Sorbonne Université
Master ANDROIDE

Casser des graphes

Stage de Master 2

Réalisé par :

Antoine TOULLALAN

Encadré par :
Matthieu Latapy, LIP6, Sorbonne Université
Thomas Bellitto, LIP6, Sorbonne Université

Référent :
Thibaut Lust, LIP6, Sorbonne Université

16 août 2023

Table des matières

1 Introduction **1**

2 État de l'art **3**

3 Contributions **4**
3.1 Données OpenStreetMap 4
3.2 Simplification du graphe initial 5
3.2.1 La suppression des noeuds de degré 2 5
3.2.2 La suppression des noeuds de degré 1 6
3.2.3 Le processus de simplification du graphe initial 7
3.3 Planarisation du graphe 9
3.3.1 Première méthode : planarisation par suppression incrémentale d'arêtes 9
3.3.2 Seconde méthode : planarisation par ajout incrémental de noeuds . 12
3.3.3 Encadrement du coût de la coupe équilibrée du graphe initial . . . 15
3.4 Application d'un algorithme de coupe équilibrée polynomiale 15
3.4.1 Un algorithme polynomiale développé par James Park et Cynthia Phillips (1993) 16
3.4.2 Description générale de l'algorithme 16
3.4.3 Comparaison de la méthode développée avec une heuristique de calcul de coupe équilibrée pour les graphes généraux 17
3.4.4 Étude de l'évolution de l'équilibre d'une coupe en fonction de son coût 19

4 Conclusion **20**

A Preuves détaillées **22**

Chapitre 1

Introduction

Ce stage a été effectué au sein du laboratoire LIP6 de Sorbonne Université, encadré par Matthieu Latapy de l'équipe "Complex Networks" et par Thomas Bellitto de l'équipe "Recherche Opérationnelle". Nous étudions la robustesse des réseaux urbains notamment face aux actions de blocage, ces actions étant de plus en plus utilisées par des activistes de divers mouvement sociaux (par exemple le groupe Extinction Rebellion). Une ville peut être représentée sous forme d'un graphe valué : les rues sont les arêtes et les intersections sont les noeuds et les poids des arêtes représentent leur difficulté à être bloquée (par exemple leur largeur). Nous assignons aussi des poids aux noeuds, le poids d'un noeud représente une surface ou un nombre de personnes. Notre objectif est donc d'étudier l'impact potentiel d'actions de blocages sur ces réseaux urbains, pour cela nous étudions s'il est possible de séparer une ville en deux parties de tailles proches.
Or ce problème est similaire au problème de coupe dans un graphe et plus particulièrement du problème de coupes équilibrées, en effet la coupe minimale de ce type de graphe représentant les réseaux urbains isole seulement quelques noeuds du reste du graphe ce qui a peu d'impacts dans le cadre d'une action de blocage. Nous allons donc essayer d'effectuer des coupes **équilibrée** de ces graphes pour modéliser des actions de blocages et ainsi étudier la robustesse des réseaux urbains.

Une coupe qui sépare un graphe en deux sous-ensemble de sommets de taille proche est donc appelée une coupe équilibrée. Mais le problème de la coupe équilibrée est NP-complet pour les graphes généraux [1].
Nous rappelons ici la définition de coupe :

Definition 1 (coupe). Etant donné un graphe $G = (V, E)$, le graphe G est valué si on considère une fonction de poids ω de $V \cup E$ dans $\mathbb{R}$. Pour tout sous-ensemble S de V ou E, on définit le poids de S par $\omega(S) = \sum_{x \in S} \omega(x)$.
Une coupe C de G est une partition de V en deux sous-ensembles X et Y : $X \cap Y = \emptyset$ et $X \cup Y = V$. On définit alors la coupe par $E_C = \{(u, v) \in E, u \in X, v \in Y\}$ et le coût de la coupe C par $C_C = \omega(E_C)$.

Une coupe équilibrée peut être crée de plusieurs manières, dans ce rapport nous allons utiliser deux définitions : la coupe b-équilibrée (b-balanced cut) et la sparsest cut.

Definition 2 (coupe b-équilibrée). La coupe C est b-équilibrée avec $0 \leq b \leq 1/2$ si et seulement si $min\{\omega(X), \omega(Y)\} \geq b * \omega(V)$. Dans la suite du rapport, nous parlerons de

coupes équilibrées pour désigner des coupes dont les deux sous-ensembles de la partition ont des poids proches, c'est à dire avec b proche de 1/2.

Definition 3 (sparsest-cut). La coupe est une sparsest-cut si c'est une coupe C de G de coût C_C en deux sous-ensembles X et Y qui minimise le quotient $C_C/min\omega(X), \omega(Y)$. La sparsest-cut donne une coupe G dont les deux sous-ensembles de noeuds sont de taille proche et dont le coût de la coupe est faible.

Mais il existe des algorithmes de complexité polynomiale pour calculer une coupe équilibrée pour les graphes planaires [2].

Nous rappelons ici la définition de planarité :

Definition 4 (plongement d'un graphe et graphe planaire). Un plongement d'un graphe $G = (V, E)$ dans le plan, ou simplement plongement de G, est un couple $p = (x, y)$ de fonctions x et y de V dans $\mathbb{R}$. Pour tout v dans V, on appelle $p(v) = (x(v), y(v))$ les coordonnées de v. Pour tout (u, v) dans E, on note $p(u, v)$ le segment reliant $p(u)$ et $p(v)$. Un graphe planaire est un graphe pour lequel il existe un plongement dans l'espace tel qu'aucune arête n'en croise une autre.

Nous faisons l'hypothèse qu'une des propriétés des graphes représentant les réseaux urbains est leur quasi-planarité c'est à dire que ces graphes, qui sont souvent non planaires , peuvent être légèrement modifié (en ajoutant ou enlevant des arêtes) afin de les rendre planaires (par exemple en enlevant des arêtes représentant des ponts ou des tunnels qui croisent d'autres arêtes).

L'objectif de ce stage est donc d'étudier l'impact potentiel d'actions de blocages dans les réseaux urbains en effectuant des coupes équilibrées sur ces graphes.

Le plan général de notre méthode de calcul de coupe équilibrée dans ce type de graphe est le suivant :

- Créer un graphe valué représentant une ville à partir de données réelles.
- Simplifier ce graphe afin d'améliorer le temps de calcul des algorithmes appliqués à ce graphe.
- Planariser le graphe en le modifiant le moins possible.
- Appliquer au moins un algorithme de coupe équilibrée de complexité polynomiale sur ce graphe planaire.
- En déduire une coupe équilibrée sur le graphe initial.

Nous pourrons ensuite comparer l'efficacité de cette méthode de coupe avec des heuristiques de coupe équilibrée pour les graphes généraux qui sont déjà implémentées.

Chapitre 2

État de l'art

Pour créer le graphe valué avec des données réelles, nous utilisons la base de données OpenStreetMap qui contient des données très complètes sur les réseaux urbains (coordonnées, taille des rues, nombre de voies...) . Il existe une librairie python qui nous permet d'importer facilement ces données pour une ville donnée et ainsi créer notre graphe valué représentant le réseau urbain de cette ville : c'est la librairie OSMnx développé par Geoff Boeing [3], nous verrons dans le chapitre suivant comment nous construisons ce graphe valué à partir de ces données.

Le problème de la coupe b-équilibrées (b-balanced cut) est NP-difficile pour les graphes généraux [1] ainsi que le problème de la sparsest cut [4]. Mais il existe des heuristiques polynomiales pour calculer la b-balanced cut pour les graphes planaires [5] qui sont plus efficaces que pour les graphes généraux. Et il existe des algorithmes de complexité polynomiale pour calculer la sparsest-cut d'un graphe planaire [2], mais aucune implémentation d'algorithme polynomiale n'a été trouvée pour calculer la sparsest-cut dans un graphe planaire. Nous avons donc implémenté l'algorithme décrit dans l'article "Finding minimum quotient-cuts in planar graphs" écrit par James K. Park et Cynthia Clark[2].

Notre méthode de calcul de coupe équilibrée pour les graphes de réseaux urbains repose sur l'hypothèse que les réseaux urbains sont quasi-planaires. Cette hypothèse s'appuie notamment sur les travaux de Geoff Boeing dans cet article[6]. Cet article présente des indicateurs de mesures de non-planarité et ces indicateurs sont appliqués pour plusieurs réseaux urbains, ce qui nous montre que certains réseaux urbains sont plus "non-planaires" que d'autres selon ces indicateurs mais globalement ces réseaux urbains sont proches de la planarité. Cet article présente aussi des méthodes pour planariser les réseaux urbains, ce sont des méthodes que nous utilisons et qui consiste à ajouter des noeuds aux intersections d'arêtes et enlever les arêtes qui en croisent d'autres.

Chapitre 3

Contributions

3.1 Données OpenStreetMap

Les données que nous utilisons pour obtenir les graphes représentant des villes viennent de la librairie python osmnx qui permet de télécharger des données géospatiale d'OpenStreetMap. OpenStreetMap fournit des cartes extrêmement précises des réseaux urbains en plus de nombreuses informations sur ces réseaux (présence de voie de bus ou de vélo, la nature des bâtiments...).

Comme nous modélisons des blocages de rues, nous considérons que, dans le graphe représentant un réseau urbain, le poids d'une arête est la largeur de la rue (ou portion de rue) correspondante. Pour obtenir la largeur d'une rue avec osmnx nous regardons le nombre de voies de cette rue et si le nombre de voies n'est pas donné par osmnx pour une rue, nous prenons la valeur par défaut de 2 voies qui est le plus courant (une voie dans un sens et une voie dans l'autre). On fait l'hypothèse qu'une voie est de largeur fixe, donc que le nombre de voies d'une rue est une bonne représentation de sa largeur.
Nous créons donc, grâce aux données d'Osmnx, un graphe dont les arêtes sont valuées par le nombre de voies de la rue (ou portion de rue) correspondante, les noeuds ont un poids initial de 1.

Ainsi, le graphe, fourni par osmnx, représentant Rouen a 2266 noeuds et 4838 arêtes et le graphe représentant Paris a 9616 noeuds et 18592 arêtes (et 55367 noeuds et 139881 arêtes pour New York) .

FIGURE 3.1 – Carte de Rouen crée avec les données d'OpenStreetMap, les rues sont coloriées en fonction de leur nombre de voies : plus la rue est clair, plus le nombre de voies est grand sur cette rue.

3.2 Simplification du graphe initial

Avant d'appliquer des algorithmes de planarisation, nous simplifions le graphe représentant une ville afin de réduire le temps de calcul des algorithmes que nous appliquerons ensuite au graphe. Ainsi, nous enlevons les noeuds qui ne correspondent pas à des intersections dans la ville, c'est à dire les noeuds de degré 1 (qui sont des culs-de-sac) et les noeuds de degré 2, la suppression de ces deux types de noeuds se fait de manière itérative.

3.2.1 La suppression des noeuds de degré 2

Nous supprimons de manière itérative les noeuds de degré 2 car ils ne représentent pas une intersection dans le graphe. Cette suppression se fait avec l'algorithme suivant :

Algorithme 1 Suppression des noeuds de degré 2

Entrées *un graphe* G

Résultat : Le graphe G sans noeuds de degré 2

1 $liste\_des\_noeuds\_de\_degre\_2 = liste\_des\_noeuds\_de\_degre\_2(G)$
2 **for** $noeud\_de\_degre\_2 \in liste\_des\_noeuds\_de\_degre\_2$ **do**
3 $\quad G = creer\_une\_arete\_entre\_les\_voisins\_du\_noeud\_de\_degre\_2(noeud\_de\_degre\_2, G)$
`/* on remplace le noeud de degré 2 par une arête entre ses deux voisins */`
4 $\quad G = supprime\_le\_noeud\_de\_degre\_2(noeud\_de\_degre\_2, G)$
5 **end**
6 return G

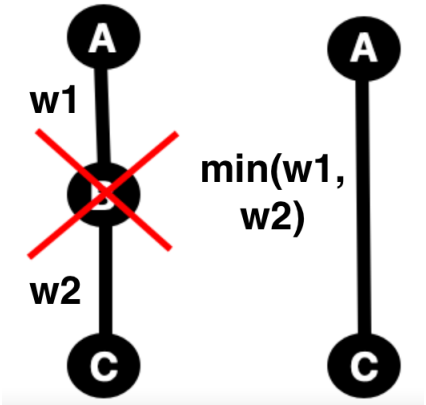


FIGURE 3.2 – Schéma de la suppression d'un noeud de degré 2 et de l'ajout d'une arête valuée pour remplacer ce noeud.

Dans le cas d'un blocage de rue telle que représenté sur le schéma ci-dessus, bloquer la partie de la rue la moins large (avec le poids le plus faible) revient à bloquer la rue entière. Ainsi, Soit v un noeud de degré 2, w_1 et w_2 les poids des arêtes adjacentes au noeud de degré 2, la nouvelle arête crée sera de poids $\min(w_1,w_2)$.

Nous modifions aussi les poids des noeuds car afin que les coupes équilibrées sur le graphe simplifié soient aussi des coupes équilibrées sur le graphe initial, nous répartissons le poids du noeud supprimé sur le noeud qui est l'extrémité de l'arête qui a le plus petit cout par rapport aux coûts des arêtes adjacentes au noeud à supprimer. Car, soit n_1 et n_2 les voisins du noeud de degré 2 à supprimer et a_1 et a_2 les arêtes adjacentes, dans le graphe initial une coupe de coût minimum qui sépare les noeuds n_1 et n_2 inclut obligatoirement une des deux arêtes a_1 ou a_2. Comme la coupe est de coût minimum, l'arête faisant partie de la coupe sera l'arête avec le plus faible coût.

La complexité de cet algorithme pour un graphe avec n sommets et m arêtes est donc $O(n^2)$ dans notre implémentation car on itère le processus de suppression d'un noeud de degré 2 au plus n fois, et chaque processus de suppression de noeud est de coût $O(n)$. Et le processus de recherche de noeuds de degré 2, qu'on n'exécute qu'une fois, est de complexité $O(n)$.
Il existe des méthodes pour optimiser notre implémentation et réduire la complexité de l'algorithme. Ainsi nous pouvons représenter les noeuds du graphe sous forme de compteurs triés en fonction de leur degré. Grâce à cette structure chaque suppression ainsi que la mise à jour de la structure de données a une complexité de $O(1)$. Comme on supprime au plus n noeuds, avec cette structure, la complexité de l'algorithme est $O(n)$.

3.2.2 La suppression des noeuds de degré 1

Nous supprimons aussi les noeuds de degré 1 de manière itérative car ils représentent des "culs-de-sac" dans le graphe et ne nous intéresse pas pour notre objectif de modélisation d'un blocage (il n'y a pas d'intérêt à bloquer une impasse). La suppression d'un noeud de degré 1 se fait avec le transfert du poids de ce noeud sur son unique noeud voisin.

Cette suppression se fait avec l'algorithme suivant :

Algorithme 2 Suppression des noeuds de degré 1

Entrées *un graphe* G

Résultat : Le graphe G sans noeuds de degré 1

```
while il_y_a_un_noeud_de_degre_1_dans_le_graphe(G) do
    noeud_de_degre_1 = noeud_de_degre_1(G)
    G = transfert_poids_noeud(noeud_de_degre_1,G) /* on transfert le poids du
        noeud de degré 1 sur son unique noeud voisin                          */
    G = supprime_le_noeud_de_degre_1(noeud_de_degre_1,G)
end
return G
```

La complexité de cet algorithme pour un graphe avec n sommets et m arêtes est donc $O(n^2)$ car on itère le processus de suppression d'un noeud de degré 1 au plus n fois. Et chaque processus de suppression est de coût $O(n)$ car nous utilisons dans le code python la fonction remove_node de la librairie Networkx pour retirer le noeud de degré 1 qui est de coût $O(n)$ et le processus de recherche d'un noeud de degré 1 dans le graphe est $O(n)$.

Grâce à la structure de données où les sommets sont des compteurs triés en fonction de leur degré, chaque suppression ainsi que la mise à jour de la structure de données a une complexité de $O(1)$. Comme on supprime au plus n noeuds, avec cette structure, la complexité de l'algorithme est $O(n)$.

3.2.3 Le processus de simplification du graphe initial

Nous remarquons que l'application de l'algorithme de suppression des noeuds de degré 1 peut créer des noeuds de degré 2, par exemple dans le cas d'un noeud de degré 3 dont un des voisins est un noeud de degré 1 : après suppression du noeud de degré 1, le noeud de degré 3 devient un noeud de degré 2. Et l'application de l'algorithme de suppression des noeuds de degré 2 peut créer des noeuds de degré 1, comme dans le cas d'un cycle. Pour que le graphe simplifié n'ait plus de noeud de degré 1 et 2 nous appliquons l'algorithme 2 suppression des noeuds de degré 1) puis l'algorithme 1 (suppression des noeuds de degré 2) jusqu'à ce qu'il n'y ait plus de noeuds de degré 1 ou 2 (voir l'algorithme ci-dessous).

Algorithme 3 Suppression des noeuds de degré 1 et 2

Entrées *un graphe* G

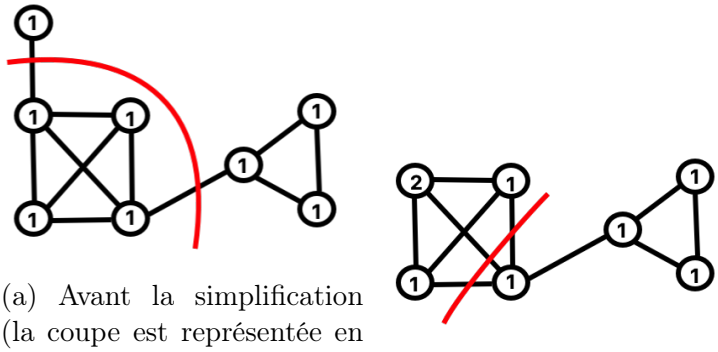
Résultat : Le graphe G sans noeuds de degré 1 et 2

```
while noeud_de_degre_1_ou_2 do
    G = Suppression_des_noeuds_de_degr_1(G) /* Algorithme 2                  */
    G = Suppression_des_noeuds_de_degr_2(G) /* Algorithme 1                  */
    noeud_de_degre_1_ou_2 = noeud_de_degre_1_ou_2(G)
        /* "noeud_de_degre_1_ou_2 " est un booléen qui vaut True s'il
        existe un noeud de degré 1 ou 2 dans le graphe, False sinon           */
end
return G
```

Dans notre implémentation, la complexité de cet algorithme pour un graphe avec n sommets et m arêtes est donc $O(n^3)$ car on itère le processus de suppression d'un noeud de degré 1 et 2 au plus n fois, et à chaque itération on exécute deux algorithmes de complexité $O(n^2)$.
Grâce à la structure de données où les sommets sont des compteurs triés en fonction de leur degré, chaque suppression ainsi que la mise à jour de la structure de données a une complexité de $O(1)$. Comme on supprime au plus n noeuds, avec cette structure, la complexité de l'algorithme est $O(n)$.

Dans le graphe représentant le réseau urbain de Rouen, le graphe initial a 2266 noeuds et 4838 arêtes. Après le processus de simplification le graphe a 1508 noeuds et 2415 arêtes. Pour Paris, le graphe initial a 9616 noeuds et 18592 arêtes, après le processus de simplification le graphe a 8115 noeuds et 13399 arêtes.
Donc le processus de simplification permet bien d'avoir un graphe de plus petite taille (en nombre de noeuds et d'arêtes) de manière significative tout en conservant les informations du graphe.

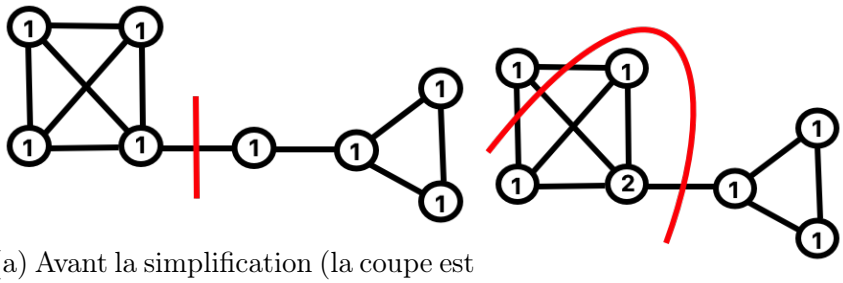
Nous remarquons aussi que ce processus de simplification peut changer le coût de la coupe équilibrée.



(a) Avant la simplification (la coupe est représentée en rouge)

(b) Après la simplification

FIGURE 3.3 – Le coût minimal de la coupe équilibrée augmente de 2 à 3 avec la simplification à cause de la suppression du noeud de degré 1



(a) Avant la simplification (la coupe est représentée en rouge)

(b) Après la simplification

FIGURE 3.4 – Le coût minimal de la coupe équilibrée augmente de 1 à 4 avec la simplification à cause de la suppression du noeud de degré 2

Donc la simplification du graphe peut faire augmenter le coût de la coupe équilibrée, mais nous faisons l'hypothèse que dans le cas des réseaux urbains la simplification aura peut d'impact sur la coupe équilibrée que nous calculerons car nous verrons que nous ne

calculons pas une coupe avec un équilibre exacte et seulement quelques cas particulier (en partie représentés dans les schémas ci-dessus) lors de la simplification augmentent le coût de la coupe, on suppose que ces cas sont peux nombreux lors des simplification de graphes représentant des réseaux urbains.

3.3 Planarisation du graphe

Grâce aux algorithmes vus précédemment, nous avons un graphe valué représentant un réseau urbain simplifié, mais pour quasiment toute les villes le graphe que nous obtenons n'est pas planaire. Nous cherchons donc à planariser ce graphe pour pouvoir ensuite appliquer des algorithmes de coupes équilibrées efficaces.C'est à dire nous voulons le modifier afin qu'il existe un dessin de ce graphe tel qu'aucune des arêtes ne se croisent. Les deux heuristiques que nous allons présenter pour planariser le graphe utilisent les croisements des arêtes du graphe plongé dans l'espace, pour déterminer comment planariser le graphe en le modifiant le moins possible.
Dans une ville, une rue entre deux intersections devient une arête dans le graphe et on considère que chaque rue est droite, c'est à dire que la géométrie de la rue consiste en un segment entre ses deux extrémités. Nous considérons qu'il y a un croisement entre deux rues (deux arêtes dans le graphe) lorsque les deux segments correspondants aux rues se croisent.
Pour déterminer l'ensemble des croisements entre les arêtes avant d'appliquer nos algorithmes nous comparons les arêtes deux par deux, ce processus a un coût de $O(m^2)$ pour un graphe avec m arêtes. Nous pouvons améliorer cette complexité en utilisant l'algorithme de Bentley-Ottmann[7] de complexité $O(n+k)*log(n)$ (l'algorithme de Bentley-Ottmann[7]) où n est le nombre de sommets et k le nombre de croisements, qui renvoie l'ensemble des croisements des arêtes. Or dans le cas des graphes représentant des réseaux urbains k est généralement de l'ordre de la centaine pour des graphes de plusieurs dizaines de milliers de noeuds, donc cet algorithme est plus efficace dans le cas de ces graphes spécifiques que notre implémentation.

On définit la fonction $f_{nombre\_croisements\_aretes}$ qui, pour chaque arête, donne le nombre de croisements qui existe dans le plongement spatial du graphe avec cette arête. Soit le graphe G=(V,E) V étant l'ensemble des sommets et E l'ensemble des arêtes :
$f_{nombre\_croisements\_aretes} : E \rightarrow \mathbb{N}$

Nous présentons les deux méthodes de planarisation du graphe.

3.3.1 Première méthode : planarisation par suppression incrémentale d'arêtes

La première méthode pour planariser ces graphes est la suppression d'arêtes : nous supprimons des arêtes du graphe initial de manière itérative jusqu'à ce que le graphe soit planaire.

Théorème 3.3.1. Soit un graphe G=(V,E) non planaire et G' le graphe planaire obtenu en supprimant des arêtes de G. Si C est le coût de la coupe équilibrée optimale de G et C' le coût de la coupe équilibrée optimale de G', alors $C' \leq C$.

Voir la preuve du théorème dans l'annexe A.

Nous cherchons donc un sous-graphe planaire de taille maximale du graphe initial afin d'avoir un graphe planaire dont le coût de la coupe est proche du coût de la coupe du graphe initial (non planaire). Pour cela nous allons utiliser la méthode générique suivante avec laquelle nous créerons des variantes :

Algorithme 4 Méthode générique de suppression incrémentale d'arêtes pour planariser le Graphe

Entrées *un graphe G*
Résultat : Le graphe G planarisé

```
19 le_graphe_n_est_pas_planaire = le_graphe_n_est_pas_planaire(G)
20 while le_graphe_n_est_pas_planaire do
21 |   arete_a_supprimer = choix_d_arete_a_supprimer(G) /* on choisit une arête
   |      du graphe à supprimer                                              */
22 |   G = suppression_d_arete(G, arete_a_supprimer)
23 end
24 return G
```

Nous modifions la fonction choix_d_arete_a_supprimer pour créer un algorithme de planarisation du graphe par suppression incrémentale d'arêtes.
D'abord nous effectuons le choix d'arêtes à supprimer de manière aléatoire, c'est à dire que la fonction choix_d_arete_a_supprimer va renvoyer une arête du graphe de manière aléatoire. La complexité de l'algorithme avec cette version de la fonction "choix_d_arete_a_supprimer" pour un graphe avec n sommets et m arêtes est $O(m * n)$ car on itère le processus de suppression d'une arête au plus m fois, et à chaque itération on exécute l'opération de choix d'arête aléatoire qui est de coût $O(1)$ et le processus de suppression d'arête est de complexité $O(n)$ ainsi que le test de planarité ,qui est celui de la librairie Networkx ("Left-Right Planarity Test"), de coût $O(n)$, la complexité globale est donc $O(m * (n + n))$ soit $O(m * n)$.
Nous pouvons améliorer le temps de calcul de cette implémentation en utilisant un test de planarité dynamique. Cela permet de vérifier la planarité du graphe après la suppression d'une arête sans le ré-analyser en entier avec la méthode "Left-Right Planarity Test". La complexité de cette méthode est $O(log(n)^2)$ à chaque itération, la complexité de la méthode aléatoire est donc la complexité globale est donc $O(m * (n + log(n)^2))$ soit $O(m * n)$.
Nous regardons l'évolution du plus petit nombre d'arêtes supprimées en fonction du nombre d'exécutions de cet algorithme.

Nous voyons que après 50 ordres aléatoires testés, le nombre minimum d'arêtes supprimées stagne à environ 560 arêtes supprimées (sur 2415).

Nous allons ensuite créer une heuristique pour supprimer le moins d'arêtes possible pour planariser le graphe. Cette heuristique consiste à supprimer du graphe l'arête qui a le plus de croisements dans le plongement spatial du graphe (parmi les arêtes qui ne sont pas déjà supprimées). Dans le graphe G=(V,E), on définit l'arête qui a le plus de

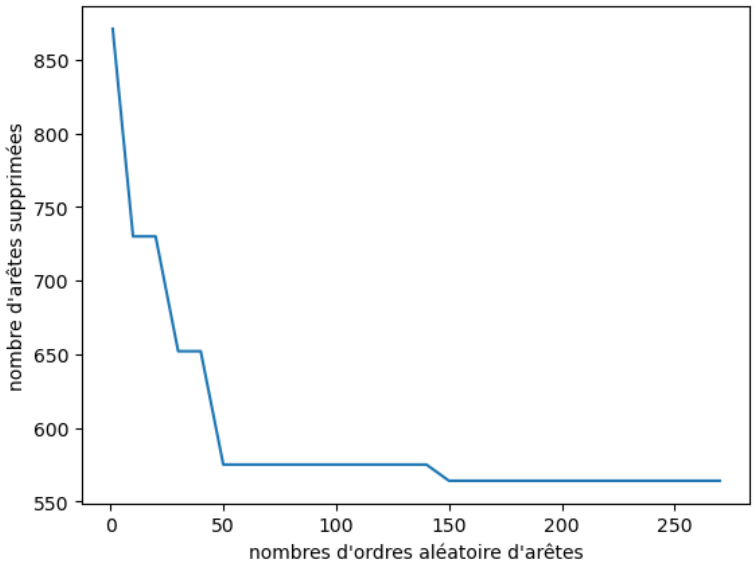


FIGURE 3.5 – plus petit nombre d'arêtes supprimées en fonction du nombre d'exécutions de l'algorithme aléatoire dans le graphe de Paris

croisements : $e_{max\_croisement} = argmax_{e \in E}\{f_{nombre\_croisements\_aretes}(e)\}$.
Pour implémenter cette heuristique, nous modifions la fonction de choix d'arête à supprimer dans la méthode générique.

Algorithme 5 choix d'arête à supprimer basé sur les croisements d'arêtes

Entrées *un graphe* G
Résultat : une arête "arête_a_supprimer"
25 $arete\_a\_supprimer = arete\_avec\_le\_plus\_de\_croisement(G)$
26 return $arete\_a\_supprimer$

La complexité de cet algorithme pour un graphe avec n sommets et m arêtes est $O(m^2 + m * n)$ car on itère le processus de suppression d'une arête au plus m fois, et à chaque itération on exécute l'opération de recherche d'arêtes avec le plus de croisement qui est de coût $O(m)$, le test de planarité de complexité $O(n)$ [8] et le processus de suppression d'arête est de complexité $O(n)$.

Nous pouvons aussi améliorer le temps de calcul de cette implémentation en utilisant un test de planarité dynamique. Ainsi, La complexité de la méthode de test de planarité dynamique est $O(log(n)^2)$ à chaque itération et la complexité de la recherche de l'arête avec le plus de croisement est en $O(m)$ donc la complexité globale est $O(m * (m + log(n)^2))$ soit $O(m^2 + m * log(n)^2)$.

Avec cette heuristique pour le graphe représentant Paris, nous supprimons 183 arêtes sur 13399 de manière itérative pour que le graphe soit planaire. Parmi ces 183 arêtes supprimées nous pouvons en remettre 55 dans le graphe en le laissant planaire. Donc 128 arêtes ont été retirés du graphe représentant Paris pour le rendre planaire. Donc cette méthode est bien meilleure que la méthode aléatoire car dans notre solution basée sur les croisements d'arêtes, le nombre d'arêtes supprimées est divisé par 20 par rapport à la solution aléatoire (nous avons des résultats similaires pour le graphe de Rouen).

3.3.2 Seconde méthode : planarisation par ajout incrémental de noeuds

Une autre méthode pour planariser le graphe est d'ajouter des noeuds de poids 0 aux intersections d'arêtes dans le graphe plongé dans l'espace.

Soit le graphe G=(V,E), avec $e1 \in E$ et $e2 \in E$ tel que e1 et e2 se croisent dans le plongement spatial de G. On rappelle que le plongement spatial d'un graphe représentant un réseau urbain est sa représentation graphique tel que les noeuds sont disposés sur le plan en fonction des coordonées spatiales du réseaux urbains (dasn cette représentation les arêtes sont représentés comme des segments entre les noeuds). On note n1d et n1g les extrémités de l'arête e1, et n2d et n2g les extrémités de l'arête e2.On supprime les arêtes e1 et e2 de G et on ajoute le noeud n à G. Puis on ajoute les arêtes (n1g,n),(n,n1d), (n2g,n) et (n,n2d).

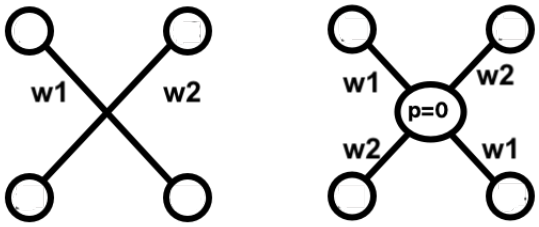


FIGURE 3.6 – Schéma d'ajout de noeud d'un poids 0 à un croisement d'arêtes

Nous prouvons ci-dessous que le graphe planarisé avec cette méthode permet d'avoir une borne supérieure du coût de la coupe optimale équilibrée du graphe initial (non planaire).

Théorème 3.3.2. Soit un graphe G=(V,E) non planaire et G' le graphe obtenu en ajoutant des noeuds de poids 0 aux intersections d'arêtes dans le graphe plongé dans l'espace.Si C est le coût de la coupe équilibrée optimale de G et C' le coût de la coupe équilibrée optimale de G', alors $C \leq C'$.

La preuve de ce théorème est dans l'annexe A.
Nous cherchons donc un graphe planaire obtenu par ajout de noeud sur les arêtes du graphe initial. Pour cela nous allons utiliser la méthode générique suivante avec laquelle nous créerons des variantes :

Algorithme 6 Méthode générique d'ajout incrémental de noeud pour planariser le Graphe

Entrées *un graphe G*
Résultat : G planaire

```
27 le_graphe_n_est_pas_planaire = le_graphe_n_est_pas_planaire(G)
28 while le_graphe_n_est_pas_planaire do
29 |   liste_de_croisements_auquel_on_ajoute_un_noeud = choix_de_croisements(G)
   |      /* on choisit une liste de croisements auxquels on ajoute des noeuds */
30 |   G = ajout_des_noeuds(liste_de_croisements_auquel_on_ajoute_un_noeud)
31 end
32 return G
```

Nous modifions la fonction choix_de_croisements pour créer un algorithme de planarisation du graphe par ajout incrémental de noeuds.

D'abord nous effectuons le choix de croisements de manière aléatoire, c'est à dire que la fonction choix_de_croisements va renvoyer un croisement de manière aléatoire. La complexité de cet algorithme ,dans le cas aléatoire, pour un graphe avec n sommets et m arêtes est $O(m! * n)$ car on exécute le processus d'ajout de noeuds au plus $m!$ fois car il y au plus $m!$ croisements dans un graphe à m arêtes. À chaque itération on exécute l'opération d'ajout de noeud à un croisement aléatoire qui est de complexité $O(n)$ car ce processus consiste en supprimant l'arête où il y a le croisement pour la remplacer par deux arêtes. Enfin, le test de planarité est celui de la librairie Networkx ("Left-Right Planarity Test") de coût $O(n)$.

Nous pouvons améliorer le temps de calcul de cette implémentation en utilisant un test de planarité dynamique. Cela permet de vérifier la planarité du graphe après la suppression d'une arête sans le ré-analyser en entier avec la méthode "Left-Right Planarity Test". La complexité de cette méthode est $O(log(n)^2)$ à chaque itération, la complexité de la méthode aléatoire est donc $O(m! * (n + log(n)^2))$ soit $O(m! * n)$.

Nous affichons le plus petit nombre de noeuds ajoutés avec cet algorithme en fonction du nombre d'exécutions.

Nous voyons qu'après 200 exécutions de l'algorithme, le nombre minimum de noeuds ajoutés pour atteindre la planarisation stagne à environ 98 noeuds.

On applique une heuristique à cette méthode générique : à chaque itération, on choisit les croisements qui sont présents sur l'arête avec le plus de croisements, c'est à dire l'arête $e_{max\_croisement}$.

Algorithme 7 choix de croisement non aléatoire

Entrées *un graphe G*
Résultat : un croisement d'arêtes

```
33 e_max_croisement = arete_avec_le_plus_de_croisement(G)
34 croisements_auquel_on_ajoute_un_noeud = croisements_de_l_arete(e_max_croisement)
      /* La fonction renvoie les croisements de l'arête en argument, c'est à
      dire e_max_croisement                                                  */
35 return croisements_auquel_on_ajoute_un_noeud
```

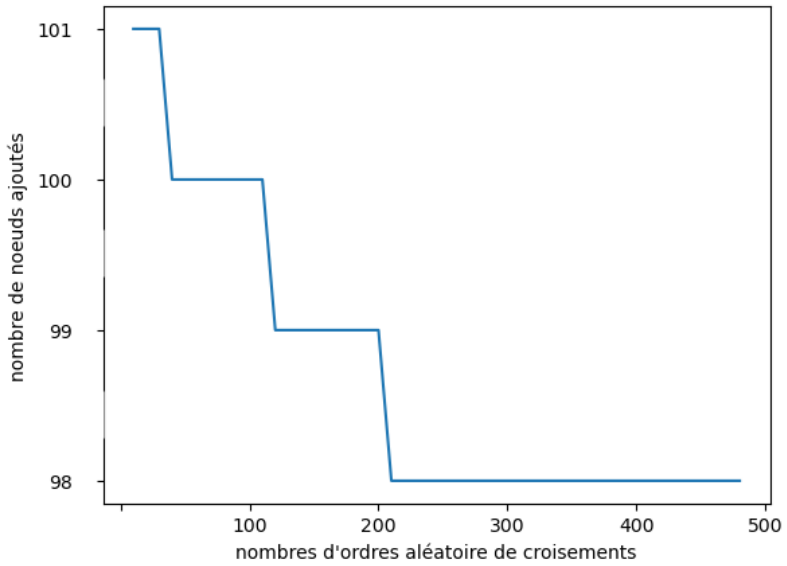
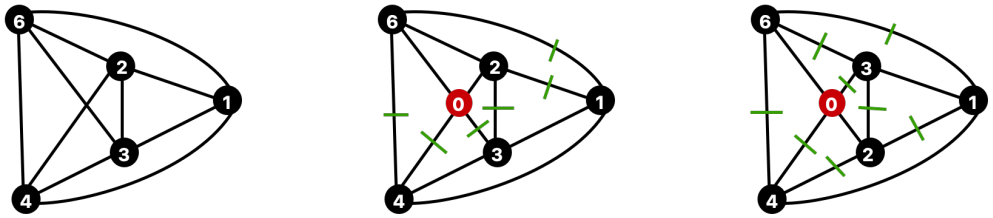


FIGURE 3.7 – plus petit nombre de noeuds ajoutés en fonction du nombre d'exécutions de l'algorithme aléatoire

La complexité de cet algorithme pour un graphe avec n sommets et m arêtes est $O(m^2 * n)$ car on exécute le processus d'ajout de noeud au plus m fois. Et à chaque itération, on exécute l'opération de recherche d'arêtes avec le plus de croisement qui est de coût $O(m)$. Le processus d'ajout de noeuds est de coût $O(m * n)$ car chaque arête a au plus $m - 1$ croisements et à chaque ajout de noeud on supprime une arête pour en ajouter 2 ce qui est de complexité $O(n)$.Et le test de planarité est celui de la librairie Networkx ("Left-Right Planarity Test") de coût $O(n)$. Donc la complexité est $O(m * (m + m * n + n)$, soit $O(m^2 * n)$.
Nous pouvons améliorer le temps de calcul de cette implémentation en utilisant un test de planarité dynamique de complexité $O(log(n)^2)$ à chaque itération.Donc la complexité globale est $O(m * (log(n)^2) + m * n + n)$ soit $O(m^2 * n)$.

Pour le graphe représentant Rouen, il y a 101 croisements spatiaux et avec l'algorithme incrémental, 100 noeuds sont ajoutés au graphe (à 100 croisements) pour que le graphe soit planaire. On voit que cet algorithme ne permet pas de réduire de manière significative le nombre de noeuds à ajouter pour planariser par rapport à la solution qui est d'ajouter un noeud à chaque croisement. Donc notre solution avec 100 noeuds ajoutés n'est pas optimale et notre heuristique est peu efficace car la méthode aléatoire nous donne une meilleure solution. Par comparaison, dans le graphe représentant Paris, il y a 514 croisements, et avec notre heuristique nous ajoutons 512 noeuds.

On remarque aussi que changer la disposition des noeuds dans l'espace peut modifier le coût de la coupe équilibrée après la planarisation par ajout de noeuds au croisements.



(a) (b) (c)

FIGURE 3.8 – Le changement de disposition spatiale des noeuds peut modifier le coût de la coupe équilibrée après la planarisation par ajout de noeuds au croisements

Soit le graphe G représenté en (a) avec un poids de 1 à chaque noeud. On voit dans le schéma (b) que si on planarise ce graphe en ajoutant un noeud (en rouge, de poids 0) la coupe équilibrée optimale serait la coupe représentée en vert de cout minimum 6.On voit dans (c) que si on échange les noeuds de poids 2 et 3 dans l'espace, puis on place le noeud au croisement pour planariser (en rouge, de poids 0), le cout minimum de la coupe équilibrée sera 8.

3.3.3 Encadrement du coût de la coupe équilibrée du graphe initial

Nous avons donc montré l'encadrement du coût de la coupe équilibrée optimale du graphe initiale par les coût des coupes des graphes planarisés avec les deux méthodes présentées.

Théorème 3.3.3. Soit un graphe non planaire G dont le coût de la coupe équilibrée optimale est C. La méthode de suppression d'arêtes pour planariser G permet d'avoir le graphe planaire G' dont le coût de la coupe équilibrée optimale est C'. Et la méthode d'ajout de noeuds pour planariser G permet d'avoir le graphe planaire G" dont le coût de la coupe équilibrée optimale est C''. On a alors $C' < C < C''$.

3.4 Application d'un algorithme de coupe équilibrée polynomiale

Les deux méthodes qui nous avons vues nous permettent d'avoir un graphe planaire. Nous pouvons donc exploiter cette propriété de planarité en appliquant un algorithme de coupe équilibrée de complexité polynomiale sur ce graphe. On rappelle que le problème de la coupe équilibrée est NP-difficile pour les graphes généraux et polynomiale[1] pour les graphes planaires [2].

3.4.1 Un algorithme polynomiale développé par James Park et Cynthia Phillips (1993)

Une partie de ce stage a été dédié à étudier les travaux déjà effectué dans le domaine des coupes équilibrées des graphes planaires mais aucune implémentation d'algorithme permettant de calculer de telles coupes n'a été trouvé [1].
Nous avons donc implémenté un algorithme de coupe décrit dans l'article "Finding minimum-quotient cuts in planar graphs" écrit par James Park et Cynthia Phillips[2]. Cet article décrit un algorithme qui permet de calculer en un temps polynomial une coupe dans un graphe planaire qui minimise le quotient $C_C/min(\omega(X), \omega(Y))$, si C est une coupe du graphe G en deux sous-ensembles X et Y de coût C_C, c'est à dire de calculer la sparsest-cut. Soit W la somme des poids des noeuds du graphe en entrée et m le nombre d'arêtes du graphe, la complexité de cet algorithme est $O(m^2 W log(mW))$. La minimisation de ce quotient donne une coupe équilibrée du graphe car nous minimisons le cout de la coupe (nominateur du quotient) et nous maximisons la taille du plus petit sous-ensemble de la partition (dénominateur du quotient), ce problème fait partie des problème NP-difficiles pour les graphes généraux[4].

3.4.2 Description générale de l'algorithme

L'algorithme implémenté se divise en deux parties principales : les transformations du graphe planaire en entrée et la recherche de la meilleure coupe dans le graphe modifié en calculant une table T qui énumère une partie des coupes possibles.

Nous rappelons la définition d'un graphe dual :

Definition 5 (graphe dual). Etant donné un graphe $G = (V, E)$ associé à un plongement planaire, chaque composante connexe (ou cellule) dans ce plongement est munie d'un point définissant un sommet du graphe dual. Chaque arête du graphe initial définit une arête du graphe dual reliant les composantes (ou cellules) qui la bordent.

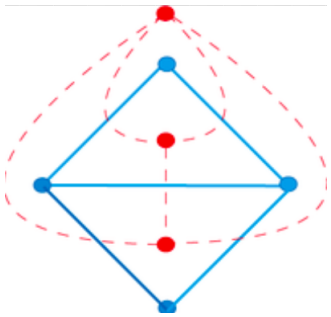


FIGURE 3.9 – En bleu le graphe primal et en rouge le graphe dual.

Théorème 3.4.1. Une coupe dans un graphe planaire correspond à un cycle simple dans son graphe dual, où les deux sous-ensembles de noeuds de la coupe correspondent aux cellules à l'intérieur et à l'extérieur du cycle, l'ensemble des arêtes de la coupe dans le graphe primal correspond aux arêtes associées dans le graphe dual.

1. Les chercheurs Shay Mozes (un des co-créateurs du site planarity.org), Christian Schulz et Alexander Noe (contributeurs de la librairie de partition de graphes KaHIP[9]) ont été contacté mais ne voient pas d'implémentation d'algorithmes calculant une telle coupe.

La recherche d'une coupe qui minimise le quotient vus précédemment correspond donc à la recherche d'un cycle dans le graphe dual qui minimise ce quotient.

L'algorithme est subdivisé en ces étapes avec le graphe planaire G en entrée :

(a) Création du graphe dual de G, Gd.

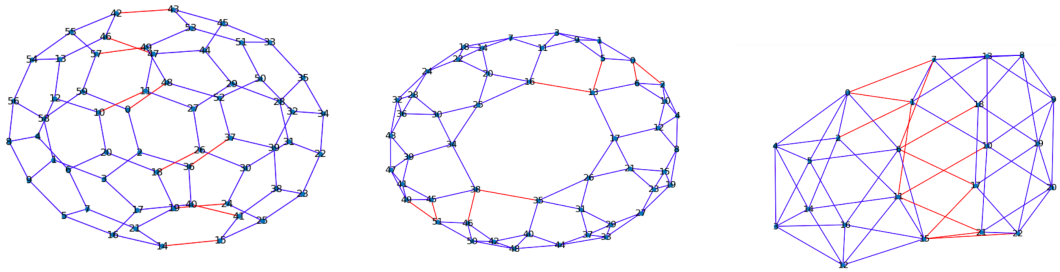
(b) Modification de Gd en un graphe orienté Gs permettant de calculer rapidement la somme des poids des cellules à l'intérieur et l'extérieur d'un cycle dans Gs (ce qui correspond à connaitre le poids des sous-ensemble dans la coupe correspondante dans G).

(c) Énumération des cycles simples dans Gs. Cette énumération est de complexité $O(m^2Wlog(mW))$, m étant le nombre d'arêtes de G et W la somme des poids des noeuds de G.

(d) Parmi les cycles énumérés, on choisit le cycle C qui minimise le quotient.

(e) À partir du cycle C dans Gs on en déduit un cycle C' dans Gd. A partir du cycle C' dans le graphe dual, on en déduit la sparsest-cut dans G.

Nous présentons des exemples de coupes équilibrées de graphes calculés avec cet algorithme :



(a) quotient = 2/3 (b) quotient = 8/13 (c) quotient = 24/11

FIGURE 3.10 – 3 exemples de coupes équilibrées calculées avec l'algorithme de Park et Phillips : en rouge les arêtes de la coupe, chaque noeud a un poids de 1 et chaque arête a un coût de 2

3.4.3 Comparaison de la méthode développée avec une heuristique de calcul de coupe équilibrée pour les graphes généraux

L'algorithme de Park et Phillips que nous avons implémenté calcule en un temps polynomial la sparsest-cut du graphe planaire en entrée, et nous avons montré que les méthodes permettant de simplifier et planariser le graphe initial représentant le réseaux urbain ont une complexité polynomiale. Nous pouvons déduire de la coupe du graphe planarisé une coupe dans le graphe initial, donc notre méthode de calcul de sparsest-cut pour des graphes représentant des réseaux urbains est de complexité polynomiale. La complexité de cette méthode correspond à la complexité de l'algorithme de Park et Phillips, c'est à dire $O(m^2Wlog(mW))$.

Cette méthode a été appliquée au graphe représentant Paris mais après les étapes de simplification et de planarisation, l'algorithme de Park et Phillips nécessite trop de temps de calcul pour obtenir un résultat. Cela s'explique par une étape de l'algorithme qui consiste à créer un graphe "étendu" du graphe Gd. Pour le graphe de Paris, ce graphe étendu a 123 millions d'arêtes et 27 millions de noeuds. Il faut ensuite calculer des plus court chemins (grâce à l'algorithme de Djikstra) dans ce graphe "étendu" mais dans ce cas, chaque chemin dans ce graphe met plus d'une heure à calculer. Donc nous n'avons pas de résultat pour le graphe représentant Paris (ou d'autres villes) avec notre méthode. Nous pouvons améliorer cette étape de l'algorithme de recherche de meilleure cycle en appliquant sur le graphe Gd un algorithme d'énumération de tous les cycles simples plus efficace. Ainsi Donald Johnson[10] a décrit un tel algorithme d'énumération de cycles simples borné dans le temps et dans l'espace de façon polynomiale.

Nous avons vu que la planarisation du graphe en entrée (simplifié) par une des deux méthodes présentées fait que la sparsest-cut du graphe planarisé ne correspond pas forcément à la sparsest-cut du graphe initial. Notre méthode de calcul de coupe équilibrée est donc une **une méthode approchée**. Nous comparons cette méthode approchée avec d'autres méthodes approchées de calcul de coupes équilibrées dans des graphes généraux.

Nous comparons ainsi cet algorithme avec une heuristique de coupe équilibrée pour les graphes généraux (nommée "KaFFPa") appartenant à la famille de programmes de partitions KaHIP[9]. Nous appliquons cette heuristique aux graphes représentant des villes comme Rouen ou Paris et l'heuristique renvoie en environ 0.5 secondes une coupe très équilibrée (voir exemple ci-dessous avec la graphe non simplifié de Paris) et de faible coût par rapport à la taille du graphe et de l'équilibrage de la coupe.

FIGURE 3.11 – coupe équilibrée du graphe représentant Paris. Coût : 76, les sous-ensembles de noeuds sont de poids 4809 et 4807 (quasi-égalité), temps de calcul : 0.5 secondes

Donc la performance de notre algorithme de minimisation de quotient est très inférieure aux heuristiques de coupes équilibrées déjà existantes.

3.4.4 Étude de l'évolution de l'équilibre d'une coupe en fonction de son coût

Nous avons ensuite voulu étudier l'évolution de l'équilibre maximal d'une coupe en fonction de son coût maximum. Cela est en lien avec notre objectif de modélisation de perturbation du réseaux urbain car lors de blocages de rues par des individus, il y a un nombre de voies maximal que ces individus peuvent bloquer (le "coût maximal"), car ils sont limité par leur nombre.
Nous essayons de visualiser cette évolution pour le graphe représentant Paris. Pour cela nous utilisons l'heuristique KaHip (qui possède un paramètre *imbalance* permettant de varier l'équilibre de la coupe) qui permet de calculer des coupes équilibrées.
Nous faisons l'hypothèse que pour un équilibre désiré de la coupe, la solution renvoyée par KaHip est proche de l'optimal, c'est à dire que la coupe renvoyée vérifie l'équilibre désiré et est de coût proche du coût minimal pour cet équilibre.
En faisant varier le paramètre *imbalance* pour KaHip, nous pouvons construire une courbe qui associe le meilleur équilibre de coupe possible en fonction du coût maximal de la coupe. Ici l'ordonnée est le déséquilibre de la coupe, c'est à dire pour une coupe $(C1, C2)$, le déséquilibre est égale à $max(|C1|, |C2|) * 2/(|C1| + |C2|)$.

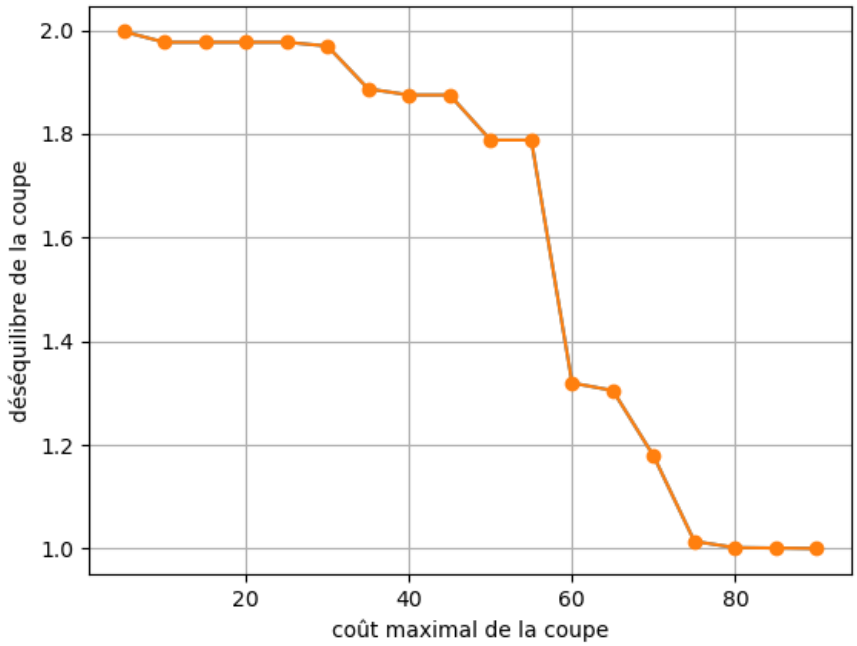


FIGURE 3.12 – Évolution du déséquilibre de la coupe en fonction du coût maximal

Nous observons que le déséquilibre diminue avec le coût maximal de la coupe car une coupe très équilibrée de coût minimal a un coût supérieur à une coupe peu équilibrée de coût minimal. Nous observons aussi une zone de la courbe où une petite augmentation du coût maximal (de coût 55 à 60) est associée à une forte diminution du déséquilibre de la coupe (de 1.8 à 1.5). En effet, la coupe de coût maximal 60 et de déséquilibre 1.5 est la solution qui consiste à couper tout les ponts dans Paris, comme la taille des sous-ensembles de noeuds situés de part et d'autres des ponts sont proches c'est une solution qui fait fortement diminuer le déséquilibre de la coupe.

Chapitre 4

Conclusion

L'objectif de ce stage est d'étudier l'impact potentiel d'actions de blocages dans les réseaux urbains en effectuant des coupes équilibrées sur les graphes représentant ces réseaux. Nous avons fait l'hypothèse que ces types de graphes sont quasiment planaires, c'est à dire que des modifications peu importantes peuvent rendre ces graphes planaires, ce qui permet d'appliquer des algorithmes de complexité polynomiale sur ces graphes . Cette hypothèse a été vérifiée par les fonctions de planarisations des graphes qui ont été developpés et qui ont peu modifiés les graphes pour les rendre planaires.
Afin d'obtenir les graphes représentant des réseaux urbains, nous avons importés des données géographiques de villes venant d'OpenStreetMap. Les graphes initiaux ont été simplifiés en supprimant itérativement les noeuds qui ne sont pas des intersections (c'est à dire les noeuds de degré 1 et 2). Nous planarisons ces graphes simplifiés grâce à deux méthodes : l'ajout de noeuds aux intersections et la suppression d'arêtes qui se croisent. Le graphe planaire obtenu nous permet d'appliquer un algorithme de recherche de coupe de plus petit quotient de complexité polynomiale alors que le problème de recherche de coupe de plus petit quotient est NP-difficile pour les graphes généraux.

Mais malgré sa complexité polynomiale, l'application de cet algorithme aux graphes représentant des villes est très lente (plusieurs heures) et n'est pas performante par rapport aux heuristiques de coupes équilibrées de coupes généraux qui existe déjà.

Une perspective de ce stage serait d'implémenter des heuristiques de coupes équilibrées qui utilisent la propriété de planarité pour les graphes planaires (l'algorithme déjà implémenté est un algorithme de recherche de coupe équilibrée exacte) car il existe des heuristiques de coupes équilibrées qui ont une complexité assez faible notamment pour la coupe b-balanced[5]. Ces heuristiques pourraient être comparées aux heuristique de coupes équilibrées pour les graphes généraux.

Bibliographie

[1] L. Stockmeyer M.R. Garey D.S. Johnson. *Balanced Graph Partitioning*. Theoretical Computer Science Volume 1, Issue 3, February 1976, Pages 237-267, 1976 (pages 1, 3, 15).

[2] Cynthia A. Phillips James K. Park. « Finding Minimum-Quotient Cuts in Planar Graphs ». In : (1993) (pages 2, 3, 15, 16).

[3] JGeoff Boeing. « OSMnx : New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks ». In : (2017) (page 3).

[4] F. Shahrokhi D.W. Matula. « The maximum concurrent flow problem and sparsest cuts ». In : (1986) (pages 3, 16).

[5] Henning Meyerhenke Aydin Buluç et Christian Schulz. *Recent advances in Graph Partitioning*. LNTCS volume 9220, 2010 (pages 3, 20).

[6] JGeoff Boeing. « Planarity and Street Network Representation in Urban Form Analysis ». In : (2020) (page 3).

[7] Thomas Ottmann Jon Louis Bentley. *Algorithms for Reporting and Counting Geometric Intersections*. IEEE Transactions on Computers (Volume : C-28, Issue :9), 1979 (page 9).

[8] Ulrik Brandes. *The Left-Right Planarity Test*. Manuscript submitted for publication, 2009 (page 11).

[9] Peter Sanders et Christian Schulz. « Think Locally, Act Globally : Highly Balanced Graph Partitioning ». In : *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*. T. 7933. Springer, 2013, p. 164–175 (pages 16, 18).

[10] Donald B Johnson. « Finding all the elementary circuits of a directed graph ». In : *SIAM Journal on Computing* 4.1 (1975), p. 77–84 (page 18).

Annexe A

Preuves détaillées

Preuve du théorème 3.3.1 :
Soit un graphe G=(V,E) non planaire et G' le graphe planaire obtenu en supprimant des arêtes de G. Si C est le coût de la coupe équilibrée optimale de G et C' le coût de la coupe équilibrée optimale de G', alors $C' \leq C$.

Soit G le graphe initial avec n sommets et m arêtes, non planaire, et P = {A,$\bar{\text{A}}$} la coupe équilibrée de coût minimum de G, c'est à dire $\sum_{s\in A} poids(s) \approx \sum_{s'\in \bar{A}} poids(s')$. Et soit S = $\{e_1, e_2, ..., e_t\}$ les arêtes de G ayant une extrémité dans A et l'autre extrémité dans $\bar{\text{A}}$. Le coût C de la coupe équilibrée est donc C = $\sum_{e\in S} cout(e)$.
Soit G' le graphe planarisé avec la méthode de suppression d'arêtes et E l'ensemble d'arêtes de G supprimées.
Si $E \cap S = \emptyset$ la coupe P = {A,$\bar{\text{A}}$} est un coupe équilibrée de G' de coût C.
Si $E \cap S \neq \emptyset$ la coupe P = {A,$\bar{\text{A}}$} est un coupe équilibrée de G' de coût C'<C car C' = $\sum_{e\in S} cout(e) - \sum_{e'\in E\cap S} cout(e')$.
Donc G' a une coupe équilibrée de coût inférieure ou égale au coût de la coupe équilibrée de G. Donc la méthode de planarisation par suppression d'arêtes permet d'avoir une borne inférieure du coût de la coupe équilibrée.

Preuve du théorème 3.3.2 :
Soit un graphe G=(V,E) non planaire et G' le graphe obtenu en ajoutant des noeuds de poids 0 aux intersections d'arêtes dans le graphe plongé dans l'espace.Si C est le coût de la coupe équilibrée optimale de G et C' le coût de la coupe équilibrée optimale de G', alors $C \leq C'$.

Nous prouvons qu'un graphe planarisé avec cette méthode permet d'avoir une borne supérieure du coût de la coupe optimale équilibrée du graphe initial (non planaire).
Soit G le graphe initial et G' le graphe planarisé à partir de G par ajout de noeuds. On suppose qu'il existe une coupe équilibrée de G' de coût minimal : P'={A,$\bar{\text{A}}$} et S' = $\{e'_1,e'_2,...,e'_t\}$ l'ensemble des arêtes de G' ayant une extrémité dans A et l'autre extrémité dans $\bar{\text{A}}$. le coût de la coupe P' est donc $C' = \sum_{e\in S'} cout(e)$. Nous associons à chaque arête e' de S, une arête dans G de la manière suivante : si e' est une arête de G, on lui associe elle-même (e') et si e' n'est pas une arête de G alors c'est une arête dont au moins une des extrémités est un sommet qui a été ajouté lors de la planarisation, on associe alors à e', l'arête e de G auquel on a rajouté des sommets pour planariser G et qui a crée l'arête e'.On a dans les deux cas $cout(e') = cout(e)$ d'après notre méthode de planarisation. Nous

avons donc une ensemble d'arêtes S = $\{e_1$,e_2,...,$e_t\}$ de G associé à l'ensemble d'arêtes S'. S défini une coupe C de G, tel que P=$\{B, \bar{B}\}$. Soit $C = \sum_{e \in S} cout(e)$ le coût de P, on a $C \leq C'$ et l'équilibre des coupes C et C' sont identiques car les ensembles d'arêtes S et S' séparent les mêmes ensembles de noeuds de G (les noeuds ajoutés dans G' sont de poids 0 donc on ne les prend pas en compte) donc l'ensemble des sommes des poids de B et de $\bar{B}$ dans la coupe C est égale à l'ensemble des sommes de poids de A et $\bar{A}$ dans C'.
Donc pour chaque coupe équilibrée dans le graphe planarisé avec ajout de noeuds G', il existe une coupe de même équilibre et de coût inférieur ou égal dans G. Donc si C est le coût de la coupe équilibrée optimale de G et C' le coût de la coupe équilibrée optimale de G', alors $C \leq C'$.