résultats

Dans cette partie, nous exposons un certain nombre de tests visant à connaitre la capacité moyenne nécessaire dans un véhicule électrique, afin de satisfaire toutes les demandes en utilisant l'heuristique MCWS.

1. réalisant les contraintes de consommation

6.2.1 Type d'instances et Hypothèses

Nos instances support sont des instances générées aléatoirement de la manière suivante : n sites clients générés uniformément dans un carré unité. Le dépôt est au centre du carré (0.5,0.5). Nous fixons le nombre de stations de rechargement à différents ratio de n. Nous supposons que la consommation est linéaire et proportionnelle à la distance parcourue pour simplifier le modèle. Nous relaxons la contrainte de T_{max}, ie nous n'imposons pas de durée maximale sur une tournée, seule la contrainte de consommation sera prise en compte.

6.2.2 Protocole de test et résultats

Notre démarche peut être considérée plutôt comme une approche numérique pour estimer la capacité nécessaire pour un véhicule électrique afin de pouvoir satisfaire la demande d'un certain nombre de clients situés sur une carte de façon uniforme. Pour cela, Nous fixons un couple comportant : un certain nombre de sites clients n et une distance de couverture maximale d_{max} (lié linéairement à la capacité). Nous fixons le nombre de stations à $\lfloor \frac{n}{10} \rfloor$. Nous mesurons pour chaque combinaison le nombre de clients servis. Les résultats sont synthétisée dans ls tableau ci-dessous.

$n \backslash d_{max}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5
20	0,01	0,05	0,1	0,22	0,35	0,49	0,62	0,7	0,8	0,88	0,95	0,98	0,99	1	1
40	0,02	0,05	0,12	0,21	0,34	0,47	0,59	0,69	0,78	0,87	0,94	0,98	1	1	1
60	0,01	0,03	0,09	0,2	0,34	0,45	0,58	0,67	0,77	0,87	0,94	0,98	1	1	1
80	0,01	0,05	0,1	0,2	0,32	0,46	0,58	0,68	0,77	0,86	0,93	0,97	1	1	1
100	0,01	0,04	0,1	0,18	0,32	0,45	0,58	0,68	0,78	0,87	0,94	0,97	1	1	1
120	0,01	0,04	0,09	0,18	0,3	0,45	0,57	0,67	0,78	0,87	0,94	0,98	1	1	1
140	0,01	0,04	0,1	0,19	0,33	0,47	0,58	0,69	0,78	0,87	0,94	0,97	1	1	1
160	0,01	0,04	0,09	0,18	0,32	0,45	0,58	0,68	0,78	0,88	0,94	0,98	1	1	1
180	0,01	0,04	0,08	0,18	0,31	0,45	0,58	0,68	0,78	0,88	0,94	0,98	1	1	1
200	0,02	0,04	0,09	0,17	0,28	0,43	0,55	0,66	0,76	0,86	0,93	0,97	0,99	1	1

TABLE 6.1 – Ratio des clients visités par un tour avec nombre de stations égal à $\lfloor \frac{n}{10} \rfloor$.

Nosu remarquons que seule la capacité de la batterie a de l'influence sur le ratio de clients servis, car toutes les colonnes sont presque identiques. Cette approche numérique montre qu'il est nécessaire de procéder de cette façon avant la mise en place d'un réseau de distribution par véhicules électrique. Cela permet de mesurer le nombre de stations de rechargement nécessaire (qui sont souvent elles mêmes mise en place grâce à des approches de localisations par clustering), mais aussi de tester la faisabilité d'une telle infrastructure. Une approche d'optimisation topologique par descente peut être combiné au même procéder afin de placer les stations de rechargement dans des endroits qui permettent à la fois de servir tous les sites clients et de minimiser la longueur d'une tournée de livraison. Les approches par essais successifs (numériques) sont utilisées en cas de la présence d'un modèle fidèle à la situation réelle.

6.3 Conclusion

Ce chapitre a permi de montrer la nécessité de l'approche expérimentale numérique dans la conception de réseaux de distribution à véhicules électrique. La nature spécifique de la problématique qui limite le nombre de clients servis par un tel service impose de procéder à des tests de faisabilité (numériques similaires au tests ci-dessus).

Bilan de projet et conclusion

Les livrables de ce projet consistent en un certain nombre de techniques algorithmiques proposées. Les techniques ont été implémentée et évaluées sur des jeux de données permettant de montrer leur principales propriétés en terme de performance, qualité et pertinence. Les codes produits (en C++ et en Matlab) seront fournis à l'encadrant, avec les instances de tests élaborés.

Une partie du projet a été communniquée et présentée à un congrès national (ROADEF 2015 à Marseille).

En conclusion et sur un plan personnel, ce travail m'a permi d'apprendre énormément de choses. La latitude était large et très riche, ce qui m'a permi de découvrir différentes techniques de domaines variés. Ce projet m'a aussi offert l'opportunité de participer à une conférence nationale avec mon encadrant Ameur Soukhal (ROADEF 2015), une expérience que je trouve très enrichissante. Nous avons proposé des approaches et techniques d'une moyenne à une grande efficacité à travers ce projet. Je finis avec la citation suivante de Donald E. Knuth : Everyday life is like programming, I guess. If you love something you can put beauty into it.

Annexe

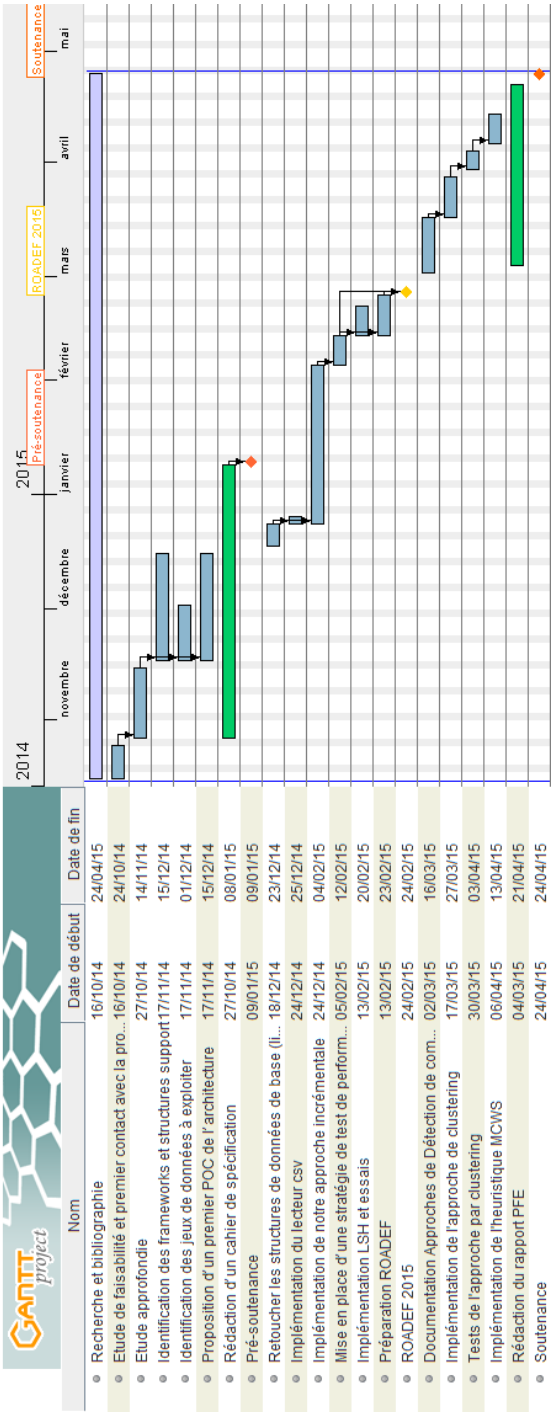


FIGURE 8.1 – Diagramme de Gantt du déroulemnt du projet

Bibliographie

[1] J. Eisner, S. Funke, and Storandt. Optimal Route Planning for Electric Vehicules in Large Network. *In Proceedings of the Twenty-Fifth AAAI Conference on articficial Intelligence*. 2011.

[2] Moritz Baum, Julian Dibbelt, Thomas Pajor, Dorothea Wagner. Energy-Optimal Routes for Electric Vehicules. *SIGSPATIAL* 2013. 54–63.

[3] Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1(1)* :269–271.

[4] Johnson, Donald B. Efficient algorithms for shortest paths in sparse networks, Journal of the ACM 24 (1) : 1–13. (1977).

[5] Sevgi Erdogan, Elise Miller-Hooks. A Green Vehicle Routing Problem. *Transportation Research Part E 48* : 100–114. (2012).

[6] Marshall L. Fisher and Ramchandran Jaikumar. A Generalized Assignment Heuristic for Vehicle Routing. *Networks*, 11 :109-124, 1981.

[7] Ryan G. Conrad and Miguel Andres Figliozzi. The Recharging Vehicle Routing Problem. *In Proceedings of the 2011 Industrial Engineering Research Conference*.

[8] Michael Schneider, Andreas Stenger, Dominik Goeke. The Electric Vehicle Routing Problem with Time Windows and Recharging Stations. *Technical report*. https://bisor.wiwi.uni-kl.de/fileadmin/bisor.wiwi.uni-kl.de/papers/unprotected/tr_electric_vehicles.pdf.

[9] Momtazpour, Marjan and Butler, Patrick and Hossain, M. Shahriar and Bozchalui, Mohammad C. and Ramakrishnan, Naren and Sharma, Ratnesh. Coordinated Clustering Algorithms to Support Charging Infrastructure Design for Electric Vehicles. *Proceedings of the ACM SIGKDD International Workshop on Urban Computing 2012*. p126-p133.

[10] Hakim Hacid, Abdelkader Djamel Zighed. An Effective Method for Locally Neighborhood Graphs Updating. DEXA 2005.

[11] Hakim Hacid and Tetsuya Yoshida. Incremental Neighborhood Graphs Construction for Multidimensional Databases Indexing. Canadian AI 2007, LNAI 4509.

[12] Godfried T. Toussaint. The Relative Neighbourhood Graph of a Finite Planar Set.

[13] Tianyang Liu, Fatma Bouali, Gilles Venturini : EXOD : A tool for building and exploring a large graph of open datasets. Computers & Graphics 39 : 117-130 (2014)

[14] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter and Complex Systems* Vol. 38, No. 2. (25 March 2004), pp. 321-330.

[15] Kernighan, B. W. ; Lin, Shen (1970). "An efficient heuristic procedure for partitioning graphs". Bell System Technical Journal 49 : 291–307.

[16] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Open acces at* :http://arxiv.org/pdf/cond-mat/0309508.pdf.

[17] Aristides Gionis, Piotr Indyky, Rajeev Motwaniz. Similarity Search in High Dimensions via Hashing. *Proceedings of the 25th International Conference on Very Large Data Bases* : 518–529 (1999).

[18] Malcolm Slaney and Michael Casey. Locality-Sensitive Hashing for Finding Nearest Neighbors. *Lecture notes available on* : http://www.slaney.org/malcolm/yahoo/Slaney2008-LSHTutorial.pdf

Tournées de véhicules électriques : une approche basée sur les graphes de proximité relative

Département Informatique
5e année
2014 - 2015

Rapport Final de PFE

Résumé : Dans ce travail, nous étudions la problématique des tournées de véhicules électrique. Une approche Cluster first Route Second est étudiée, avec proposition d'une nouvelle approche de clustering basée sur les graphes de proximité relative. Nous proposons aussi une nouvelle approche efficace pour la construction des graphes de proximité relative.

Mots clefs : GVRP, Cluster First - Route Second, Cluster, RNG, Algorithmes

Abstract: In this project, we study the electric vehicule routing problem with special focus on a Cluster first Route second heuristic scheme. Therefore we propose a new clustering approach based on Relative Neighborhood Graphs. We also propose a new and efficient algorithm for constructing RNG graphs.

Keywords: GVRP, Cluster First - Route Second, Cluster, RNG, Algorithms

---

Encadrants
Ameur SOUKHAL
ameur.soukhal@univ-tours.fr

Etudiants
Nasreddine FERGANI
nasreddine.fergani@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2014 - 2015