

ÉCOLE NORMALE SUPÉRIEURE DE LYON

CASSER DES GRAPHES

Rapport de stage au LIP6
15 août 2024

Corto Cristofoli

Sous la direction de Clémence Magnien et Matthieu Latapy
Encadré par Alexis Baudin et Bastien Legay



Table des matières

1	Introduction	1
1.1	Définitions	2
1.2	Méthode de partitionnement de graphes	2
1.3	Paris et les graphes urbains	3
2	Coupes itératives rendues successivement insécables	3
2.1	Problématisation	3
2.2	Algorithme d'itérations successivement insécables	4
2.3	Évolution des coupes au cours des itérations	5
2.4	Étude du coût des coupes itératives	7
2.5	Diversité des coupes itératives	8
2.5.1	Distance entre deux coupes	8
2.5.2	Diversité d'un ensemble de coupes	9
2.5.3	Analyse de la diversité des coupes obtenues par l'algorithme	9
3	Coupes résistantes sur les graphes urbains	10
3.1	Définition d'une coupe résistante	10
3.2	Algorithme de coupe résistante	10
3.2.1	Preuve formelle de l'algorithme	11
3.3	Optimisations de l'algorithme	12
3.3.1	Optimisation des copies du graphe	12
3.3.2	Structure de donnée d' <i>Union-Find</i>	12
3.3.3	Élagage des r -uplets	13
3.4	Analyse d'une coupe résistante	14
3.5	Contraintes sur les manières de couper	15
4	Conclusion	17
A	Situation et déroulement du stage	17
B	Exemples de coupes résistantes	18

1 Introduction

De nombreux mouvements sociaux et écologistes recourent à des tactiques disruptives comme les **blocages de rues** ou les sabotages d'infrastructures pour faire entendre leurs revendications [Mal20]. Des groupes tels que la CGT et *Dernière Rénovation* utilisent des actions ciblées, impliquant un petit nombre d'individus, pour paralyser la circulation dans certaines zones. D'autres activistes s'attaquent directement aux réseaux de communication en sectionnant des câbles ou en endommageant des antennes.

Les conséquences réelles ou potentielles de ces actions sont encore mal comprises et peu étudiées. Par exemple, peut-on isoler complètement un quartier ou une ville ? Quels sont les moyens et les coûts nécessaires pour atteindre un tel objectif ? Comment mesurer l'ampleur des perturbations causées par ces attaques ?

En informatique, cette question peut être envisagé à l'aide de la théorie des graphes et, plus précisément, via la problématique de la **coupe de graphes**. Une coupe est un ensemble d'éléments (arêtes ou sommets) dont la suppression divise un réseau en plusieurs parties distinctes.

Le but de ce stage, réalisé dans l'équipe *Complex Network* du Laboratoire d'Informatique de Paris 6 (LIP6), entre le 3 juin et le 2 aout 2024, a été d'analyser l'impact potentiel des blocages dans les réseaux urbains, en s'appuyant sur des données réelles détaillées à l'échelle de la capitale parisienne et en utilisant des algorithmes de coupes de graphes performants. Toute la démarche aura alors de comprendre et d'analyser des méthodes de coupes de Paris et de comprendre comment rendre ces attaques plus efficaces.

Après avoir présenté le cadre et les définitions de l'expérience nous analyserons d'abord un moyen de **diversification des attaques de graphes** avant d'étudier comment rendre ces mêmes attaques plus **résilientes aux perturbations** potentielles.

1.1 Définitions

Le stage, intitulé "Casser des Graphes", s'intéresse à la coupe de graphes particuliers : les graphes urbains.

Définition 1 (Graphe) *Un graphe est un ensemble de n sommets V et un ensemble de m arêtes E . Dans notre étude, ce graphe possède plusieurs caractéristiques :*

1. *Les arêtes sont **pondérées** et ne sont **pas orientées**.*
2. *Chaque sommet a une valeur $v \in \mathbb{N}$: ils sont **valués**.*
3. *Le graphe est **simple** : il n'y a pas d'arête connectant un sommet à lui-même ni plusieurs arêtes distinctes connectant les 2 mêmes sommets.*

Généralement, les graphes que nous utilisons sont des graphes de réseau urbain, qui ne sont pas nécessairement connexes.

Définition 2 (Partition de sommets) *Soit G un graphe, une **partition des sommets** de G est un ensemble de sous-ensemble V_1, \dots, V_k ($k \geq 2$) de V tel que les $(V_i)_{i \in [k]}$ sont distincts deux à deux et $\bigcup_{i \in [k]} V_i = V$.*

Définition 3 (Coupe) *Soit G un graphe et $k \geq 2$, soit V_1, \dots, V_k une partition des sommets de G . On définit la **coupe** C sur la partition des sommets comme le sous-ensemble de E tel que :*

$$C := \{(u, v) \in E \mid \exists i \neq j, u \in V_i \wedge v \in V_j\}$$

Remarquons que l'application qui à une partition associe une coupe est injective (grâce à la définition constructive de la coupe ci-dessus) mais sa réciproque ne l'est absolument pas. À partir d'une coupe C donnée, on peut construire de multiples partitions différentes. C'est pourquoi l'algorithme *KaFFPa* que nous utilisons pour réaliser nos coupes durant ce stage, renvoie, en pratique, une partition des sommets du graphe.

1.2 Méthode de partitionnement de graphes

Le travail réalisé s'appuie sur des travaux précédents en matière de partitionnement de graphes. Durant toute la durée du stage, il a fallu comprendre puis utiliser *KaHIP* (*Karlsruhe High Quality Partitioning* [SS13]), un outil de partitionnement de graphe me permettant de réaliser des coupes de graphe de manière rapide. Il s'agit d'une bibliothèque contenant des heuristiques de coupes de graphes, que nous étudions dans le contexte des réseaux urbains.

KaHIP, ou plus précisément *KaFFPa* (*Karlsruhe Fast Flow Partitioner*, inclus dans l'outil) est un algorithme de coupe de graphe, prenant en paramètre un graphe G et un réel $0 < x \leq 1$, valeur de déséquilibre. L'algorithme renvoie une partition de V en deux ensembles V_1 et V_2 telle

que la différence de taille proportionnelle entre V_1 et V_2 est inférieure à x (plus x est proche de 0 plus ces deux ensembles seront de tailles similaires) et telle que la coupe C en résultant est de coût presque minimal. C'est une heuristique donc cette coupe n'est pas nécessairement optimale. Les principaux intérêts de cette méthode sont la vitesse de la coupe (une demi seconde par coupe de Paris) et l'aspect non-déterministe des coupes obtenues. Lancer l'algorithme plusieurs fois sur un même graphe donne des coupes différentes, ce qui est intéressant pour notre démarche d'analyse et de recherche de coupes diverses et variées sur une ville. C'est pourquoi nous utilisons cet algorithme par la suite.

1.3 Paris et les graphes urbains

La ville étant le principal théâtre des diverses attaques et blocages, nous avons choisis d'étudier ces diverses méthodes de coupes dans les graphes urbains. Nous nous sommes concentré plus précisément sur le graphe parisien. Ce choix s'explique par plusieurs raisons :

1. C'est la capitale de la France, et c'est donc un point stratégique de blocage pour divers groupes souhaitant avoir un impact.
2. C'est une ville bien délimitée par le périphérique, ce qui permet donc d'avoir une zone d'étude circonscrite spatialement et de réduire certains effets de bords qui pourraient être causées par le choix de bordures arbitraires.
3. La ville est assez petite, permettant des calculs relativement rapides, notamment pour la génération de coupes.
4. La ville a une topographie singulière et presque organique. La Seine, au centre, constitue une barrière naturelle et il y a une diversité dans la structure des rues, entre les grands boulevards haussmanniens et les nombreuses ruelles. L'étude a ainsi plus d'intérêt pour une analyse informatique que Manhattan par exemple, dont le plan est géométrique et homogène.

Pour construire le graphe de Paris que nous utilisons pour la suite, nous utilisons le module de python *OSMnx* [Boe17] afin de récupérer les données d'*OpenStreetMap*. Une fois ce graphe récupéré, il est nécessaire, avant toute utilisation, de rendre le graphe non orienté en transformant les arêtes multiples en arêtes simples, afin que le graphe puisse être utilisé par l'algorithme *KaFFPa* (ne fonctionnant pas pour les graphes orientés). La valeur attribuée à chaque sommet est ainsi mise à 1 tandis que le poids des arêtes varie, selon la taille des routes, pouvant aller 1 à 10 environ.

Une fois le graphe de Paris traité, on se retrouve avec un graphe comprenant 40 593 sommets et 46 759 arêtes avec un degré moyen de 2.3.

C'est dans ce graphe de Paris que nous analysons nos différentes attaques, en commençant par une méthode de diversification de ces dernières afin de rendre les défenses plus difficiles.

2 Coups itératives rendues successivement insécables

2.1 Problématisation

L'algorithme *KaFFPa*, bien que non déterministe, propose des coupes relativement similaires. Cela s'explique par la topographie parisienne : la présence de la Seine au centre de la ville est une barrière naturelle que *KaFFPa* peut utiliser à son avantage afin de proposer une coupe efficace. La grande majorité des coupes que propose l'algorithme a la forme de l'une des deux coupes présentes sur la figure 1.

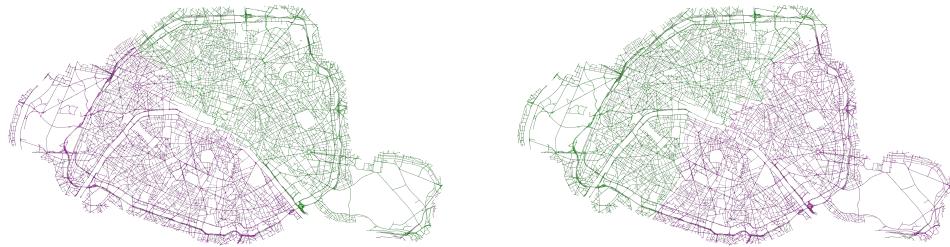


FIGURE 1 – Coupes classiques de Paris produites par *KaFFPa*

Or un des intérêts de notre recherche est d'essayer d'exhiber une plus grande variété de coupes, pour rendre les attaques plus imprévisibles et donc complexifier les réponses à ces dernières. Même si le terme "variété" reste à définir, l'idée est d'essayer de trouver des coupes différentes de celles que l'on obtient par l'algorithme *KaFFPa* seul. Cela suppose d'ajouter des contraintes à *KaFFPa* afin de le "forcer" à produire des coupes différentes, certes plus coûteuses à construire, mais aux formes plus diversifiées. La méthode que nous avons retenue est d'interdire à l'algorithme d'utiliser certaines arêtes, en les rendant "insécables". En effet, puisque l'algorithme ne peut couper ces arêtes rendues insécables, il en est détourné et doit donc en choisir d'autres à couper ; cela modifie la forme globale de la coupe.

Comment choisir les arêtes à rendre insécables pour favoriser l'apparition de coupes différentes, conformément à l'objectif ? Quelle est l'influence de cette méthode sur le coût des coupes obtenues ? Comment définir puis mesurer la diversité des coupes et quels sont les résultats obtenus ?

2.2 Algorithme d'itérations successivement insécables

Si l'on souhaite réaliser un blocage des rues parisiennes et que l'on sait qu'une certaine rue est très intéressante à bloquer¹ mais qu'elle est bien gardée, ne serait-il pas plus profitable de bloquer des rues "de traverse", certes peut-être moins centrales, afin de ne pas gaspiller des ressources à tenter de bloquer cette rue protégée ?

L'algorithme que nous utilisons se fonde sur un concept d'arête insécable, pour représenter des rues protégées que l'on ne peut couper. Prenons un graphe G et appliquons successivement le traitement suivant : réaliser une coupe de G , rendre les arêtes de la coupe obtenue insécables, en remplaçant leur ancien poids par un poids ∞ , puis recommencer avec le graphe mis à jour. Au fur et à mesure de l'algorithme, le nombre d'arêtes insécables augmente jusqu'à être totalement majoritaires dans G . On peut ensuite regarder les coupes obtenues au fil des itérations de la procédure.

Le concept d'arêtes de poids ∞ est théoriquement compréhensible mais n'est pas applicable en pratique. Nous choisissons donc une valeur arbitraire à ces arêtes : $\infty = 10\,000$. Le poids des rues étant compris entre 1 et 10, 10 000 est une valeur assez grande en pratique pour être considérée "insécable". Notons que la somme des poids de toutes les arêtes du graphe peut être supérieure à 10 000 (ce qui est le cas pour Paris) mais que cela ne rend pas l'arête de poids 10 000 moins insécable.

Nous avons utilisé cet algorithme sur Paris. La figure 2 montre l'évolution de Paris au fil des itérations avec les arêtes insécables coloriées en jaune. On voit bien la proportion augmenter, confirmant le fonctionnement de l'algorithme.

1. Une arête peu être intéressante à bloquer si elle est centrale dans le graphe [Bar04]

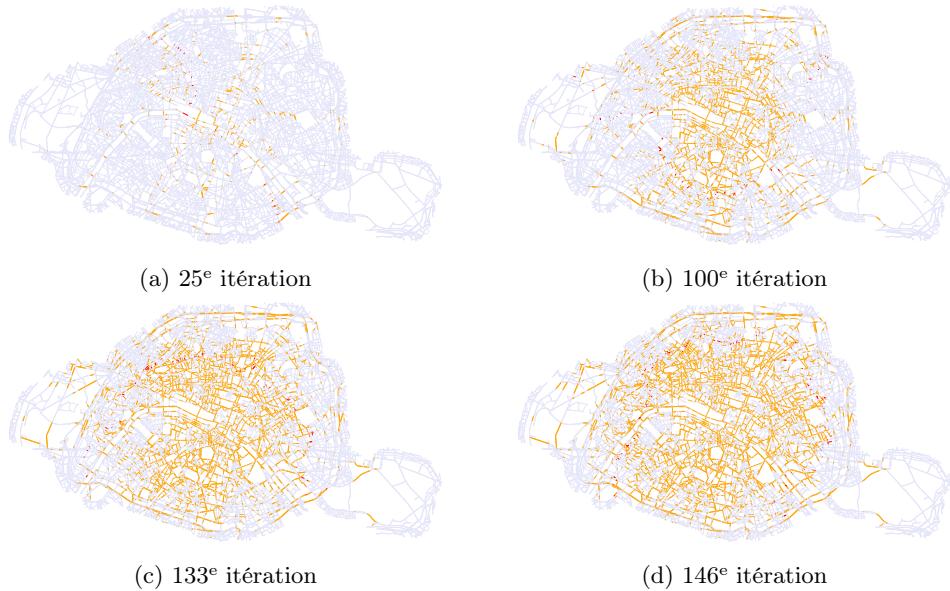


FIGURE 2 – Évolution des arêtes insécables dans Paris au fil des itérations de notre algorithme

Une fois notre algorithme implémenté, le reste du travail a été consacré à l'étude de l'évolution des coupes obtenues au fil des itérations de celui-ci.

2.3 Évolution des coupes au cours des itérations

Constatons premièrement, sur un exemple, l'apparition de coupes différentes de celles produites initialement par *KaFFPa*. On peut voir sur la figure 3 des coupes séparant presque le centre Paris de sa bordure extérieure, tandis que sur la 146^e itération figure désormais une bande centrale verticale qui laisse place à 3 parties distinctes.



FIGURE 3 – Coupes de Paris produites par notre algorithme

Ces coupes diffèrent des coupes classiques vues à la figure 1. Pourtant, à partir d'un certain nombre d'itérations, les coupes redeviennent semblables aux coupes classiques que produit *KaFFPa*. Ce phénomène est en fait un phénomène de **saturation** des arêtes du graphe par les arêtes insécables. À partir d'un certain nombre d'itérations (directement reliées à la structure de la ville), toutes les arêtes utilisables pour créer une coupe sont insécables et pour toute nouvelle

coupe, *KaFFPa* devra utiliser des arêtes insécables pour la construire. Ce moment arrive autour de la 150^e itération pour Paris, comme on peut l'observer sur la figure 4.

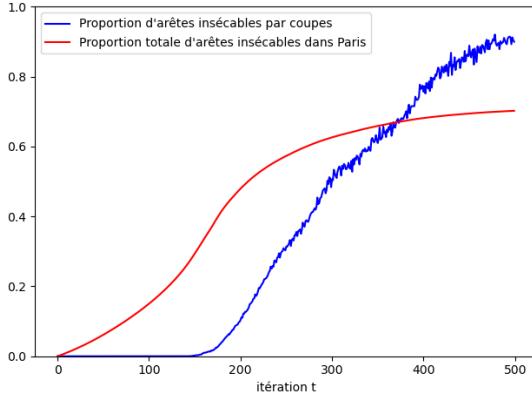


FIGURE 4 – Proportion moyenne d’arêtes insécables dans Paris sur 200 coupes au fil des itérations

Ce sont deux courbes représentant l’évolution de la proportion d’arêtes insécables, l’une dans chaque coupe et l’autre dans tout Paris. On peut observer un deuxième moment critique autour de la 300^e itération : la proportion d’arêtes insécables dans Paris semble converger aux alentours de 70% de l’ensemble des arêtes. Ce moment de stabilisation montre le cas limite de notre algorithme ; c’est le moment où toutes les arêtes utilisables pour créer une coupe sont devenues insécables. À chaque itération, on ne transforme presque plus aucune arête en arête insécable, ce qui explique la stabilisation de cette courbe. À cet instant, les valeurs de toutes les arêtes sont alors homogènes et le poids ∞ n’a plus aucun intérêt puisque, désormais, ces arêtes insécables ne sont plus comparées à des arêtes de poids faible.



FIGURE 5 – Rues parisiennes après 500 itérations de notre algorithme

Cette convergence vers environ 70% des arêtes qui deviennent insécables implique qu’environ 30% des arêtes parisiennes ne seront jamais utilisées dans des coupes. Cela peut s’expliquer par le fait que le graphe parisien produit par *OpenStreetMap* n’est pas connexe et qu’il existe donc des zones indépendantes assez petites pour que *KaFFPa* n’y coupe jamais. Cependant, cette proportion non connexe du graphe représente 0.4% du graphe et, par conséquent, ces 30% ne sont pas seulement dus à cela. Les principales causes sont plutôt les impasses et les petites

boucles telles que, si l'on choisit de couper une rue à ces endroits alors la coupe ne sera pas plus avancée et on devra tout de même couper une arête insécable. Il n'est donc jamais intéressant pour *KaFFPa* de couper ces arêtes puisque toute coupe avec une de ces arêtes sera plus coûteuse que la même coupe sans cette dernière. Sur la figure 5, représentant Paris après 500 itérations de l'algorithme, on voit ce phénomène avec, en jaune, les arêtes insécables et, en gris, les autres.

Ce changement de régime par la saturation progressive de toutes les arêtes est aussi visible lorsque l'on étudie l'évolution du coût des coupes au cours des itérations notre algorithme.

2.4 Étude du coût des coupes itératives

L'algorithme *KaFFPa* cherche à minimiser le coût des coupes réalisées. Étudier comment varie le coût des coupes au fur et à mesure des itérations nous permet de savoir si les coupes proposées – qui sont plus coûteuses que celles initialement proposées – ont quand même un coût acceptable pour envisager de les utiliser à la place des coupes *quasi* optimales. On cherche à savoir s'il n'y a pas une explosion trop rapide du coût de ces coupes.

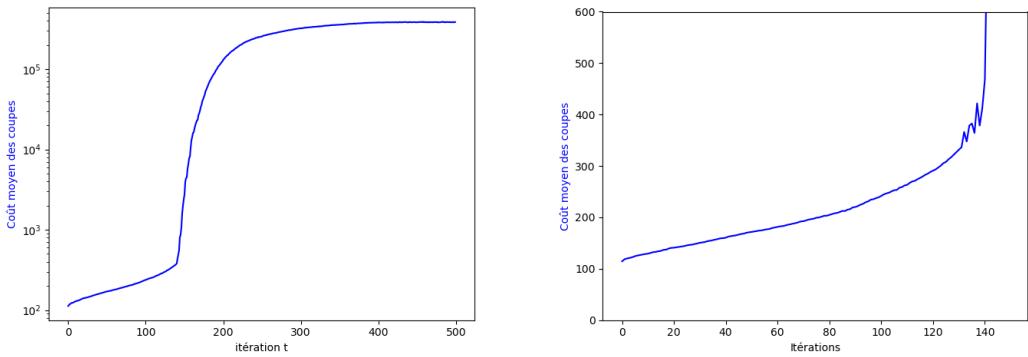


FIGURE 6 – Évolution du coût moyen des 400 coupes réalisées à chaque itération

Sur la première courbe de la figure 6 on observe le coût moyen sur 200 coupes de Paris, obtenues grâce à notre algorithme jusqu'à un total de 500 itérations. On peut y observer les deux moments critiques décrits précédemment. Vers la 150^e itération, on observe un changement brusque dans le coût des coupes : c'est le moment où les coupes commencent à contenir des arêtes insécables. Vers la 300^e itération, on observe le même aplatissement que sur la figure 4. Notons que sur cette figure l'axe des ordonnées est logarithmique, afin de ne pas écraser le coût des premières itérations par rapport à celui des suivantes. L'algorithme propose ainsi une centaine de coupes acceptables, i.e. non saturées. Ce sont ces coupes précisément qui nous intéressent en premier lieu ; c'est pourquoi, la seconde courbe se concentre sur le poids de celles-ci.

Sur cette seconde courbe, l'axe des ordonnées n'est plus logarithmique et on observe une évolution quasi linéaire du coût au fil des itérations, passant d'un coût moyen de 100 à un peu plus de 350 entre la 1^{ère} itération et le premier moment critique. Il y a par exemple environ 80 coupes ayant un coût compris entre 100 et 200, qui peuvent ainsi offrir des options diverses à des personnes capables d'investir jusqu'au double d'effort dans une attaque de ce type. L'augmentation du coût au fil des itérations est donc attestée, mais à quoi ressemble ces coupes plus coûteuses ?

La figure 7 permet d'analyser ce à quoi ressemble ces coupes. Notons premièrement que les coûts moyens des arêtes utilisées sont indépendants des modifications d'insécabilité apportées par l'algorithme durant les itérations. Ces coûts représentent le coût *réel* de l'arête, sans prendre en

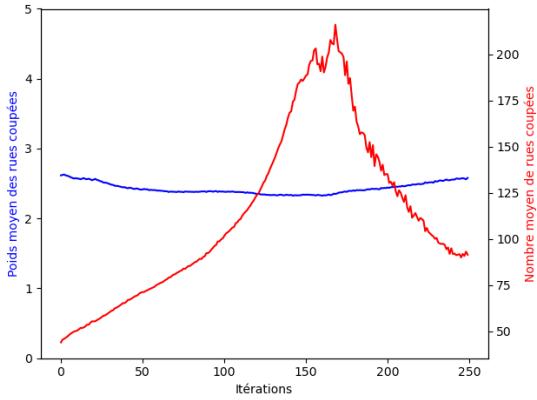


FIGURE 7 – Nombre et poids moyens des arêtes de chaque coupe au fil des itérations (sur 400 coupes)

compte le fait qu'elle est ou non rendue insécable. C'est pour cela qu'il n'y a pas une explosion du coût moyen des arêtes sur cette figure. La courbe en rouge nous indique que le nombre d'arêtes par coupe augmente significativement au cours des itérations. On passe, en effet, d'une cinquantaine d'arêtes à plus de 200. C'est bien en accord avec l'augmentation générale du coût que montre la figure 6. En bleu nous pouvons voir le poids moyen des arêtes appartenant à chaque coupe en fonction du moment de l'algorithme. On observe une très légère diminution du poids des arêtes, indiquant une utilisation plus importante de petites rues dans les coupes au fur et à mesure que l'on avance dans l'algorithme pour compenser le blocage impossible des axes principaux. On retrouve ici encore le premier moment critique, qui correspond à un changement de variation, par diminution du nombre d'arêtes utilisées. Elle est due au fait que les coupes, après ce moment critique, utilise à nouveau d'anciennes arêtes utilisées et donc qu'elles n'ont plus besoin de "contourner" une arête insécable.

Il y a une augmentation claire du coût général des coupes au cours de ces itérations et ce, principalement à cause de la hausse du nombres d'arêtes par coupe. La question est maintenant de savoir si ces nouvelles coupes, bien que plus coûteuses, présentent un intérêt et sont réellement plus diverses.

2.5 Diversité des coupes itératives

2.5.1 Distance entre deux coupes

Afin de pouvoir donner une définition précise de la diversité d'un ensemble de coupes, il faut pouvoir mesurer une différence entre deux coupes, être capable de déterminer si deux coupes sont similaires ou non. La définition d'une distance entre coupes est un moyen simple de déterminer cette différence. Une coupe étant un ensemble d'arêtes et les arêtes pouvant être spatialement localisées, on peut ramener une coupe à un ensemble de points du plan et utiliser une distance par rapport à ceux-ci.

En pratique, pour une arête (u, v) donnée, on calcule la position de l'arête comme étant le milieu entre la position du point u et celle du point v (que l'on a grâce à *OpenStreetMap*). Pour ce qui est de la distance, le choix de la **distance de Chamfer** [AS03] a été naturel.

Définition 4 (Distance de Chamfer) Soit une coupe A et une coupe B , $\|.\|$ la distance eu-

clidienne entre deux arêtes, on pose :

$$d(A, B) = \left(\sum_{a \in A} \min_{b \in B} \{ \|b - a\| \} \right) + \left(\sum_{b \in B} \min_{a \in A} \{ \|b - a\| \} \right)$$

En reprenant les notations de la définition, la distance de Chamfer d est donc la somme des distances des points de A par rapport à B , à laquelle on ajoute la somme des distances des points de B par rapport à A . Une distance étant symétrique, une faible distance indique que toutes les arêtes de A sont globalement proches de B et que toutes les arêtes de B sont globalement proches de A .

2.5.2 Diversité d'un ensemble de coupes

Définissons désormais la notion de diversité d'un ensemble de coupes \mathcal{C} . La diversité est, d'une certaine manière, le nombre de groupes de coupes similaires. Pour calculer cette diversité il est nécessaire de pouvoir regrouper les coupes similaires, en les rassemblant si leur distance est inférieure à un certain seuil.

Définition 5 (Diversité) Soit s un réel positif, la diversité $\in [0, 1]$ d'un ensemble de coupe \mathcal{C} en fonction du seuil s est :

$$\text{Diversité}_s(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \times \text{Card}(\{X \mid \forall c_1, c_2 \in \mathcal{C}, d(c_1, c_2) \leq s \Rightarrow \{c_1, c_2\} \subset X\})$$

2.5.3 Analyse de la diversité des coupes obtenues par l'algorithme

La notion de diversité définie, intéressons-nous à son application dans le cadre de notre algorithme de coupes successives. Nous avons appliqué 400 fois cette méthode itérative, ce qui signifie que pour chaque itération i nous avons un ensemble de coupes comprenant 400 éléments. La figure 8 nous montre les résultats des groupes obtenus pour un seuil $s = 3$ et un seuil $s = 5$. Tandis que la courbe rouge représente la taille moyenne des groupes de coupes, la courbe en bleue mesure le nombre de groupes présents à chaque itération. Cette dernière courbe est donc la valeur de la diversité non normalisée en fonction de s .

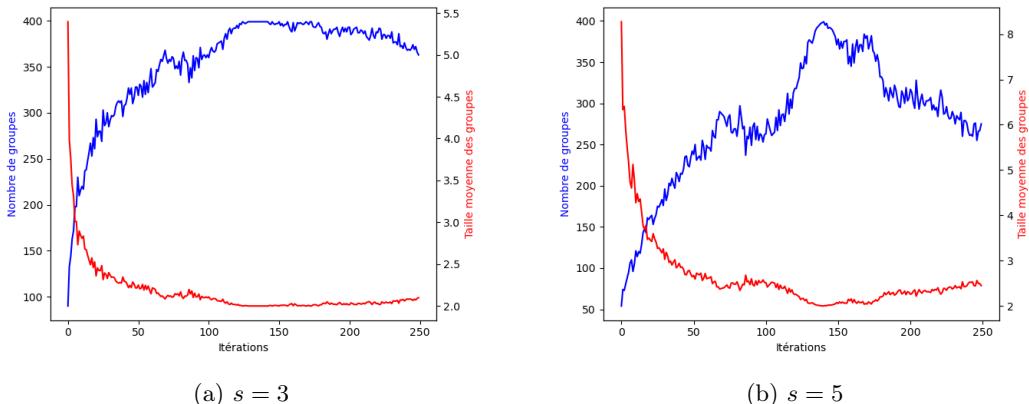


FIGURE 8 – Diversité des coupes en fonction du moment de l'itération

Précisons que cette diversité est calculée sur l'ensemble des coupes obtenues à l'itération i en appliquant 400 fois l'algorithme et non sur l'ensemble total des coupes obtenues depuis la première

itération. Ce calcul alternatif de la diversité aurait été plus précis pour mesurer l'évolution de la diversité au fur et à mesure des itérations mais aurait nécessité un temps de calcul beaucoup plus important. En effet, au lieu de regrouper 400 coupes pour les 250 itérations, il aurait fallu regrouper 400 coupes pour la première, 800 pour la deuxième, 1 200 pour la troisième... C'est donc pour cela que nous avons choisi cette méthode, donnant des résultats tout de même pertinents en un temps plus court. En considérant l'évolution de la diversité pour chaque itération, on espère montrer que les coupes proposées sont plus diverses entre elles plus l'itération i est grande.

L'analyse de ces deux courbes montre une même tendance : il y a une augmentation de la valeur de la diversité jusqu'au 1er moment critique de l'algorithme puis une diminution. Tandis qu'à l'itération 0 le nombre de groupes est avoisine la vingtaine, à l'itération 100 on compte 350 groupes pour un seuil de 3 et 250 pour un seuil de 5. Au fil des itérations, les coupes produites par l'algorithme sont donc plus diverses jusqu'au moment critique où les coupes tendent de nouveau vers des formes que l'on trouvait aux premières itérations.

Cet algorithme de coupes itératives a bien un résultat conforme à nos intuitions : il produit des coupes qui, bien que plus coûteuses, ont l'avantage d'être plus diverses et donc de proposer de multiple manières de bloquer une ville, rendant ces attaques plus difficiles à prévoir. Pour améliorer encore l'efficacité de ces attaques potentielles, il est aussi possible de rendre ces coupes résistantes à diverses tentatives de déblocage.

3 Coupes résistantes sur les graphes urbains

Dans le cadre de notre modélisation, une coupe est susceptible d'être attaquée. Une attaque est l'action de débloquer une route, c'est à dire de retirer une arête de la coupe. En prenant en compte cette contrainte, il est alors nécessaire de concevoir des coupes pouvant "résister" à un certain nombre de ces attaques.

Quel algorithme peut-on mettre en oeuvre afin de produire des coupes résistant aux attaques ? Quels moyens peut-on utiliser afin d'optimiser ce dernier ?

3.1 Définition d'une coupe résistante

Avant d'entrer au coeur de cette partie, il est nécessaire de définir précisément la notion de coupe résistante. Une coupe résistante C est une coupe qui vérifie la propriété de résistance : il existe $k \in \mathbb{N}^*$ tel que, si l'on retire n éléments de C , C reste une coupe, le graphe est toujours séparé en deux. On peut alors nommer une r -coupe résistante une coupe vérifiant cette propriété pour l'entier r défini. Notons qu'une 0-coupe résistante est une coupe simple.

On cherche un algorithme nous donnant une coupe vérifiant cette propriété.

3.2 Algorithme de coupe résistante

Tout le principe de l'algorithme repose sur la création de nouvelles coupes grâce à *KaFFPa* dans un graphe modifié afin de forcer un comportement.

À partir d'une coupe C d'un graphe G et une arête e_0 de C , on cherche une coupe C' disjointe de C tel que $(C \cup C') \setminus \{e_0\}$ reste une coupe. Pour ce faire, on retire dans G tous les éléments de C sauf l'arête e_0 puis on rend e_0 insécable (de la même manière que dans la partie 2.2). On force ainsi *KaFFPa* à produire une nouvelle coupe C' (ligne 14 de l'algorithme 1) ne comprenant pas e_0 (car insécable) ni toutes les autres arêtes de C (car supprimées du graphe). En reprenant le graphe G d'origine, $C \cup C'$ est alors une coupe (puisque C l'était) et $(C \cup C') \setminus \{e_0\}$ est aussi une coupe, par construction de C' . On a donc construit une coupe $C \cup C'$ résistant à la suppression de e_0 . $\forall e' \in C', (C \cup C') \setminus \{e'\}$ est aussi une coupe, puisque C en est une.

Algorithme 1 : Coupe résistante (version naïve)

```
Entrées : Graphe  $G$ , résistance  $r$ 
Résultat : Une  $r$ -coupe résistante  $C$ 
1 Fonction Coupe( $G$ ) :
2   retourner Coupe KaFFPa du graphe  $G$ 
3 Fonction CoupeRésistante( $G, r$ ) :
4   si  $r = 0$  alors
5     retourner Coupe( $G$ )
6    $C \leftarrow \text{CoupeRésistante}(G, r - 1)$ ; /*  $C$  est une coupe, un ensemble d'arêtes */
    */
7   pour tous  $e \in C$  faire
8     retirer  $e$  de  $G$ ;
9    $C_{res} \leftarrow C$ ;
10  pour tous les  $A$ ,  $r$ -uplets disjoints d'arêtes de  $C$  faire
11     $G' \leftarrow$  copie de  $G$ ; /* début du renforcement de la coupe sur une copie
      de  $G$  */
12    pour tous  $e \in A$  faire
13      ajouter  $e$  à  $G'$  avec un poids  $\infty$ ;
14     $C_{temp} \leftarrow \text{Coupe}(G')$ ;
15    pour tous  $e$  dans  $C_{temp}$  faire
16      retirer  $e$  de  $G'$ ;
17     $C_{res} \leftarrow C_{res} \cup \{e\}$ ;
18  retourner  $C_{res}$ 
```

Cependant, $C \cup C'$ n'est pas nécessairement une 1-coupe résistante. En effet, $\forall e \in C \setminus \{e_0\}$, on ne peut pas savoir si $(C \cup C') \setminus \{e\}$ est toujours une coupe. L'idée de l'algorithme est ainsi de propager/reproduire cette opération à toute les arêtes de C : $\forall e_i \in C$, on construit C_i une coupe de G où l'on a préalablement retiré $C \setminus \{e_i\}$ de E et avec e_i insécable. En considérant alors l'ensemble $\tilde{C} = C \cup (\bigcup_{e_i \in C} C_i)$, on peut prouver de la même manière que précédemment que $\forall e \in \tilde{C}$, $\tilde{C} \setminus \{e\}$ est une coupe. Donc \tilde{C} est une 1-coupe résistante. Notons qu'à la ligne 16, on supprime les arêtes de la coupe C_{temp} , rendant donc l'intersection des C_{temp} au cours des tours de boucle vide alors que $\bigcap_{e_i \in C} C_i$ n'est pas forcément vide. Cela ne change rien théoriquement de décider que l'intersection des C_i doit être vide et on l'applique dans notre algorithme afin de ne pas supprimer 2 fois une même arête ; on évite ainsi un certain nombre de calculs.

On obtient une 2-coupe résistante par récurrence en remplaçant C par \tilde{C} (ligne 6) et en prenant désormais des 2-uplets d'arêtes au lieu de simples arêtes dans la boucle (ligne 10). Pour une 3-coupe résistante on prendra des 3-uplets et ainsi de suite.

Afin d'obtenir une 2-coupe résistante, il suffit de répéter ces opérations en remplaçant C par \tilde{C} , et ainsi de suite pour une r -coupe résistante grâce à la récursion ligne 6.

3.2.1 Preuve formelle de l'algorithme

Soit G un graphe (connexe) et $r \in \mathbb{N}$, montrons par récurrence sur r que l'algorithme *CoupeResistante* ci-dessus renvoie bien une r -coupe résistante. On suppose par ailleurs que *KaFFPa* renvoie bien une coupe.

- ★ Si $r = 0$ alors l'algorithme renvoie une coupe simple via $KaFFPa$, ce qui est bien une 0-coupe résistante.
- ★ Soit $r \in \mathbb{N}^*$, on suppose que $CoupeResistante(G, r - 1)$ renvoie bien une $(r - 1)$ -coupe résistante. On note C_0 le résultat obtenu par $CoupeResistante(G, r - 1)$ durant l'appel à $C = CoupeResistante(G, r)$. Soit $A \subset C$ un ensemble de r arêtes que l'on retire de C . Montrons que $C \setminus A$ reste une coupe et donc que C est une r -coupe résistante.
 - Si $\exists e \in A, e \notin C_0$ alors $|A_0| = |A \cap C_0| \leq r - 1$. Par hypothèse de récurrence, C_0 est une $(r - 1)$ -coupe résistante donc $C_0 \setminus A$ reste bien une coupe de G . On a donc que/Par conséquent, $C \setminus A$ est aussi une coupe de G .
 - Sinon, $\forall e \in A, e \in C_0$, alors A est un des r -uplets que l'on calcule durant l'algorithme. On a, par conséquent, réalisé une coupe C' , durant l'exécution de l'algorithme en privant C_0 de A . C étant l'union de C_0 et de toutes les coupes réalisées durant l'exécution de l'algorithme, on a que $(C_0 \cup C') \subset C$. Par définition de C' , $(C_0 \cup C') \setminus A$ est une coupe dans G . Donc $C \setminus A$ est une coupe de G .

Ainsi, $CoupeResistante(G, r)$ est bien une r -coupe résistante.

On a donc bien montré la correction de l'algorithme.

3.3 Optimisations de l'algorithme

Un des intérêts de cet algorithme est la possibilité de sortir des r -coupes résistantes avec r quelconque. Malheureusement l'algorithme est rapidement très long dès lors que la valeur de r augmente. Sur le graphe parisien, pour $r = 1$ l'algorithme met déjà 40 secondes, pour $r = 2$ c'est 1h30, pour $r = 3$ ça n'est même plus la peine d'essayer (260h selon des estimations de calcul fournies par le module `tqdm` de *Python* [Cos19]). Il a donc été primordial d'optimiser cet algorithme afin de pouvoir analyser plus rapidement des coupes de plus en plus résistantes.

3.3.1 Optimisation des copies du graphe

La première optimisation est une optimisation purement pratique : il est plus coûteux de copier à chaque tour de boucle le graphe G que de le copier une seule fois et de simplement remettre, à chaque fin de tour de boucle, les arêtes que l'on coupe au départ de ce même tour. Cette modification accélère déjà grandement le processus en passant à 15s pour $r = 1$ et 15 min pour $r = 2$. On peut donc maintenant calculer une coupe de résistance 3 en 45h environ.

3.3.2 Structure de donnée d'*Union-Find*

Pour la suite des optimisations, nous avons besoin de la structure d'*Union-Find*. C'est une structure permettant de travailler efficacement sur des ensembles **disjoints** [Tar79], ce qui est le cas des ensembles de sommets appartenant à la même composante connexe. L'intérêt de cette structure est de pouvoir réaliser efficacement des **unions** d'ensembles (*Union*) et de **rechercher** rapidement à quel ensemble appartient un élément (*Find*). Cependant, cette structure n'enregistre pas explicitement les ensembles eux-mêmes. Il n'est donc pas efficace de parcourir tous les éléments d'un ensemble.

La structure en elle-même est une forêt, un ensemble d'arbres dont les noeuds sont les éléments des ensembles. Chaque arbre représente un ensemble, disjoint des autres ensembles et la racine de ces arbres est le représentant de l'ensemble. Soit x un élément, l'opération $Find(x)$ revient à trouver la racine de l'arbre auquel appartient x , elle se fait donc en $O(h)$ avec h la profondeur de l'arbre. Soit x, y deux éléments, l'opération $Union(x, y)$ revient à fusionner deux arbres, soit x_0

et y_0 les représentants de l'arbre de x et de l'arbre de y ; x_0 , et donc tout l'arbre qu'il représente, devient fils de y_0 . Cette opération se fait en temps constant si x et y sont déjà les représentants de leur ensemble. Sinon elle a la même complexité que l'opération *Find*.

Nous cherchons à calculer et mettre à jour rapidement les composantes connexes d'un graphe donc, dans notre cas, nous utiliserons cette structure afin de pouvoir rapidement fusionner deux composantes et connaître la taille de ces dernières. La taille d'un ensemble n'est pas donnée par la structure classique d'*Union-Find* mais elle est très rapide à implémenter telle que *Size* ai la même complexité que *Find*, sans changer la complexité de *Find* et *Union*. En effet, on peut créer un tableau stockant la taille de l'ensemble auquel appartient chaque élément puis, lors d'une union mettre à jour la taille de l'ensemble fusionné dans le tableau, à la position du représentant de ce nouvel ensemble. Pour connaître la taille de l'ensemble auquel appartient un élément quelconque x il suffit de trouver le représentant de cet ensemble (avec l'opération *Find*) puis de récupérer la taille stockée dans le tableau à la position du représentant (en temps constant).

3.3.3 Élagage des r -uplets

La principale optimisation algorithmique est en fait l'idée que la plupart du temps, durant un tour de boucle à la ligne 10 du pseudo code, le r -uplet d'arêtes que l'on choisit de remettre dans le graphe ne reconnecte pas les 2 principales composantes connexes du graphe. Dans ce cas là, il n'y a donc pas besoin de trouver une coupe puisque que le graphe est déjà coupé. Il nous faut donc un moyen de calculer rapidement si le graphe est bien coupé en 2 ou non afin de savoir si l'on doit appliquer l'algorithme. On utilise, pour cela, la structure d'*Union-Find* définie ci-dessus afin de stocker les composantes connexes de notre graphe. On l'initialise au début de la récursion avec les arêtes de G privées de celles de la coupe récursive obtenue précédemment. À chaque tour de boucle on effectue une copie de cette structure. On applique ensuite autant d'union qu'il y a d'arêtes à remettre dans le graphe puis on teste si la taille de la composante est inférieure à 75% de la taille totale du graphe. Dans ce cas là, le graphe n'est pas séparé en 2 composantes connexes de taille équivalente, cela signifie que le graphe est toujours coupé et donc qu'il n'y a pas besoin de réaliser une nouvelle coupe. Cette optimisation permet de passer à environ 15s pour une 1-coupe, 150s pour une 2-coupe et 8h pour une 3-coupe.

Résistance r	Nombre de r -uplets d'arêtes	Coût	Temps de calcul avec optimisation
0	20	110	0.2s → 0.2s
1	50	230	40s → 15s
2	6000	390	1h30 → 150s
3	900000	570	260h → 8h

FIGURE 9 – Résultats obtenus sur les coupes résistantes

Précisons que le choix de 75% est totalement arbitraire mais est convenable vis-à-vis des graphes de villes étudiés dans notre cas. Avec *OpenStreetMap*, le graphe récupéré n'est souvent pas connexe. On ne peut donc pas faire de test de connexité pour savoir si la coupe est toujours présente ou non. Sachant que même si le graphe n'est pas connexe, sa plus grande composante fait plus de 95% de sa taille totale, regarder la taille de la plus grande composante a un sens. En effet, *KaFFPa* tente toujours de séparer le graphe en parties presque égales. Si la composante connexe la plus grande fait plus de 75% du graphe alors il n'y pas de coupe (cf figure 9).

On peut mesurer l'importance de cette implémentation via le nombre de r -uplets d'arêtes que l'on élague, en ne calculant pas le corps de la boucle. Par exemple, sur une 3-coupe résistante,

l'algorithme évite de calculer 4 604 coupes sur les 4 656 à la profondeur 2 de l'algorithme puis 682 576 sur 682 640 à la profondeur 3. En moyenne, par cette technique *CoupeResistante* élague 99% pour $r = 2$ et 99.99% pour $r = 3$ des r -uplets d'arêtes à tester.

Voici l'algorithme 2 avec les optimisations ajoutées.

Algorithme 2 : Coupe résistante (avec *Union-Find*)

Entrées : Graphe G , résistance r
Résultat : Une r -coupe résistante C

- 1 **Fonction** Coupe(G) :
- 2 | retourner Coupe KAFFPa du graphe G
- 3 **Fonction** CoupeRésistante(G, r) :
- 4 | si $r = 0$ alors
- 5 | retourner Coupe(G)
- 6 | $C \leftarrow$ CoupeRésistante($G, r - 1$)
- 7 | $G' \leftarrow$ copie de G ;
- 8 | pour tous $e \in C$ faire
- 9 | | retirer e de G' ;
- 10 | $U \leftarrow$ Union-Find représentant le graphe G' par ses composantes connexes.;
- 11 | $C_{res} \leftarrow C$;
- 12 | pour tous les A , r -uplets disjoints d'arêtes de C faire
- 13 | | pour tous $e \in A$ faire
- 14 | | ajouter e à G' avec un poids ∞ ;
- 15 | | $u, v \leftarrow$ les sommets de e ;
- 16 | | $U \leftarrow$ Union(u, v)
- 17 | | si $\max_{u \in V}\{Size(u)\} \geq 0.75 \times |V|$ alors
- 18 | | $C_{temp} \leftarrow$ Coupe(G') ; /* On est dans le cas où une coupe est nécessaire */
- 19 | | pour tous e dans C_{temp} faire
- 20 | | | retirer e de G' ;
- 21 | | | $C_{res} \leftarrow C_{res} \cup \{e\}$;
- 22 | | pour tous $e \in A$ faire
- 23 | | | retirer e de G' ; /* On doit enlever les arêtes car on fait la copie en dehors de la boucle */
- 24 | retourner C_{res}

3.4 Analyse d'une coupe résistante

Nous voyons sur la figure 10 un exemple de coupe résistante (la coupe entière est en annexe). La couleur des rues coupées correspond à l'étape de récursion où la rue a été coupée : en rouge on voit la coupe initiale (0-coupe résistante), en vert les nouvelles arêtes ajoutées afin que la coupe soit une 3-coupe résistante.

En analysant les poids de chaque r -coupe résistante en fonction de r sur la figure 9 on observe que le poids ajouté à chaque itération augmente. Pour passer d'une coupe simple à une 1-coupe résistante le poids double. On a ajouté l'équivalent d'une nouvelle coupe à la précédente afin de

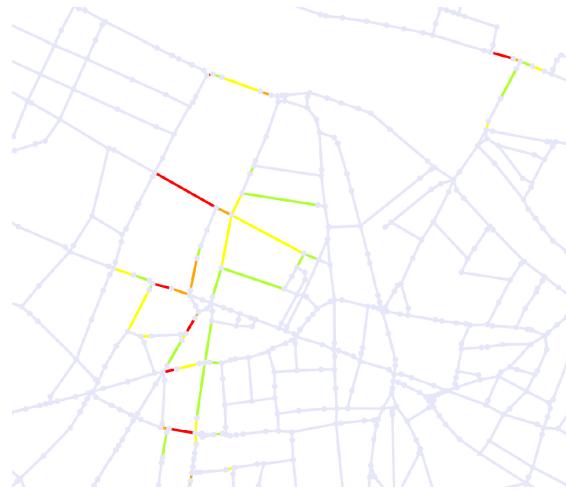


FIGURE 10 – 3-coupe résistante de Paris

la rendre 1-résistante. Pour passer de 2 à 3 par contre, on ajoute un coût d'environ 180, ce qui est presque l'équivalent de 2 coupes simples. D'une manière générale, la tendance semble être à l'augmentation du coût total des nouvelles arêtes ajoutées à chaque itération. Il est difficile de décrire précisément cette tendance étant donné l'accroissement du temps de calcul lors de l'augmentation de r .

3.5 Contraintes sur les manières de couper

Ces coupes résistantes sont intéressantes mais souvent les arêtes coupées semblent appartenir à une même rue. Or débloquer une arête devrait consister à débloquer une rue entière et non une portion de rue. Il est donc important de prendre en compte cette question dans l'algorithme afin qu'il choisisse des arêtes n'appartenant pas à une même rue. Avant ceci, définissons ce que nous entendons spécifiquement par "rue".

Définition 6 (Rue) Soit $e \in E$, on note d le degré. On définit la rue R de e de manière constructive :

1. $R_0(e) = \{e\}$
2. $\forall i \in \mathbb{N}^*, R_i(e) = \{(x, y) \in E \mid \exists z \in V, (x, z) \in R_{i-1} \wedge d(x) \leq 2\}$
3. $R = R_\infty$

La suite des $(R_i)_{i \in \mathbb{N}}$ se stabilise donc R_∞ a un sens. Intuitivement, $R(e)$ est l'ensemble des arêtes voisines de e tel qu'on ne croise pas d'intersection dans le graphe, c'est-à-dire de sommet de degré strictement supérieur à 2.

Le but recherché par les coupes résistantes a été théoriquement atteint par l'algorithme précédent. Cependant, le résultat, bien qu'attendu, a un aspect peu satisfaisant. Si l'on regarde sur la figure 10, on voit que la plupart du temps les arêtes de la coupe sont très proches, si bien que souvent plusieurs arêtes appartiennent à une même rue.

Une amélioration intéressante serait alors d'empêcher notre algorithme de couper plusieurs arêtes d'une même rue en rendant ces arêtes insécables. La manière la plus simple de modifier l'algorithme est d'insérer entre la ligne 20 et 21 de l'algorithme 2 un appel à la fonction `RueInsécable(G, e)` (algorithme 3).

Algorithme 3 : Rue insécable

Entrées : Graphe G , arête e

Résultat : Modification en place de G

1 **Fonction** RueInsécable(G, e) :

```

2   |    $Pile \leftarrow \{e\};$ 
3   |    $DejaVu \leftarrow \emptyset;$ 
4   |   tant que  $Pile \neq \emptyset$  faire
5   |   |    $(u, v) \leftarrow$  dépile  $Pile;$ 
6   |   |    $DejaVu \leftarrow DejaVu \cup \{(u, v)\}$  rendre  $(u, v)$  insécable dans  $G;$ 
7   |   |   si degré de  $u$  dans  $G \leq 2$  alors
8   |   |   |   pour  $x$  voisin de  $u$  faire
9   |   |   |   |   si  $(x, u) \notin DejaVu$  alors
10  |   |   |   |   |   empile  $(x, u)$  dans  $Pile;$ 
11  |   |   |   si degré de  $v$  dans  $G \leq 2$  alors
12  |   |   |   |   pour  $x$  voisin de  $v$  faire
13  |   |   |   |   |   si  $(x, v) \notin DejaVu$  alors
14  |   |   |   |   |   |   empile  $(x, v)$  dans  $Pile;$ 

```

Cette nouvelle version donne des coupes avec des arêtes qui sont moins proches, puisqu'elles ne peuvent plus appartenir à la même rue. La figure 11 montre cet répartition des arêtes par rues différentes. Si deux arêtes coupées sont adjacentes c'est que leur sommet commun est une intersection : il a un degré strictement supérieur à 2.



FIGURE 11 – 3-coupe résistante de Paris, avec distinction des rues

Le temps de calcul pour ce type de coupe est cependant plus long. Pour une 2-coupe résistante on passe de 2 minutes à 6 minutes tandis que pour une 3-coupe résistante les 8h deviennent 25h. Cela est dû au traitement de l'insécabilité des rues qui rajoute des calculs supplémentaires non négligeables ainsi qu'à l'éparpillement des arêtes coupées, obligeant très souvent à couper plus d'arêtes. On passe en effet d'environ Le coût d'une 3-coupe résistante montre aussi ce point

puisque l'on passe d'un coût d'environ 570 à un coût de plus de 900.

4 Conclusion

L'objectif de ce stage était d'étudier la méthode pour rendre plus efficace le blocage du réseau viaire d'une ville, ici Paris. La première manière de rendre un blocage efficace a été de produire des coupes de Paris les plus diverses possibles, en contrepartie d'une augmentation relative de leur coût, afin que ces attaques soient plus imprévisibles. Nous avons ainsi proposé un algorithme répondant à cette problématique et étudié les coupes produites par ce dernier, en constatant une réelle évolution de la diversité au fil des itérations de l'algorithme. La seconde approche a été de rendre les blocages plus résilients, en produisant des coupes capables de résister à un certain nombre de perturbations. C'est ainsi que nous avons pu présenter une méthode de création récursive de ces coupes et prouvé son efficacité. Le principal point a été d'optimiser cet algorithme afin qu'il soit en mesure de proposer des coupes résistantes en un temps plus court.

Quelle que soit la méthode utilisée, le fait de rendre **une attaque plus imprévisible ou résiliente ajoute des contraintes** à cette attaque. Si les coupes produites par *KaFFPa* peuvent être considérées comme ayant un poids quasi optimal, une attaque vise souvent à être difficile à contrer et donc cette optimalité hypothétique n'est plus l'objectif que l'on souhaite atteindre. Il est ainsi naturel d'observer l'augmentation du coût de l'attaque lorsque l'on cherche à la rendre résiliente.

Références

- [AS03] Vassilis ATHITSOS et Stan SCLAROFF. "Estimating 3D hand pose from a cluttered image". In : 2003 IEEE Computer Society Conference. T. 2. IEEE. 2003, p. II-432.
- [Bar04] Marc BARTHELEMY. "Betweenness centrality in large complex networks". In : The European physical journal B 38.2 (2004), p. 163-168.
- [Boe17] Geoff BOEING. "OSMnx : New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In : Computers, environment and urban systems 65 (2017), p. 126-139.
- [Cos19] Casper O da COSTA-LUIS. "tqdm : A fast, extensible progress meter for python and cli". In : Journal of Open Source Software 4.37 (2019), p. 1277.
- [Mal20] Andreas MALM. Comment saboter un pipeline. La fabrique éditions, 2020.
- [SS13] Peter SANDERS et Christian SCHULZ. "Think locally, act globally : Highly balanced graph partitioning". In : International Symposium on Experimental Algorithms. Springer. 2013, p. 164-175.
- [Tar79] Robert Endre TARJAN. "A class of algorithms which require nonlinear time to maintain disjoint sets". In : Journal of computer and system sciences 18.2 (1979), p. 110-127.

A Situation et déroulement du stage

Mon stage s'est déroulé à Jussieu, au *Laboratoire d'Informatique de Paris VI* (LIP6), dans l'équipe de recherche *Complex Network*. J'ai été très bien accueilli par toute l'équipe, composée d'une dizaine de chercheurs et presque autant de stagiaires. Encadré par Clémence Magnien, c'est Bastien Legay et Alexis Baudin qui m'ont suivi et dirigé pendant tout le stage. L'arrivée dans

l'équipe s'est faite simplement, dans une bonne entente générale ; j'ai pu découvrir et discuter avec des personnalités diverses durant les pauses ou les séminaires organisés au sein du LIP6. La première semaine m'a permis de comprendre et de me familiariser avec le travail de Louis, un autre stagiaire en Master 1 d'informatique, ayant posé les bases sur lesquelles j'ai pu travailler par la suite. Assez libre dans ma manière de gérer mon travail et avec la possibilité de télétravailler, j'ai été suivi de manière assidue par Bastien et Alexis via des contacts réguliers et une réunion par semaine au minimum afin de discuter de mes avancées et d'apporter un regard critique sur mon travail en me partageant leurs connaissances. Les jeux olympiques de Paris 2024 ont malheureusement contraint le LIP6 à fermer plus tôt que prévu et à terminer mon stage en télétravail. Cela ne m'a pas empêché de rester en contact avec Bastien et Alexis jusqu'au bout.

Ce stage a été enrichissant, me permettant de voir de plus près ce à quoi peut ressembler le monde de la recherche et, en particulier, de comprendre que ce travail n'est en aucun cas solitaire ; bien au contraire, il s'appuie sur les travaux des uns et des autres, sur de nombreuses discussions enrichissantes et sur l'entretien d'un bel esprit collectif.

B Exemples de coupes résistantes



FIGURE 12 – 3-coupe résistante de Paris (PDF)

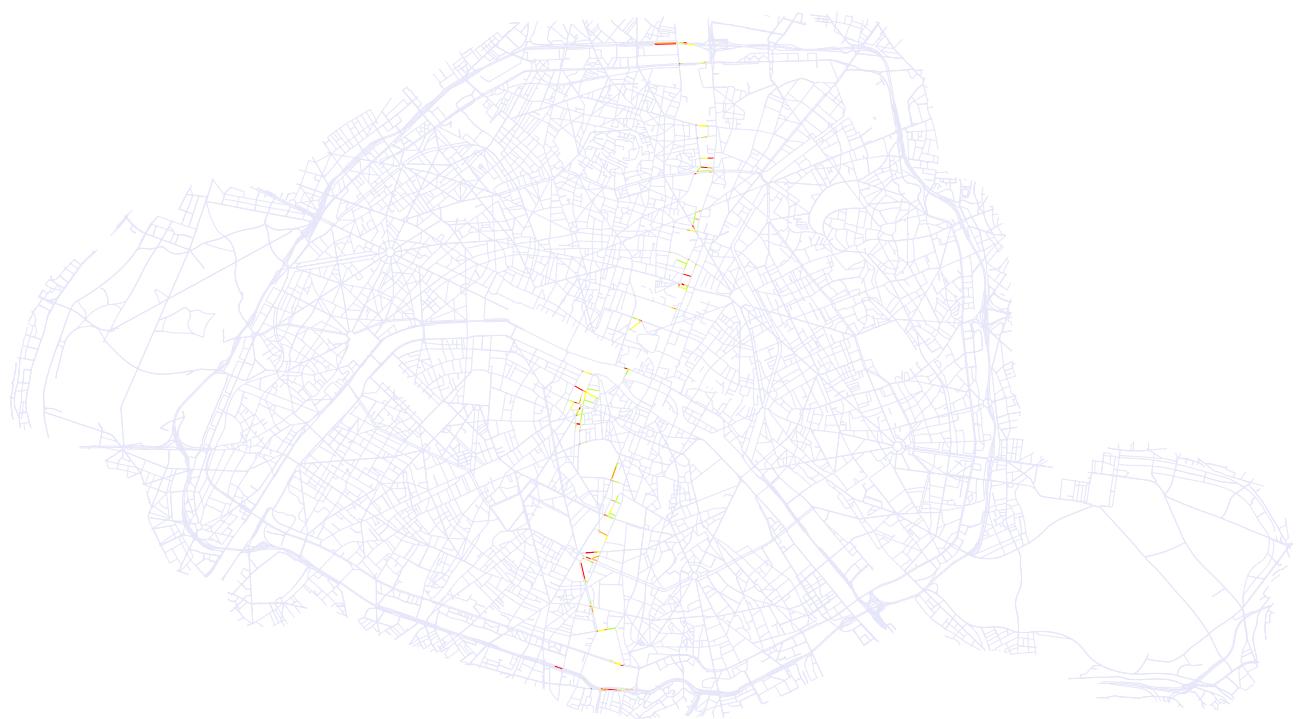


FIGURE 13 – 3-coupe résistante de Paris, avec contraintes de coupe (PDF)