
ÿ Développement d une extension logicielle implémentant un graphe par temps régressif

Auteur : Lacroix, Julien

Promoteur(s) : Donnay, Jean-Paul

Faculté : Faculté des Sciences

Diplôme : Master en sciences géographiques, orientation géomatique et géométrie, à finalité spécialisée

Année académique : 2018-2019

URI/URL : <http://hdl.handle.net/2268.2/7414>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.



**Université de Liège
Faculté des sciences
Département de géographie**

Développement d'une extension logicielle implémentant le parcours d'un graphe par temps régressif

Mémoire présenté par :

Julien LACROIX

Pour l'obtention du titre de :

**Master en Sciences géographiques, orientation géomatique et
géométrie, à finalité spécialisée**

Président du Jury : Pr Jean-Paul Donnay

Promoteur : Pr Jean-Paul Donnay

Lecteurs : Pr Jean-Marie Halleux & Dr
Jean-Paul Kasprzyk

Année académique 2018-2019

Date de défense : septembre 2019

Remerciements

Je tiens d'abord à remercier mon promoteur, Jean-Paul Donnay, pour ses remarques, ses précieux conseils, sa grande disponibilité et son accompagnement tout au long de la réalisation de ce travail.

Je remercie également mon professeur de l'Université de Sherbrooke, Yves Voirin, pour ses conseils liés à l'aspect technique de ce mémoire.

Merci également à Marie Trotta pour l'obtention des données du réseau routier, sans lesquelles ce travail n'aurait pu avoir lieu.

Enfin, je remercie grandement ma famille pour son soutien inébranlable au fil des années, ainsi que mes amis pour leur support moral durant la réalisation de ce travail.

Résumé

Le parcours de graphes routiers est couramment utilisé, notamment par les applications de calcul d'itinéraires. La transposition de ces parcours de graphes en mode maillé utilise un algorithme de propagation appliqué au réseau routier désormais matérialisé par un ensemble de pixels. Cette transposition est utilisée dans le cadre du profilage géographique, afin de mettre en évidence le point d'ancrage d'un auteur de plusieurs crimes localisés dans l'espace et dans le temps en se basant sur un ensemble de postulats. Le parcours de ce graphe en mode maillé est réalisé depuis chaque site de crime, à partir duquel on « remonte le temps » en se propageant sur l'ensemble du réseau routier rasterisé en fonction de la vitesse autorisée en chaque pixel. Le point d'ancrage est alors donné par le pixel minimisant la différence entre l'heure de passage maximale et minimale (i.e. l'étendue des heures de passage).

L'objectif de ce mémoire est de construire une extension logicielle implémentant ce processus, tout en étudiant les différentes erreurs qui affectent le résultat final et en tentant de les résoudre afin d'améliorer celui-ci.

Pour ce faire, des données spatio-temporelles arbitraires ont été utilisées. La méthode de l'étendue des heures de passage a été implémentée ainsi qu'une méthode calculant la variance de ces mêmes heures, qui a été proposée afin de déterminer si elle pouvait améliorer le résultat. Une analyse comparative a montré que la méthode de l'étendue est plus efficace étant donné qu'elle apparaît plus discriminante et plus proche de la solution.

Certaines erreurs ont été solutionnées (amélioration des coûts propagés, rattachement des sites de crimes sur le graphe routier), là où d'autres n'ont pu être résolues pour cause de manque de données (adaptation du temps de parcours à la date et à l'heure). Deux méthodes relatives à la prise en compte respectivement du sens de circulation et de la non-planéité des graphes routiers ont été proposées et implémentées, par mise en évidence des mouvements non permis entre pixels. Cependant, ces méthodes sont en pratique fortement limitées par le résultat de la rasterisation, et n'ont donc pas pu être ajoutées au processus.

Au final, l'extension logicielle a bien été développée avec la méthode de l'étendue, ainsi que deux autres solutions de mise en évidence d'un point d'ancrage également énoncées dans le cadre du profilage géographique : la minimisation de la moyenne et de la variance des temps de déplacement du point d'ancrage vers les différents crimes. Une comparaison des temps d'exécution montre que l'efficacité des méthodes est proche, et par ailleurs que cet ajout de méthodes a du sens étant donné que le temps d'exécution combiné des trois méthodes implémentées n'est pas significativement plus important que celui de la seule méthode de l'étendue.

Abstract

Road graphs traversal is commonly used, especially by route calculation applications. The transposition of these graphs traversals in raster mode uses a propagation algorithm applied to road network materialized from now on by a set of pixels. This transposition is used for geographic profiling, in order to highlight the anchor point of an author of several space and time localized crimes according to some assumptions. This raster graph traversal is realized from each crime site, from which one “go back in time” by propagation on the whole rasterized road network in function of the authorized speed in each pixel. The anchor point is then given by the pixel that minimizes the difference between maximal and minimal hours passing through each pixel (i.e. range method of these hours).

The objective of this master's thesis is to build a plugin that implements this process, while studying the possible errors that affect the final result and trying to resolve them in order to improve it.

To do so, arbitrary space-time data has been used. The range method has been implemented, as well as another method calculating the variance of hours passing through each pixel that has been proposed in order to determine if it could improve the result. A comparative analysis showed that the range method is more efficient as it appears more discriminant and closer to the solution.

Some errors have been solved (improvement of the propagated costs, connection of crime sites on road graph), while others have not been fixed because of the lack of data (adaptation of time traversal to date and time). Two methods taking into consideration respectively the traffic direction and the non-planar aspect of road graphs have been proposed and implemented, by revealing movements that are not allowed between two pixels. However, these methods are in practice forcefully limited by the result of the rasterization, and then have not been added to the process.

At the end, the plugin has been correctly developed with the range method, as well as two other solutions of highlighting an anchor point also announced in the geographic profiling literature: the minimization of journeys-to-crime mean and variance. A comparison of the times of execution shows that the efficiency of the methods is similar, and also that this addition makes sense as the total time of execution of the three implemented methods is not significantly greater than the one with the range method only.

Table des matières

1.	Introduction.....	8
2.	Etat de l'art.....	10
2.1	Principes inhérents au calcul des surfaces de coût.....	10
2.1.1	Surface de friction.....	10
2.1.2	Surface de coût.....	10
2.2	Algorithmes disponibles	11
2.2.1	Mode vecteur	11
2.2.2	Mode raster	13
2.3	Applications géographiques.....	16
2.4	Problèmes courants	18
2.4.1	Erreurs sur l'espace.....	18
2.4.2	Erreurs sur le temps.....	20
2.4.3	Erreurs sur la surface de friction	20
3.	Objectifs.....	22
3.1	Pistes de solution.....	22
3.1.1	Positionnement des crimes sur le réseau routier	22
3.1.2	Adaptation du temps de parcours à la date et l'heure	22
3.1.3	Prise en compte de l'imprécision sur le temps.....	23
3.1.4	Calcul des distances post-rasterisation.....	23
3.1.5	Prise en compte des données attributaires du mode vectoriel.....	23
3.2	Plan de l'analyse	24
3.3	Construction du jeu de données	24
4.	Amélioration de la surface de friction.....	26
4.1	Rasterisation et construction de la surface de friction	26
4.2	Amélioration des temps de parcours sur le réseau rasterisé.....	27
4.3	Validation sur les données	28
5.	Algorithme de propagation	30
5.1	Identification.....	30
5.2	Transposition.....	30
5.3	Paramétrage.....	32
5.4	Validation sur le jeu de données	32
6.	Implémentation du processus.....	33
6.1	Processus général	33
6.2	Implémentation pratique	37

6.2.1	Rasterisation.....	37
6.2.2	Création de la surface de friction améliorée	38
6.2.3	Rattachement des données spatio-temporelles au réseau routier	39
6.2.4	Création des surfaces de coût.....	42
6.2.5	Délimitation de la zone d'ancrage.....	42
6.3	Validation.....	46
6.4	Comparaison des méthodes.....	48
6.4.1	Configuration géométrique non optimale	49
6.4.2	Erreurs temporelles	51
6.4.3	Configuration géométrique non optimale et erreurs temporelles	52
6.4.4	Conclusion	54
6.5	Conclusion sur la délimitation de la zone d'ancrage	54
7.	Amélioration des surfaces de coût	56
7.1	Prise en compte des contre-sens.....	56
7.2	Prise en compte de la non-planéité des graphes routiers.....	58
7.3	Amélioration des méthodes.....	59
7.4	Limitations	61
7.5	Conclusion	62
8.	Développement de l'extension logicielle.....	63
8.1	Initialisation de l'extension.....	63
8.2	Transposition théorique.....	64
8.3	Modification de l'interface graphique.....	65
8.4	Modification du code source de l'extension	68
8.4.1	Structure du fichier.....	68
8.4.2	Liaisons avec l'interface graphique	69
8.4.3	Transposition du processus	73
8.5	Efficacité de l'extension logicielle.....	74
8.6	Accès à l'extension logicielle	75
9.	Conclusions et perspectives	76
9.1	Conclusions.....	76
9.2	Perspectives.....	78
10.	Références.....	79
	Liste des annexes	83

Identification des éléments informatiques

<i>Méthode</i>	Nom de méthode
LIBRAIRIE	Nom de librairie
Fonction	Nom de fonction
'Classe'	Nom de classe
'variable'	Nom de variable
'attribut'	Nom d'attribut
« fichier »	Nom de fichier

1. Introduction

Les algorithmes propres au parcours de graphes valués, représentant le plus souvent un réseau routier, sont omniprésents dans la société actuelle. Les applications de calcul d'itinéraires en temps réel telles que Google Maps ou Waze en sont un bon exemple, elles qui utilisent un certain nombre d'informations propres aux segments routiers (Russel, 2013). Ces informations sont utilisées par ce type d'applications afin de calculer le temps de parcours de tronçons routiers en utilisant un mode de transport donné, permettant finalement d'estimer, en temps réel, le temps d'arrivée en un lieu déterminé.

Ces algorithmes de la théorie des graphes sont relatifs au mode vectoriel, caractérisé par des primitives géométriques associées à des données attributaires, ce qui implique deux restrictions majeures (Donnay et Ledent, 1995). D'abord, l'accessibilité est uniquement calculée aux nœuds du graphe, autrement dit aux extrémités de chaque arête. L'accessibilité de tout point situé entre celles-ci ne peut être calculée que par interpolation entre les deux nœuds délimitant l'arête, ce qui peut entraîner de fortes incohérences dans le cas de deux nœuds présentant la même accessibilité ou encore des impossibilités dans le cas de nœuds ne présentant aucune accessibilité. Ensuite, il est impossible d'obtenir une valeur d'accessibilité concernant les points situés en dehors du graphe, et par conséquent de délimiter des courbes de même accessibilité.

La solution à ces limitations du mode vectoriel requiert de considérer l'information d'accessibilité de manière non plus discrète mais spatialement continue, en passant du mode vectoriel au mode maillé (ou *raster*, terme également utilisé dans la suite de ce travail) par rasterisation. Ce mode *raster* est caractérisé par une image, constituée d'un ensemble de pixels qui présentent une certaine valeur d'accessibilité. La transposition de ces algorithmes de la théorie des graphes du mode vectoriel au mode maillé est réalisée par propagation d'un coût de pixel en pixel sur le réseau routier rasterisé représenté auparavant en mode vectoriel par le graphe routier. Ce coût propagé peut être exprimé en différentes unités telles que la distance ou le temps, et ce *raster* obtenu traduisant l'accessibilité depuis un pixel donné est appelé « surface de coût ».

Le parcours d'un graphe valué de voirie en mode maillé est utilisé dans le cadre du profilage géographique (*geographic profiling*) via la cartographie criminelle (*crime mapping*), introduit à la fin des années 90 afin d'aider la police américaine à résoudre une série de viols perpétrés par une seule et même personne (MacKay, 1999). Ce type de profilage peut être défini comme une méthodologie d'investigation utilisant les différents lieux d'une série de crimes connectés afin de déterminer le point d'ancrage la plus probable du criminel (Trotta *et al*, 2011). Depuis quelques années, la police belge commence également à s'intéresser aux résultats découlant du profilage géographique (Donnay *et al*, 2013).

Dans ce contexte, plusieurs méthodologies d'investigation différentes ont été développées, en considérant divers postulats. L'une de celles-ci se base sur un triple postulat : le criminel part d'un même endroit constituant son point d'ancrage, à une même heure, et utilise un même mode de transport sur le réseau routier pour rejoindre les sites de crime. Le parcours de ce graphe par temps régressif est alors réalisé depuis chaque crime connu de manière spatio-

temporelle, à partir duquel on va « remonter le temps » en se propageant sur l'ensemble du réseau routier rasterisé en fonction de la vitesse autorisée en chaque pixel. L'objectif final est de délimiter une zone de convergence aussi réduite que possible depuis l'ensemble de ces crimes, qui correspondra vraisemblablement à la zone d'ancrage du criminel. En pratique, cette zone de convergence est constituée des pixels présentant des différences minimales entre l'heure de passage maximale et minimale depuis les multiples crimes relevés.

La question de recherche associée à ce mémoire porte donc sur la délimitation d'une zone d'ancrage sur base d'une combinaison de surfaces de coût temporel régressif depuis une série de crimes connus de manière spatio-temporelle. L'opérationnalité de la méthode sera testée à travers un prototype d'extension du logiciel-SIG *open source* QGIS. Ce travail se focalise plus particulièrement sur l'identification des problèmes inhérents au contexte introduit précédemment et tente d'y apporter des solutions afin d'améliorer la délimitation de cette zone d'ancrage.

La raison de la création d'une extension logicielle est qu'aucun logiciel-SIG ne possède d'outil permettant de répondre complètement et de manière efficace à la question de recherche. L'objectif, à travers le développement de cette extension, est de rassembler toutes les procédures nécessaires au processus en un seul et même outil, dans un logiciel-SIG *open source* puissant et largement utilisé (QGIS). Ce choix de logiciel-SIG permet, dès lors, d'apporter de la transparence sur le processus mis en place.

La structure de ce mémoire est la suivante : les notions de base liées à la question de recherche et nécessaires à la compréhension de ce travail sont introduites dans le chapitre 2. Le chapitre 3 formule l'hypothèse autour de laquelle ce travail s'inscrit, ainsi que le plan de l'analyse pour y répondre. Le chapitre 4 traite la manière d'améliorer la surface de friction, tandis que le chapitre 5 est consacré à l'algorithme de propagation utilisé dans ce mémoire. Le chapitre 6 explique l'ensemble du processus implémenté dans le cadre de ce travail, d'abord de manière générale et ensuite de manière plus exhaustive. Le chapitre 7 traite les possibles améliorations relatives aux surfaces de coût tandis que le chapitre 8 explique le développement de l'extension logicielle sur QGIS ainsi que la transposition dans celle-ci du processus précédemment implémenté. Enfin, le neuvième et dernier chapitre présente les conclusions et perspectives de ce travail.

2. Etat de l'art

2.1 Principes inhérents au calcul des surfaces de coût

2.1.1 Surface de friction

Une surface de friction est un *raster* dans lequel est attribuée, en chaque pixel, une valeur positive. Cette valeur, associée à la notion de résistance, représente le « coût » requis pour traverser ce pixel : plus sa valeur est élevée et plus il sera difficile de le franchir. Eastman (1989) propose très tôt d'incorporer les effets de friction dans le calcul de distances en mode *raster*, afin de créer de nouveaux rasters qu'il nomme « distance de coût » (*Cost distance*). Il précise que ce coût peut être calculé en une multitude d'unités possibles, en citant la distance, le temps ou encore le coût financier. Il spécifie enfin qu'une friction unitaire constitue la norme, et que toutes les valeurs de friction sont définies par rapport à celle-ci. Quelques années plus tard, Douglas (1994) puis Donnay et Ledent (1995) transposent ces surfaces de friction à l'exemple d'un réseau routier dans lequel la vitesse maximale autorisée sur chaque tronçon constitue la friction liée à ce même tronçon de voirie. Chaque pixel constituant un même tronçon présente donc le même coût, celui-ci traduisant la résistance à la vitesse. Douglas (1994) considère alors une vitesse « normale » dont le coût est fixé à l'unité, et énonce qu'un segment routier ayant une vitesse permise supérieure à la vitesse normale aura un poids entre 0 et 1 (résistance moindre à la vitesse), et inversement. Tout ceci implique qu'un coût négatif est impossible, car on ne peut pas perdre de la distance parcourue ou réduire le temps en se déplaçant dans l'espace. Ce coût négatif est pourtant utilisé dans certains logiciels tels qu'Idrisi qui l'affecte à une barrière infranchissable alors que d'autres utilisent une valeur « *No Data* » (ESRI 2016a) ou encore des valeurs de coûts infinies pour les définir. Celles-ci sont utiles dans les situations qui nécessitent d'empêcher la propagation du coût en certains pixels : on peut citer le cas des pixels présentant une pente trop importante que pour pouvoir être franchis, ou de tout pixel se trouvant en dehors d'un réseau routier.

2.1.2 Surface de coût

La surface de friction précédemment introduite est indispensable à la création de la surface de coût. Eastman (1989) puis Douglas (1994) ont montré la manière dont cette surface de coût est construite, par propagation pixel par pixel en partant d'un pixel-source pour lequel le coût est nul (cf. 2.2 pour l'explication de l'algorithme). Vanraes *et al* (1998) ont, par après, appliqué ces surfaces de coût dans le cadre de l'analyse du réseau de transport public liégeois afin de représenter la distribution spatiale de son offre. Cette propagation pixel par pixel est effectuée sur l'ensemble du réseau routier, de manière à ce que tout pixel appartenant à celui-ci ait une valeur de coût total accumulé entre lui et le pixel-source d'où démarre l'algorithme. Ce coût peut se traduire tant en distance à laquelle le pixel se trouve de la source qu'en temps qu'il faut pour y parvenir depuis celle-ci. Il est donc possible de propager la surface de coût sur le réseau routier selon un temps régressif depuis l'heure précise à laquelle un crime localisé a été commis, et donc de « revenir dans le temps ». Dans le cas du coût temporel, l'idée est de déterminer l'heure de passage en un pixel donné sachant qu'à l'heure du crime il devra avoir atteint le pixel correspondant au site de crime.

2.2 Algorithmes disponibles

2.2.1 Mode vecteur

Il est nécessaire de débiter cette partie consacrée aux algorithmes disponibles par la distinction des modes vectoriel et raster, car les algorithmes développés en mode raster sont des transpositions des algorithmes de la théorie des graphes qui étaient déjà opérationnels en mode vecteur. Ces algorithmes relatifs au mode vecteur sont des algorithmes de recherche du plus court chemin (*shortest path*), dont un des plus connus est celui de Dijkstra (1959). Si plusieurs algorithmes avaient été énoncés avant lui tels que celui de Shimbél (1955), sa spécificité est, pour un graphe d'ordre N , de permettre de trouver les plus courts chemins vers les $N - 1$ autres nœuds du graphe en partant d'une seule source, là où les autres algorithmes trouvaient les plus courts chemins entre tous les nœuds d'un graphe. Trouver tous ces chemins les plus courts revient à relancer Dijkstra autant de fois qu'il existe de nœuds autres que le nœud de départ. Ces algorithmes effectuent par conséquent les calculs dans le pire des cas en $(N - 1) * (N - 1)$ étapes là où Dijkstra tournera en $N - 1$ étapes. Comme le montre la figure 1 ci-dessous prenant un exemple de graphe valué d'ordre 4, l'algorithme de Dijkstra initialise trois éléments :

- un tableau de distances T entre le nœud de départ et tous les nœuds du graphe, distance initialisée à 0 pour le nœud de départ (A) lui-même, à la valeur de l'arête reliant les nœuds adjacents à ce nœud de départ pour ceux-ci (B) et une distance infinie pour les autres nœuds du graphe non reliés à A (C et D) ;
- une queue Q contenant tous les nœuds du graphe non parcourus (B, C, D) ;
- une liste L qui contiendra les nœuds visités par l'algorithme (ne contient initialement que le nœud de départ, A).

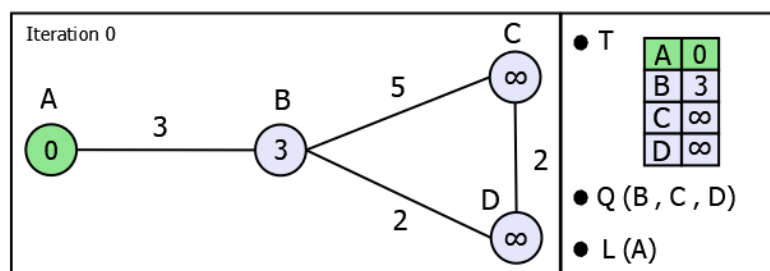


Figure 1 : itération 0 de l'algorithme de Dijkstra pour un graphe d'ordre 4

L'algorithme procède alors de la manière suivante :

- Il cherche le nœud qui n'est pas présent dans L et qui présente la distance minimale. Ce nœud qui présente la plus faible distance est retiré de Q et inscrit dans L (itération 1 à 3, cf. Figures 2 à 4).

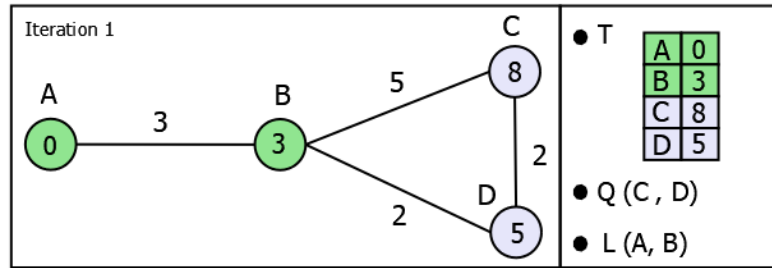


Figure 2 : itération 1 de l'algorithme de Dijkstra pour un graphe d'ordre 4

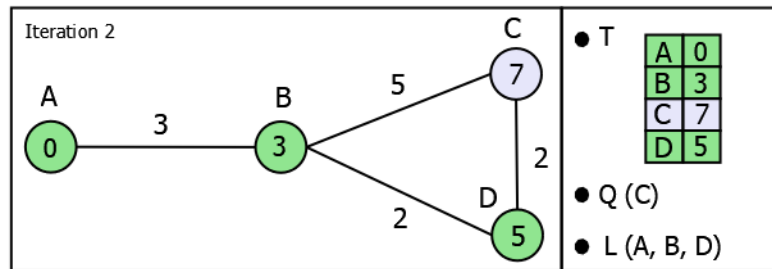


Figure 3 : itération 2 de l'algorithme de Dijkstra pour un graphe d'ordre 4

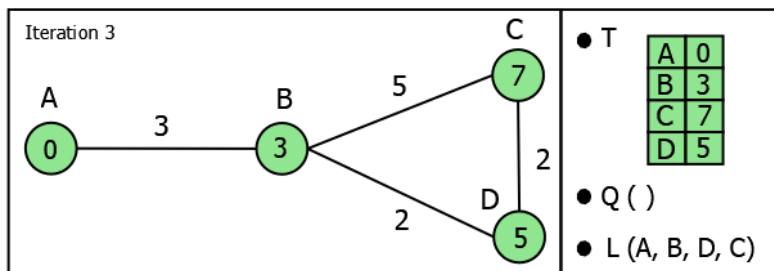


Figure 4 : itération 3 de l'algorithme de Dijkstra pour un graphe d'ordre 4

- b) Les distances des nœuds adjacents au nœud courant sont mis à jour dans le tableau T si la distance du nœud courant au nœud source augmenté de la distance entre le nœud courant et un nœud adjacent « m » considéré est inférieure à la distance entre le nœud source et ce nœud « m » (cas du nœud C lors de l'itération 2, qui passe de 8 à 7 via le nœud D, cf. Figure 3).

L'algorithme se termine lorsque la queue Q est vide, et le tableau T donne les plus courts chemins pour chaque nœud du graphe depuis le nœud de départ. Dans le pire des cas, c'est-à-dire lorsqu'un seul nœud est choisi à chaque itération parmi toutes les possibilités, il se déroule en $N - 1$ étapes avec N l'ordre du graphe (autrement dit le nombre de nœuds que comprend le graphe). Lorsque plusieurs nœuds peuvent être simultanément sélectionnés durant au moins l'une des itérations de l'algorithme, celui-ci se déroulera en moins de $N - 1$ itérations. Il est à noter encore une fois que le coût-distance pris dans cet exemple peut être généralisé en coût temporel ou monétaire.

Certains algorithmes ont par la suite été créés pour considérer d'autres situations. Citons celui de Bellman-Ford (Ford, 1956 ; Bellman, 1958), qui se différencie de l'algorithme de Dijkstra par la prise en compte de poids négatifs, ce que ne permet pas Dijkstra. L'algorithme de

Floyd-Warshall (Floyd, 1962 ; Warshall, 1962) résout quant à lui le problème du plus court chemin entre toutes les paires de nœuds d'un graphe, ce que pourrait effectuer Dijkstra en relançant la procédure pour chaque nœud. Enfin, l'algorithme A* (Hart *et al.*, 1968) développé quelques années plus tard, dont Dijkstra est un cas particulier, recherche le chemin le plus court jusqu'au nœud final non pas en parcourant tous les nœuds du graphe mais en choisissant ceux qui semblent les plus prometteurs. Pour ce faire, l'algorithme se base sur une fonction heuristique qui approxime le coût entre les nœuds de départ et d'arrivée. Cet algorithme sera par conséquent plus rapide que Dijkstra mais il ne garantit pas le chemin le plus court, seulement une solution qui se rapproche de celui-ci. En d'autres termes, la première solution n'est pas nécessairement la meilleure, mais une des meilleures. Au final, la principale différence entre ces différents algorithmes est leur complexité en temps (*time complexity*) qui fera qu'en fonction de la configuration d'un problème donné, il sera préférable de porter son choix sur un algorithme en particulier.

2.2.2 Mode raster

Un algorithme de propagation en mode *raster* démarre d'un pixel de départ (ou pixel-source) et reporte, à chaque itération, la valeur du pixel courant sur les valeurs de tous les pixels voisins 8-connexes accessibles (i.e. ceux qui ne présentent pas de valeur *No Data*). On peut prendre l'exemple de la propagation d'un poids unitaire de pixels en pixels jusqu'à couvrir l'entièreté du *raster*, comme le montre la figure 5 ci-dessous reprenant l'exemple d'un réseau routier rasterisé (pixels verts). Plus on s'éloigne du pixel de départ, c'est-à-dire le pixel associé au coin supérieur gauche de l'image, plus le poids propagé sera important.

1	0	0	4
0	2	3	0
0	0	0	4

Figure 5 : exemple de propagation d'un poids unitaire à travers un réseau routier rasterisé (pixels verts)

Les algorithmes de surface de coût sont un cas particulier de cet algorithme de propagation qui apparaît lorsque le coût est fonction de la distance. En effet, une propagation aux pixels diagonaux (« *bishop* ») voisins du pixel courant nécessite une distance plus importante à prendre en compte. Deux types de déplacements distincts sont donc envisageables (et un troisième dans le cas de l'algorithme de GRASS, cf. *knight move* ci-dessous) : horizontal-vertical et diagonal. La prise en compte de ce second type de déplacement est réalisée en introduisant dans le calcul un facteur $\sqrt{2}$ (diagonale d'un pixel de côté unitaire). Le calcul du coût dans ce cas particulier de l'algorithme de propagation qu'est la surface de coût, est également différent.

Ce coût associé à un pixel considéré est le coût du lien imaginaire entre le centre de ce pixel et celui du pixel courant, comme le montre la figure 6 ci-dessous dans le cas d'un

déplacement horizontal-vertical et diagonal. Le coût de ce lien est une moyenne des frictions des deux pixels, éventuellement multipliée par un facteur $\sqrt{2}$ dans le cas d'un déplacement diagonal. Il faut néanmoins noter que le coût alloué aux pixels dépendra des valeurs de x, y et z : si pour rejoindre le pixel correspondant à la friction z il est plus coûteux de se déplacer en diagonale plutôt qu'un double déplacement horizontal puis vertical, ce dernier sera privilégié. Au final, l'algorithme calcule le coût cumulé de chacun des 8 pixels voisins qui peuvent être considérés au départ du pixel courant, c'est-à-dire ceux qui sont franchissables, et choisira celui présentant le coût cumulé minimal (nouveau pixel courant). Enfin, le coût associé au pixel de départ est toujours nul (quelle que soit la métrique utilisée).

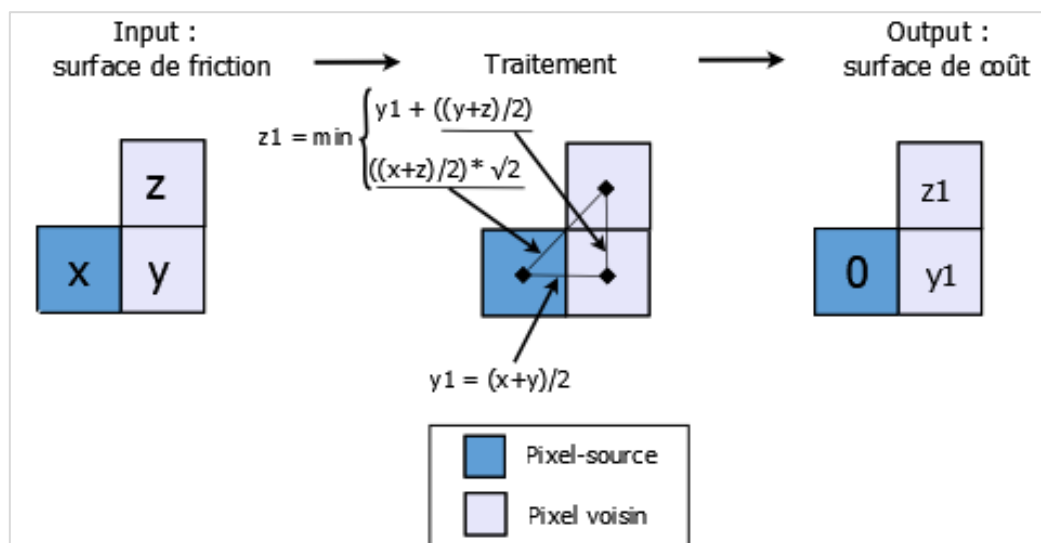


Figure 6 : fonctionnement de l'algorithme de surface de coût en mode maillé

Pour aller plus en détails, ESRI (2016a) montre que l'algorithme débute à la source (pixel de départ) et utilise durant tout le processus une liste ordonnée (*heap*) mise à jour à chaque itération. Comme le montre l'annexe 1, cette liste contient les coûts cumulés depuis le pixel de départ, de tous les pixels voisins accessibles depuis chacun des pixels déjà traités par l'algorithme. Le pixel correspondant au premier élément de cette liste (coût minimum) est choisi, et sa valeur sauvegardée dans le futur raster créé en sortie (*output*). La liste est alors mise à jour pour prendre en compte le voisinage du nouveau pixel courant, qui vient s'ajouter aux pixels déjà disponibles précédemment mais non sélectionnés. L'algorithme procède de cette manière jusqu'à atteindre soit les bords de la fenêtre, soit une valeur seuil de coût. Enfin, s'il trouve un chemin plus avantageux pour un pixel qui a déjà été attribué, ce dernier est mis à jour et ses voisins (déjà attribués ou non) se retrouvent dans la liste. Ceci permet d'assurer un coût cumulé minimum en chaque pixel de l'image depuis la source.

Comme mentionné au 2.1.2, les premiers auteurs à énoncer des algorithmes de surfaces de coût sont Eastman (1989, pour le logiciel Idrisi) et Douglas (1994). Les logiciels-SIG existants et ceux qui ont été développés par la suite ont progressivement incorporé un outil de création de surface de coût : Idrisi, GRASS (2003), SAGA (2004) et ArcGIS (ESRI, 2018a) en sont des exemples. Les algorithmes de ces logiciels-SIG comportent toute une série d'options avancées qui peuvent permettre de les différencier. ArcGIS (ESRI, 2018a) propose

par exemple un taux de résistance pouvant symboliser la fatigue d'un voyageur, en augmentant la friction des pixels visités au fur et à mesure que le coût cumulé depuis le pixel-source est important. GRASS (2003) propose quant à lui d'arrêter le processus lorsque des pixels donnés sont atteints par la surface de coût, mais également de considérer 16 pixels voisins au lieu de 8 dans le calcul de la surface de coût, selon le mouvement en L du cavalier au échec (*Knight's move*) repris à la figure 7 ci-dessous :

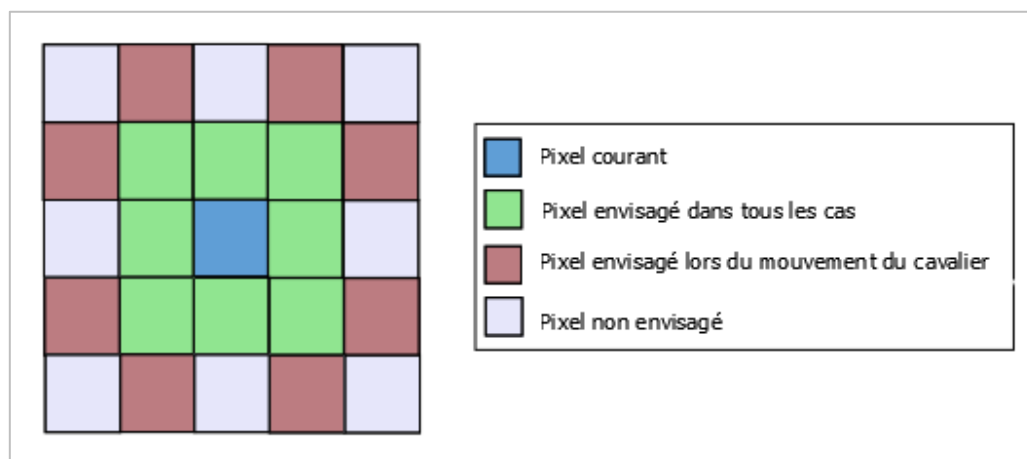


Figure 7 : option du mouvement du cavalier (*Knight's move*) via l'algorithme de surface de coût du logiciel GRASS

La prise en compte de ce mouvement améliore la précision du calcul, car l'algorithme va désormais considérer huit pixels supplémentaires à chaque itération, calculer pour chacun le coût total cumulé et choisir celui qui le minimise. En revanche, cette contrainte supplémentaire utilise plus de ressources et est responsable de ralentissements non négligeables affectant le processus.

Par après, certains de ces algorithmes ont été implémentés dans des bibliothèques accessibles par d'autres logiciels-SIG comme c'est le cas de SAGA (2004) et GRASS (2003) accessibles sur le logiciel QGIS, ou encore accessibles par des scripts Python (cas d'ArcGIS (ESRI, 2018a) avec ARCPY (ESRI, 2018b) et de R avec IGRAPH (The Igraph Core Team, 2015) notamment). Enfin, des bibliothèques Python disposent de tels algorithmes comme c'est le cas de NETWORKX (NetworkX developers, 2019) et SCIKIT – IMAGE (van der Walt *et al*, 2014) . Pour résumer les possibilités d'utilisation des principaux algorithmes :

- GRASS : logiciel open source mais algorithme programmé en C++, ce qui le rend difficilement transposable en Python ;
- ArcGIS : logiciel propriétaire, algorithme par conséquent impossible à consulter et donc à modifier ;
- SAGA : cf. GRASS ;
- Idrisi : cf. ArcGIS ;
- R (IGRAPH) : logiciel open source, avec bibliothèque IGRAPH transposée en plusieurs langages de programmation dont Python. L'inconvénient de cette bibliothèque est que l'implémentation en mode raster ne semble pas être développée, seul le mode

vectorel est a priori utilisable selon la documentation. Cette implémentation est cependant disponible directement sur le logiciel R ;

- NETWORKX : librairie Python open source, mais essentiellement en mode vectoriel. La documentation liée au raster est très peu développée ;
- SCIKIT – IMAGE : librairie open source, mais Python et Cython (extension au langage Python permettant de manipuler une partie du code en C pour des questions de performance). L'utilisation de la librairie est fonctionnelle, et est celle qui est reprise dans la documentation officielle de GDAL / OGR (Erickson, J. *et al*, 2013). Néanmoins, son utilisation requiert une version récente de Python (Python 3.7 a été utilisé dans le cadre de ce travail) et l'installation d'une version de Microsoft Visual Studio (MVS) compatible avec la version de Python utilisée (la version 2017 de MVS dans ce cas).

Comme mentionné précédemment, aucun logiciel-SIG ne possède un outil permettant de répondre de manière complète et efficace à la question de recherche. Celle-ci porte, pour rappel, sur la délimitation d'une zone d'ancrage sur base d'une combinaison de surfaces de coût temporel régressif depuis une série de crimes connus de manière spatio-temporelle, tout en se focalisant sur la solution d'erreurs affectant le processus.

En effet, si ces logiciels possèdent généralement les fonctions suffisantes pour effectuer l'ensemble du processus, l'obtention d'un résultat identique nécessite néanmoins un nombre d'étapes considérablement plus important. De plus, leurs fonctions ne sont pas toujours optimales, comme c'est par exemple le cas de la fonction de création de surface de coût du logiciel Idrisi (**costgrow**), qui est extrêmement lente. Sachant que cet algorithme est utilisé autant de fois que le nombre de crimes, et ce sur des *rasters* volumineux, il paraît évident d'éviter son utilisation. Enfin, toutes les pistes de solutions proposées pour la résolution des problèmes affectant le processus (cf. 3.1) ne peuvent être évaluées via les outils existants des logiciels-SIG.

2.3 Applications géographiques

Si les applications géographiques utilisant les algorithmes de surfaces de coût sont nombreuses, ces derniers peuvent être adaptés au contexte de l'application considérée.

C'est notamment le cas de l'hydrographie, dont la transposition des principes des surfaces de coût permet le calcul et la modélisation d'un flux quelconque dans un bassin versant (généralement des précipitations, mais le cas d'un polluant déversé suite à l'accident d'un camion en amont peut par exemple être considéré). Le résultat de ce calcul est l'importance de l'accumulation du flux en chaque pixel de l'image correspondant au bassin versant. De ceci, il ressort que l'algorithme doit prendre en compte plusieurs éléments, ce qui explique la raison pour laquelle les logiciels spécialisés tels qu'Idrisi (Jenson & Domingue, 1988), GRASS (2019) et ArcGIS (ESRI, 2016b) ont implémenté un algorithme spécifique à l'hydrographie (fonctions **runoff**, **r.accumulate** et **Accumulation de flux** respectivement). D'abord, la surface de friction est toujours liée aux élévations des pixels (ce qui nécessite un MNT – Modèle Numérique de Terrain), car le déplacement d'un fluide se fait selon la ligne de plus grande pente topographique. Ensuite, l'algorithme calcule une valeur d'accumulation

en chaque pixel qui n'est pas définitive : tant qu'un des huit pixels voisins du pixel contenant le flux à l'instant t présente une élévation moins élevée que celui-ci, le flux continue de se propager. En pratique, ces algorithmes tels qu'implémentés par ArcGIS (ESRI, 2016b) et GRASS (2019) n'ont pas directement besoin d'une surface de friction mais d'un raster contenant la direction du flux (qui est néanmoins calculée grâce à un MNT). Ce raster de direction du flux spécifie, pour chaque pixel courant dans une fenêtre 3x3, le pixel voisin parmi les huit possibles où le flux est censé s'écouler. Dans le cas de l'algorithme d'ArcGIS (ESRI, 2016b), l'encodage des directions de ce *raster* (cf. figure 8, à gauche) est réalisé selon un système binaire, de 2^0 (i.e. 1, correspondant à l'est) à 2^7 (c'est-à-dire 128, pour le nord-est). L'algorithme de GRASS (2019), repris également à la figure 8 (à droite), utilise quant à lui un encodage correspondant à l'orientation en degrés depuis l'est (valeur de 360 degrés). Par exemple, pour un écoulement d'un flux vers le nord-est, une valeur de 45 (degrés) sera appliquée au pixel courant, ou encore 315 degrés si l'écoulement est réalisé vers le sud-est.

32	64	128
16		1
8	4	2

135	90	45
180		360
225	270	315

Figure 8 : comparaison des encodages des valeurs de directions autour d'un pixel courant selon ArcGIS (gauche) et GRASS (droite)

La figure 9 reprend la conversion d'un raster d'élévation en *raster* de direction de flux avec l'encodage des directions selon ArcGIS. On peut déduire de celui-ci que l'écoulement va se concentrer vers le pixel associé à la valeur 4 le plus au sud du *raster*. Au final, le déplacement d'un pixel à un autre dans cette configuration est immuable et ne dépend que des différences d'élévations entre un pixel et chacun de ses 8 voisins. Autrement dit, on ne s'intéresse pas à la quantité à propager mais à la direction dans laquelle cette quantité va s'écouler, ce qui diffère de l'algorithme de surface de coût expliqué plus tôt pour lequel la valeur d'un pixel peut être modifiée si l'algorithme trouve par après un chemin qui apportera une valeur plus faible en ce pixel.

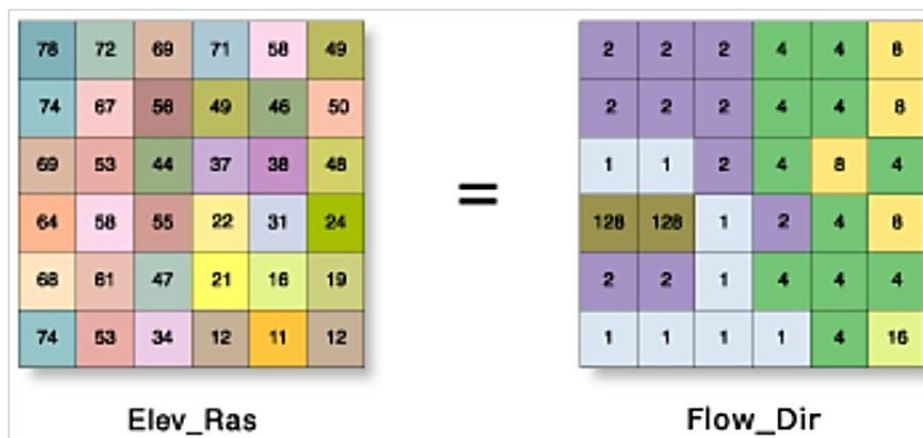


Figure 9 : exemple de traitement de l'algorithme de direction de flux sur un *raster* d'élévation (ESRI, 2016c)

L'autre application fortement utilisée pour laquelle l'algorithme de surface de coût expliqué au 2.2.2 est développé est le transport. En fonction de ce que l'on propage sur un réseau routier, on peut y déterminer le chemin le plus court, le chemin le plus rapide ou encore le chemin qui réduit le coût financier au maximum. De nombreux logiciels prennent en compte ces informations pour estimer le temps d'arrivée à destination (Google maps, Waze, Michelin), quel que soit le moyen de transport utilisé, et permettent souvent d'effectuer ces calculs en temps réel en fonction des aléas de la route. Les surfaces de coût propageant la distance sont également essentielles dans le cadre des modèles gravitaires, comme celui de Reilly-Converse (Reilly, 1931 ; Converse, 1949) qui détermine l'aire d'influence de centres urbains ou encore le modèle de Huff (Huff, 1963) qui délimite une zone d'influence probable d'un centre d'achat et évalue sa masse de clientèle potentielle. Un autre modèle utilisant ces surfaces de coût est le modèle de localisation des activités économiques de Weber et Friedrich (1929), qui énonce trois éléments influençant le choix de localisation parmi lesquels la minimisation des coûts de transport (Mérenne-Schoumaker, 2011). Enfin, de telles surfaces sont utilisées en profilage géographique (Trotta *et al*, 2011) comme expliqué précédemment, ou peuvent encore être utilisées dans des modèles d'accessibilité, par exemple celle des hôpitaux d'une région afin de mettre en évidence les zones trop peu desservies par les services hospitaliers (Bilasco *et al*, 2015).

2.4 Problèmes courants

2.4.1 Erreurs sur l'espace

S'il a été mentionné dans l'introduction l'avantage du mode maillé par rapport au mode vectoriel, celui-ci présente également de nombreux désavantages par rapport au mode vectoriel.

D'abord, la rasterisation nécessaire pour passer d'un mode à l'autre amène de l'imprécision due à la considération non plus d'un point (0D – élément ponctuel) mais d'un pixel (2D – élément surfacique), et elle sera d'autant plus importante que la résolution choisie pour la rasterisation est grossière. Un algorithme de rasterisation d'un élément linéaire couramment utilisé est celui de Bresenham (1965), développé à l'origine pour les tables traçantes servant pour les impressions graphiques d'éléments vectoriels. Cet algorithme, dont un exemple de résultat est repris à la figure 10, a pour objectif de sélectionner les pixels d'une image qui correspondent le mieux à un élément linéaire donné, en considérant les huit mouvements linéaires que peut suivre le traceur, repris à la figure 11.

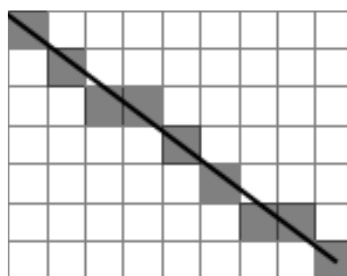


Figure 10 : exemple de résultat de rasterisation d'un élément linéaire avec l'algorithme de Bresenham (StackExchange, 2014 ; modifiée)

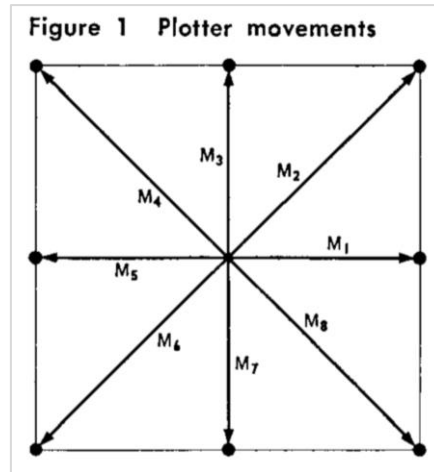


Figure 11 : huit mouvements linéaires pouvant être réalisés par la table traçante (Bresenham, 1965)

Un second algorithme de rasterisation possible est le *Polygonal Map Quadtree*, ou *PM Quadtree* (Samet & Webber, 1985). Cet algorithme est une méthode d'indexation spatiale également utilisée dans le cadre de processus de rasterisation des polygones et polygones, car elle est plus efficace que l'algorithme de Bresenham (Shaffer & Ursekar, 1992). Elle fonctionne par subdivisions successives de l'espace en 4 cellules carrées (arbre quaternaire, les nœuds ont 4 enfants) tant que certains critères de décomposition ne sont pas respectés.

Dans le cas d'éléments linéaires tels que des tronçons routiers (et également des limites de polygones), cette rasterisation s'accompagne d'un « effet escalier » (ou *aliasing*) pour les éléments linéaires qui ne sont ni à la verticale ni à l'horizontale (Donnay, 2015), comme le montre la figure 12 (figure de gauche). Il faut toutefois noter que si des algorithmes *anti-aliasing* ont été développés pour pallier à ce problème, ils ne sont pas utilisables dans notre cas. En effet, la figure 12 (à droite) montre comment ces algorithmes répartissent l'intensité du remplissage de pixels entre plusieurs pixels candidats au déplacement linéaire, afin de réduire cet effet escalier. Cette technique ne fonctionne pas dans le cas d'images binaires, pour lesquelles seulement deux intensités sont possibles : le pixel est soit dans le réseau routier, soit en dehors, mais il ne peut pas y être en partie.

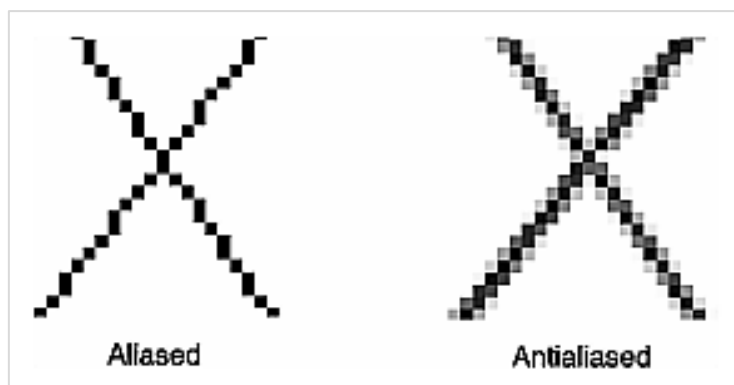


Figure 12 : comparaison entre la rasterisation avec et sans *anti-aliasing* (DEI-ISEP, 2007)

Un deuxième désavantage inhérent au mode maillé est la manipulation de graphes non planaires, comme c'est le cas dans un réseau de transport classique. En effet, un réseau routier ne peut pas s'inscrire dans un plan dû à la présence de ponts et de tunnels, et cette spécificité n'est pas prise en compte en mode maillé. Il n'y a aucune différence entre un carrefour standard et une route qui « croise » un pont ou un tunnel : la propagation se fera de la même manière dans les deux cas, ce qui induit automatiquement une erreur dans le calcul des coûts.

Enfin, une dernière erreur sur l'espace tant en mode maillé qu'en mode vecteur est due à la localisation des sites de crime, qui est a priori hors du réseau routier. En mode vectoriel, Ritsema van Eck et de Jong (2002) ont étudié la manière de rattacher un point associé à des données hors d'un graphe routier sur celui-ci, en fonction de divers cas. Outre des questions sur la pertinence du rattachement en fonction de la précision des données par rapport à celle du réseau routier et sur la définition de barrière infranchissable pour le rattachement, ils font la distinction entre le rattachement à un nœud du graphe et à n'importe quel point du graphe. En mode maillé, cette différence de configurations ne s'applique pas car on enlève cette distinction lorsque l'ensemble du réseau est transformé en un ensemble de pixels. Au final, il faudra rattacher le point de données au pixel du réseau routier le plus proche du site de crime enregistré.

2.4.2 Erreurs sur le temps

Une deuxième catégorie d'erreurs non négligeables est l'erreur sur le temps. Trotta *et al* (2011) en mentionnent trois sources dans le cas du profilage géographique :

- l'impossibilité pour la police de capturer les détails temporels ;
- l'incapacité pratique / psychologique de la victime de spécifier une heure de crime ;
- la variabilité des habitudes du criminel, telles que l'heure de départ du crime.

D'une manière générale, toute mesure de temps contient des erreurs. Cette erreur temporelle se propage à travers le réseau routier et influence inévitablement le résultat obtenu. Par exemple, si on considère un crime relevé à 19:05 alors qu'il s'est en réalité déroulé à 19:00, la propagation temporelle régressive sur le réseau routier présentera une erreur temporelle de 5 minutes en chaque pixel. Étant donné que l'on considère plusieurs crimes qui présentent chacun une erreur temporelle propre, toutes ces erreurs vont se combiner sur le résultat final et induiront, en conséquence, une erreur spatiale plus ou moins importante sur la délimitation de la zone d'ancrage. Si ces erreurs ne peuvent pas être corrigées, elles peuvent en revanche être prises en compte dans le calcul (cf. 3.1.3).

2.4.3 Erreurs sur la surface de friction

Une dernière catégorie de sources d'erreurs est celle des erreurs sur la surface de friction. Une première erreur de cette catégorie découle d'une erreur sur l'espace lors de la rasterisation. En effet, on ne mesure plus la distance euclidienne comme auparavant, mais désormais la distance liée au nombre de pixels et leur configuration. Cette distance surestime généralement la distance euclidienne comme le montre la figure 13, induisant une différence plus ou moins importante en fonction de la configuration. Étant donné que les valeurs de la

surface de friction dépendent de la taille des pixels (i.e. résolution spatiale) et de la vitesse maximale autorisée, cette surestimation de la distance va induire une surestimation des valeurs de friction.

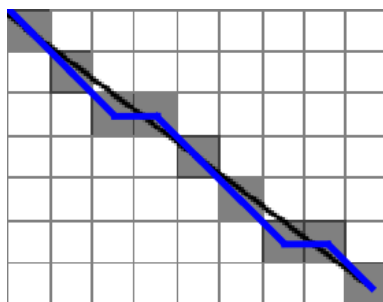


Figure 13 : comparaison entre la distance euclidienne (noir) et la distance après rasterisation (bleu) d'un élément linéaire (StackExchange, 2014 ; modifiée)

Une seconde erreur provient de l'incapacité du mode maillé à fournir de l'information liée au réseau routier. En mode vectoriel par exemple, un tronçon possède un ensemble d'attributs dont le sens de circulation, qui est important dans cette analyse car il permet d'éviter la propagation du coût sur les tronçons routiers où un conducteur ne peut pas s'engager. Ce mode présente également une information sur les nœuds du graphe (i.e. les extrémités d'une ou plusieurs arêtes), de laquelle on peut extraire leur complexité et potentiellement y définir un temps de franchissement spécifique en fonction de celle-ci mais également en prenant en compte la présence ou non de feux de circulation routière. Ces deux informations peuvent ensuite être utilisées pour augmenter la friction en ces nœuds en fonction de leur importance effective : à un important carrefour sera appliquée une forte friction ; à l'inverse, la friction à un simple croisement routier ne sera que peu ou pas modifiée. En mode maillé par contre, l'absence d'attributs en dehors de la friction empêche la considération dans le calcul des sens uniques, pour reprendre l'exemple mentionné plus tôt. De plus, les nœuds sont de simples pixels parmi tant d'autres, la complexité du nœud ne peut dès lors pas être extraite. Tout ceci constitue une limite importante à l'exactitude de la surface de friction.

Enfin, une dernière erreur sur la surface de friction provient de la variabilité du trafic routier et des aléas que l'on peut rencontrer durant un déplacement, pouvant notamment être causés par un accident ou un embouteillage. Cette variabilité n'est pas reflétée dans les données de vitesse, qui sont statiques car associées aux vitesses maximales officiellement autorisées sur le réseau routier. Cette absence de prise en compte des données en temps réel introduit une erreur de friction proportionnelle à la différence existante entre les vitesses réelle et théorique. Dans certains cas extrêmes, le trajet effectué peut même différer du trajet théoriquement le plus court précisément pour éviter ces zones de ralentissements, et donc amener une erreur encore plus substantielle.

3. Objectifs

La question de recherche associée à ce mémoire porte, pour rappel, sur la création d'un processus de délimitation d'une zone d'ancrage sur base d'une combinaison de surfaces de coût temporel régressif, depuis une série de crimes connus de manière spatio-temporelle. Ce travail doit prouver l'opérationnalité de la méthode à travers un prototype d'extension logicielle, et se concentre en particulier sur les problèmes inhérents au processus.

Ces problèmes courants ont été repris au 2.4. Ils introduisent nécessairement des erreurs qui se propagent sur le résultat final, c'est-à-dire la délimitation de la zone d'ancrage. Plus ces erreurs sont nombreuses et importantes, plus cette délimitation risque d'être inexacte et imprécise. Elle sera inexacte lorsque le pixel correspondant au point d'ancrage réel recherché ne fait pas partie des pixels minimisant les différences d'heures de passage, imprécise quand les valeurs des pixels ne permettent pas de pouvoir utiliser le résultat obtenu afin, par exemple, de débiter des recherches pour trouver le repaire d'un criminel.

L'hypothèse posée lors de ce travail est alors la suivante : la levée de ces erreurs, notamment à travers la modification de la rasterisation, permet d'améliorer le processus de délimitation de la zone d'ancrage via le parcours d'un graphe par temps régressif en mode maillé. Pour lever ces diverses erreurs, plusieurs pistes de solution sont envisagées et sont présentées ci-dessous.

3.1 Pistes de solution

3.1.1 Positionnement des crimes sur le réseau routier

Le crime n'est généralement pas localisé sur le réseau routier, où la propagation des coûts est réalisée. Or, l'origine depuis laquelle démarre cet algorithme de surface de coût doit nécessairement être située sur ce réseau routier. Il apparaît donc nécessaire de rattacher le point correspondant au site de crime, au pixel du réseau routier le plus proche spatialement de celui-ci. Ce pixel présentera alors un coût nul et sera le point de départ de l'algorithme. Cette méthode présente néanmoins un problème lors d'importantes distances entre le réseau routier et le site de crime, ce qui pourrait conduire à introduire une autre friction hors réseau routier pour prendre en compte la marche effectuée entre le réseau routier et le site de crime.

3.1.2 Adaptation du temps de parcours à la date et l'heure

La connaissance de l'état du trafic routier durant la période de temps précédant l'heure du crime est essentielle pour limiter l'erreur sur la surface de friction. En effet, les aléas que peut rencontrer un automobiliste tels qu'une importante densité du trafic, l'occurrence d'accidents ou encore la présence de travaux, modifient nécessairement son temps de parcours. Dans certains cas, ces aléas peuvent même conduire le criminel à modifier son parcours, qui sera distinct d'un trajet « rationnel ». Etant donné que par défaut, la surface de friction ne permet que la considération d'un trajet idéal durant lequel l'automobiliste ne rencontrera aucun aléa routier et se déplacera selon les vitesses maximales autorisées, l'erreur occasionnée par ces aléas peut rapidement devenir considérable. Tout ceci explique pourquoi ces informations sont précieuses dans cette analyse, et permettent d'améliorer la mise en évidence de la zone d'ancrage du criminel.

Pour ce faire, différents outils permettent en théorie, via leur interface de programmation d'application (API – *Application Programming Interface*), de récupérer ce type d'informations. Une API est un ensemble de fonctions qui permettent d'accéder aux fonctionnalités d'une application au travers d'un langage de programmation, tel que Python par exemple. En pratique, nous verrons que la récupération de ces données via ces API souffre de nombreux problèmes. Elles n'ont par conséquent pas pu être utilisées dans le cadre de ce travail (cf. chapitre 4).

3.1.3 Prise en compte de l'imprécision sur le temps

Les erreurs temporelles inévitables mentionnées au 2.4.2 ont pour conséquence qu'il est peu probable, voire impossible, d'obtenir un pixel où la différence temporelle maximale de passage entre toutes les surfaces de coût est nulle. Les valeurs des pixels de chaque surface de coût présentent nécessairement une erreur plus ou moins importante sur l'heure de passage, et l'importance de cette erreur est propre à chacune d'entre elles.

Devant le postulat d'une heure de départ constante du point d'ancrage vers les différents sites de crime, la méthode de l'étendue des heures de passage prend en compte les erreurs restantes qui affectent le processus, dont ces erreurs sur le temps font partie. L'objectif est précisément de limiter les autres erreurs affectant le processus, qui peuvent théoriquement être solutionnées, afin de ne garder que cette erreur sur le temps qui elle peut uniquement être prise en compte.

Par ailleurs, l'imprécision sur l'heure du crime peut être considérée par la création de deux surfaces de coût pour un même crime. Par exemple, une imprécision de 5 minutes sur un crime relevé à 19h peut se refléter via deux surfaces de coût, l'une démarrant du site de crime à 18h55 et l'autre à 19h05. Ces deux surfaces peuvent alors être introduites dans le processus en lieu et place de l'unique surface de coût de départ.

Enfin, les valeurs minimales des étendues obtenues témoignent du respect ou non des postulats évoqués plus tôt (cf. triple postulat, introduction). En effet, si celles-ci sont trop importantes, on peut supposer qu'au moins un des postulats d'une heure de départ constante, de l'utilisation d'un même mode de transport ou encore d'un point d'ancrage constant n'est pas vérifié.

3.1.4 Calcul des distances post-rasterisation

Comme mentionné au 2.4.3, la résolution des pixels est utilisée dans la construction de la surface de friction, ce qui signifie que l'on prend en compte la distance de rasterisation et non la distance euclidienne. Cette distance tend à surestimer la distance réelle, introduisant une erreur. Ce problème peut être résolu par le calcul des distances après la rasterisation, en reliant la distance connue des tronçons (données attributaires du réseau routier vectoriel) avec leur nombre de pixels obtenus lors de la rasterisation. Ainsi, la friction temporelle totale du tronçon est reliée à sa longueur exacte.

3.1.5 Prise en compte des données attributaires du mode vectoriel

Il a été mentionné au 2.4.1 et 2.4.3 que le mode *raster* ne prend en compte ni la non-planéité des graphes routiers, ni le sens de circulation des tronçons routiers. Ces informations, perdues

lors de la rasterisation, sont reprises dans les données attributaires du mode vectoriel et peuvent être prises en compte dans une liste annexe au *raster*. Cette liste peut par exemple reprendre l'ensemble des pixels appartenant à un pont ou un tunnel concernant le premier cas, tandis que pour le second cas elle peut contenir les possibilités de déplacement aux pixels des tronçons routiers voisins en partant d'un pixel donné. Dans les deux cas, cette prise en compte doit se faire via une modification de l'algorithme de surface de coût. Au final, la résolution de l'un et/ou l'autre de ces problèmes permet d'améliorer les surfaces de coût obtenues.

3.2 Plan de l'analyse

La réalisation de ce travail est divisée en plusieurs parties, constituant chacune un chapitre. Ces parties, selon leur ordre d'apparition dans l'analyse, sont les suivantes :

- construction de la surface de friction améliorée, par prise en compte des longueurs réelles connues et non des longueurs provenant de la rasterisation ;
- transposition d'un algorithme de surface de coût adéquat, avec comme objectif de le rendre modifiable en vue de l'amélioration des surfaces de coût (cf. 3.1.5) ;
- explication de l'implémentation du processus, d'abord de manière générale, ensuite de manière plus technique ;
- tentative d'amélioration des surfaces de coût, par prise en compte de données attributaires du mode vectoriel en mode maillé (cf. 3.1.5) ;
- déploiement du processus créé dans un prototype d'extension logicielle.

De manière générale, chacune des parties mentionnées ci-dessus nécessite la validation des parties précédentes afin de pouvoir être considérée. Ces validations se réalisent à l'aide sur logiciel-SIG QGIS. L'ensemble du travail a quant à lui été développé au moyen du langage de programmation Python version 3.7, dont les versions récentes de QGIS sont munies. L'environnement de développement intégré (IDE – *Integrated Development Environment*) utilisé dans le cadre de ce mémoire est PyCharm 2018.2.2. La version de QGIS est quant à elle la version 3.4.7 couplée à GRASS 7.6.1. Enfin, la version 4.9.0 du logiciel Qt Creator est utilisée pour la construction de l'interface graphique de l'extension logicielle.

3.3 Construction du jeu de données

L'hypothèse énoncée précédemment ne peut être validée (ou rejetée) sans un jeu de données préalablement récupéré / construit. Si des données existent notamment au niveau de la police, qui possède des informations sur les crimes situés sur son territoire, celles-ci sont confidentielles et ne peuvent par conséquent pas être utilisées dans le cadre de ce travail. Par contre, des données arbitraires relatives à un cas théorique peuvent l'être, et pour ce faire, deux cas d'école relatifs à la logique de déplacement d'un auteur de plusieurs crimes existent et sont repris à la figure 14. Comme le mentionne Trotta (2012) citant Canter et Larkin (1993), le premier comportement est celui du maraudeur : le criminel part de son point d'ancrage et commet des crimes tout autour de celui-ci, de manière uniforme. Le second comportement est le navetteur : le criminel parcourt une certaine distance plus ou moins grande depuis son point d'ancrage avant de commettre ses méfaits.

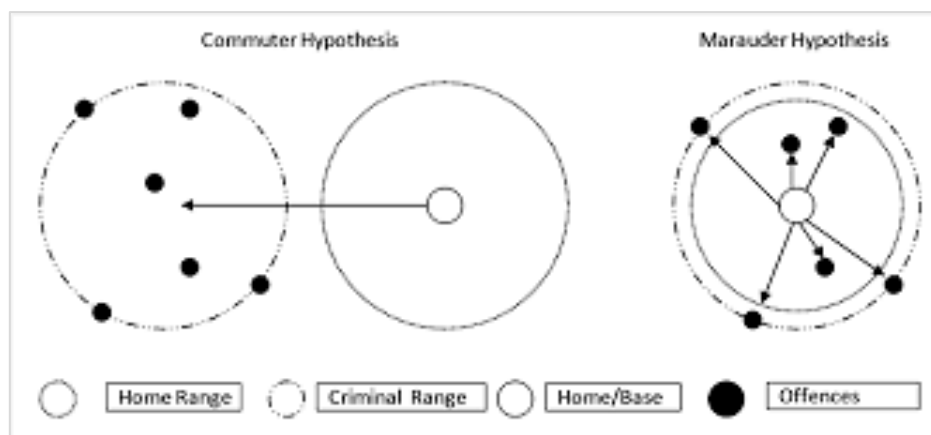


Figure 14 : comparaison entre les comportements de maraudeur et de navetteur (source : http://www.hellenicpolice.gr/images/stories/periodiko/289_3.pdf)

Pour tester la validité du processus, un point d'ancrage arbitraire du criminel a été choisi, et une surface de coût partant de ce point a été construite. Arbitrairement toujours, cinq sites de crimes ont été définis autour du point d'ancrage, dont le coût des pixels associés depuis le pixel correspondant au point d'ancrage du criminel est donc connu. Ensuite, on « oublie » le point d'ancrage du criminel, puis on tente de le retrouver grâce aux données spatio-temporelles définies. Si on arrive à retrouver le point d'ancrage à partir des cinq sites de crimes arbitraires, alors le processus est validé. Les données spatiales arbitraires relatives aux sites de crimes sont ici définies selon le comportement du maraudeur.

Enfin, il ne faut pas oublier que ce travail n'est pas réalisable sans les données vectorielles du réseau routier. Celles-ci peuvent être obtenues soit à l'échelle régionale via le géoportail¹ de la Wallonie, soit à l'échelle nationale via le géoportail² de l'Etat fédéral de la Belgique. Les données vectorielles utilisées pour ce travail ont quant à elles été récupérées via Marie Trotta de l'Université de Liège, données qui lui avaient servi dans le cadre de ses recherches en profilage géographique (Trotta, 2012). Celles-ci sont relatives au réseau routier belge à l'échelle nationale connu avec une précision de 5 m.

¹ <http://geoportail.wallonie.be/home.html>

² <https://www.geo.be/#!/home?l=fr>

4. Amélioration de la surface de friction

4.1 Rasterisation et construction de la surface de friction

Comme mentionné précédemment dans ce travail, la surface de friction rend compte du coût que requiert la traversée d'une certaine portion du territoire matérialisée par un pixel, coût ici calculé en un temps de parcours. La construction de cette surface de friction débute par l'obtention de ces pixels, par rasterisation du réseau routier vectoriel. Pour ce faire, des bibliothèques spécialisées dans la manipulation de données vectorielle et maillée (*raster*) ont été munies de cet algorithme de rasterisation de données vectorielles. Il existe aujourd'hui deux bibliothèques majeures spécialisées dans la manipulation de données *raster*. La plus connue est la bibliothèque GDAL, qui est utilisée dans de nombreux logiciels tels qu'ArcGIS, QGIS, SAGA, GRASS, IDRISI, Google Earth et R. La bibliothèque RASTERIO est un second choix possible pour effectuer cette rasterisation. Celle-ci est en réalité calquée sur GDAL, sauf qu'elle exploite pleinement les fonctionnalités de Python là où GDAL est basé sur le GDAL C API, signifiant que les programmes Python qui utilisent GDAL tendent à fonctionner comme un programme en C (Rasterio, 2018). A noter que les deux algorithmes de rasterisation utilisent l'algorithme de Bresenham, et non celui du *PM Quadtree* (cf. 2.4.1)

Ces algorithmes proposent diverses options afin de contrôler certains aspects de la rasterisation. Outre les options « standard » de choix de la résolution, des valeurs « *No Data* », des transformations entre systèmes de coordonnées et de la rasterisation en fonction d'une valeur passée en argument ou bien d'un champ d'attributs, ces algorithmes présentent également l'option « *all_touched* ». Celle-ci permet de choisir si la rasterisation doit prendre en compte tous les pixels qui croisent l'élément linéaire ou bien si elle doit simplement effectuer une rasterisation « standard ». Pour comprendre la différence entre les deux, on compare un tronçon du réseau routier vectoriel (dont on connaît la longueur exacte) à la distance obtenue après rasterisation. Dans cet exemple, la longueur du tronçon vectoriel est de 74.22 m et chaque pixel mesure 10 m de côté. Dans le cas où l'on ne prend pas en compte cette option (cf. figure 15, à gauche), le parcours de ce tronçon nécessitera 4 déplacements diagonaux et 2 horizontaux, ce qui donne :

$$4 * 10 * \sqrt{2} + 2 * 10 = 76.8 \text{ m}$$

soit une surestimation de la longueur du tronçon d'un peu moins de 3.48 % de sa longueur totale. Si on prend maintenant en considération cette option (cf. figure 15, à droite), le parcours de ce tronçon va désormais être constitué de 4 déplacements horizontaux en plus qu'auparavant, portant le total à 4 déplacements diagonaux et 6 déplacements horizontaux. On obtient donc ici :

$$4 * 10 * \sqrt{2} + 6 * 10 = 116.8 \text{ m}$$

soit une surestimation de la longueur du tronçon d'un peu plus de 57.36 % de sa longueur totale.

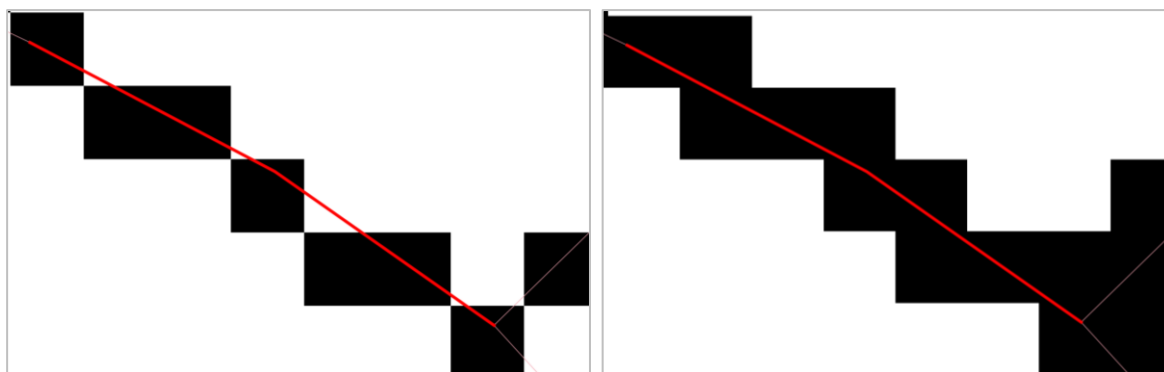


Figure 15 : comparaison de rasterisations sans (gauche) et avec (droite) considération de l'option *all_touched*

Évidemment, cette différence est d'autant plus marquée que le tronçon routier considéré est diagonal : pour un tronçon parfaitement horizontal ou vertical, il n'y a aucune différence entre la prise en compte ou non de cette option. Il ressort donc de cette comparaison que cette option ne doit pas être prise en compte, d'abord parce qu'elle surestime substantiellement la longueur des tronçons routiers diagonaux, ensuite parce que le choix de déplacements à l'intérieur d'un même tronçon n'a pas vraiment de sens et complexifie le problème.

Une dernière option proposée par GDAL est la possibilité d'effectuer des requêtes SQL afin de rasteriser les entités qui satisfont à certains critères spécifiés. RASTERIO ne possédant pas cette fonctionnalité, son algorithme apparaît par conséquent comme moins complet que celui de GDAL.

Au final, pour en revenir à la construction de la surface de friction dans notre cas, celle-ci est en théorie construite par division de la résolution du pixel, par la vitesse autorisée sur ce pixel. Par exemple, si on considère une vitesse de 90 km/h (ce qui, ramené en mètre par seconde, donne 25 m/s) sur ce pixel de 10 m de côté, on obtient la friction temporelle t suivante (cas d'un déplacement vertical ou horizontal) :

$$t = \frac{d}{v} = \frac{10}{25} = 0.4 \text{ s}$$

avec d la distance parcourue qui correspond à la résolution du pixel et v la vitesse autorisée en ce pixel. Cette rasterisation soulève néanmoins un problème non négligeable : la longueur totale du tronçon routier dépendra du nombre de pixels créés durant sa rasterisation, et sera donc modifiée alors qu'elle est connue avec précision.

4.2 Amélioration des temps de parcours sur le réseau rasterisé

Pour résoudre ce problème de perte d'information de la longueur totale du tronçon routier lors de sa rasterisation, il faut introduire cette donnée directement dans le processus de construction de la surface de friction. L'idée est de lier le nombre de pixels correspondant à chaque tronçon routier, à sa longueur connue avec précision. Pour ce faire, le coût temporel total du tronçon est calculé par division de sa longueur totale reprise dans les données attributaires du réseau routier vectoriel, par la vitesse v étant autorisée. Ensuite, ce coût temporel est normalisé en le divisant par le nombre de pixels couvrant ce tronçon routier

rasterisé. Ainsi, on obtient une friction temporelle unitaire (i.e. par pixel) calculée depuis une longueur indépendante à la fois de la résolution spatiale et du nombre de pixels.

Ensuite, il a été mentionné plus tôt que la prise en compte du temps de parcours réel précédant l'heure du crime peut améliorer le processus de délimitation de la zone d'ancrage de manière non négligeable. Divers outils mettent à disposition des développeurs leur interface de programmation d'application (API). L'objectif de l'utilisation de ces API est la récupération des temps de parcours précédant l'heure du crime, afin de prendre en considération les conditions de circulation effectives et non plus théoriques. Il n'y aurait par conséquent plus une seule surface de friction pour tous les crimes, mais bien une surface de friction par crime étant donné que les conditions de circulation sont a priori différentes pour chacun d'entre eux. En pratique, plusieurs API existent : Google Maps, Bing, Tomtom, Here, Open Transport Map... Si ces API ont l'avantage d'être gratuites, moyennant de ne pas dépasser un certain nombre de requêtes aux serveurs, plusieurs problèmes se posent toutefois :

- la disponibilité temporelle restreinte des données : celles-ci ne semblent en effet être disponibles qu'une semaine en arrière dans le meilleur des cas ;
- l'exactitude du résultat : ces données sont calculées grâce aux GPS (des voitures, des téléphones), ce qui signifie que les routes moins empruntées présenteront moins de données. De plus, des vérifications ont été opérées et certains API présentaient des valeurs de vitesses inexactes ;
- le manque d'information : certains API n'offrent que la possibilité de récupérer des données d'incidents routiers et non d'état du trafic.

Pour toutes ces raisons, ces API n'ont pas pu être utilisées dans le cadre de ce travail.

4.3 Validation sur les données

Pour valider cette surface de friction améliorée, reprenons le tronçon pris comme exemple au 4.1. Celui-ci mesure 74.22 m et présente une vitesse autorisée de 70 km/h soit 19.44 m/s. Le temps de parcours de ce tronçon est donc : $\frac{74.22 \text{ m}}{19.44 \text{ m/s}} = 3.818 \text{ s}$. En normalisant le temps de parcours par le nombre de pixels (5), on obtient un coût par pixel d'environ 0.763 s, ce qui correspond au résultat obtenu après vérification de la surface de friction obtenue sur QGIS.

Une comparaison visuelle peut également être opérée entre le réseau routier vectoriel classifié en 3 catégories de vitesses et ce même réseau routier rasterisé à une résolution spatiale de 10 m, sur lequel le calcul de la friction a été réalisé et classé en 3 catégories également. Bien que cette comparaison ne soit pas une validation en soi, elle montre la tendance existante entre d'une part la vitesse autorisée et le nombre de pixels, d'autre part la friction. Les figures 16 et 17 reprennent un échantillon du réseau routier centré sur le village de Grand-Leez (province de Namur, Belgique) en mode vectoriel et *raster*, permettant cette comparaison visuelle. Les vitesses autorisées ont été classées en 3 classes : vitesse d'agglomération (en-dessous de 50 km/h), vitesse intermédiaire (70 km/h) et vitesse des axes plus importants / hors agglomération (90 et 120 km/h). Les frictions calculées ont également été classées en 3 classes dans un souci de comparaison, selon les quantiles du jeu de données. En comparant

les deux figures, la relation entre vitesse autorisée et friction (qui sont par ailleurs inversement proportionnelle, plus la vitesse augmente et plus la friction diminue) apparaît clairement, de même que certains pixels présentant des valeurs plus importantes de friction (1.6 s par exemple) due à la rasterisation d'un tronçon en un seul pixel (couplé à une vitesse autorisée de 11 km/h).

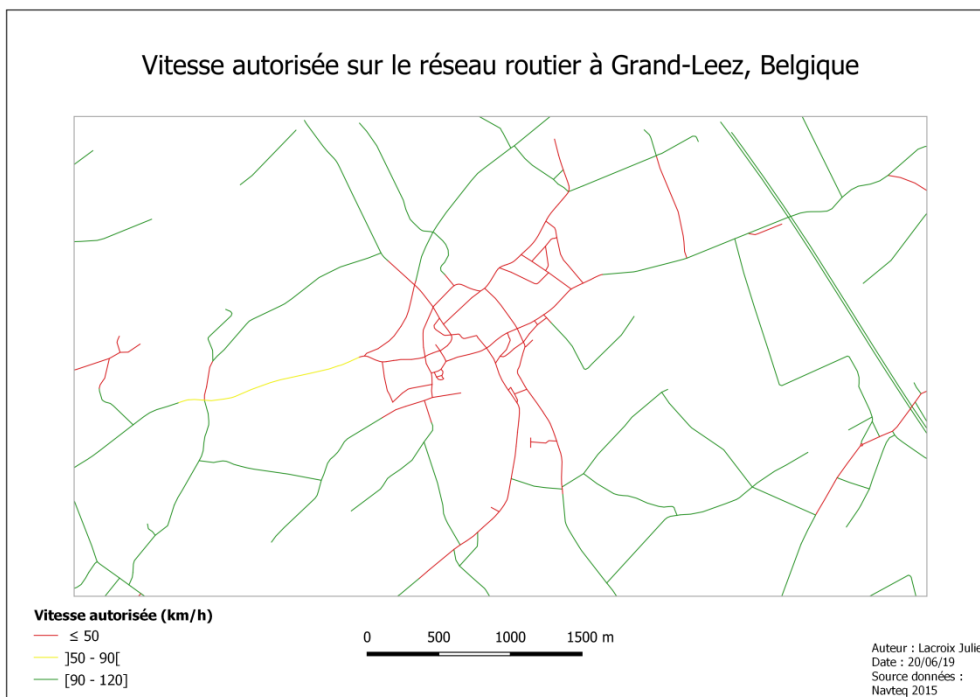


Figure 16 : vitesse autorisée sur le réseau routier à Grand-Leez, Belgique

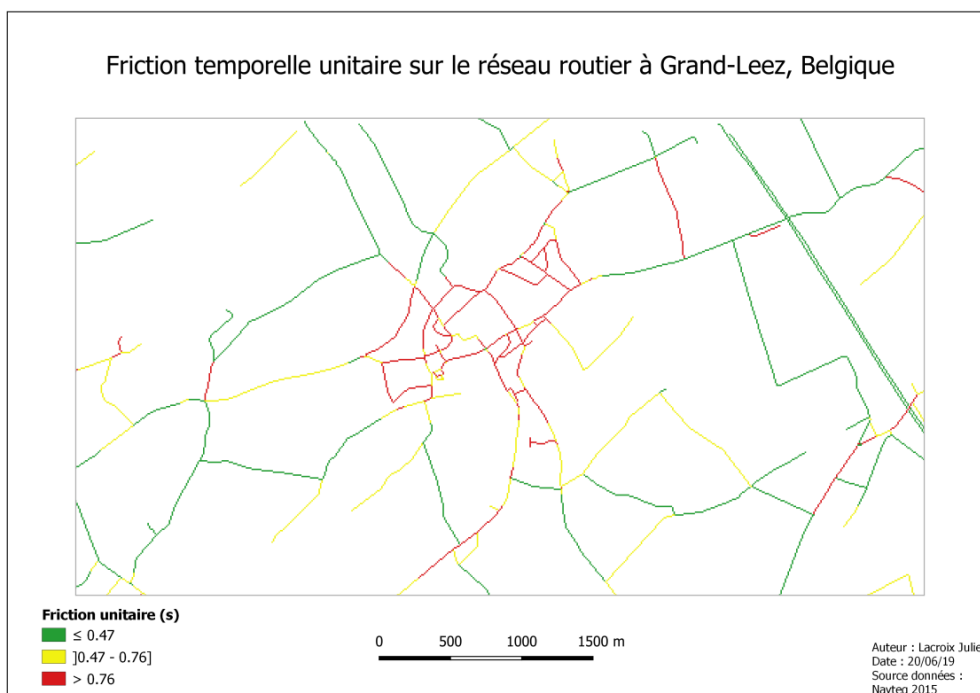


Figure 17 : friction temporelle unitaire sur le réseau routier à Grand-Leez, Belgique

5. Algorithme de propagation

5.1 Identification

Étant donné que plusieurs algorithmes de propagation existent (ceux-ci sont présentés au 2.2.2), un choix doit être opéré. Les critères de sélection de l’algorithme sont d’abord relatifs à la disponibilité du code source (*open source*), ce qui exclut l’utilisation des logiciels propriétaires ArcGIS et Idrisi. Ensuite, le langage de programmation associé entre en considération dans la sélection. En effet, le langage C++ avec lequel sont écrits les algorithmes de surface de coût de GRASS et SAGA est d’une part syntaxiquement plus compliqué à comprendre que le langage Python, d’autre part cela nécessite la traduction du C++ en Python ce qui est particulièrement lourd à réaliser. Le dernier critère de sélection de l’algorithme de propagation est l’implémentation de la propagation en mode raster et sa documentation : la librairie IGRAPH du logiciel R transposée au mode raster est implémentée pour le mode vectoriel uniquement, l’implémentation du mode raster n’étant disponible que dans R. La librairie NETWORKX semble quant à elle pouvoir réaliser l’opération voulue, cependant la documentation ne couvre que le mode vectoriel, la fonction de surface de coût ne possède qu’une série limitée d’options et l’utilisation de leur fonction n’est pas des plus aisées.

Pour l’ensemble de ces raisons, l’algorithme de création de surface de coût de la librairie SCIKIT-IMAGE a été utilisé ici. Il s’agit plus précisément d’une méthode nommée *find_costs*. SCIKIT-IMAGE est une librairie *open source* spécialisée en traitements d’images et développée principalement en Python. Elle est également développée en Cython pour certains algorithmes tels que l’algorithme de surface de coût utilisé dans le cadre de ce travail. Cython est un langage de programmation qui mêle les performances offertes par le C (langage compilé) et la facilité de compréhension et d’écriture du langage Python (langage interprété de haut niveau). Il permet d’écrire des extensions en C pour Python, car il est traduit en langage C/C++ et compilé en tant que module d’extension Python (Behnel *et al.*, 2019a). Il permet également de déclarer des variables et des paramètres associés aux types de données du C (Behnel *et al.*, 2019b). Enfin, du Python peut être directement traduit en C/C++ via Cython dans la majorité des cas. Notons enfin que l’extension propre au fichiers Cython est « .pyx » et que c’est ce format de fichier qui est compilé par Cython afin d’obtenir un fichier à l’extension « .c ».

5.2 Transposition

L’algorithme à utiliser est désormais connu, il est accessible et utilisable via la librairie SCIKIT-IMAGE mais n’est cependant pas modifiable. La démarche pour disposer de l’algorithme de surface de coût en « local » avec possibilité de le modifier est expliquée ci-dessous.

La possibilité d’écrire du code Cython dans un fichier au format « .pyx » puis de le compiler afin d’être accessible sous forme de module d’extension Python a été mentionnée au 5.1. L’idée est donc d’importer la méthode *find_costs* dans un fichier « .pyx » afin de le compiler, importation qui ne peut être réalisée sans également prendre en compte le module auquel appartient cette méthode. L’ensemble du dossier correspondant à ce module est disponible

sur le service web d'hébergement Github³ et a été repris dans le cadre de ce mémoire. Ce dossier comprend notamment un fichier « setup.py » (cf. Figure 18) responsable de la compilation de tous les modules d'extensions compris dans le dossier. Dans ce cas-ci, trois fichiers Cython, à savoir « _spath.pyx », « _mcp.pyx » et « heap.pyx », sont convertis au format « .c », et correspondent à l'ensemble du module contenant la méthode *find_costs*.

```

from skimage._build import cython
import os.path

base_path = os.path.abspath(os.path.dirname(__file__))

def configuration(parent_package='', top_path=None):
    from numpy.distutils.misc_util import Configuration, get_numpy_include_dirs

    config = Configuration('graph', parent_package, top_path)
    config.add_data_dir('tests')

    # This function tries to create C files from the given .pyx files. If
    # it fails, try to build with pre-generated .c files.
    cython(['_spath.pyx',
            '_mcp.pyx',
            'heap.pyx'], working_path=base_path)

    config.add_extension('_spath', sources=['_spath.c'],
                        include_dirs=[get_numpy_include_dirs()])
    config.add_extension('_mcp', sources=['_mcp.c'],
                        include_dirs=[get_numpy_include_dirs()])
    config.add_extension('heap', sources=['heap.c'],
                        include_dirs=[get_numpy_include_dirs()])

    return config

if __name__ == '__main__':
    from numpy.distutils.core import setup
    setup(maintainer='scikit-image Developers',
          maintainer_email='scikit-image@python.org',
          description='Graph-based Image-processing Algorithms',
          url='https://github.com/scikit-image/scikit-image',
          license='Modified BSD',
          ** (configuration(top_path='').todict())
    )

```

Figure 18 : code source du fichier « setup.py » responsable de la compilation des modules d'extensions

La compilation du code Cython de ces trois fichiers .pyx peut alors être réalisée au moyen de la ligne de commande suivante : `py - 3.7 - m setup.py build - ext - -inplace`. Cette commande va, dans un premier temps, compiler le fichier Cython en créant un fichier C/C++, et ensuite compiler ce dernier fichier .c en un module d'extension directement importable depuis Python (Behnel *et al.*, 2019c). Notons par ailleurs que cet import se réalise comme un import classique en Python.

L'algorithme peut alors être appelé en « local ». En pratique, la librairie SCIKIT-IMAGE est tout de même utilisée afin de construire un graphe par instantiation de la classe 'MCP_Geometric' (cf. Figure 19) à laquelle on fournit une matrice *data*. La méthode *find_costs* est alors appliquée à ce graphe, moyennant des paramètres expliqués ci-dessous.

³ <https://github.com/scikit-image/scikit-image/tree/master/skimage/graph>


```

mcp = graph.MCP_Geometric(data)
cost_surface, traceback = mcp.find_costs([[y, x]], ends=None,
                                        find_all_ends=False,
                                        max_coverage=1,
                                        max_cumulative_cost=None,
                                        max_cost=None)

```

Figure 19 : application de l'algorithme de surface de coût à l'objet de la classe 'MCP_Geometric' instancié avec une matrice *data*

5.3 Paramétrage

L'algorithme de surface de coût contient divers paramètres (cf. Figure 19) :

- liste des pixels-source : pixels depuis lesquels la surface de coût est construite, qui vont donc présenter des coûts nuls. Dans notre cas, il s'agit d'un pixel unique par surface de coût qui correspond au pixel routier associé au point du réseau routier vectoriel le plus proche du site de crime (qui est a priori hors réseau) ;
- liste de pixels de fin : l'algorithme s'arrête lorsque tous les pixels spécifiés dans cette liste ont été traités par l'algorithme. Ici, donner une liste de pixels n'a aucun sens étant donné que l'objectif est de propager le coût sur l'ensemble du réseau routier rasterisé afin de pouvoir, par après, effectuer les calculs pour délimiter la zone d'ancrage. Autrement dit, il est impossible de savoir à l'avance à quel endroit du réseau routier rasterisé s'arrêter ;
- coût minimum en chaque pixel de fin : option qui permet de ne trouver le coût minimum qu'en des points spécifiés dans la liste des pixels de fin. Une fois qu'un coût est associé à ce/ces pixel(s) de fin, l'algorithme s'arrête. Au contraire, si cette option n'est pas considérée ou qu'aucun pixel de fin n'est spécifié, l'algorithme calcule le coût en chaque pixel de fin, i.e. en chaque pixel du réseau routier comme tel est le cas dans ce travail ;
- couverture maximum : option permettant d'allouer un coût à une proportion donnée des pixels de l'image (*raster*). Dans ce cas-ci, on demande à ce que tous les pixels routiers, qui constituent déjà une proportion fortement minoritaire des pixels de l'image, soient alloués. Par conséquent, cette option présente ici la valeur 1 (100 % de l'image couverte), d'autant plus qu'en fonction de la densité du réseau routier et de la résolution spatiale, cette proportion peut varier de manière non négligeable, ce qui rend cette option particulièrement compliquée à paramétrer pour un usage généralisé.

5.4 Validation sur le jeu de données

La validation de l'algorithme de surface de coût est dans un premier temps réalisée en vérifiant si le pixel où l'algorithme débute est bien le pixel-source défini, et présente bel et bien un coût nul. Elle est ensuite réalisée par comparaison avec la surface de friction. En effet, les incréments de coût entre deux pixels doivent correspondre au coût total cumulé depuis la source jusqu'à l'un d'eux, augmenté de la moyenne des frictions de ces deux pixels (moyenne multipliée par la racine de 2 dans le cas diagonal, cf. 2.2). Une comparaison avec le logiciel R a également été réalisée (fonction `accCost`) et le résultat final obtenu est identique.

6. Implémentation du processus

Le processus peut être implémenté, maintenant que le choix de l'algorithme est réalisé. Son implémentation s'opère dans un premier temps hors de l'extension logicielle, en raison de la complexité supplémentaire inutile apportée par la création de l'extension. De cette manière, le processus est validé avant d'être transposé dans l'extension logicielle, ce qui amène une plus grande simplicité. A noter que le processus développé hors de l'extension n'est pas optimal tant que sa validation n'est pas réalisée. Il sera par la suite amélioré dans l'extension logicielle.

Pour plus de lisibilité, le processus implémenté est d'abord expliqué de manière générale dans un premier sous-chapitre (cf. 6.1), il sera ensuite expliqué plus en détails de manière technique dans un second sous-chapitre (cf. 6.2)

6.1 Processus général

La première étape du processus est le chargement des données vectorielles du réseau routier, que l'on va par la suite rasteriser. Pour ne pas alourdir le processus de rasterisation, un échantillon de ce réseau a été sélectionné, repris à la figure 20. Cette rasterisation est réalisée en fonction d'un attribut correspondant à l'identifiant des segments routiers, et ce à une résolution de 10 m. Étant donné que l'identifiant présente une valeur différente pour chaque tronçon du réseau routier, il est de ce fait possible de les différencier. Cette différenciation sera utilisée afin de créer la surface de friction améliorée (cf. 4).

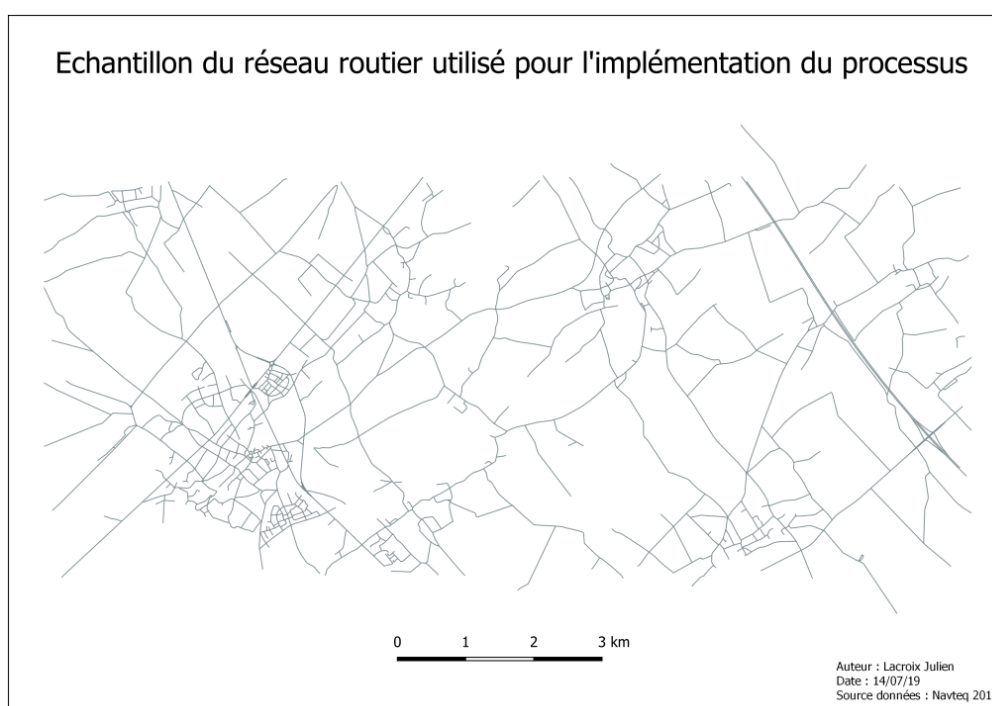


Figure 20 : échantillon du réseau routier vectoriel utilisé pour l'implémentation du processus

Pour construire cette surface de friction améliorée, on démarre du *raster* créé à l'étape précédente, dont les pixels du réseau routier possèdent la valeur de l'identifiant du tronçon routier auquel ils appartiennent. Cette construction nécessite alors deux éléments : le nombre

de pixels constituant chacun de ces tronçons routiers, et le coût temporel total associé à leur traversée. On peut, de la sorte, normaliser le coût temporel du tronçon par le nombre de pixels qui le constituent. Ces deux informations sont récupérées via deux sources différentes : le coût temporel est un attribut des données vectorielles tandis que le nombre de pixels peut être récupéré via le *raster* créé précédemment. Chacune de ces informations est reliée à l'identifiant, ce qui permet de les associer à la manière d'une jointure d'attributs. Notons que le coût de traversée des différents tronçons a été obtenu en divisant la longueur du tronçon par la vitesse autorisée sur celui-ci comme expliqué au 4.2. Ce calcul ayant déjà été réalisé sur QGIS, l'attribut a simplement été repris mais il est évident que ce calcul aurait pu être intégré au processus grâce aux informations de longueur du tronçon et de vitesse autorisée sur ce dernier. Encore une fois, ce qui importe ici est la validation du processus, et non son optimisation qui sera faite lors de la construction de l'extension logicielle.

Un échantillon du résultat obtenu concernant cette surface de friction améliorée est présenté à la figure 21. Il apparaît clairement que les tronçons présentant les valeurs les plus faibles de friction sont l'autoroute et les segments routiers reliant les villages, où la vitesse autorisée est la plus élevée. À l'opposé, les rues de ces villages présentent quant à elles des valeurs plus fortes de friction liées à une vitesse restreinte en agglomération.

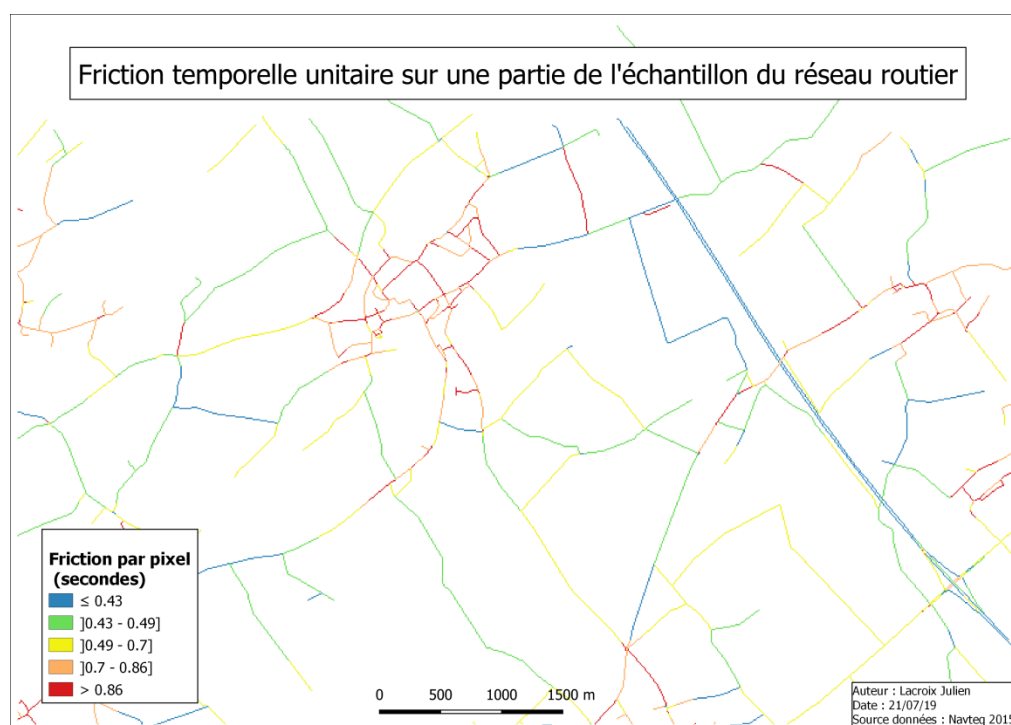


Figure 21 : friction temporelle unitaire (par pixel) sur un échantillon du réseau routier

Les données spatio-temporelles liées aux crimes peuvent maintenant être prises en compte. Il y a trois données pour chaque crime : les deux coordonnées du site de crime et l'heure à laquelle il a été commis. Pour rappel, les données spatio-temporelles utilisées pour ce test ont été définies arbitrairement (cf. 3.3). Comme le résume la figure 22 reprenant l'échantillon du réseau routier utilisé pour la construction du processus, un point d'ancrage théorique ainsi

que cinq sites de crimes qui l'entourent ont été définis. Ces sites de crimes ont été disposés à proximité directe du réseau routier.

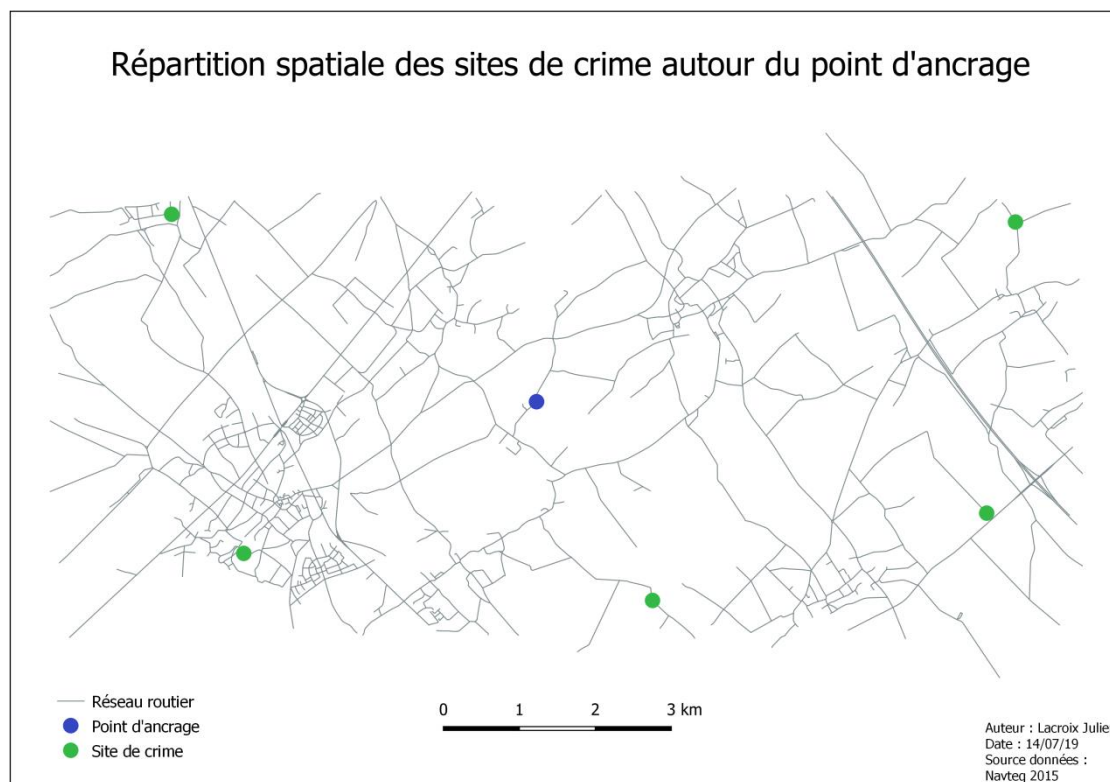


Figure 22 : répartition des sites de crimes autour du point d'ancrage

Les valeurs du coût temporel depuis le point d'ancrage pour atteindre ces sites de crimes sont connues et reprises au tableau 1. Pour les obtenir, l'algorithme de surface de coût a été mis en œuvre depuis le point d'ancrage théorique. Ainsi, il est possible de choisir des coordonnées relatives à un pixel et d'y consulter le coût temporel associé. Celui-ci est d'abord exprimé en secondes dans la surface de coût, puis converti en minutes pour insérer les informations sous format HH:MM:SS. A noter que l'identifiant 0 correspond au point d'ancrage (coût temporel nul).

ID	X	Y	Coût (s)	Coût (min)
0	176630	140080	0	0
1	171544.5	142666.8	468.68	7m48s
2	172494.4	138203.6	358.43	5m58s
3	177875.9	137584.3	225.92	3m45s
4	182654.4	142564.7	565.46	9m25s
5	182275.3	138733.7	514.81	8m34s

Tableau 1 : coordonnées cartographiques (Lambert 72) et coûts temporels du point d'ancrage et des sites de crimes

Les coordonnées cartographiques (Lambert 72) des points correspondant aux sites de crimes étant connues, il est possible de les rattacher aux pixels respectifs du réseau routier les plus proches de ceux-ci (cf. 3.1.1). Pour ce faire, deux fonctions principales ont été créées.

La première permet, au départ de coordonnées cartographiques correspondant à un site de crime, de retourner les coordonnées cartographiques du point du réseau routier qui minimise la distance euclidienne par rapport à celui-ci. L'objectif de la seconde fonction est, quant à elle, de transformer des coordonnées cartographiques en des coordonnées-image. Elle s'applique donc ici aux coordonnées du point du réseau routier obtenues grâce à la première fonction. Un problème peut toutefois subsister si le pixel associé au point du réseau routier intersecté ne fait pas partie de la rasterisation de ce réseau routier. Dans ce cas, la fonction recherche parmi les pixels voisins de ce pixel, celui qui appartient au réseau routier et minimise la distance euclidienne au point obtenu via la première fonction, afin de départager plusieurs candidats éventuels. Enfin, trois fonctions secondaires ont été créées pour les calculs respectivement d'une projection orthogonale, d'une distance euclidienne et du passage de coordonnées – image en coordonnées cartographiques.

Les coordonnées-image du pixel du réseau routier associé à un site de crime, depuis lesquelles la surface de coût doit être construite, peuvent désormais être connues. Les n surfaces de coût correspondant aux n crimes peuvent donc être engendrées. Celles-ci démarrent d'un coût nul au point de départ, coût qui s'accroît au fur et à mesure que l'on s'éloigne de celui-ci (coût temporel progressif). La surface est dès lors modifiée afin de rendre ce coût temporel régressif depuis l'heure à laquelle le crime s'est déroulé. Elle est modifiée une seconde fois pour ramener l'intervalle de valeurs pouvant être prises par un pixel dans l'intervalle $[0; 86400[$, c'est-à-dire entre minuit et 23:59:59. Au final, chaque pixel du réseau routier d'une surface de coût aura comme valeur l'heure de passage (en secondes) telle qu'en partant de ce pixel à cette heure, le pixel correspondant au site de crime serait atteint à l'heure à laquelle le crime s'est déroulé. Chaque surface de coût comportant une heure de passage, il y aura par conséquent n heures de passage en chaque pixel.

A partir de ces surfaces de coût, deux méthodes ont été mises en place et testées afin de délimiter la zone d'ancrage. La première est la méthode énoncée dans le cadre du profilage géographique, à savoir la prise en compte de l'étendue des heures de passage des pixels homologues des surfaces de coût. L'étendue d'un ensemble de données est la différence entre les valeurs maximale et minimale (Triola *et al*, 2012). Dans le cas qui nous intéresse, il s'agit de la différence entre l'heure de passage la plus tardive et l'heure de passage au plus tôt en chaque pixel. Pour ce faire, deux *rasters* sont construits à partir des surfaces de coût, l'un « au plus tôt » reprenant l'heure minimale de passage en chaque pixel tandis que l'autre « au plus tard » traduit l'heure maximale de passage en ces mêmes pixels. Ces *rasters* sont évalués à chaque nouvelle construction d'une surface de coût, et mis à jour si au moins un pixel de cette nouvelle surface de coût présente une heure inférieure (respectivement supérieure pour le *raster* « au plus tard ») à celle de référence obtenue jusqu'alors. Lorsque toutes les surfaces de coût ont été comparées aux *rasters* « au plus tôt » et « au plus tard », un troisième *raster* est créé et reprend la différence entre ceux-ci (i.e. « au plus tard » – « au plus tôt »). Idéalement, la différence entre les heures de passages du *raster* « au plus tard » et celles du

raster « au plus tôt » doit être proche de 0 aux alentours du point d'ancrage. Elle doit ensuite théoriquement augmenter au fur et à mesure que l'on s'en éloigne.

La seconde méthode est également une mesure de la dispersion d'un ensemble de données, il s'agit de la variance. Celle-ci, notée σ^2 pour une population, désigne la somme des carrés des écarts à la moyenne, divisée par n ($n - 1$ dans le cas d'un échantillon), et est donnée par la formule suivante (Triola *et al*, 2012) :

$$\sigma^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}$$

Cette méthode est proposée dans le cadre de ce travail, afin de déterminer si elle peut améliorer la délimitation de la zone d'ancrage par rapport à la méthode de l'étendue. Appliquer ce calcul revient ici à calculer la variance des heures de passage des pixels homologues des n surfaces de coût, une fois qu'elles ont toutes été créées. Ces valeurs de variances sont toutes enregistrées dans un *raster*. Comme dans le cas de l'étendue, le point d'ancrage le plus vraisemblable est le pixel qui présente une valeur minimale, autrement dit celui qui minimise la variance des heures de passage.

Pour résumer ce sous-chapitre et rendre les explications du processus plus visuelles, un *workflow* a été créé et est disponible à l'annexe 2.

6.2 Implémentation pratique

6.2.1 Rasterisation

Le processus de rasterisation, repris à la figure 23 ci-dessous, commence par la définition du fichier *raster* via la méthode *Create* de la librairie GDAL. Quelques éléments sont précisés afin de définir ce fichier, tels que son nom, les nombres de colonnes et de lignes, le nombre de bandes et le type de données GDAL. La variable '*target_ds*' permet alors d'accéder à ce raster. On applique ensuite une transformation affine au *raster* de manière à passer des coordonnées-pixel à des coordonnées cartographiques, ce qui nécessite divers paramètres tels que la coordonnée minimale en abscisse ('*x_min*') et la coordonnée maximale en ordonnée ('*y_max*') du fichier vectoriel du réseau routier, la résolution de rasterisation ('*resolution*') tandis que les valeurs de rotation n'ont pas lieu d'être (image orientée au nord) et sont fixées à 0. La projection du *raster* est enfin définie suivant la projection du fichier vectoriel du réseau routier utilisé en *input* ('*source_srs*'), i.e. le Lambert Belge 1972.

```
target_ds = driver_gdal.Create(name_raster, col, row, 1, gdal.GDT_Float32)
target_ds.SetGeoTransform((x_min, resolution, 0, y_max, 0, -resolution))
target_ds.SetProjection(source_srs.ExportToWkt())

band = target_ds.GetRasterBand(1)
band.SetNoDataValue(NoData_value)

gdal.RasterizeLayer(target_ds, [1], source_layer, None,
                   options=["ATTRIBUTE=%s" % 'OBJECTID'])
```

Figure 23 : rasterisation du réseau routier vectoriel

L'unique bande de ce fichier .tif est alors chargée (variable *'band'*) et tous les pixels qui la composent sont définis par défaut à une valeur *No Data* (*'NoData_value'*) égale à -99, qui est une valeur arbitraire. Toute autre valeur négative aurait pu être choisie, le principal est de ne pas définir une valeur positive ou nulle car cet intervalle de valeurs est réservé au résultat de la rasterisation. Enfin, la rasterisation est réalisée en fonction de l'attribut *'OBJECTID'* du réseau routier vectoriel, qui représente l'identifiant des différents tronçons routiers. D'autres paramètres sont repris dans cette rasterisation, tels que le nom du fichier *raster* qui contiendra le résultat (*'target_ds'*), sa bande (1, seule et unique bande), la couche vectorielle du réseau routier (*'source_layer'*) et la valeur de rasterisation par défaut qui n'est pas utile dans ce cas (*'None'*) étant donné qu'on utilise l'attribut *'OBJECTID'*.

6.2.2 Création de la surface de friction améliorée

Le *raster* obtenu à la suite de la rasterisation est composé d'un ensemble de pixels présentant la valeur *No Data* -99 s'ils ne sont pas associés au réseau routier, et la valeur de l'identifiant du tronçon dans le cas contraire (i.e. valeur de l'attribut *'OBJECTID'*). Comme expliqué au 6.1, l'amélioration de la surface de friction nécessite notamment la connaissance du nombre de pixels par tronçon routier. La fonction **unique** de la librairie NUMPY, librairie Python spécialisée dans la manipulation de matrices, permet précisément d'obtenir ce résultat. Comme le montre la figure 24, cette fonction permet l'obtention de toutes les valeurs distinctes présentes dans une matrice, ainsi que leur nombre d'occurrences. Un *raster* étant une matrice de nombres qui représentent les valeurs des pixels (variable *'datas'*, cf. figure 25), il est possible d'appliquer cette fonction afin de récupérer toutes les valeurs uniques de cette matrice (i.e. les identifiants des segments routiers, variable *'unique_elements'*) et l'occurrence à laquelle chaque identifiant apparaît (i.e. le nombre de pixels, variable *'count_elements'*).

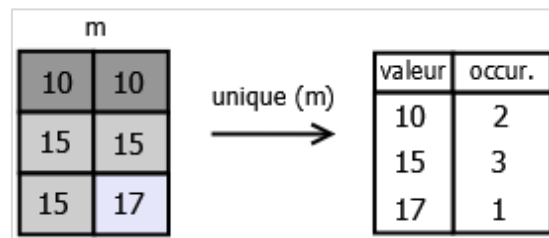


Figure 24 : exemple de fonctionnement de la fonction unique de NUMPY pour une matrice 3x2

La seconde information indispensable à l'amélioration de la surface de friction est le coût temporel du parcours des tronçons routiers. Pour ce faire, chaque élément (*'feature'*, cf. Figure 25) composant la couche du fichier vectoriel du réseau routier (*'source_layer'*), autrement dit les tronçons de voiries, est parcouru de manière à récupérer l'identifiant (attribut *'OBJECTID'*) et le coût temporel associé (attribut *'cost'*) dans une liste *'tuple_ID_cost'*, vide à l'initialisation. Elle contiendra une liste de tuples identifiant – coût.

Ces deux informations obtenues, chacune liée à l'identifiant des tronçons routiers, sont jointes grâce à ce dernier via deux boucles imbriquées. L'idée est de parcourir la liste des identifiants uniques (*'unique_elements'*), et pour chacun de ces identifiants parcourir les tuples identifiant – coût *'tuple_ID_cost'*. Lorsque les identifiants coïncident, le coût temporel

du tronçon, noté '*cost_by_pixel*', est normalisé en le divisant par le nombre de pixels qui le composent. Tous les pixels de la matrice '*datas*' qui présentent l'identifiant courant sont ensuite modifiés et leur valeur est désormais le coût temporel unitaire calculé. Enfin, une fois que tous les identifiants ont été traités, on enregistre les modifications via la méthode *WriteArray* appliquée à la bande '*band*'.

```

datas = target_ds.GetRasterBand(1).ReadAsArray()
unique_elements, count_elements = np.unique(datas, return_counts=True)

tuple_ID_cost = []
for feature in source_layer:
    tuple_ID_cost.append([int(feature.GetField("OBJECTID")),
                        feature.GetField("cost")])

for i in range(1, len(unique_elements), 1):
    for d in tuple_ID_cost:
        if unique_elements[i] == d[0]:
            cost_by_pixel = d[1]/count_elements[i]
            datas[datas == d[0]] = cost_by_pixel

band.WriteArray(datas, 0, 0)

```

Figure 25 : amélioration de la surface de friction

6.2.3 Rattachement des données spatio-temporelles au réseau routier

Comme expliqué précédemment (cf. 6.1), deux fonctions principales et trois fonctions secondaires ont été créées afin d'implémenter ce rattachement des données des sites de crimes au réseau routier.

La première fonction principale, **closest_point** (cf. Annexe 3), recherche le point du réseau routier le plus proche spatialement du site de crime. Elle débute par la création d'un point via la librairie OGR (sous-librairie de GDAL permettant de manipuler des données vectorielles), auquel on associe les coordonnées du site de crime ('lon', 'lat'). Deux variables sont ensuite définies : '*buffer_radius*' et '*intersection*'. La première correspond au rayon d'un espace-tampon (ou *buffer*) créé autour du point associé au site de crime. La raison de la création de ce *buffer* est de limiter le nombre de tronçons routiers testés et ne garder que ceux qui sont les plus proches spatialement du site de crime. La valeur du rayon de ce *buffer* est initialisée arbitrairement à 100 m, et l'intersection entre ce *buffer* et tout segment routier vectoriel est alors testée au moyen de la méthode *Intersects* de la librairie GDAL. Dans le cas où aucun élément du réseau routier ne se trouve à moins de 100 m du site de crime, autrement dit si l'intersection est nulle, le rayon du *buffer* est alors augmenté de 100 m. Ce processus est répété jusqu'à ce qu'au moins un tronçon routier soit intersecté par le *buffer*. Dans ce cas, la variable booléenne '*intersection*', initialisée à *False* est redéfinie à *True* afin de sortir de la boucle lorsque tous les tronçons routiers ont été passés en revue. Toujours dans cette boucle, si l'intersection entre le tronçon routier et le *buffer* existe, la distance euclidienne minimale entre le site de crime et ce tronçon est évaluée grâce à la méthode *Distance* (librairie GDAL). À chaque itération (c'est-à-dire à chaque tronçon routier intersecté par le *buffer*), si une

distance inférieure à la distance jusqu'alors la moins importante est obtenue, on remplace cette dernière. Cette distance de référence intitulée '*distance*' est initialisée à la valeur du *buffer* incrémenté d'une unité pour obligatoirement trouver une distance inférieure. Un problème reste toutefois à résoudre : celui de déterminer les coordonnées exactes de ce point d'intersection du tronçon routier minimisant la distance euclidienne depuis le site de crime. La librairie GDAL ne semblant pas comporter cette fonction, une fonction **proj_to_segment** (cf. Annexe 4) a été créée afin de déterminer celui-ci. Cette fonction est appelée par **closest_point** lorsque le tronçon routier le plus proche a été obtenu. **proj_to_segment** fonctionne de la manière suivante : soient $A(x_A, y_A)$ et $B(x_B, y_B)$ deux points constituant un segment de la polyligne correspondant au tronçon routier et soit $P(x_P, y_P)$ le point associé au site de crime. Comme le montre la figure 26, on considère les vecteurs \vec{v} et \vec{u} formés respectivement par les points A et B d'une part et d'autre part par les points A et P. Ces vecteurs forment entre eux un angle θ . La projection orthogonale du vecteur \vec{u} sur le vecteur \vec{v} , notée \vec{u}' , est alors donnée par (Bastin, 2014) :

$$\vec{u}' = \frac{\|\vec{u}\| \|\vec{v}\| \cos \theta}{\|\vec{v}\|^2} \vec{v} = \frac{\vec{u} \cdot \vec{v}}{\|\vec{v}\|^2} \vec{v}$$

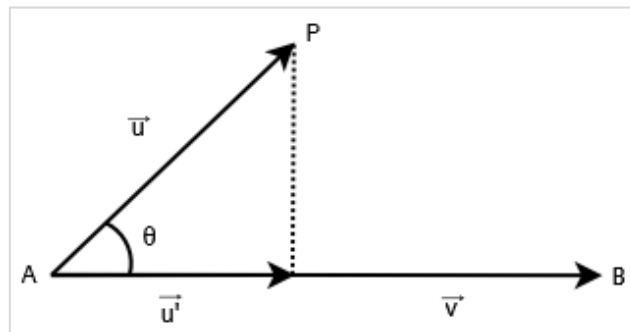


Figure 26 : projection orthogonale du vecteur \vec{u} sur le vecteur \vec{v}

Cette projection minimise la distance du point P au segment $|AB|$ et fournit le ratio du vecteur \vec{v} où le vecteur \vec{u} (et donc P) est projeté. Si elle vaut 0, cela signifie que la projection est sur le point A ; elle vaudra 1 si cette la projection du point P correspond au point B. Pour toute valeur entre 0 et 1, la position du point projeté sur le vecteur \vec{v} (i.e. le ratio) est connue, par conséquent on obtient les coordonnées d'intersection $(x_{intersect}, y_{intersect})$ de la manière suivante :

$$x_{intersect} = X_A + \vec{u}' * (X_A - X_B)$$

$$y_{intersect} = Y_A + \vec{u}' * (Y_A - Y_B)$$

Deux cas restent à définir : celui où la valeur de la projection orthogonale est négative et celui où cette même valeur est strictement supérieure à 1. Dans le premier cas, cela signifie que l'angle θ entre les deux vecteurs est supérieur à $\pi/2$, et donc que la projection orthogonale se situe dans le prolongement du segment à gauche de A. Une projection supérieure à 1 correspond quant à elle à une projection orthogonale dans le prolongement du segment à droite de B. Dans chacun de ces cas, le point le plus proche du segment est son extrémité,

autrement dit le point A pour une projection orthogonale négative et le point B pour une valeur strictement supérieure à 1. En pratique, la fonction **proj_to_segment** est appelée autant de fois que le nombre de segments composant la polyligne. La projection orthogonale est alors calculée et en fonction de la valeur, le point d'intersection est déterminé. Pour chaque point obtenu (i.e. pour chaque segment), la distance euclidienne est calculée pour ne garder au final que le point minimisant celle-ci. Si elle est inférieure à celle de référence, notée '*nearest_point_distance*' et initialisée arbitrairement à 1000, on enregistre les coordonnées et la distance du nouveau point d'intersection le plus proche. Enfin, une fois que tous les segments du tronçon routier ont été testés, on appelle la fonction **coord_to_pixel** (cf. Annexe 5) en lui passant en argument les coordonnées cartographiques de ce point d'intersection.

Cette seconde fonction, **coord_to_pixel**, sert à transformer des coordonnées cartographiques en coordonnées-image. Elle débute par la récupération d'informations propres au *raster* de la surface de friction obtenu au 6.2.2 : les coordonnées du point supérieur gauche ('*upper_left_x*', '*upper_left_y*'), la résolution en x et en y ('*x_size*' et '*y_size*' respectivement) qui est par ailleurs identique, et les rotations qui sont ici nulles. A l'aide de ces informations, il est possible de passer aux coordonnées-image (*px* et *py*) en soustrayant les coordonnées du coin supérieur gauche aux coordonnées passées en arguments (coordonnées de la projection du site de crime sur le réseau routier vectoriel), le tout divisé par la résolution spatiale. Un problème peut toutefois subsister comme le montre la figure 27, si le point du réseau routier intersecté ne fait pas partie de la rasterisation du réseau routier, et donc présente la valeur *No Data* -99.

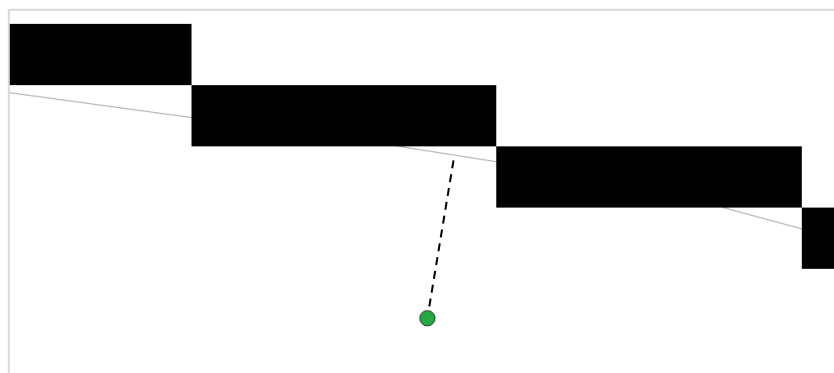


Figure 27 : cas de figure où le point du réseau routier vectoriel intersecté ne correspond pas à un pixel du réseau routier rasterisé

Dans ce cas, la fonction recherche parmi les pixels voisins celui qui appartient au réseau routier rasterisé, et qui minimise la distance euclidienne au point intersecté afin de départager plusieurs éventuels candidats. Par exemple, le pixel qui sera sélectionné à la figure 27 sera celui au nord du pixel courant présentant la valeur *No Data*. Ce calcul de distance euclidienne nécessite de passer des coordonnées – image aux coordonnées cartographiques, transformation inverse réalisée grâce à la fonction **pixel_to_coord** (cf. Annexe 6).

6.2.4 Création des surfaces de coût

La création de chaque surface de coût (cf. figure 28) débute par la création d'un nouveau *raster*, en copiant celui de la surface de friction 'ds' à l'aide de la méthode *CreateCopy* de la librairie GDAL. La fonction *closest_point* est ensuite appelée en passant en argument les coordonnées cartographiques du site de crime, et renvoie les coordonnées-image 'y' (rangée) et 'x' (colonne) associées à sa projection sur le réseau routier. Par après, un graphe SKIMAGE (sous-librairie de la librairie SCIKIT – IMAGE) est construit en passant en argument la matrice 'data' correspondant au *raster* de la surface de friction améliorée. La méthode *find_costs* peut alors être appliquée au graphe tout juste créé, en spécifiant les coordonnées-image d'où l'algorithme doit débiter. Les autres arguments de cette méthode ont, quant à eux, été expliqués au 5.3. Comme mentionné au 6.1, le résultat de cette méthode est l'obtention d'une surface de coût 'cost_surface', dont le coût temporel est progressif depuis la source. Pour le rendre régressif depuis l'heure d'occurrence du crime, on soustrait chaque valeur valable de la matrice (i.e. autre que *No Data*) à cette heure d'occurrence du crime ramenée en secondes ('event_time_seconds'). Enfin, la valeur d'une journée (en secondes) est ajoutée à toute valeur négative afin d'obtenir uniquement des valeurs dans l'intervalle d'une journée [0;86400[. Le *raster* est alors mis à jour avec la surface de coût créée et modifiée.

```

name = 'source_%d.tif' % number
target_ds2 = driver_gdal.CreateCopy(name, ds)

y, x = closest_point(float(latlon[0]), float(latlon[1]))

mcp = graph.MCP_Geometric(data)
cost_surface, traceback = mcp.find_costs([[y, x]], ends=None,
                                       find_all_ends=False,
                                       max_coverage=1,
                                       max_cumulative_cost=None,
                                       max_cost=None)

cost_surface = event_time_seconds - cost_surface
cost_surface[cost_surface < 0] += 86400

bandbis = target_ds2.GetRasterBand(1)
bandbis.WriteArray(cost_surface, 0, 0)

```

Figure 28 : création et enregistrement d'une surface de coût

6.2.5 Délimitation de la zone d'ancrage

a) Étendue des heures de passage

Cette méthode utilise deux *rasters*, qu'elle met à jour à chaque création de surface de coût (« au plus tôt » et « au plus tard »). Ces deux *rasters* sont d'abord créés et initialisés avant la boucle de création de surface de coût, c'est-à-dire avant le 6.2.4. Ils sont nommés respectivement 'sooner' pour le temps au plus tôt et 'later' pour le temps au plus tard (cf. figure 29). Dans les deux cas, les *rasters* sont créés de la même manière que les surfaces de

coût (cf. 6.2.4) en copiant le *raster* de la surface de friction. Ils sont par la suite lus en tant que matrices puis initialisés à des valeurs cohérentes au regard de leur fonction. En effet, la valeur d'un pixel du *raster* du temps au plus tôt '*sooner*' sera modifiée si une heure de passage plus tôt que cette valeur est trouvée. Par conséquent, tous les pixels doivent être initialisés à une valeur très grande afin qu'une valeur inférieure soit obligatoirement trouvée pour la remplacer. Pour ce faire, une nouvelle valeur de 99999 secondes est définie à toutes les valeurs supérieures à 0, autrement dit toutes les valeurs correspondant au réseau routier. Cette valeur de 99999 secondes est amplement suffisante étant donné que la valeur maximale possible de l'heure de passage est 23:59:59, soit 86399 secondes. Concernant le *raster* au plus tard '*later*', le raisonnement est le contraire : les valeurs du réseau routier doivent être initialisées à une valeur telle que toute valeur présente dans la matrice de la surface de coût sera supérieure. La valeur minimale possible étant 0 (00:00:00), toutes les valeurs du réseau routier ont été définies à -1.

```
target_sooner = driver_gdal.CreateCopy("sooner.tif", ds)
sooner = target_sooner.GetRasterBand(1).ReadAsArray()
sooner[sooner >= 0] = 99999

target_later = driver_gdal.CreateCopy("later.tif", ds)
later = target_later.GetRasterBand(1).ReadAsArray()
later[later >= 0] = -1
```

Figure 29 : création des *rasters* et initialisation des matrices des temps au plus tôt et au plus tard

Les deux *rasters* créés, les opérations de comparaisons entre ceux-ci et les surfaces de coût peuvent être implémentées. Ces opérations se déroulant à chaque itération, c'est-à-dire à chaque nouvelle création de surface de coût, elles sont placées dans la boucle après le 6.2.4. L'idée est la suivante : deux matrices vont être créées ('*sooner_comparison*' et '*later_comparison*') grâce aux fonctions **greater** et **less** de la librairie NUMPY dont un exemple de fonctionnement est repris à la figure 30. Chaque élément de ces matrices est un booléen qui vérifie une relation entre deux éléments homologues de deux matrices de même dimension. Dans le premier cas (fonction **greater**), la relation entre les deux matrices passées en argument est que la valeur de l'élément de la première matrice ('*sooner*') est strictement supérieure à celle de l'élément de la seconde ('*cost_surface*'). La valeur d'un élément de la matrice '*sooner_comparison*' sera donc égale à *True* si une heure de passage inférieure à celle de référence jusqu'alors a été trouvée. Dans le second cas au contraire (fonction **less**), la relation est que la valeur de l'élément de la première matrice ('*later*') est strictement inférieure à celle du même élément de la seconde ('*cost_surface*'). Dans ce cas-ci, on en déduit que la valeur d'un élément de la matrice '*later_comparison*' sera égale à *True* si une heure de passage supérieure à celle jusqu'alors de référence a été obtenue. Il ne reste plus alors qu'à parcourir les deux matrices, chacune via une double boucle – une parcourant les rangées ('*p*') et l'autre parcourant les colonnes ('*d*') – et affecter la nouvelle valeur obtenue lorsqu'un booléen *True* est trouvé (cf. Figure 31). Deux incréments ('*index_row*' et '*index_col*') sont définis pour garder l'indice de l'élément de la matrice afin d'effectuer la modification.

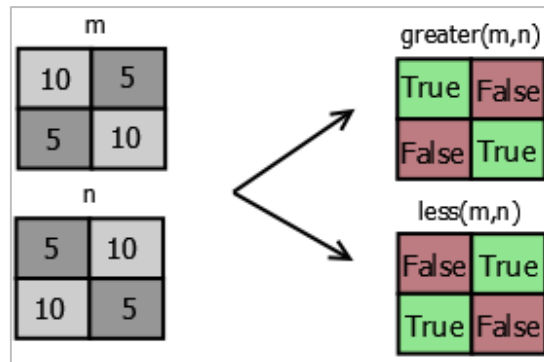


Figure 30 : exemple d'utilisation des fonctions greater et less sur deux matrices 2x2

```

sooner_comparison = np.greater(sooner,cost_surface)
index_row = 0
for d in sooner_comparison:
    index_col = 0
    for p in d:
        if p: # (if p == True)
            sooner[index_row,index_col] = cost_surface[index_row,index_col]
            index_col += 1
    index_row += 1

later_comparison = np.less(later,cost_surface)
index_row = 0
for d in later_comparison:
    index_col = 0
    for p in d:
        if p:
            later[index_row, index_col] = cost_surface[index_row, index_col]
            index_col += 1
    index_row += 1
    
```

Figure 31 : modification des rasters au plus tôt et au plus tard

Lorsque les n surfaces de coût ont été créées et les *rasters* 'sooner' et 'later' modifiés (ou du moins comparés) n fois, un troisième *raster* nommé 'time_difference' et destiné à contenir les différences de temps entre les deux *rasters* est finalement créé. Pour ce faire, une double boucle parcourt tous les éléments et effectue le calcul $later - sooner$. Une fois tous les éléments parcourus, les modifications sont enregistrées (cf. Figure 32).

```

target_difference = driver_gdal.CreateCopy("time_difference.tif", ds)
band_difference = target_difference.GetRasterBand(1)
difference = band_difference.ReadAsArray()

for index_row in range(0,row,1):
    for index_col in range(0,col,1):
        if difference[index_row,index_col] != -99:
            difference[index_row, index_col] = \
                later[index_row, index_col] - sooner[index_row,index_col]

band_difference.WriteArray(difference,0,0)
    
```

Figure 32 : création du raster de différence entre l'heure de passage maximale et minimale

b) Variance temporelle des heures de passage

Le calcul de cette variance de tous les pixels homologues nécessite de sauvegarder les informations propres à chaque surface de coût (donc à chaque itération). Cette sauvegarde (temporaire) est réalisée en créant une liste '*cost_surfaceS*', vide à l'origine, se remplissant d'une surface de coût à chaque itération grâce à la méthode *append*. Il s'agira donc d'une liste de matrices.

Une fois cette liste remplie par les *n* surfaces de coût, l'idée est de parcourir ces dernières élément par élément, autrement dit pixel par pixel, afin d'obtenir un calcul cohérent sur des éléments homologues. La fonction *zip* de la librairie NUMPY, dont un exemple de fonctionnement est présenté à la figure 33, permet d'effectuer ce traitement. Dans cet exemple, une liste '*rasters*' contient deux matrices bidimensionnelles '*m*' et '*n*' de même dimension (3x3). La fonction *zip* est appliquée une première fois à cette liste '*rasters*', avec le symbole « * » permettant d'extraire les deux matrices de la liste. Ainsi *zip(*rasters)* revient à effectuer *zip(m,n)*, et le résultat de cette fonction est trois listes '*arr1*', '*arr2*' et '*arr3*'. Chacune de ces listes contient deux matrices unidimensionnelles correspondant à deux rangées homologues des matrices '*m*' et '*n*'. Par exemple, la liste '*arr1*' contient la première rangée de la matrice '*m*' suivie de la première rangée de la matrice '*n*'. On applique ensuite une deuxième fois la fonction *zip*, cette fois sur les trois listes créées à l'étape précédente. Comme dans le premier cas, le symbole « * » est utilisé afin de mettre en argument les deux matrices unidimensionnelles. Notons que pour des matrices bidimensionnelles contenant *r* rangées, il faudra appliquer *r* fois cette fonction *zip*. Le résultat de cette fonction est l'obtention de listes '*tuple1*', '*tuple2*' et '*tuple3*' contenant les éléments homologues des deux matrices au niveau le plus précis, c'est-à-dire pixel par pixel.

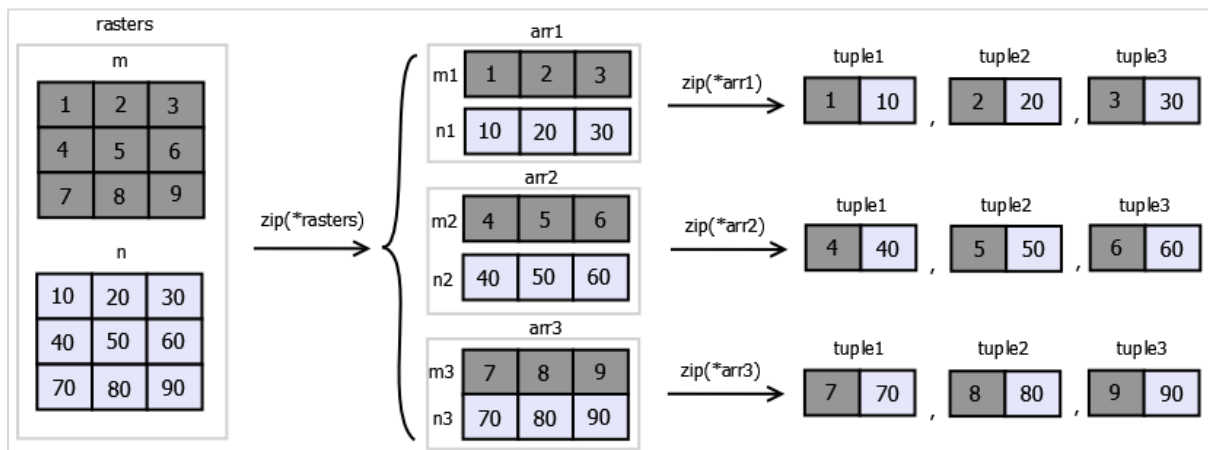


Figure 33 : exemple de fonctionnement de la fonction *zip* d'abord sur une liste de matrices bidimensionnelles ('*rasters*'), ensuite sur les résultats de la première fonction *zip* ('*arr1*', '*arr2*' et '*arr3*')

Transposons maintenant cet exemple de la fonction *zip* à nos surfaces de coûts (cf. Figure 34) : la liste '*cost_surfaceS*' contient les *n* surfaces de coût, qui sont des matrices bidimensionnelles au même titre que '*m*' et '*n*' dans l'exemple ci-dessus. En appliquant deux fois cette fonction *zip*, il est ainsi possible d'accéder à tous les pixels homologues des *n* surfaces de coût, sous forme de liste. De là, la variance peut être calculée pour chaque liste.

En pratique, on commence par définir une liste *'variances'*, vide à l'origine. Cette liste contiendra tous les résultats des calculs de variance. Une première fonction **zip** est ensuite appliquée à la liste *'cost_surfaceS'*, dont le résultat est parcouru par l'incrément *'arr'* permettant d'accéder aux listes successives des rangées homologues des surfaces de coût. La fonction **zip** est appelée une seconde fois pour chacune de ces listes, afin d'accéder à la liste des éléments homologues (i.e. pixels) *'element'* des surfaces de coût. Pour chacun de ces éléments *'element'*, la présence d'une valeur infiniment négative est évaluée de manière à savoir si l'élément évalué correspond à un élément hors du réseau routier. Si elle ne le contient pas, donc si l'élément appartient au réseau routier, la variance de la liste est calculée. Cette variance correspond pour rappel à la variance des heures de passage de chaque surface de coût en un pixel donné, et est calculée au moyen de la fonction NUMPY **var**. Elle est ensuite ajoutée à la liste *'variances'*. Dans le cas d'un élément hors du réseau routier, une valeur *No Data* est ajoutée à la liste pour garder l'ordre des éléments. Lorsque tous les éléments ont été parcourus, la liste *'variances'* est transformée en matrice puis redimensionnée en matrice bidimensionnelle grâce aux informations du nombre de rangées *'row'* et de colonnes *'col'* via la fonction **reshape** de NUMPY. Le fichier image « *variances.tif* » est enfin créé et enregistré avec les informations adéquates.

```

variances = []
for arr in zip(*cost_surfaceS):
    for element in zip(*arr):
        if float("--inf") not in element:
            var = np.var(element, dtype=np.float32)
            variances.append(float(var))
        else:
            variances.append(NoData_value)

variances = np.array(variances)
variances = np.reshape(variances, (row, col))

target_var = driver_gdal.CreateCopy("variances.tif", ds)

band_var = target_var.GetRasterBand(1)
band_var.SetNoDataValue(NoData_value)
band_var.WriteArray(variances, 0, 0)

```

Figure 34 : calcul des variances des heures de passage et création du *raster*

6.3 Validation

La validation de ces deux processus est dictée par trois éléments :

- a) la valeur du pixel minimisant l'étendue / la variance est proche de la valeur nulle, étant donné que les coûts temporels depuis la source des pixels associés aux sites de crime (cf. Tableau 1) sont quasiment exacts. Ils ne sont néanmoins pas complètement exacts car arrondis à la seconde près, ce qui explique pourquoi cette valeur ne peut pas être nulle ;
- b) ce pixel correspond au pixel associé au point d'ancrage théorique défini arbitrairement au 6.1 ;

- c) les valeurs des pixels voisins sont supérieures et augmentent au fur et à mesure que l'on s'éloigne de cette source.

Les figures 35 et 36 reprennent les résultats obtenus avec la prise en compte respectivement de l'étendue et la variance.

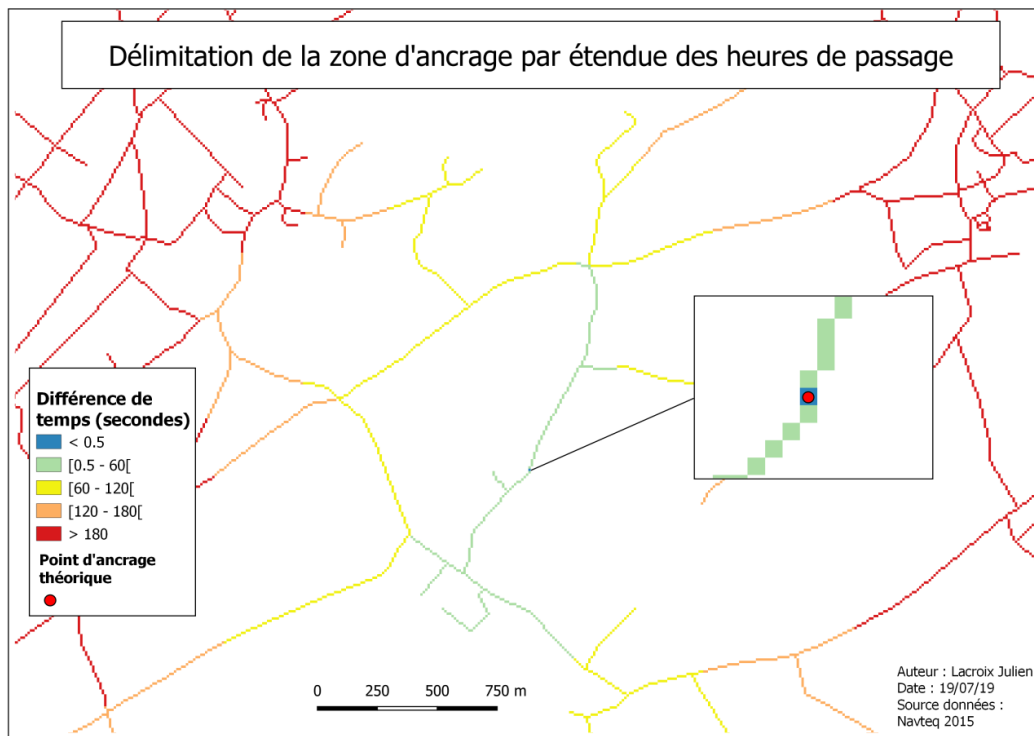


Figure 35 : délimitation de la zone d'ancrage avec prise en compte de l'étendue des heures de passage

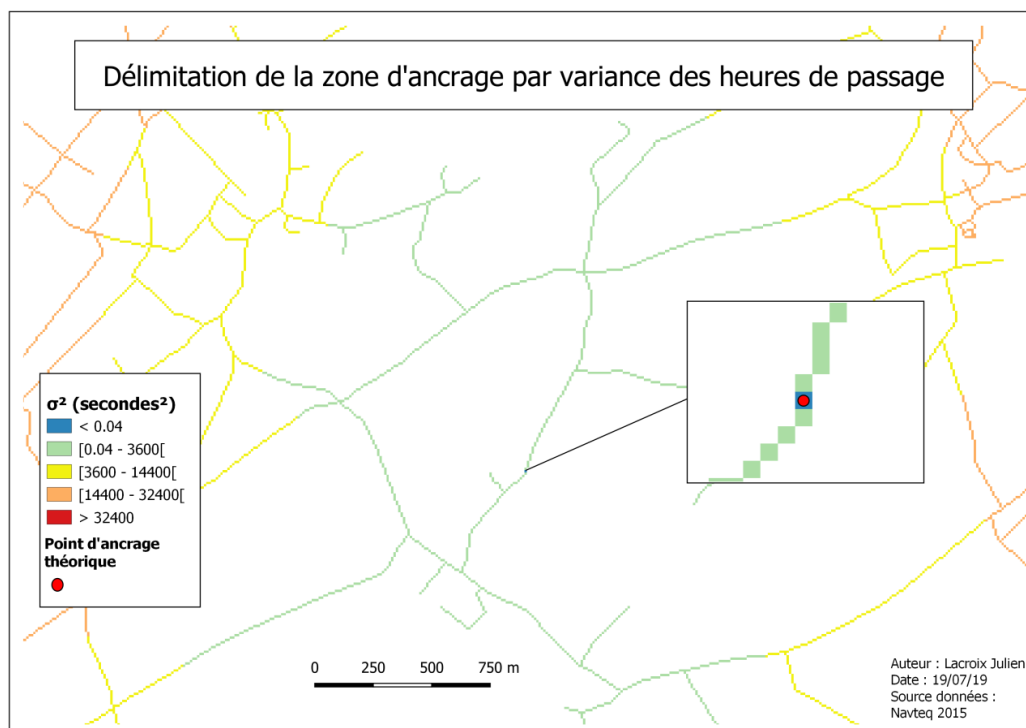


Figure 36 : délimitation de la zone d'ancrage avec prise en compte de la variance des heures de passage

Dans les deux cas, 5 seuils ont été définis, et le premier a pour but de délimiter uniquement le pixel présentant la valeur minimale (pixel associé au point d'ancrage). Dans le cas de l'étendue, les seuils sont ensuite définis minute par minute. Ainsi, le second seuil reprend tous les pixels dont la différence entre les heures de passage maximale et minimale est inférieure à 60 secondes, le troisième correspond à moins de 120 secondes, le quatrième 180 secondes (3 minutes) et le dernier comprend les pixels dont l'étendue temporelle est supérieure à 3 minutes. Concernant la variance, les seuils sont également définis en minutes, mais les unités sont ici les secondes au carré. Ces seuils seront donc 3600 secondes, 14400 et 32400 correspondant à une, deux et trois minutes respectivement.

Au final, il apparaît que les deux processus sont fonctionnels au vu des trois éléments de la validation mentionnés plus haut. Il ressort néanmoins que la méthode de l'étendue des heures de passage apparaît plus discriminante, étant donné que le passage aux différents seuils temporels définis est plus rapide que pour la méthode de la variance. En effet, le passage au dernier seuil (en rouge) dans le cas de l'étendue est plus rapide que pour la variance, qui n'est d'ailleurs même pas visible à la figure 36. Pour aller plus loin dans l'analyse des résultats de ces méthodes, et afin de transposer la meilleure d'entre elles dans l'extension logicielle, une comparaison de celles-ci a été réalisée au 6.4.

6.4 Comparaison des méthodes

Le sous-chapitre précédent a montré que les résultats obtenus avec les méthodes d'étendue et de variance des heures de passage sont assez proches et permettent tous les deux de valider leur méthode respective. Cette validation a été réalisée en prenant en compte une géométrie homogène des sites de crime autour du point d'ancrage ainsi que des heures de crimes quasiment exactes. Pour comparer ces deux méthodes plus en détails, l'idée est de tester leur robustesse face à une « mauvaise » configuration géométrique des sites de crimes et/ou des erreurs temporelles associées aux heures des crimes. Pour ce faire, un nouvel échantillon de réseau routier plus étendu a été sélectionné, afin d'augmenter les temps de déplacement du point d'ancrage vers les sites de crimes. Cet échantillon est repris à la figure 37, avec la disposition des données spatio-temporelles et le point d'ancrage théorique. Le point d'ancrage théorique (en bleu) a été choisi en agglomération afin de considérer un point d'ancrage plausible, tandis que les sites de crime ont été répartis en trois catégories : les sources utilisées pour une bonne géométrie (en vert), les sources utilisées pour une mauvaise géométrie (en rouge) et celles utilisées dans les deux dispositions (en rouge et vert). Ainsi, les sources dans le cadre d'une bonne géométrie sont réparties autour du point d'ancrage alors que dans le cadre d'une mauvaise géométrie elles sont toutes disposées entre le nord-ouest et le sud-ouest du point d'ancrage. A noter que le réseau routier est une nouvelle fois rasterisé à une résolution de 10 m.

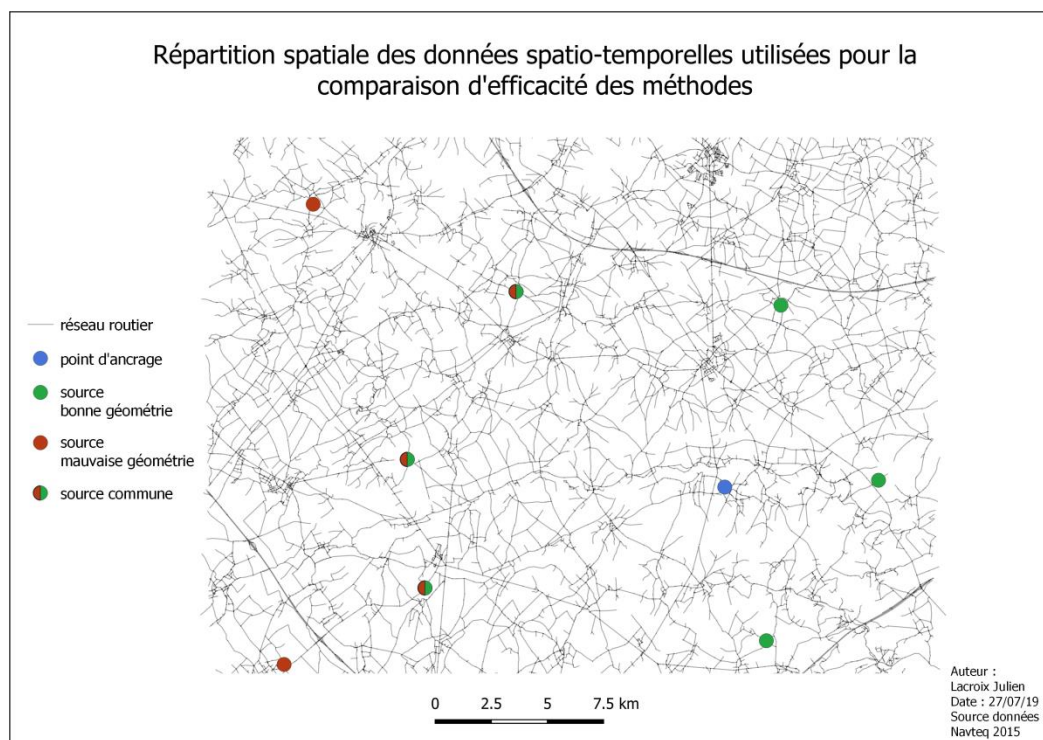


Figure 37 : répartition spatiale des données temporelles utilisées pour la comparaison des deux méthodes

6.4.1 Configuration géométrique non optimale

Le premier test est réalisé en prenant en compte les sources rouges et rouges – vertes de la figure 37 correspondant donc à une géométrie non optimale, avec des données temporelles exactes. Les résultats sont présentés aux figures 38 et 39, respectivement pour l'étendue et la variance. Comme pour la validation (cf. 6.3), les pixels ont été répartis en différentes classes dont la première est une nouvelle fois obtenue par seuillage de la valeur correspondant au point d'ancrage. Ensuite, des valeurs seuils d'erreurs temporelles de 4, 5 et 6 secondes ont été choisies pour la délimitation des autres classes, ce qui donne des valeurs seuils de variances de 16, 25 et 36 secondes² afin de pouvoir comparer les méthodes.

Il ressort de ces figures que la prise en compte d'une configuration géométrique non optimale des sources n'amène pas une différence importante entre les deux méthodes. Chacune admet les mêmes pixels présentant des valeurs inférieures à celle du point d'ancrage, disposés par ailleurs selon un axe est-ouest. Cette disposition des pixels minimisant l'étendue / la variance paraît logique étant donné que toutes les sources sont réparties à l'ouest de ce point d'ancrage. Ce qui distingue en revanche les deux méthodes, c'est une nouvelle fois la rapidité à laquelle on passe au-dessus du seuil de 6 secondes / 36 secondes². En effet, il apparaît une nouvelle fois que l'étendue temporelle est plus discriminante, les valeurs des pixels dépassent plus rapidement les seuils des classes par rapport à la variance temporelle. Cela se remarque visuellement par l'existence de nombreux pixels verts (classe 0.2 – 16) avec la variance, par rapport aux pixels oranges (classe 5 – 6) de l'étendue temporelle.

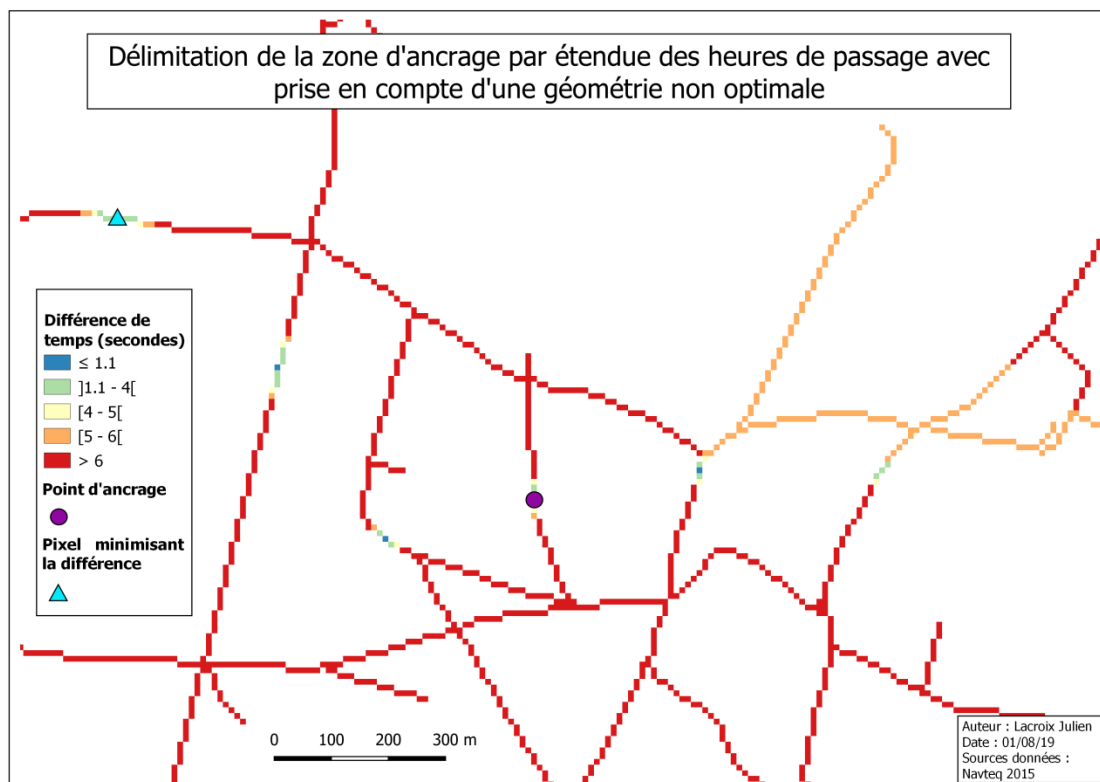


Figure 38 : délimitation de la zone d’ancrage par étendue des heures de passage avec prise en compte d’une géométrie non optimale

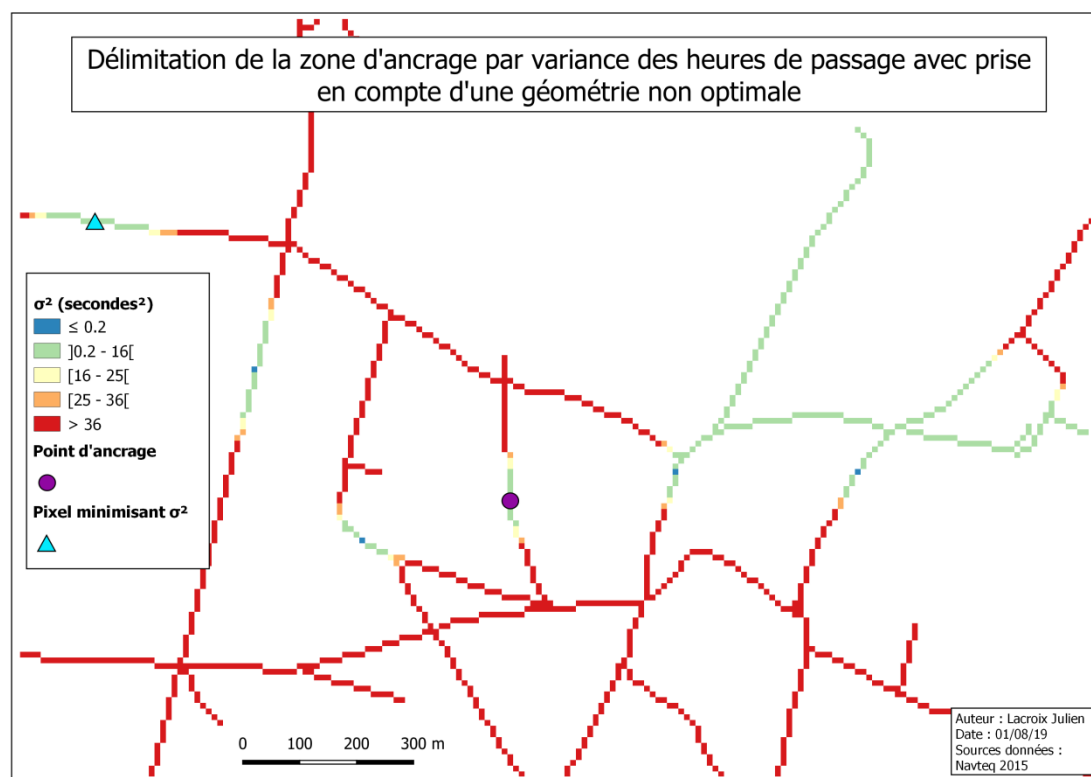


Figure 39 : délimitation de la zone d’ancrage par variance des heures de passage avec prise en compte d’une géométrie non optimale

6.4.2 Erreurs temporelles

Le deuxième cas traité est celui où la configuration des sources est homogène autour du point d'ancrage, correspondant dans la figure 37 aux points verts et rouges – verts. Les heures des crimes ont quant à elles été modifiées en ajoutant et en retirant entre 2 et 5 minutes. Les résultats sont repris aux figures 40 et 41. La comparaison entre les deux méthodes au moyen de seuils équivalents n'est désormais plus possible, parce que les valeurs d'étendue deviennent trop importantes par rapport à celles de la variance. La comparaison porte alors uniquement sur les pixels se trouvant sous le seuil de la valeur du pixel associé au point d'ancrage.

Il apparaît d'abord que la prise en compte d'erreurs temporelles influence considérablement plus le résultat que la prise en compte d'une configuration géométrique non optimale. Ensuite, l'aire délimitée par le seuillage de la valeur du pixel d'ancrage est différente selon les méthodes : elle est plus centrée sur le point d'ancrage avec la méthode d'étendue temporelle que celle de la variance. Cela implique dans ce cas-ci que le pixel correspondant au point d'ancrage dans le premier cas est entièrement compris dans la délimitation de l'aire, autrement dit il est entouré de valeurs inférieures ou égales, tandis que dans le second cas, ce pixel fait partie d'une augmentation constante des valeurs. De plus, le nombre de pixels appartenant à cette zone est plus restreint dans le cas de l'étendue temporelle, et les deux solutions qui minimisent la différence sont plus proches du point d'ancrage que la solution qui minimise la variance. Enfin, il faut noter la différence entre la valeur du pixel d'ancrage et celle de celui minimisant l'étendue / la variance. Dans le cas de la variance, cette valeur est de 4452 secondes² soit plus de 66 secondes alors que dans le cas de l'étendue cette valeur atteint un peu plus de 0.77 secondes.

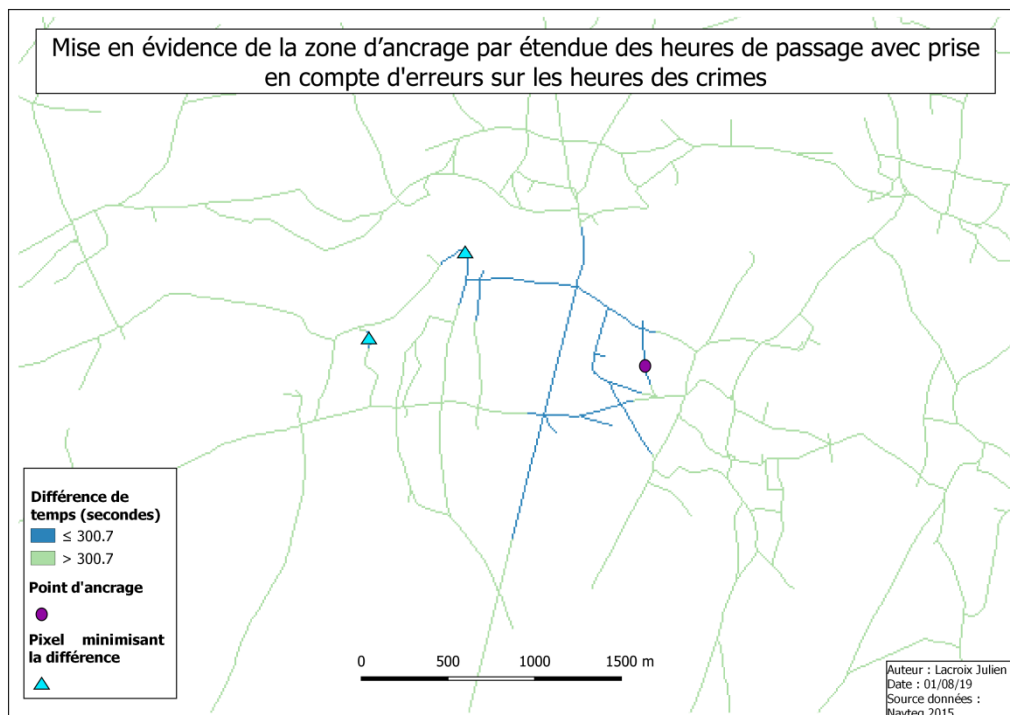


Figure 40 : mise en évidence de la zone d'ancrage par étendue des heures de passage avec prise en compte d'erreurs sur les heures des crimes

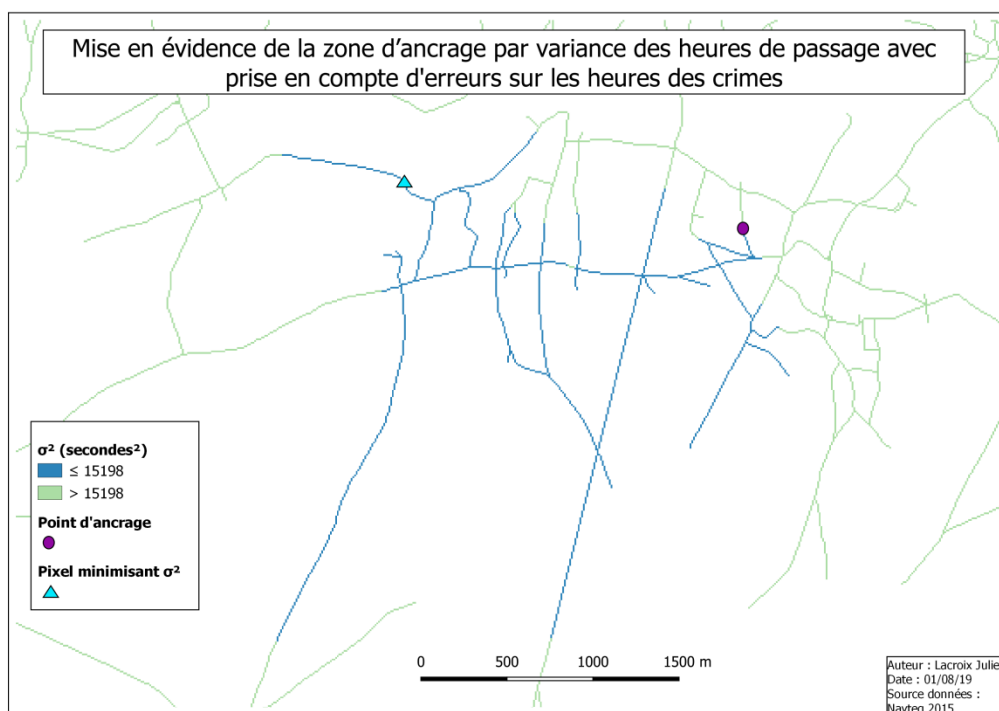


Figure 41 : mise en évidence de la zone d'ancrage par variance des heures de passage avec prise en compte d'erreurs sur les heures des crimes

6.4.3 Configuration géométrique non optimale et erreurs temporelles

Le troisième et dernier cas considéré est la combinaison des deux sources d'erreurs mentionnées et analysées ci-dessus. Pour ce faire, les sources rouges et rouges – vertes de la figure 37 ont été prises en compte comme pour le premier cas, avec des heures de crimes qui diffèrent comme dans le second cas. Pour la même raison que le second cas, la comparaison des seuils équivalents entre les deux méthodes n'a pas pu être mise en place. La comparaison porte donc sur les pixels se trouvant sous le seuil de la valeur du pixel associé au point d'ancrage (cf. Figures 42 et 43).

La comparaison entre les figures 42 et 43 est analogue à celle présentée au 6.4.2 : la méthode de l'étendue minimise le nombre de pixels dont la valeur est inférieure à celle du point d'ancrage, le *pattern* créé par ces pixels se concentre de manière générale autour de ce point d'ancrage (selon un axe nord-ouest, sud-est) et les pixels minimisant la différence de temps sont plus proches du point d'ancrage qu'avec la prise en compte de la variance.

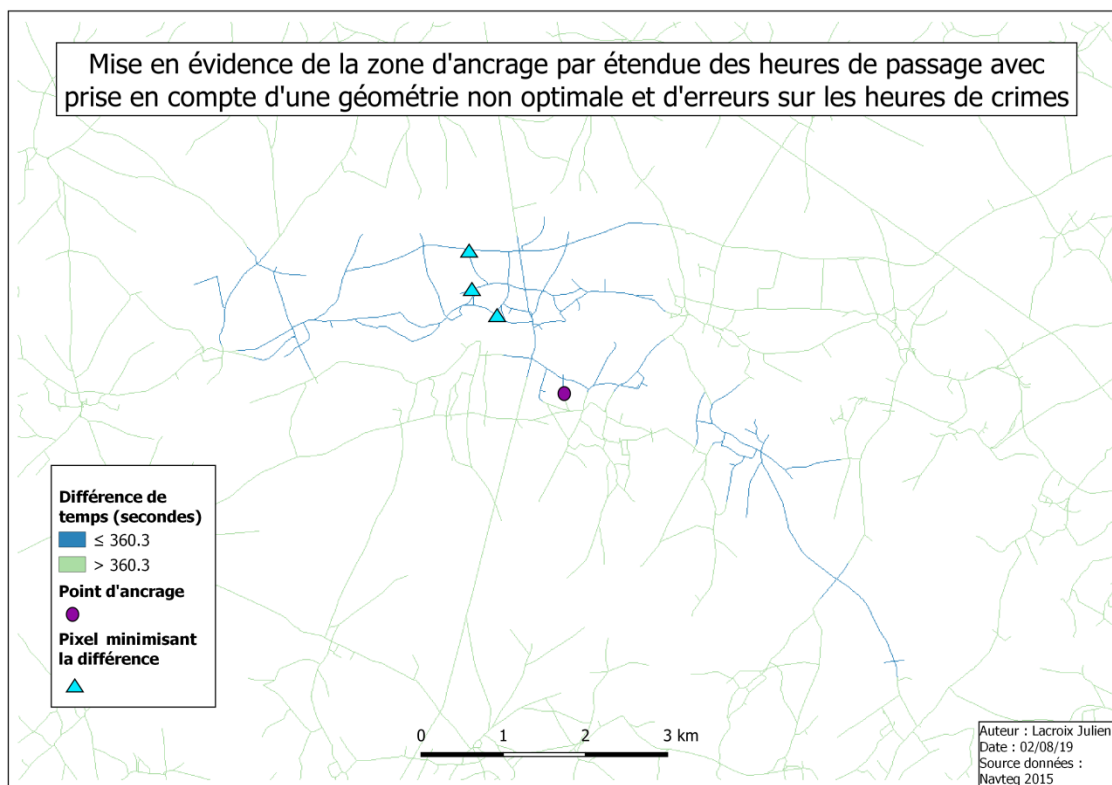


Figure 42 : mise en évidence de la zone d'ancrage par étendue des heures de passage avec prise en compte d'une géométrie non optimale et d'erreurs sur les heures des crimes

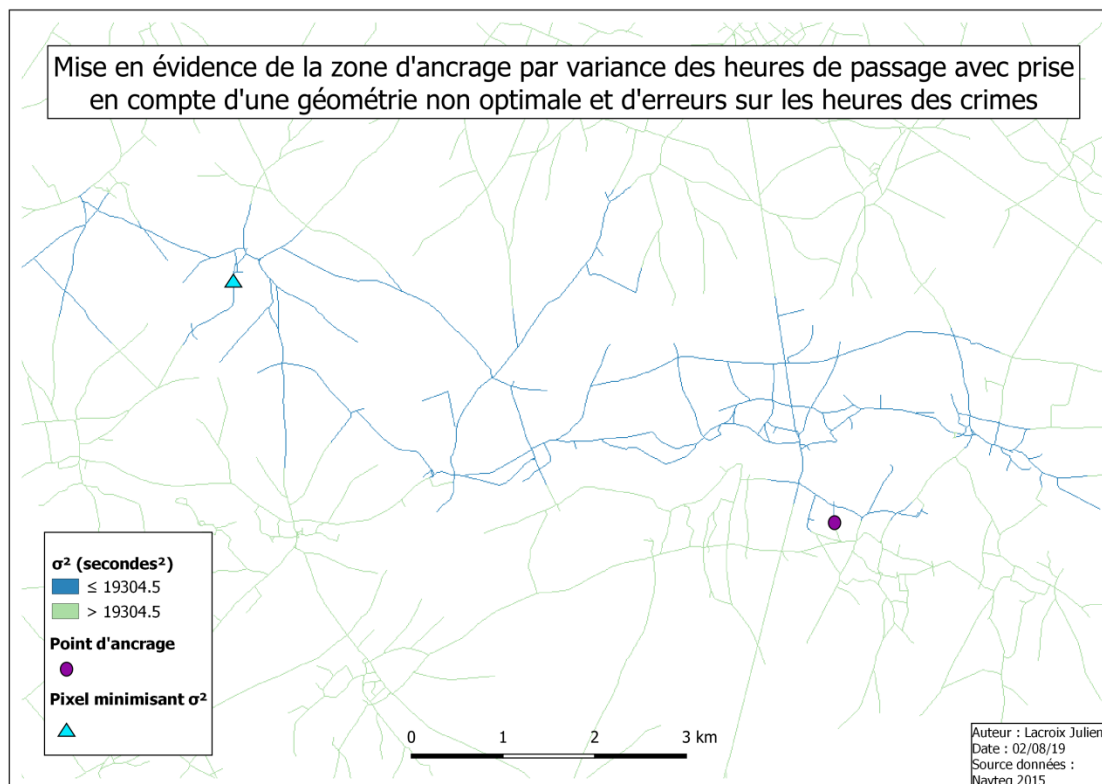


Figure 43 : mise en évidence de la zone d'ancrage par variance des heures de passage avec prise en compte d'une géométrie non optimale et d'erreurs sur les heures des crimes

6.4.4 Conclusion

Il ressort de ces résultats que la méthode de calcul de l'étendue des heures de passage est plus efficace que celle de la variance, qu'importe le type d'erreur pris en compte (configuration géométrique, heures des crimes, les deux combinés). D'abord, les solutions issues de la méthode de l'étendue sont plus proches du point d'ancrage recherché que via la méthode de la variance. Ensuite, la prise en compte de l'étendue des heures de passage est plus discriminante que la variance. En effet, pour des classes présentant des seuils équivalents entre les deux méthodes, la méthode de l'étendue les atteint plus rapidement. Cette meilleure discrimination peut s'expliquer par le fait que l'étendue est particulièrement sensible aux mesures aberrantes, ce qui au final aide à mieux discriminer les pixels proches du point d'ancrage par rapport à ceux plus éloignés. Cette meilleure discrimination induit par ailleurs une limitation du nombre de pixels dont la valeur est inférieure à celle du pixel d'ancrage. Pour toutes ces raisons, la méthode de calcul de l'étendue temporelle est celle qui est transposée dans l'extension logicielle. Enfin, il faut souligner le fait que les erreurs temporelles ont sensiblement plus d'impact qu'une mauvaise configuration géométrique.

6.5 Conclusion sur la délimitation de la zone d'ancrage

Les figures 35, 36, 38 et 39 ont montré comment délimiter la zone d'ancrage à partir des *rasters* obtenus à la fin des processus. Cette délimitation est réalisée par seuils correspondant à des secondes voire des minutes en fonction du résultat obtenu. Il apparaît sur ces figures que la zone d'ancrage délimitée est très restreinte, ce qui s'explique par l'absence d'erreurs temporelles dans le processus.

La figure 44 reprend quant à elle la délimitation de la zone d'ancrage dans le cas de la méthode de l'étendue des heures de passage, en considérant cette fois les erreurs sur les heures de crime (qui varient de 2 à 5 minutes). Cette délimitation n'avait pour rappel pas été appliquée, étant donné que la comparaison par seuils équivalents avec la méthode de la variance n'était pas possible. Cette figure 44 fait logiquement apparaître des différences de temps plus importantes, avec un minimum de plus de 4 minutes. Il ressort également de cette figure que le point d'ancrage ne fait pas partie des pixels minimisant l'étendue (en bleu). Néanmoins, il appartient au seuil supérieur (entre 5 et 6 minutes) ce qui délimite une zone d'ancrage de quelques kilomètres.

Évidemment, ce résultat lié à 5 crimes dont les heures sont connues de 2 à 5 minutes près constitue un cas quasiment idéal. En pratique, les heures de crimes sont susceptibles de présenter des approximations plus importantes, ce qui risque d'augmenter en proportion la taille de la délimitation de la zone d'ancrage. Cependant, ce résultat donne un ordre d'idée de du résultat qu'il est possible d'obtenir moyennant ces approximations introduites sur les heures de crimes.

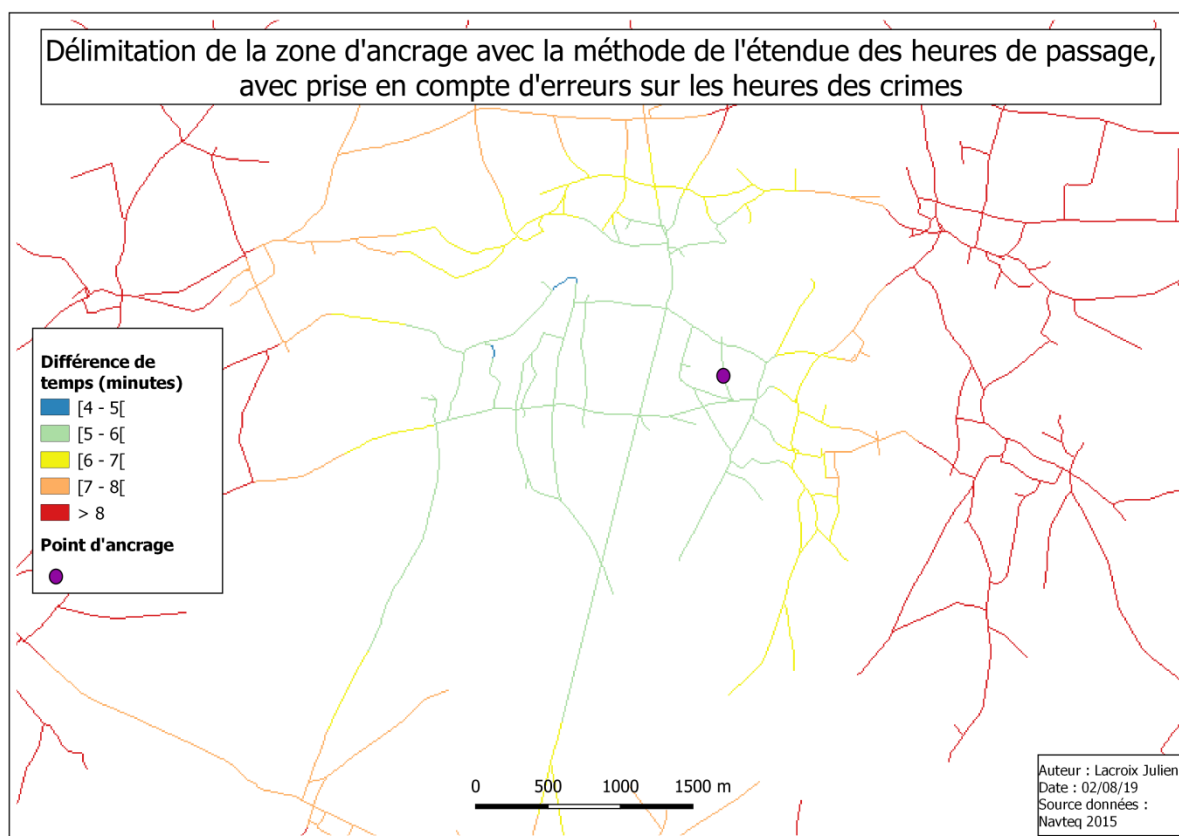


Figure 44 : délimitation de la zone d'ancrage avec la méthode de l'étendue des heures de passage, en considérant les erreurs sur les heures de crime

7. Amélioration des surfaces de coût

Le sous-chapitre 2.4 a mentionné certains problèmes courants inhérents au passage du mode vectoriel au mode maillé, notamment le manque de prise en compte de l'aspect non planaire des graphes routiers et la perte d'information quant au sens de circulation lors de la rasterisation. Pour rappel, cette non-planéité des graphes routiers provient du fait qu'il est impossible de les représenter dans un plan sans que des arêtes ne se croisent, comme c'est le cas des ponts et des tunnels.

Ces informations quant au sens de circulation sont accessibles en mode vectoriel via la table d'attributs. Lors du passage du mode vecteur au mode *raster*, ces données attributaires sont perdues et ne reste plus alors que la valeur des pixels, qui équivaut dans notre cas à l'identifiant du tronçon auxquels ils appartiennent.

Les deux problèmes mentionnés ci-dessus présentent un point commun : ils peuvent être résolus grâce aux données relatives au sens de circulation, par interdiction de la propagation du coût dans des cas bien définis. En effet, l'engagement dans un tronçon routier à contre-sens peut ainsi être évité, de même que la propagation depuis un pont / tunnel vers un tronçon routier « normal » passant en-dessous / au-dessus, et inversement.

7.1 Prise en compte des contre-sens

La solution proposée pour remédier au problème de la propagation du coût à contre-sens est la construction d'une liste contenant tous les mouvements non permis au départ de pixels correspondant aux extrémités des tronçons routiers. Cette liste contient quatre éléments par rangée : les deux coordonnées-image du pixel de départ et les deux coordonnées-image de celui d'arrivée. Si deux pixels voisins se trouvent sur une même rangée de cette liste, cela signifie qu'en partant du premier, il est interdit de rejoindre le second. Au final, l'idée est de modifier l'algorithme de surface de coût présenté au chapitre 5 afin qu'en chaque pixel il consulte cette liste, voie si les coordonnées-image du pixel en font partie et si c'est le cas, prenne en compte ces mouvements non permis dans la construction de la surface de coût.

Trois attributs du réseau routier sont nécessaires pour définir le sens de circulation : la direction du trafic routier et les identifiants des nœuds constituant chaque extrémité. Pour distinguer ceux-ci, un nœud est dit « de référence » (noté R) tandis que l'autre est dit « de non-référence » (noté NR). Le déplacement se fera différemment suivant la valeur de la direction du trafic routier, qui peut prendre une des trois valeurs F, T, B :

- F : le déplacement sur le tronçon se fait depuis le nœud R vers le nœud NR ;
- T : le déplacement sur le tronçon est réalisé, à l'inverse, depuis le nœud NR vers le nœud R ;
- B : le déplacement est possible dans les deux sens.

À partir de ces informations, il faut maintenant déterminer tous les cas pour lesquels le déplacement n'est pas permis, et pour ce faire des cas spéciaux tels qu'une sortie d'autoroute et un rond-point ont été analysés. Il ressort que les cas de figure pour lesquels le déplacement n'est pas permis sont au nombre de huit. Pour chaque cas, il faut déterminer la combinaison

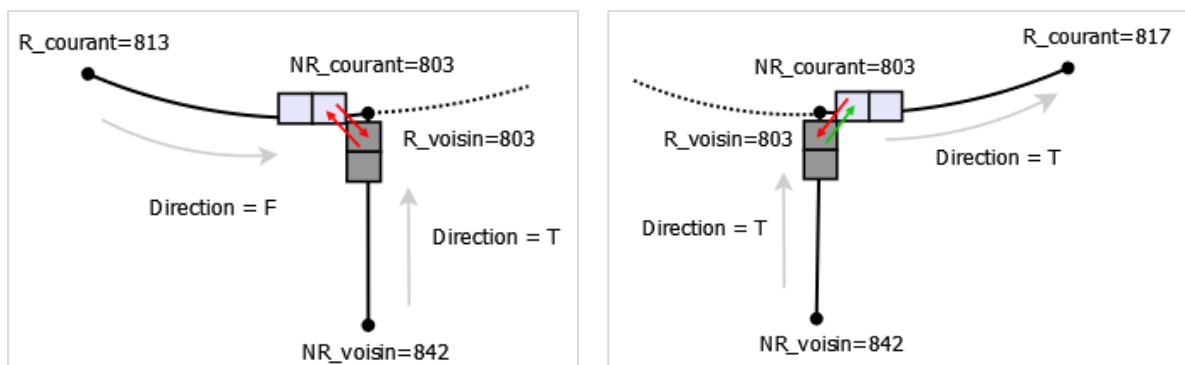
des trois attributs qui permet de mettre en évidence un déplacement interdit. Les figures 45 et 46 reprennent l'exemple d'un tronçon routier en rejoignant un second qui ne présente qu'un seul sens de circulation. Il s'agit dans ce cas-ci d'un rond-point, mais l'exemple d'une sortie d'autoroute rentre aussi dans ce cas de figure. Chacune de ces figures reprend un cas de déplacement interdit d'un pixel à un autre. Dans le premier cas (Figure 45), il s'agit d'empêcher d'une part la sortie du rond-point via une de ses entrées (pixel gris clair vers gris foncé), d'autre part le déplacement à contre-sens dans le rond-point en s'y engageant (pixel gris foncé vers gris clair). A noter que les noms des nœuds R et NR (« courant » et « voisin ») sont associés à la première configuration (i.e. pixel gris clair vers gris foncé), la seconde configuration est simplement obtenue en intervertissant ces noms. Ces deux configurations respectent les mêmes conditions :

- les deux directions sont différentes de « B » ;
- les deux directions sont différentes entre elles (ce qui est le cas avec « F » et « T ») ;
- la valeur du nœud NR du pixel courant est égale à la valeur R du pixel voisin (première configuration) ou bien la valeur du nœud R du pixel courant est égale à la valeur NR du pixel voisin (seconde configuration, si on considère l'intervertissement des noms). Pour les deux configurations, cette valeur est 803.

Dans le second cas (Figure 46), l'engagement dans le rond-point est possible mais pas l'inverse. Si on considère par exemple que l'algorithme de surface de coût débute au pixel venant du rond-point (en gris clair), il ne peut sortir du rond-point en ce sens vers le pixel gris foncé. Les conditions à respecter pour mettre en évidence ce cas de figure sont les suivantes :

- les deux directions de tronçons sont égales à « T » ;
- la valeur du nœud NR du pixel courant est égale à celle du nœud R du pixel voisin (dans ce cas-ci, 803).

Les autres cas sont traités de manière analogue. Dès qu'un cas est vérifié, les coordonnées-image du pixel courant et du pixel voisin sont enregistrées dans la liste.

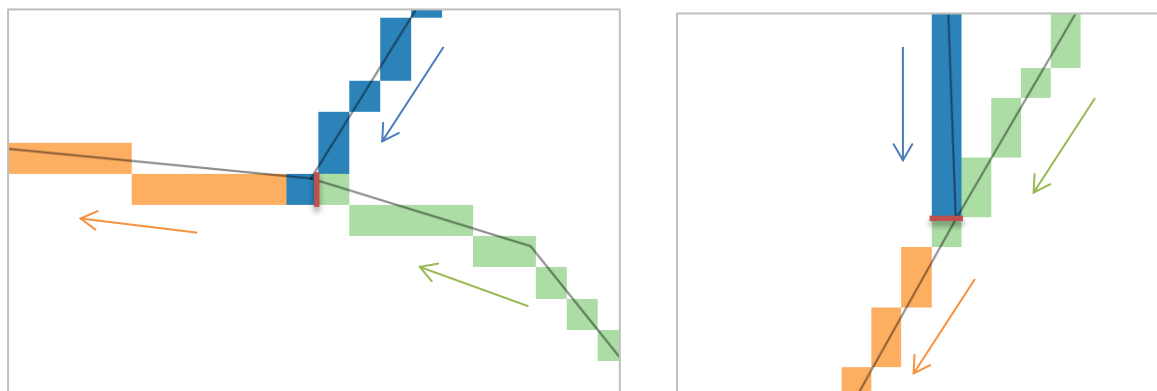


Figures 45 et 46 : détermination de deux cas de déplacement impossible d'un pixel à un autre

En pratique, on part du *raster* obtenu après rasterisation en fonction de l'identifiant des tronçons routiers. Les pixels du *raster* sont parcourus jusqu'à en trouver un présentant une valeur autre que celle correspondant à *No Data*. Dès qu'un tel pixel est trouvé, les pixels voisins sont évalués pour déterminer si leur identifiant diffère de celui du pixel courant et de

la valeur *No Data*. Si c'est le cas, cela signifie que le pixel voisin appartient à un tronçon routier voisin, et donc que les huit cas de figure mentionnés précédemment peuvent être testés. Si l'un d'eux est respecté, alors les coordonnées-image du pixel courant et du pixel voisin sont enregistrées dans la liste. Cet algorithme a été implémenté avec succès et est disponible à l'annexe 7. Il a donc pour résultat cette liste contenant tous les mouvements qui ne sont pas permis depuis les pixels des extrémités des tronçons routiers. Les mouvements repris dans la liste et symbolisés par deux coordonnées-image successives de pixels voisins ont été validés sur un échantillon du réseau routier vectoriel et du réseau routier rasterisé en fonction de l'identifiant.

Cette méthode décrite ci-dessus présente néanmoins une importante limite, et celle-ci provient de la rasterisation. Considérons les figures 47 et 48 ci-dessous, représentant les cas respectivement d'un rond-point et d'une sortie d'autoroute avec trois tronçons routiers distincts. Les flèches colorées indiquent le sens de circulation associé à chaque tronçon, tronçons rasterisés à une résolution d'1 m afin de montrer que cette limite est indépendante de la résolution spatiale choisie. Il apparaît que le pixel correspondant à l'intersection des tronçons routiers de la figure 47 appartient au tronçon menant vers le rond-point (en bleu) tandis qu'à la figure 48, il s'agit d'un pixel du tronçon routier (en vert) sur lequel arrive la sortie d'autoroute (en bleu) qui est rasterisée à cette intersection. Le problème dans ces deux cas est une restriction du déplacement, symbolisée par un trait vertical / horizontal rouge, suivant la méthode expliquée précédemment : dans le premier cas (Figure 47), un pixel du tronçon vert ne peut se propager sur un pixel du tronçon bleu étant donné que ce dernier ne constitue pas une sortie de rond-point. Dans le second cas (Figure 48), le pixel final de la sortie d'autoroute ne pourra pas non plus se déplacer sur le tronçon routier précédant l'intersection (en vert).



Figures 47 et 48 : exemple de la limite de la méthode sur un rond-point (gauche) et sur une sortie d'autoroute (droite), par restriction d'un déplacement autorisé (symbolisée par le trait vertical / horizontal rouge)

7.2 Prise en compte de la non-planéité des graphes routiers

Les mouvements non permis depuis un pixel du *raster* vers un autre, comme expliqué au 7.1, peuvent être étendus à la prise en compte de la non-planéité des graphes routiers. Ces mouvements interdits réunissent dans ce cas, à chaque rangée de la liste, deux pixels de tronçons routiers qui ne sont pas contigus. L'un de ces deux pixels doit, de plus, appartenir à

un pont / tunnel, suivant la valeur des attributs 'BRIDGE' (pont) et 'TUNNEL' du réseau routier vectoriel. Contrairement au 7.1, il n'y a donc qu'un cas unique à considérer ici. Il suffit uniquement de comparer les valeurs des nœuds « de référence » et « de non-référence » des tronçons routiers associés aux deux pixels, dont l'un est un tunnel ou un pont. Si aucune valeur n'est identique, alors les deux pixels voisins appartiennent bien à deux tronçons routiers qui ne sont pas contigus, et le déplacement n'est donc pas permis. Cette méthode présente néanmoins deux importants problèmes.

D'abord, le résultat de la rasterisation aux intersections a pour conséquence de perdre l'information d'un des deux tronçons routiers. Un seul des deux tronçons est en effet rasterisé, puisque le pixel ne peut présenter qu'une valeur unique. Ceci est repris à la figure 49, qui montre deux types d'éléments linéaires, à savoir un tronçon routier « normal » (en rouge) et un tunnel (en jaune), qui se croisent à deux reprises. Après rasterisation, il apparaît que les deux pixels correspondant aux deux intersections sont associés au tunnel (en gris). Le tronçon routier « normal » (en noir) est quant à lui divisé en trois parties. Le problème de cette méthode est d'empêcher à tort la propagation du coût depuis le tronçon routier « normal » (en noir) vers le tunnel (en gris).

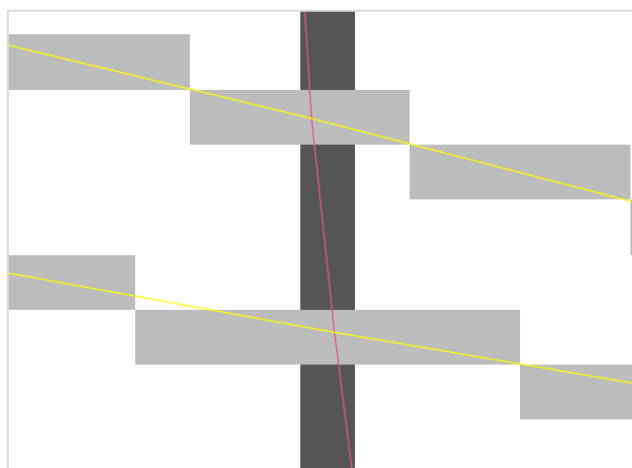


Figure 49 : rasterisation appliquée à un tronçon « normal » (en rouge) et à un tunnel (en jaune) qui se croisent à deux reprises. Les deux pixels relatifs aux intersections correspondent au tunnel, ce qui divise le tronçon routier « normal » en 3 parties

Le second problème provient des données vectorielles du réseau routier utilisé dans le cadre de ce travail. Les données attributaires relatives aux ponts et aux tunnels ne sont pas complètement correctes ; ainsi, de nombreux ponts et tunnels ne sont pas répertoriés dans les données attributaires. Ceci constitue plutôt une limitation à la méthode, dans le sens où cela ne l'empêche pas de fonctionner mais limite toutefois son efficacité.

7.3 Amélioration des méthodes

Il est à souligner que les dysfonctionnements affectant ces deux méthodes sont similaires : la propagation du coût est arrêtée alors qu'elle est pourtant valide. Pour résoudre ces dysfonctionnements, un algorithme amélioré des méthodes est ici proposé :

- a) les informations relatives aux sens de circulation et au caractère de pont et de tunnel sont stockées dans un dictionnaire. Un dictionnaire est une structure permettant de stocker des données qui seront accessibles via une clé, et non plus des indices comme le cas des listes. Il est important de le souligner étant donné que cela amène une grande simplicité : chacun des attributs mentionnés ci-dessus sont accessibles en connaissant la clé, qui est simplement l'identifiant du tronçon.
- b) les pixels du *raster* sont parcourus jusqu'à en trouver un présentant une valeur autre que celle correspondant à *No Data*. Dès qu'un tel pixel est trouvé, les pixels voisins sont évalués pour déterminer si leur identifiant diffère de celui du pixel courant et de la valeur *No Data*. Si c'est le cas, on distingue les deux cas :
- prise en compte des contre-sens : vérification des huit cas mentionnés au 7.1. Si l'un des huit cas de figures est validé, cela signifie qu'un mouvement interdit a été détecté et l'algorithme passe à l'étape suivante (cf. c) ;
 - prise en compte de la non-planéité du graphe : dans ce cas-ci, et comme mentionné au 7.2, il s'agit d'un cas unique à considérer. Si ce cas de figure est détecté, l'algorithme passe également au point c.
- c) les voisins du pixel bloquant la propagation sont alors considérés, sauf le pixel depuis lequel on vient, les pixels *No Data* et les pixels voisins appartenant au même réseau routier que celui empêchant la propagation. Ce dernier cas permet, si l'on reprend la figure 49, d'éviter la propagation du coût sur les autres pixels du tunnel. Les deux méthodes sont alors encore à distinguer :
- prise en compte des contre-sens : les huit cas de configuration sont une nouvelle fois vérifiés, cette fois entre le pixel depuis lequel la propagation est arrêtée et le voisin de celui bloquant cette même propagation. Un neuvième cas est également à considérer : celui où les pixels appartiennent à deux tronçons routiers non contigus comme au 7.2. Les deux pixels doivent en effet obligatoirement appartenir à des tronçons contigus pour permettre le déplacement. Si aucun des cas de configuration n'est respecté, cela signifie que le mouvement est empêché à tort. Dans le cas contraire, l'arrêt de la propagation est bien valide ;
 - prise en compte de la non-planéité du graphe : le prolongement du tronçon routier segmenté à l'intersection avec un autre (les pixels noirs à la figure 49 par exemple) est soit ce même tronçon routier, soit un autre tronçon contigu. Dans ces deux cas, il suffit de vérifier qu'une des valeurs R et NR des tronçons routiers associés aux pixels comparés soient égales. Si c'est le cas, le mouvement est empêché alors qu'il est valide.
- d) création d'une liste contenant les coordonnées-image du pixel au départ du mouvement, celles du pixel bloquant à tort la propagation et enfin les coordonnées-image du pixel vers lequel le mouvement est empêché à tort.

L'annexe 8 présente cette méthode concernant la prise en compte de la non-planéité du graphe. Celle de la prise en compte des contre-sens n'est pas présentée parce qu'elle utilise l'algorithme du 7.1 et la recherche des voisins du pixel empêchant la propagation comme présenté à l'annexe 8. Au final, la liste permet théoriquement à l'algorithme de surface de coût de mettre en évidence les mouvements considérés à tort comme interdits. Il permet également de lui fournir les coordonnées-image du pixel vers lequel cette propagation est valide. Malheureusement, ces améliorations souffrent également de problèmes dus à la rasterisation.

7.4 Limitations

Ces méthodes subissent des limitations substantielles. La figure 50 reprend une configuration complexe du réseau routier comprenant une partie de rond-point (en rouge) et des tunnels (en jaune), qui permet de mettre en évidence les limites de ces deux méthodes.

D'abord, l'intersection de deux éléments linéaires jusqu'ici considérée était relativement orthogonale. Pour des tronçons routiers parallèles, il y a une perte complète d'information d'un des deux tronçons sur plus d'un pixel. En conséquence, aucun coût ne pourra être propagé chaque fois que cette perte d'information s'étale sur plus d'un pixel. Par exemple, la rasterisation à droite du rond-point (Figure 50) est uniquement relative au tronçon provenant du rond-point, l'information quant à la présence du tunnel est dès lors complètement supprimée et la propagation du coût est impossible.

Ensuite, en fonction de la résolution spatiale de la rasterisation et de la configuration des tronçons routiers, le mouvement peut être empêché par le fait que le pixel voisin du pixel empêchant la propagation ne fait pas partie d'un tronçon routier contigu à celui du pixel de départ. Si on considère une nouvelle fois la figure 50, un pixel de départ peut être celui portant le numéro 1. Celui portant le numéro 2 est relatif à la route arrivant à droite du rond-point, tandis que le numéro 3 est associé au tunnel. La méthode proposée va considérer le pixel numéro 4 étant donné que le 3 n'est pas accessible depuis le numéro 1. Ce pixel numéro 4 ne respecte pas les conditions pour pouvoir propager le coût et ce pour deux raisons. D'une part, ce pixel appartient au tronçon routier arrivant au rond-point ; d'autre part, même s'il était associé au tronçon routier quittant le rond-point, ce dernier n'est pas contigu au tronçon routier du pixel numéro 1, ce qui rend impossible le déplacement.

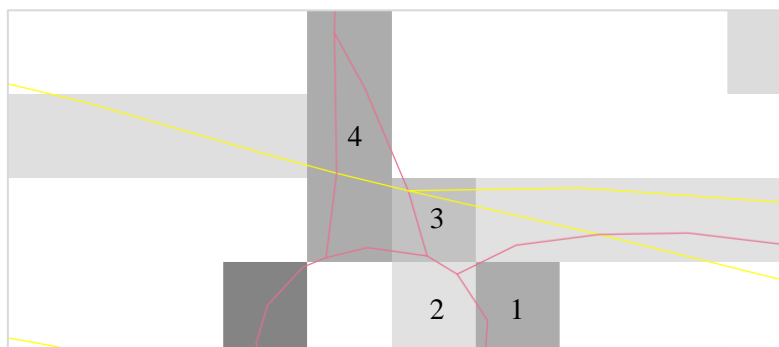


Figure 50 : configuration complexe du réseau routier comprenant un rond-point (en rouge) et des tunnels (en jaune)

7.5 Conclusion

Il ressort que les méthodes présentées pour solutionner ces deux problèmes sont fortement limitées par le résultat de la rasterisation, ce qui a été illustré par deux exemples de problèmes qui se présentent lors d'une configuration routière complexe. Dès lors, elles ne s'avèrent pas assez efficaces que pour être ajoutées aux fonctionnalités de l'algorithme de surface de coût présenté au chapitre 5. L'ajout de ces méthodes à l'algorithme de surface de coût aurait pour conséquence d'arrêter la propagation à de multiples endroits, alors même que celle-ci est valide. Cela induirait au final que tous les pixels du réseau routier ne présentent pas de valeur de coût, ce qui est plus préjudiciable encore que les erreurs dues à la non prise en compte de ces deux problèmes.

8. Développement de l'extension logicielle

8.1 Initialisation de l'extension

Une extension logicielle, ou *plugin*, est un module qui vient s'ajouter à un logiciel existant afin d'y ajouter certaines fonctionnalités. On peut notamment prendre l'exemple de l'extension *PDF Viewer*, permettant d'afficher un fichier PDF directement dans le navigateur web. Sur un logiciel-SIG tel que QGIS, cela permet à un utilisateur d'implémenter et valider une procédure qu'il a préalablement conçue. Ces extensions développées par des utilisateurs de QGIS peuvent être définies comme expérimentales ou stables par leur développeur. Ces deux types d'extensions peuvent être déposés sur le répertoire⁴ des extensions QGIS afin de pouvoir être téléchargées et utilisées par tout autre utilisateur de QGIS, le caractère expérimental servant simplement d'avertissement à l'utilisateur qui télécharge l'extension. Pour donner un ordre d'idées, les extensions les plus plébiscitées dépassent généralement la centaine de milliers de téléchargements, et peuvent atteindre le million de téléchargements. Dans notre cas, il s'agira d'une extension expérimentale uniquement accessible en local.

La création de cette extension requiert deux éléments en dehors de QGIS : un IDE (cf. précédemment, Pycharm) afin de la programmer et le logiciel Qt Creator qui permet de créer l'interface graphique d'applications (l'interface de QGIS est d'ailleurs créée grâce à ce logiciel). La création de l'extension débute sur QGIS, avec l'extension *QGIS plugin builder* (version 3.1). Celle-ci permet de créer un *template* de l'extension, en demandant au développeur de fournir quelques informations générales sur celle-ci telles que son nom, l'auteur qui l'a développée, une brève description, la version minimum du logiciel QGIS pouvant l'utiliser, l'onglet du menu principal de QGIS dans lequel il sera accessible (vecteur, raster...) et le type de *template* à utiliser parmi ceux proposés par défaut par QGIS. Le *template* « *tool button with dialog* » a ici été utilisé, il s'agit d'un formulaire à remplir muni d'un bouton pour lancer le processus ou bien l'annuler.

Au final, le résultat de cette extension *plugin builder* est un dossier de fichiers relatif à l'extension logicielle, créé dans le répertoire approprié de QGIS. Ce dossier contient divers sous-dossiers et fichiers dont deux sont particulièrement intéressants car ce sont ceux devant être modifiés afin de faire évoluer l'extension. Le premier de ces fichiers, portant le nom de l'extension au format « .py », est le code Python de l'extension qui est à modifier directement avec l'IDE PyCharm. Le second fichier, au format « .gui », correspond à l'interface graphique de l'extension ; le code source dans ce cas-ci est modifié indirectement en fonction des modifications graphiques de l'interface opérées sur Qt Creator. Enfin, un fichier *batch* (fichier de lignes de commandes, cf. Annexe 9) est à créer et à exécuter afin de mettre en place les liaisons entre Qt Creator et Python (et donc QGIS). L'interface graphique peut ensuite être chargée et visualisée sur QGIS. Celle-ci, disponible ci-dessous à la figure 51, constitue le visuel de l'extension logicielle dans sa version la plus primitive. A noter que le nom donné à l'extension constitue ici un exemple, pour toute la suite de ce travail l'extension créée sera nommée « *Cost_surface_generalization* ».

⁴ <https://plugins.qgis.org/plugins/>

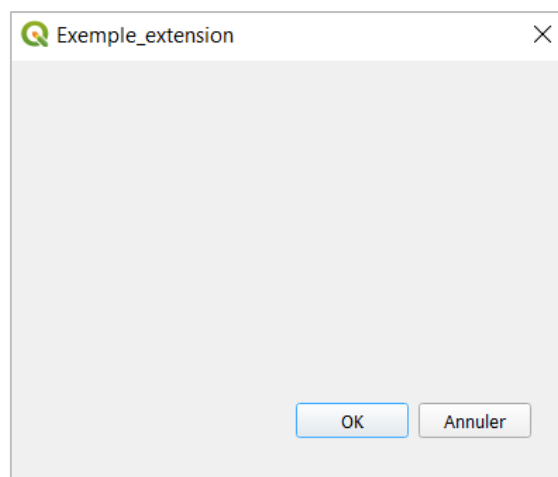


Figure 51 : *template* de base de l'extension logicielle

8.2 Transposition théorique

La modification de l'interface graphique de l'extension nécessite de connaître les informations que requiert le processus afin d'être fonctionnel. Le processus doit également être imaginé de manière plus générale, en fonction de ce que l'utilisateur pourrait introduire comme informations. Autrement dit, l'objectif est de rendre l'extension robuste, c'est-à-dire résistante aux éventuelles erreurs de l'utilisateur. Au final, la transposition du processus dans l'extension logicielle s'accompagne de trois modifications du processus précédemment développé, ainsi que de discussions sur la manière dont les données sont fournies à l'extension.

La première modification porte sur le processus de rasterisation, qui n'en fait désormais plus partie. En effet, si on imagine le cas où l'utilisateur introduit l'ensemble du réseau routier vectoriel en *input*, cette rasterisation ne sera pas possible étant donné que le fichier est trop volumineux. Il n'est pas non plus possible de déterminer une zone plus restreinte à rasteriser à partir de l'ensemble de ce réseau routier, même en fonction des données spatio-temporelles. Par exemple, une rasterisation en fonction des quatre coordonnées extrêmes des données spatio-temporelles nécessite d'être en présence d'une configuration géométrique homogène des sites de crime autour du point d'ancrage, ce qui n'est pas nécessairement vérifié. Elle pose également problème si le tronçon routier le plus proche auquel est rattaché le site de crime se trouve en dehors de cette zone rasterisée. Enfin, garder cette étape de rasterisation dans le processus nécessiterait d'effectuer la même rasterisation à chaque lancement de l'extension, ce qui est inutile. Pour toutes ces raisons, l'utilisateur effectue lui-même la rasterisation avant d'utiliser l'extension.

Par ailleurs, cette rasterisation effectuée par l'utilisateur affecte directement la création de la surface de friction améliorée, qui démarre du résultat de cette rasterisation. Pour obtenir le même résultat, la rasterisation désormais effectuée par l'utilisateur doit donc être réalisée en fonction de l'identifiant des tronçons routiers. La création de la surface de friction améliorée nécessite également la connaissance des attributs d'identifiant des tronçons routiers et du coût associé à chacun d'eux. Ce dernier, provenant des attributs de vitesse autorisée sur le tronçon et de longueur de celui-ci, est désormais inutile dans le cadre de la généralisation du

processus. Dans ce contexte, ce sont ces deux attributs de vitesse et longueur qui seront pris en compte dans le calcul du coût temporel de traversée des pixels, qui se fera dans le processus. Par conséquent, il ressort que ces informations doivent être fournies à l'utilisateur au même titre que le réseau routier vectoriel afin de déterminer le rattachement au réseau routier le plus proche de chaque site de crime.

La deuxième modification du processus est l'ajout de deux nouvelles fonctionnalités pouvant être proposées par l'extension logicielle. Un des postulats évoqués dès l'introduction de ce travail est une heure de départ constante d'un point d'ancrage vers une série de crimes connectés. Cependant, deux autres méthodes ont également été introduites dans le cadre du profilage géographique, correspondant à deux autres comportements d'un criminel. Ces deux comportements sont basés non plus sur l'analyse des heures de passage comme dans le cas de la méthode de l'étendue, mais sur l'analyse des temps de déplacement (ou distance-temps) depuis le point d'ancrage vers les différents sites de crime. Les données temporelles sont par conséquent ici inutiles. Dans le premier cas, le postulat est que le criminel, par son comportement répétitif, tend à minimiser la moyenne de ses déplacements (Snook *et al*, 2005) ; dans le second cas, il s'agit de minimiser la variance de ces mêmes déplacements (Trotta, 2012). Au final, ces calculs ne s'éloignent pas de ceux réalisés lors de l'implémentation du processus. En effet, il suffit de supprimer la transposition de l'aspect temporel progressif en aspect régressif (cf. Figure 28, création et enregistrement d'une surface de coût) et d'ensuite utiliser le processus développé dans le cadre de la prise en compte de la variance des heures de passage (la moyenne est similaire). Au final, la facilité avec laquelle ces deux nouvelles fonctionnalités peuvent être ajoutées à l'extension, correspondant à deux autres méthodes de mise en évidence d'un point d'ancrage dans le cadre du profilage géographique, justifie de rassembler toutes ces procédures dans une seule et même extension logicielle. L'utilisateur peut alors choisir celles(s) dont il a besoin et seules les procédures choisies seront exécutées.

La troisième modification est la suppression de l'enregistrement de l'ensemble des *rasters* créés pendant le processus. Tout est réalisé en mémoire, seul le *raster* final est enregistré.

Concernant la manière dont les données sont fournies à l'extension, il existe deux options : soit l'utilisateur fournit les chemins d'accès aux différents fichiers, soit une liste déroulante propose de sélectionner une couche parmi celles déjà importées dans QGIS. La seconde option a ici été choisie, parce qu'elle permet notamment d'éviter tous les paramètres variables du format « .csv » lors de son import. De plus, l'utilisateur visualise ses fichiers avant de lancer l'extension, ce qui réduit les erreurs de prise en compte de mauvais fichiers.

8.3 Modification de l'interface graphique

L'interface graphique peut maintenant être améliorée grâce au logiciel Qt Creator. Comme mentionnées ci-dessus, les informations obligatoires sont les suivantes :

- couche vectorielle : couche du réseau routier vectoriel utilisé pour le rattachement des sites de crimes au réseau routier et la récupération des trois attributs pour la création de la surface de coût améliorée (identifiant, longueur et vitesse autorisée) ;

- couche *raster* : couche du réseau routier rasterisé en fonction de l'identifiant ;
- données csv : données spatio-temporelles des crimes ;
- trois propositions de méthodes de traitement des données (cf. 8.2) ;
- trois *outputs* (un par méthode) : chemin d'accès et définition du nom du fichier en *output*.

Enfin, une description de l'extension est également envisagée de manière d'une part à expliquer à l'utilisateur ses fonctionnalités, d'autre part à donner des éclaircissements sur les informations à fournir afin de la faire fonctionner correctement. A noter que le système de projection ne fait pas partie des informations obligatoires étant donné que celui du fichier vecteur est connu. Le/les fichiers(s) finaux présenteront donc ce même système de projection.

L'esthétique de l'interface graphique étant choisie, elle peut être matérialisée sur le *template* grâce à Qt Creator. La figure 52 ci-dessous montre l'interface graphique de ce logiciel, comprenant le *template* de l'extension au centre (1).

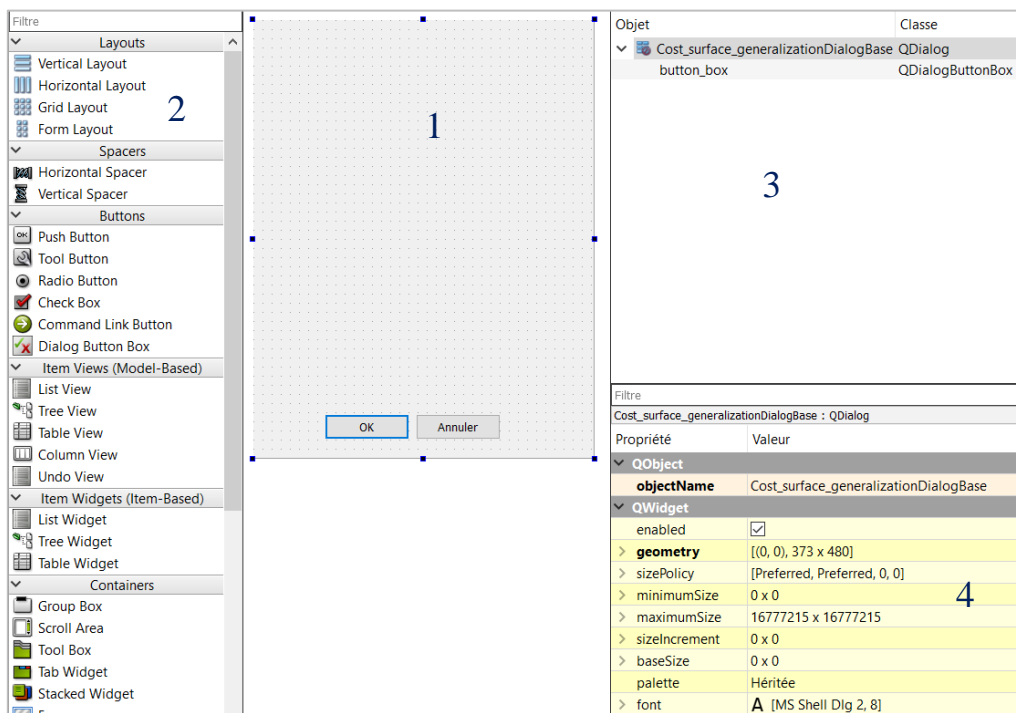


Figure 52 : interface graphique de Qt Creator, avec le *template* (1) de l'extension logicielle

La construction de l'interface graphique de l'extension logicielle se déroule par ajout d'un ensemble de composants (composants d'interface graphique ou *widgets*) formant chacun une couche. Une liste de ces composants, qui sont en réalité des objets de classes Qt Creator, est disponible à gauche de l'interface (2). On y retrouve notamment divers types de boutons, de zones de texte ou encore de listes déroulantes. Lorsqu'un composant est sélectionné et ajouté à l'interface graphique de l'extension, il s'affiche en haut à droite de l'interface graphique de Qt Creator (3). On retrouve, dans cet espace, la liste ordonnée des couches relatives aux composants formant l'interface graphique. Les deux premiers composants sont définis par défaut par le *plugin builder*, il s'agit pour le premier de la fenêtre de dialogue de l'extension, et pour le second d'un bouton d'exécution menant vers le processus. Pour chaque composant,

le nom de l'objet (modifiable) et sa classe d'appartenance sont donnés. Enfin, des informations quant à l'objet sélectionné au (3) s'affichent en bas à droite de l'interface graphique de Qt Creator (4), notamment le nom de l'objet, sa taille ou encore son emprise dans la fenêtre de dialogue.

Les éléments devant peupler l'extension ont été mentionnés ci-dessus. Pour chacun d'eux, il est nécessaire de définir le nombre de composants à leur associer ainsi que leur type :

- description : un seul composant est nécessaire, à savoir une zone de texte (classe '**QTextBrowser**' de Qt Creator) ;
- couche vectorielle, attributs d'identifiant – longueur – vitesse, couche *raster*, données CSV : deux composants sont nécessaires pour chacun de ces éléments :
 - un affichage texte (sans zone associée, uniquement du texte – classe '**QLabel**') pour préciser l'information à fournir par l'utilisateur ;
 - une liste déroulante (classe '**QComboBox**') permettant d'effectuer un choix parmi les différentes couches / les différents attributs. Il faut d'ores et déjà noter que les listes déroulantes des trois attributs sont liées à celle de la couche vectorielle et doivent être modifiées en même temps, ce qui sera réalisé au 8.4.2.
- trois propositions de méthodes : un seul composant par méthode est suffisant, il s'agit d'une case à cocher (classe '**QCheckBox**') comprenant une description textuelle ;
- fichier(s) en *output* : trois composants sont indispensables par élément. Il s'agit :
 - d'un affichage texte comme dans le second cas ci-dessus ;
 - d'un éditeur de texte (classe '**QLineEdit**') qui contiendra le chemin d'accès pour l'enregistrement de l'*output* ;
 - d'un bouton de commande (classe '**QPushButton**') permettant d'accéder au répertoire des fichiers et d'ainsi permettre à l'utilisateur de définir le chemin d'accès et le nom de fichier voulus. Dans le cas où le fichier existe déjà (i.e. même nom et même format), il est possible d'écraser celui-ci et de le remplacer par le nouveau.

L'affichage de ces éléments, autrement dit la possibilité d'exporter un fichier (*output*), dépend du cas où l'utilisateur a bien coché la case appropriée. Ces trois composants dépendent donc de la case à cocher qui leur est associée.

Au final, l'interface graphique de l'extension obtenue après prise en compte des considérations susmentionnées est disponible à la figure 53 (à gauche). A noter que par souci de clarté, des noms distincts ont été donnés aux objets-composants de l'interface graphique de l'extension étant donné qu'ils seront appelés dans la modification du code. Soulignons enfin, comme expliqué ci-dessus, que les trois affichages des trois composants de l'export de fichier sont ici visibles mais ne le sont en pratique que si leur case respective a été cochée. Par défaut, c'est-à-dire à l'initialisation de l'extension, ils sont invisibles.

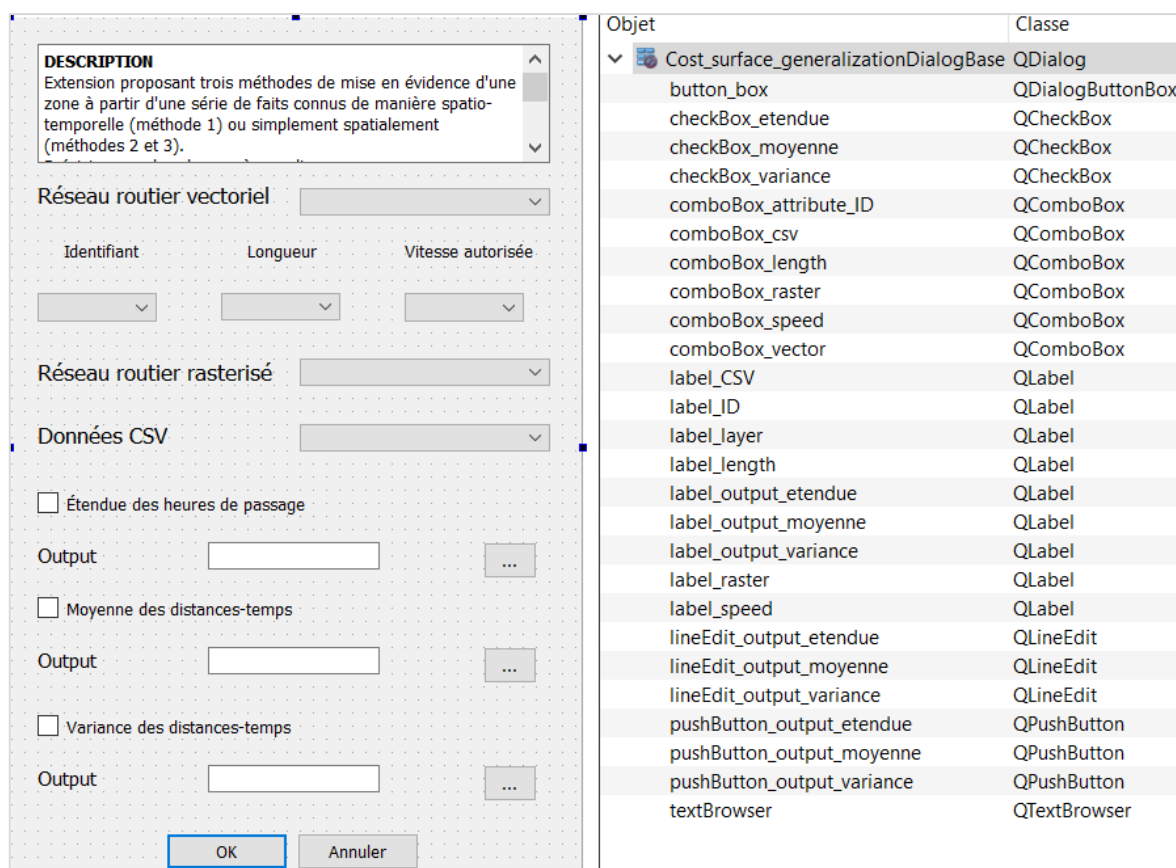


Figure 53 : interface graphique finale de l’extension logicielle (gauche) et liste des noms modifiés des objets et leur classe d’appartenance (droite)

8.4 Modification du code source de l’extension

8.4.1 Structure du fichier

La modification du code source de l’extension logicielle peut désormais être opérée. Celle-ci nécessite dans un premier temps de comprendre la structure du fichier créé automatiquement, pour rappel, par l’extension *plugin builder* de QGIS. Ce fichier Python contient, outre notamment les nombreux imports de modules de Qt Creator, la définition d’une classe. Cette classe porte le nom de l’extension logicielle (i.e. « Cost_surface_generalization »), une instance (autrement dit un objet) de celle-ci étant précisément une extension logicielle. Plusieurs méthodes (ainsi qu’un constructeur) sont définies à l’intérieur de cette classe, méthodes qui ont pour but de pouvoir s’appliquer aux objets, c’est-à-dire à l’extension. La compréhension du rôle de ce constructeur et de ces méthodes est essentielle pour connaître l’emplacement du code où les modifications doivent être réalisées. Ce constructeur et ces méthodes sont les suivants :

- `__init__` : constructeur de la classe. Il est par définition appelé à chaque instanciation de cette classe, et par conséquent avant toute autre méthode. Il permet notamment de passer en argument un objet interface de QGIS (classe ‘**QgsInterface**’);
- `tr` : méthode permettant de transformer de l’information textuelle (format *string*) en un objet *string* de Qt Creator (classe ‘**QString**’);

- **add_action** : méthode qui permet d'ajouter une icône relative à l'extension au menu de QGIS et de définir une méthode à appeler lorsque l'icône est sélectionnée ;
- **initGui** : méthode faisant appel à **add_action**. C'est elle qui va spécifier l'icône à utiliser par **add_action** et qui va donc lancer l'affichage de l'icône de l'extension dans le menu QGIS. Elle précise également la méthode devant être appelée lors du clic sur l'icône du menu QGIS, en l'occurrence la méthode **run** ci-dessous ;
- **unload** : méthode qui permet d'enlever l'interface graphique de l'extension, de l'interface graphique de QGIS ;
- **run** : méthode appelée lors du clic de l'utilisateur sur l'icône associée à l'extension logicielle. Elle permet l'affichage de l'interface graphique de l'extension dans l'interface graphique de QGIS, et définit un emplacement où est exécuté du code lorsque le bouton « ok » est déclenché. C'est par conséquent à cet endroit de cette méthode qu'est lancé le processus.

8.4.2 Liaisons avec l'interface graphique

La modification de la méthode **run** commence par la création des fonctionnalités de certains composants de l'interface graphique de l'extension. Ces fonctionnalités sont de quatre types : le remplissage des listes déroulantes, le remplissage des listes déroulantes des trois attributs en fonction de celle de la couche vectorielle, la visibilité/invisibilité des composants d'export de fichiers en fonction de l'état des cases à cocher et les exports de fichiers.

Le remplissage des listes déroulantes nécessite d'abord de recenser les couches importées dans le projet courant en fonction de certaines caractéristiques, afin de ne proposer que les types de couches qui correspondent à la liste. Par exemple, la liste déroulante des couches *raster* ne proposera que ce type de couches déjà importées dans le projet. Ce procédé est repris à la figure 54 ci-dessous, et débute par la création de trois listes qui contiendront les noms des couches vectorielles linéaires (*network_list*), des couches *raster* (*raster_list*) et des couches vectorielles non-spatiales ou spatiales ponctuelles (*points_list*). Cette dernière liste est nécessaire pour mettre en évidence les couches créées via l'import d'un fichier au format .csv. En effet, toute couche importée dans QGIS devient un objet de la classe '**QgsMapLayer**', et l'import d'un tel fichier sera un objet de type vecteur, qu'il soit importé avec ou sans géométrie. Ensuite, les couches du projet courant sont affectées à la variable '*layers*' et l'incrément '*l*' parcourt toutes les couches. Plusieurs différenciations sont alors effectuées pour classer les couches dans les listes, la première étant la différenciation sur base du type de couche (vecteur ou *raster*). Il n'y a aucune ambiguïté dans le cas du *raster*, en revanche pour le type vectoriel le caractère spatial ou non de la couche est testé afin de mettre en évidence les fichiers .csv importés sans géométrie. Enfin, les couches vectorielles spatiales sont différenciées en fonction du type de géométrie, afin de distinguer les couches ponctuelles correspondant aux crimes (et donc provenant du fichier .csv, valeur renvoyée par la méthode **geometryType** égale à 0) des couches linéaires liées au réseau routier (valeur renvoyée par cette même méthode égale à 1). Les listes sont alors ajoutées aux trois listes déroulantes.

```

networks_list = []
raster_list = []
points_list = []
layers = QgsProject.instance().mapLayers().values()
for l in layers:
    if l.type() == QgsMapLayer.VectorLayer:
        if l.isSpatial():
            if l.geometryType() == 0:
                points_list.append(l.name())
            if l.geometryType() == 1:
                networks_list.append(l.name())
        else:
            points_list.append(l.name())
    if l.type() == QgsMapLayer.RasterLayer:
        raster_list.append(l.name())

```

Figure 54 : différenciation des couches du projet QGIS courant et affectation aux listes correspondantes

À ce niveau, des erreurs peuvent intervenir dans le cas où aucune couche (vectorielle, *raster*, csv ou tout simplement aucune couche) n'a au préalable été importée dans le projet. Pour informer l'utilisateur de ces problèmes, quatre messages d'erreur (un par cas) ont été mis en place si une des situations intervient (cf. Figure 55). Pour obtenir ces erreurs, on vérifie pour chaque liste si sa longueur est nulle, si c'est le cas le message d'erreur est déclenché. Deux autres erreurs peuvent intervenir dans le cas où aucune donnée temporelle n'est présente ou bien l'est sous un format invalide (cf. Figure 56).

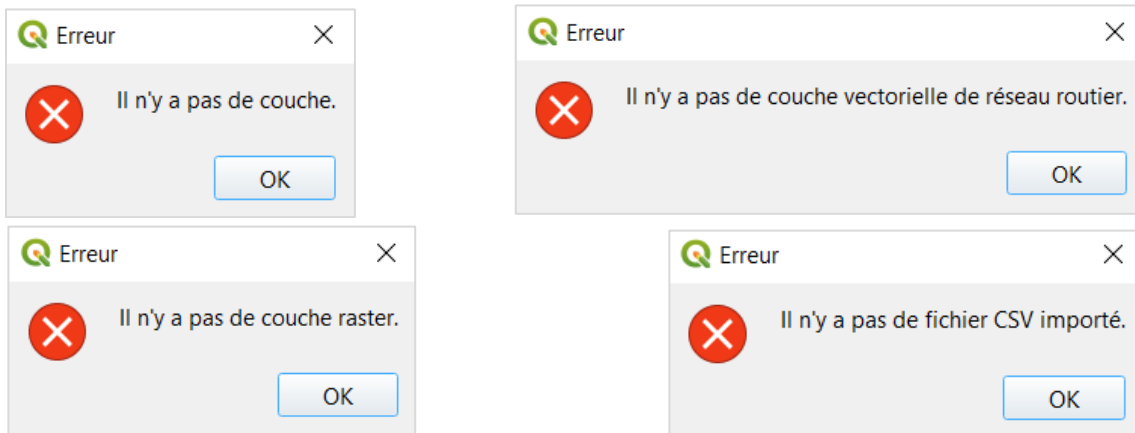


Figure 55 : quatre types de messages d'erreur apparaissant dans l'interface graphique de QGIS en fonction de la situation

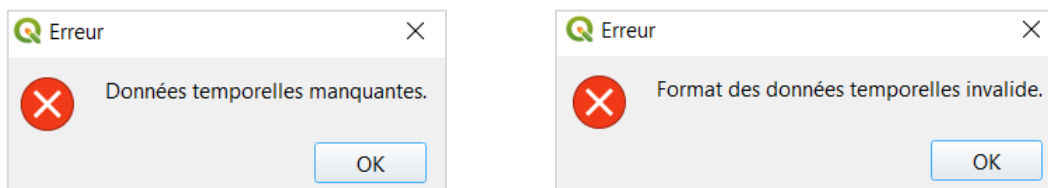


Figure 56 : messages d'erreur liés à la récupération des données temporelles

Les listes déroulantes des trois attributs (i.e. `'comboBox_attribute_ID'`, `'comboBox_length'` et `'comboBox_speed'`, cf. Figure 57) peuvent désormais être peuplées maintenant que la liste de la couche vectorielle n'est plus vide. Pour ce faire, chaque liste déroulante est d'abord vidée afin d'éviter que les attributs ne s'ajoutent à chaque exécution de l'extension. Ensuite, le nom de la couche actuellement affichée dans la liste déroulante de la couche vectorielle est affecté à une variable `'name_layer'`. On parcourt alors la liste des couches `'layers'` pour retrouver la couche `'l'` correspondant à `'name_layer'`, puis on récupère ses champs d'attribut `'fields'`. Ils sont par la suite ajoutés aux listes déroulantes grâce à la méthode `addItem`s. Une dernière considération est à prendre en compte, à savoir la modification des trois listes déroulantes en fonction de celle de la couche vectorielle à laquelle elles sont liées. Ces modifications se font via trois méthodes créées (`layer_changed_ID`, `layer_changed_speed` et `layer_changed_length`) et appelées via les méthodes `IndexChanged` et `connect` dont l'objectif est d'appeler ces méthodes lorsque l'index de la liste déroulante de la couche vectorielle `'comboBox_vector'` change. Ces trois méthodes suivent exactement la même structure que ce qui vient d'être expliqué et ne sont donc pas présentées ici, mais il faut noter qu'elles ont été implémentées au même niveau que les autres méthodes présentées au 8.4.1, c'est-à-dire en dehors de la méthode `run`.

```

self.dlg.comboBox_attribute_ID.clear()
self.dlg.comboBox_length.clear()
self.dlg.comboBox_speed.clear()
name_layer = self.dlg.comboBox_vector.currentText()
for l in layers:
    if l.name() == name_layer:
        pr = l.dataProvider()
        fields = pr.fields()
        self.dlg.comboBox_attribute_ID.addItem(field.name() for field in fields)
        self.dlg.comboBox_speed.addItem(field.name() for field in fields)
        self.dlg.comboBox_length.addItem(field.name() for field in fields)

self.dlg.comboBox_vector.currentIndexChanged.connect(self.layer_changed_ID)
self.dlg.comboBox_vector.currentIndexChanged.connect(self.layer_changed_speed)
self.dlg.comboBox_vector.currentIndexChanged.connect(self.layer_changed_length)

```

Figure 57 : remplissage des listes déroulantes des attributs à l'appel de l'extension logicielle

La troisième fonctionnalité est donc l'affichage ou non des composants de l'export de fichier en fonction de l'état des cases à cocher. Par définition, à l'initialisation de l'extension, ces neuf composants (trois par méthode) sont visibles alors qu'aucune case n'est cochée. Pour éviter d'afficher les composants alors que la case n'est pas cochée, celles-ci sont évaluées à chaque lancement de l'extension et les composants sont cachés (méthode `hide`) si les cases ne sont pas cochées. Ensuite, l'idée est qu'à chaque fois que l'utilisateur clique sur une des cases à cocher, une méthode est appelée afin d'afficher / d'enlever les trois composants associés à cette case. Un seul exemple est présenté ci-dessous étant donné que les trois méthodes sont quasiment identiques. L'appel de cette méthode se fait au moyen d'une méthode `stateChanged` (cf. Figure 58) qui appelle la méthode `in_visible_etendue` dès que l'état de la boîte à cocher est modifié (via un clic de l'utilisateur).


```
self.dlg.checkBox_etendue.stateChanged.connect(self.in_visible_etendue)
```

Figure 58 : appel de la méthode d'affichage / de suppression de l'affichage des composants de l'export de fichier à chaque changement d'état de la case à cocher

Cette méthode *in_visible_etendue* est basique (cf. Figure 59). Elle commence par vérifier l'état de la case à cocher via la méthode *isChecked*, qui renvoie un booléen. Ensuite, en fonction du cas, les éléments sont montrés via la méthode *show* ou bien cachés avec *hide*.

```
def in_visible_etendue(self):
    if self.dlg.checkBox_etendue.isChecked():
        self.dlg.label_output_etendue.show()
        self.dlg.lineEdit_output_etendue.show()
        self.dlg.pushButton_output_etendue.show()
    if not self.dlg.checkBox_etendue.isChecked():
        self.dlg.label_output_etendue.hide()
        self.dlg.lineEdit_output_etendue.hide()
        self.dlg.pushButton_output_etendue.hide()
```

Figure 59 : méthode d'affichage / de suppression de l'affichage des composants liés à la case à cocher de la première option (étendue)

Le quatrième et dernier type de fonctionnalité des composants de l'interface graphique de l'extension est précisément les trois exports de fichiers. Ces exports sont réalisés via trois méthodes mais une seule est présentée ici, *select_output_file_etendue* (cf. Figure 60) appelée selon une même structure que pour les listes déroulantes et les cases à cocher, mais avec la méthode *clicked* au lieu d'*IndexChanged* et *stateChanged*, signifiant qu'elle est déclenchée lorsque l'utilisateur clique sur le bouton pour aller vers le répertoire. La méthode *select_output_file_etendue* est alors exécutée et permet l'ouverture d'une nouvelle fenêtre de dialogue (classe '*QFileDialog*') et la sélection d'un chemin d'accès pour l'export du fichier '*output_file*' (méthode *getSaveFileName*) qui sera inscrit dans l'éditeur de texte '*lineEdit_output*'. Le format du fichier est, par ailleurs, directement défini en tant que format .tif.

```
def select_output_file_etendue(self):
    output_file_etendue, _filter = QFileDialog.getSaveFileName(
        self.dlg, "Select output file ", "", '*.tif')
    self.dlg.lineEdit_output_etendue.setText(output_file_etendue)
```

Figure 60 : méthode d'export de fichier

Les composants de l'interface sont maintenant fonctionnels, et il faut noter que l'appel des méthodes est réalisé avant l'emplacement où est exécuté le code lorsque le bouton « ok » est déclenché. Deux dernières étapes sont réalisées avant la transposition du processus : la récupération des informations fournies par l'utilisateur (y compris l'état des cases à cocher), et l'affichage de messages d'erreur. Ces derniers doivent apparaître d'abord si aucune méthode parmi les trois proposées n'a été sélectionnée par l'utilisateur ; ensuite si aucun

chemin d'accès n'a été précisé alors que la méthode a été cochée. Deux exemples de ces messages d'erreur sont repris à la figure 61 ci-dessous.

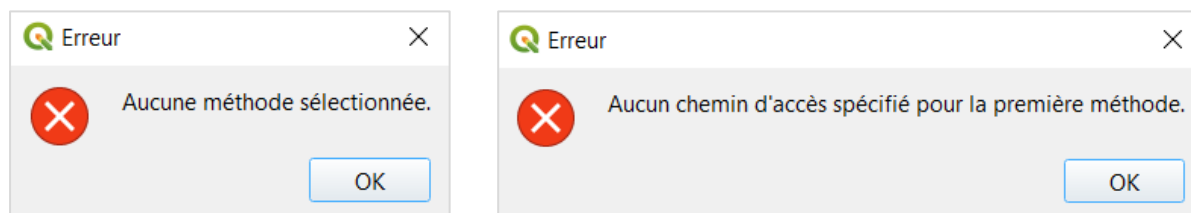


Figure 61 : exemples de messages d'erreur liés aux cases à cocher et aux exports de fichiers

8.4.3 Transposition du processus

Le processus développé au chapitre 6 peut désormais être transposé dans l'extension logicielle. Mis à part les transpositions théoriques mentionnées au 8.2, cette transposition est réalisée via deux changements majeurs : d'une part la récupération des données nécessaires au processus et d'une manière plus générale la transposition des objets GDAL aux objets QGIS, d'autre part la modification des fonctions.

Ce premier changement majeur provient du fait que toutes les couches fournies à l'extension sont désormais des objets QGIS, là où des objets GDAL étaient auparavant utilisés pour la récupération des données nécessaires au processus. Cette manière de récupérer les informations propres à ces couches, qu'elles soient vecteur ou *raster*, diffère de GDAL et la syntaxe est par conséquent à ajuster. Pour donner un exemple, la manière de lire un *raster* en tant que matrice, auparavant effectué via la méthode **ReadAsArray** nécessite désormais de passer par un autre objet QGIS nommé *block* sur lequel les opérations sont effectuées. D'une manière plus générale, tous les objets GDAL créés et/ou manipulés auparavant, tels que les points des crimes et les segments routiers à travers lesquels on itère pour trouver l'intersection (cf. 6.2), sont également des objets QGIS. Cela signifie une nouvelle fois que les méthodes appliquées à ceux-ci sont propres à la syntaxe QGIS (manière de créer un buffer, de récupérer les coordonnées, de calculer une intersection...) et doivent être ajustées.

Le second changement est la modification des fonctions du processus. Pour rappel, l'extension logicielle est considérée en tant qu'objet (cf. 8.4.1). Les fonctions développées dans le processus doivent donc ici être appelées par cet objet, elles deviennent donc par définition des méthodes. Dans ce contexte, la première modification à apporter à ces nouvelles méthodes est de leur rajouter le mot *self* comme tout premier argument. Cet argument est une règle de la définition des méthodes (et donc de la programmation orientée-objet) écrites en Python : le premier argument est une référence à l'objet lui-même et par convention il est nommé *self*. De la même manière, pour accéder aux méthodes existantes, un « *self.* » doit précéder la méthode étant donné que c'est, encore une fois, une méthode appliquée à l'objet référencé par *self*. Un exemple de cette transposition des fonctions en méthodes ainsi que leur appel dans le code est reprise à la figure 62.

```
def proj_to_segment(self, x1, y1, x2, y2, x_point, y_point):
    scalar_product = (x_point - x1) * (x2 - x1) + (y_point - y1) * (y2 - y1)
    norm = self.dist_eucl(x1, y1, x2, y2)
    ratio = scalar_product / pow(norm, 2)
```

Figure 62 : transformation des fonctions en méthodes ainsi que leur appel avec la référence « self », exemple avec la méthode *proj_to_segment*

Un dernier changement s'applique à certaines des fonctions devenues méthodes, à savoir l'ajout d'arguments. En travaillant avec les fonctions, les variables peuvent être définies en dehors de celles-ci puis être appelées à l'intérieur (variables globales). Les méthodes ne le permettent pas, ce qui signifie que les références aux variables doivent être effectuées via les arguments. Ces ajouts ne concernent cependant que les méthodes utilisant les variables contenant les couches vectorielle et *raster*.

Enfin, il est à noter que deux fonctionnalités ont été ajoutées à l'extension afin d'informer l'utilisateur sur l'état du processus en cours, il s'agit d'abord d'un onglet informant que le processus est en cours (cf. Figure 63 ci-dessous). Ensuite, à la fin du processus, un message apparaît afin d'informer l'utilisateur que le processus s'est bien déroulé et afficher le chemin d'accès vers le(s) fichier(s) créé(s).

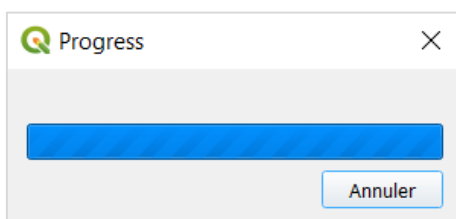


Figure 63 : message d'information du processus en cours

Au final, la validation de l'extension logicielle est réalisée par comparaison des *rasters* obtenus hors de l'extension et en résultat de celle-ci, en considérant des données identiques. Notons que les *rasters* obtenus à la fin du ou des processus choisi(s) sont automatiquement affichés dans le projet courant sur QGIS.

8.5 Efficacité de l'extension logicielle

Outre le fait que l'extension logicielle soit fonctionnelle, son efficacité se reflète également à travers le temps nécessaire pour la réalisation des processus en fonction du nombre de crimes considérés. Étant donné que trois processus ont été implémentés, il est également intéressant de voir l'efficacité des différents processus entre eux. Pour ce faire, trois catégories ont été définies : la première considère la réalisation des trois méthodes simultanément, la deuxième uniquement la réalisation du processus avec la méthode de l'étendue et la troisième uniquement celle de la variance. À noter que la méthode avec prise en compte de la moyenne des distances-temps est similaire à la variance.

Pour mesurer l'efficacité de ces processus, le nombre de crimes a été divisé en six paliers : 1, 4, 8, 12, 16 et 20 sources. Le *raster* sur lequel porte cette analyse comporte 3105 rangées ainsi que 3370 colonnes, ce qui au vu de la résolution spatiale (10 m) constitue une zone de

plus de 31 km sur 33 km. La figure 64 présente les résultats de ces temps. Il en ressort que l'efficacité des deux types de méthode (i.e. étendue et moyenne – variance) est quasiment identique si on considère jusqu'à 8 crimes, au-delà de ce seuil la méthode de l'étendue apparaît légèrement plus efficace d'une trentaine de secondes. Plus globalement, les temps des processus en fonction du nombre de sources augmentent de manière relativement linéaire et la tendance est comparable entre les différents cas considérés. Cependant, il est à noter que plus le nombre de crimes est important, plus les différences de temps entre les trois catégories sont conséquentes. Également, la prise en compte d'une unique source permet de mettre en évidence le temps minimum requis pour chaque processus, et on voit que l'augmentation du nombre de sources à partir de ce point n'a pas un impact considérable sur le temps de réalisation du processus (pentes faibles). Enfin, ce graphique montre que cela a du sens de rajouter les deux méthodes de la moyenne et de la variance sur les distances-temps, car elles utilisent en grande partie ce qui a déjà été réalisé pour l'étendue, et donc n'ajoute que peu de temps au processus.

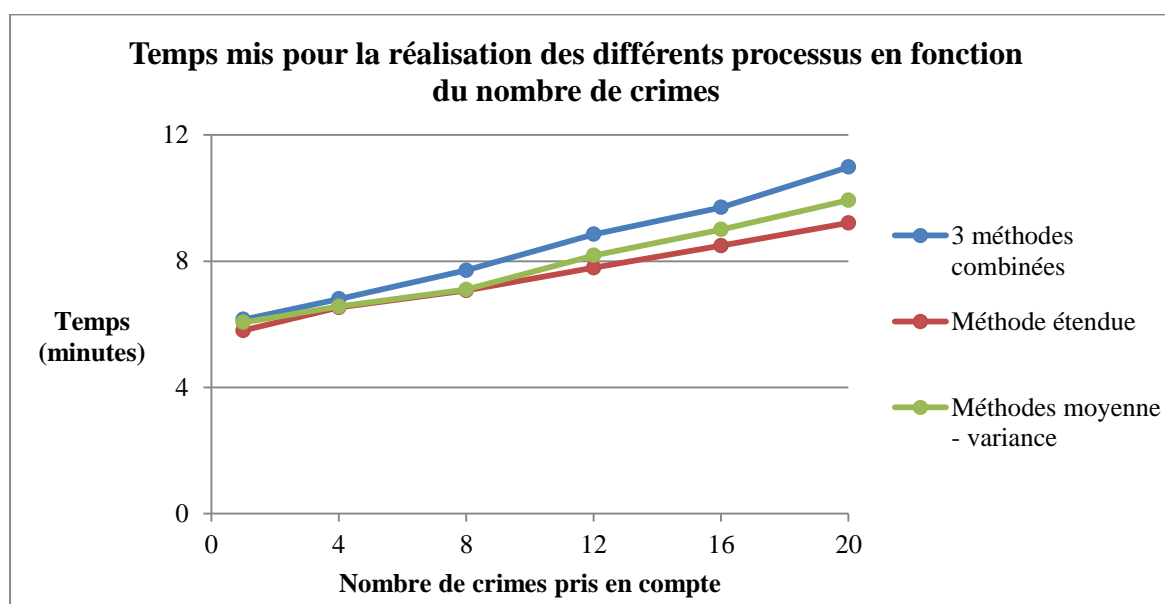


Figure 64 : temps mis pour la réalisation des différents processus en fonction du nombre de crimes pris en compte

8.6 Accès à l'extension logicielle

Le fichier compressé contenant l'extension logicielle est disponible sur MatheO⁵, associé à ce mémoire. Pour l'utiliser, il faut télécharger ce fichier et le décompresser dans le répertoire QGIS 3 propre aux extensions, dont le chemin d'accès est de la forme suivante : « C:\Users\your_name\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins ». Ensuite, le fichier batch placé dans le dossier (et par ailleurs disponible à l'annexe 9) doit être exécuté afin de mettre en place les liaisons entre Qt Creator et Python (et donc QGIS). L'extension peut alors être chargée et testée dans QGIS 3.

⁵ <https://matheo.uliege.be/>

9. Conclusions et perspectives

9.1 Conclusions

L'objectif de ce travail était de construire une extension logicielle implémentant un processus énoncé dans le cadre du profilage géographique, à savoir la délimitation d'une zone d'ancrage au départ d'une série de crimes connectés connus de manière spatio-temporelle. Ce processus se base notamment sur le postulat d'une heure de départ constante depuis le point d'ancrage du criminel vers les sites de crime. La zone d'ancrage est obtenue via le parcours d'un graphe routier par temps régressif depuis les heures auxquelles les crimes ont été commis. Ce parcours de graphe est en pratique réalisé par propagation d'un coût temporel de pixel en pixel, formant au final une surface de coût par crime. La combinaison de ces surfaces de coût permet alors de mettre en évidence les pixels qui minimisent la différence entre l'heure de passage maximale et minimale (autrement dit, l'étendue des heures de passage), correspondant à la zone d'ancrage recherchée. Ce mémoire devait, en particulier, s'attacher à étudier les erreurs affectant le résultat final et tenter de les solutionner.

L'hypothèse posée lors de ce mémoire est que la levée de ces erreurs, notamment à travers la modification de la rasterisation, permet d'améliorer le processus de délimitation de la zone d'ancrage via le parcours d'un graphe par temps régressif en mode maillé. Pour lever ces diverses erreurs, plusieurs pistes de solution ont été présentées.

Il ressort de ce travail que cette extension logicielle, implémentant le parcours d'un graphe par temps régressif, a bien été implémentée et est fonctionnelle. Pour ce faire, deux méthodes ont été développées : celle utilisée dans le cadre du profilage géographique, qui prend en compte l'étendue des heures de passage, et une seconde méthode qui considère la variance de ces mêmes heures. Cette seconde méthode a été proposée dans le but de déterminer s'il était possible d'améliorer la mise en évidence du point d'ancrage par rapport à la solution initiale de l'étendue, et pour ce faire les deux méthodes ont été comparées. En présence d'une répartition homogène des sites de crime autour du point d'ancrage et en prenant en compte les temps exacts de ceux-ci, les résultats offerts par ces deux méthodes sont similaires : le point minimisant respectivement l'étendue et la variance correspond bien au point d'ancrage. Pour tenter de les différencier, les méthodes ont été testées en fonction d'une situation plus proche de la réalité, représentée par une mauvaise configuration géométrique des sites de crime autour du point d'ancrage et des temps approximatifs. Dans cette situation, la méthode par prise en compte de l'étendue des heures de passage s'avère être la plus efficace, étant donné qu'elle est plus proche de la solution et est plus discriminante que la variance. Cela peut s'expliquer par le fait que l'étendue étant plus affectée par les valeurs extrêmes, cette particularité l'aide à mieux discriminer le point d'ancrage de ses alentours. C'est par conséquent cette méthode de l'étendue qui a été insérée dans l'extension.

Deux autres méthodes de délimitation d'une zone d'ancrage développées dans le cadre du profilage géographique ont également été implémentées dans cette extension logicielle. Celles-ci reposent sur deux autres postulats que celui de la méthode de l'étendue des heures de passage (heure de départ constante) : la minimisation de la moyenne des temps de déplacement vers les différents crimes (ou distance-temps) et la minimisation de la variance

de ces mêmes distance-temps. Ainsi, ces ajouts permettent d'élargir les fonctionnalités offertes par cette extension en prenant en compte d'autres solutions mentionnées dans le cadre du profilage géographique.

Le parcours de ce graphe routier en mode maillé souffre de plusieurs problèmes présentés dans ce mémoire, qu'ils soient inhérents à ce mode ou non. Certaines solutions ont été mises en place pour y remédier, telles que le rattachement des sites de crimes sur le réseau routier et la modification du résultat de la rasterisation par amélioration de la surface de friction, en normalisant le temps de parcours total d'un tronçon par le nombre de pixels qui le constituent. D'autres, au contraire, n'ont pu être réalisées pour diverses raisons. L'adaptation du temps de parcours à la date et à l'heure n'a par exemple pu être prise en compte étant donné l'absence de données utilisables. Dans le cas de la prise en compte de l'aspect non planaire des graphes routiers et des sens de circulation, deux méthodes de résolution assez similaires ont été proposées. Celles-ci reposent sur l'identification, dans une liste annexe et liée au *raster* de la surface de friction améliorée, de tous les mouvements qui ne sont pas permis selon les sens de circulation et les ponts et tunnels en présence.

Un problème commun aux deux méthodes s'est toutefois présenté : l'arrêt de la propagation du coût à tort à cause du pixel situé aux intersections. Les méthodes de résolution ont alors été améliorées afin d'éviter ce cas de figure et pour ce faire, les pixels voisins du pixel responsable de l'arrêt de la propagation sont considérés. L'objectif est de déterminer si un de ces voisins peut accueillir le déplacement, et donc de mettre en évidence un déplacement qui est empêché à tort. La liste liée au *raster* comporte alors les coordonnées-image du pixel au départ du mouvement, celles du pixel bloquant à tort la propagation du coût, et enfin celles du pixel voisin de ce dernier qui autorise la propagation depuis le pixel de départ.

En pratique, ces méthodes ont bien été implémentées, mais il a été montré que leur efficacité est fortement limitée par le résultat de la rasterisation du réseau routier sous certaines configurations. Ces limites ont pour conséquence que la propagation est une nouvelle fois arrêtée à tort à ces endroits, et impliquent donc que tous les pixels du *raster* ne peuvent présenter une valeur de coût cumulé. Ces méthodes n'ont donc, pour cette raison, pas été ajoutées à l'algorithme de construction de la surface de coût. Néanmoins, ce travail a montré qu'il est possible de récupérer un tel algorithme et de le modifier en « local » au lieu d'utiliser la librairie équivalente, ce qui pourrait être utile dans de futurs travaux.

Enfin, l'efficacité de l'extension logicielle a été testée à travers sa vitesse d'exécution. Pour ce faire, ces temps d'exécution ont été notés et comparés sur un même *raster* en considérant 1, 4, 8, 12, 16 et 20 crimes, et ce selon les méthodes choisies. En effet, trois catégories de combinaison de méthodes ont été considérées : uniquement celle de l'étendue, uniquement celle de la variance (la moyenne est similaire) et les trois méthodes simultanément. Il ressort d'abord que l'efficacité des méthodes de l'étendue et de la moyenne – variance sont quasiment identiques si on considère jusqu'à 8 crimes. Au-delà de ce seuil, la méthode de l'étendue apparaît légèrement plus efficace. Ensuite, l'augmentation des temps de processus en fonction du nombre de crimes n'est pas substantielle. Si on prend l'exemple de la méthode de l'étendue, le temps de réalisation du processus passe de 5,8 minutes pour une seule source

à 9,21 minutes si on en considère 20. Enfin, l'ajout des deux nouveaux processus susmentionnés est justifié par le fait que le temps supplémentaire lié à l'exécution non plus d'une (i.e. l'étendue) mais des trois méthodes ne constitue pas une augmentation considérable.

9.2 Perspectives

Plusieurs perspectives découlent de ce travail. La perspective principale est l'amélioration du parcours de ce graphe du réseau routier en mode maillé via la résolution des problèmes qui n'ont pu être solutionnés dans le cadre de ce mémoire. Il s'agit d'une part de l'adaptation du temps de parcours à la date et à l'heure grâce à l'utilisation des *API* des applications de calculs d'itinéraires, d'autre part de la prise en compte de l'aspect non planaire des graphes routiers ainsi que des sens de circulation. Dans le cadre de ce second problème, les méthodes proposées ont été implémentées ; par conséquent, de futures recherches pourraient se pencher soit sur la manière d'améliorer la rasterisation afin d'éviter les limitations qu'elle introduit sur ces méthodes, soit d'améliorer directement les méthodes. De plus, cette prise en compte ne peut se traduire qu'à travers une modification de l'algorithme de surface de coût afin d'y ajouter la méthode proposée, et ce travail a montré qu'il était possible de récupérer un tel algorithme et de le rendre modifiable.

Une deuxième perspective est d'utiliser les données d'occupation du sol afin de définir une friction hors réseau routier, que l'on peut associer à un déplacement à pied. La prise en compte de cette friction est d'autant plus intéressante que la distance entre le site de crime et le réseau routier est importante.

L'amélioration du résultat final par prise en compte de l'analyse multicritère constitue une troisième perspective. En effet, dans le cas de l'obtention d'une vaste zone d'ancrage, par conséquent peu utilisable, l'ajout de critères supplémentaires à celui du résultat obtenu avec le processus implémenté dans le cadre de ce mémoire peut aider à affiner sa délimitation.

Enfin, une quatrième et dernière perspective est d'étendre l'utilisation de cette extension logicielle à d'autres applications que le profilage géographique. Les applications ciblées sont celles utilisant le réseau routier et qui cherchent à déterminer la meilleure localisation possible. La détermination d'une localisation « a priori optimale » pour la mise en place d'un centre de distribution, en minimisant les différences de temps pour atteindre les commerces de détail ou de gros qu'il fournit, peut par exemple être considérée. Le terme « a priori optimale » est ici utilisé car le choix de localisation d'un centre de distribution est en réalité plus complexe ; cependant, et comme mentionné ci-dessus, une analyse multicritère peut être développée en prenant en compte le résultat des processus implémentés dans ce travail, afin d'affiner le choix de localisation. Un second cas considéré peut résider dans le choix de localisation pour la mise en place de bâtiments institutionnels tels qu'une caserne de pompiers à l'échelle communale qui minimiserait les différences d'accessibilité de tous les villages communaux.

10. Références

- Bastin, F. (2014). *Mathématiques générales, partim A*. Cours de 1^{er} bachelier en sciences géographiques, orientation générale, Université de Liège, 195 p.
- Behnel, S., Bradshaw, R., Seljebotn, D.S., Ewing, G., Stein, W., Gellner, G. *et al.* (2019a). *Cython – an overview – Cython 3.0a0 documentation*. Cython.org. <http://docs.cython.org/en/latest/src/quickstart/overview.html> Consulté le 8 juin 2019.
- Behnel, S., Bradshaw, R., Seljebotn, D.S., Ewing, G., Stein, W., Gellner, G. *et al.* (2019b). *Basic Tutorial – Cython 3.0a0 documentation*. Cython.org. https://cython.readthedocs.io/en/latest/src/tutorial/cython_tutorial.html Consulté le 8 juin 2019.
- Behnel, S., Bradshaw, R., Seljebotn, D.S., Ewing, G., Stein, W., Gellner, G. *et al.* (2019c). *Sources Files and Compilation – Cython 3.0a0 documentation*. Cython.org. https://cython.readthedocs.io/en/latest/src/userguide/source_files_and_compilation.html#compilation Consulté le 10 juin 2019.
- Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*. 16(1), 87-90.
DOI: <http://doi.org/10.1090/qam/102435>
- Bilasco, S.T., Filip, S., Cocean, P., Petrea, D., Vescan, I. & Fodorean, I. (2015). The Evaluation of Accessibility to Hospital Infrastructure at Regional Scale by Using GIS Space Analysis Models: the North-West Region, Romania. *Studia Universitatis Babeş-Bolyai: Geographia*, LX(1), 21-50.
- Bresenham, J.E. (1956). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1), 25-30.
<http://doi.org/10.1147/sj.41.0025>
- Canter, D. & Larkin, P. (1993). The environmental range of serial rapists. *Journal of Environmental Psychology*, 13(1), 63-69.
DOI: [http://doi.org/10.1016/S0272-4944\(05\)80215-4](http://doi.org/10.1016/S0272-4944(05)80215-4)
- Converse, P. (1949). New Laws of Retail Gravitation. *Journal of Marketing (pre – 1986)*, 14, 379.
- DEI-ISEP. (2007). *Chapter 7. Blending, Antialiasing, Fog, and Polygon Offset*. Instituto Superior de Engenharia do Porto – Departamento de Engenharia Informatica. http://www.dei.isep.ipp.pt/~matos/cg/docs/OpenGL_PG/ch07.html. Consulté le 15 avril 2019.
- Donnay, J.P. (2015). *Cartographie & SIG*. Cours de 2^{ème} bachelier en sciences géographiques, orientation générale, Université de Liège, chapitre 3, 74 p.
- Donnay, J.P. (2018a). *Cartographie mathématique*. Cours de 1^{er} master en sciences géographiques, orientation géomatique et géométrie à finalité, Université de Liège, chapitre 1, 24 p.
- Donnay, J.P. (2018b). *SIG*. Cours de 1^{er} master en sciences géographiques, orientation géomatique et géométrie à finalité, Université de Liège, chapitre 8, 46 p.
- Donnay, J.P., Ledent, P. (1995). Modelling of Accessibility Fields. In AKM Messe AG, Proceeding of the First Joint European Conference on Geographical Information. Den Haag, Pays-Bas, 1, 489 – 4994.

- Donnay, J.P., Trotta, M. & Kasprzyk, J.P. (2013). Méthodologie de recherche en cartographie criminelle. In *Police judiciaire*, 26 avril, 2013, Liège, Belgique. Orbi ULiège.
https://orbi.uliege.be/bitstream/2268/149543/1/MethodoCartoCriminelle_ULg.pdf
Consulté le 24 janvier 2019.
- Douglas, D.H. (1994). Least-cost Path in GIS Using an Accumulated Cost Surface and Slopelines. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 31(3), 37-51.
<https://doi.org/10.3138/D327-0323-2JUT-016M>
- Dijkstra, E.W. (1959). A Note on Two Problems in Connexion With Graphs. *Numerische Mathematik*, 1, 269-271.
<https://doi.org/10.1007/BF01386390>
- Eastman, J.R. (1989). Pushbroom algorithms for calculating distances in raster grids. In *Auto-carto 9: Ninth International Symposium on Computer-Assisted Cartography*, April 2 – 7, 1989, Baltimore, Maryland, USA. Auto-Carto, 288 – 297.
- Erickson, J., Daniel, C. & Payne, M. (2013). *Raster Layers – Python GDAL/OGR cookbook 1.0 documentation*. Github. https://pcjericks.github.io/py-gdalogr-cookbook/raster_layers.html#create-least-cost-path. Consulté le 18 avril 2019.
- ESRI. (2018a). *Distance de coût – Aide | ArcGIS Desktop*. ESRI. <http://pro.arcgis.com/fr/pro-app/tool-reference/spatial-analyst/cost-distance.htm>. Consulté le 6 février 2019.
- ESRI. (2018b). *What is ArcPy? ArcPy Get Started | ArcGIS Desktop*. ESRI. <https://pro.arcgis.com/fr/pro-app/arcpy/get-started/what-is-arcpy.htm>. Consulté le 15 juin 2019.
- ESRI. (2016a). *Fonctionnement des outils de distance de coût – Aide | ArcGIS Desktop*. ESRI. <http://desktop.arcgis.com/fr/arcmap/10.3/tools/spatial-analyst-toolbox/how-the-cost-distance-tools-work.htm>. Consulté le 6 février 2019.
- ESRI. (2016b). *Accumulation de flux – Aide | ArcGIS Desktop*. ESRI. <http://desktop.arcgis.com/fr/arcmap/10.3/tools/spatial-analyst-toolbox/flow-accumulation.htm>. Consulté le 14 avril 2019.
- ESRI. (2016c). *Direction de flux – Aide | ArcGIS Desktop*. ESRI. <http://desktop.arcgis.com/fr/arcmap/10.3/tools/spatial-analyst-toolbox/flow-direction.htm>. Consulté le 26 juillet 2019.
- Floyd, R.W. (1962). Algorithm 97 : Shortest path. *Communications with the ACM*, 5, 345.
DOI : [10.1145/367766.368168](https://doi.org/10.1145/367766.368168).
- Ford, L.R. (1956). *Network flow theory*. Santa Monica, Calif. : Rand Corp, Paper P-923, 17 p.
- GRASS. (2003). *GRASS GIS Manual: r.cost*. GRASS. <https://grass.osgeo.org/grass76/manuals/r.cost.html>. Consulté le 30 janvier 2019.
- GRASS. (2019). *GRASS GIS Manual: r.accumulate*. GRASS. <https://grass.osgeo.org/grass76/manuals/r.watershed.html>. Consulté le 14 avril 2019.

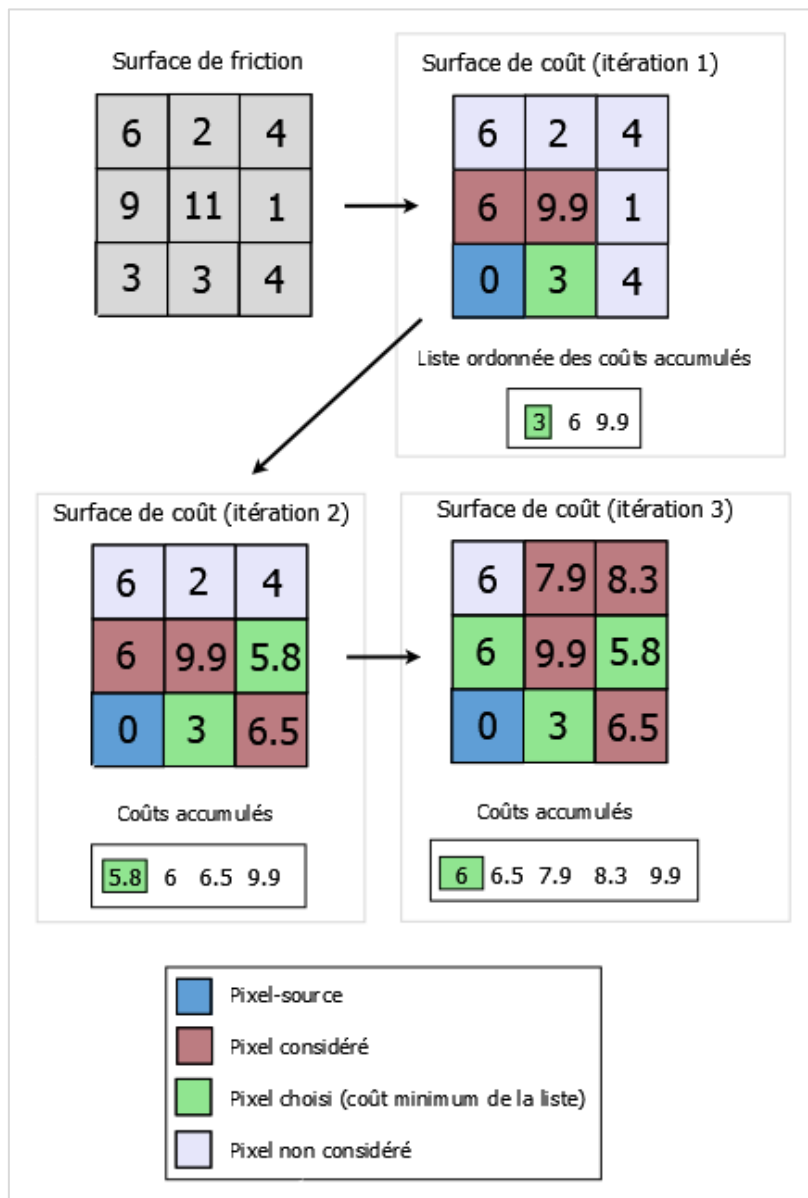
- Hart, P.E., Nilsson, N.J., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100 – 107.
DOI: [10.1109/TSSC.1968.300136](https://doi.org/10.1109/TSSC.1968.300136)
- Hoover, E.M. (1948). *The Location of Economic Activity*. New-York : McGraw-Hill Company, universallibrary, 336 p.
- Huff, D.L. (1963). A Probabilistic Analysis of Shopping Center Trade Areas. *Land Economics*, 39(1), 81-90.
DOI: [10.2307/3144521](https://doi.org/10.2307/3144521)
- Jenson, S.K. & Domingue, J.O. (1988). Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing*, 54(11), 1593 – 1600.
- MacKay, R. (1999). Geographing Profiling : A New Tool For Law Enforcement. *Police Chief*, 66(12), 51-59.
- Mérenne-Schoumaker, B. (2011). *La localisation des industries : enjeux et dynamiques*. France, Rennes : Presses Universitaires de Rennes, 263 p.
- NetworkX developers. (2019). *NetworkX – NetworkX*. NetworkX developers. <https://networkx.github.io/>. Consulté le 15 juin 2019.
- Rasterio. (2018). *Introduction – rasterio documentation*. Rasterio. <https://rasterio.readthedocs.io/en/stable/intro.html>. Consulté le 5 juin 2019.
- Reilly, W.J. (1931). *The law of retail gravitation*. New – York : W.J. Reilly, 75 p.
- Russel, R. (2013). *How does Google calculate your ETA ? – Quora*. Quora. <https://www.quora.com/Speed-Limits/How-does-Google-Maps-calculate-your-ETA>
Consulté le 13 janvier 2019.
- Ritsema van Eck, J.R. & de Jong, T. (2002). *Off the Road: From Data points to the Networks in GIS-Based Network Analysis*. In 21st Annual South African Transport Conference South Africa, 15 – 19 July 2002, South Africa. Unknown publisher.
- SAGA-GIS. (2004). *Module Accumulated Cost / SAGA-GIS Module Library Documentation (v2.2.6)*. SAGA-GIS.
http://www.saga-gis.org/saga_tool_doc/2.2.6/grid_analysis_0.html. Consulté le 30 janvier 2019.
- Samet, H. & Webber, R.E. (1985). Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics (TOG)*, 4(3), 182-222.
DOI : [10.1145/282957.282966](https://doi.org/10.1145/282957.282966)
- Shaffer, C.A. & Ursekar, M.T. (1992). Large Scale Editing and Vector to Raster Conversion Via Quadtree Spatial Indexing. In *proceedings: 5th International Symposium on Spatial Data Handling*, Aug. 3-7, Charleston, S.C., USA, 1992. Charleston : International Geographic Union (IGU) Commision on GIS. 725 p.
- Shimbel, A. (1955). Structure in communication nets. In *Proceedings of the Symposium on Information Networks*, New-York, 1954. Brooklyn : Polytechnic Press of the Polytechnic Institute of Brooklyn, 199 – 203.

- Snook, B., Zito, M., Bennell, C. & Taylor, P. (2005). On the Complexity and Accuracy of Geographic Profiling Strategies. *Journal of Quantitative Criminology*, 21(1), 1-26.
- StackExchange. (2014). *Graphics – Bresenham's line algorithm - Mathematica Stack Exchange*. Stack Exchange Inc.
<https://mathematica.stackexchange.com/questions/45753/bresenham-s-line-algorithm?noredirect=1&lq=1>. Consulté le 6 février 2019.
- van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T., the scikit-image contributors. (2014). Scikit – image : image processing in Python. *Peer J*, 453.
<https://doi.org/10.7717/peerj.453>
- The Igraph Core Team. (2015). *Python – igraph*. The Igraph Core Team.
<https://igraph.org/python/>. Consulté le 15 juin 2019.
- Trotta, M., Bidaine, B. & Donnay, J.P. (2011). Determining the Geographical Origin of a serial offender considering the temporal uncertainty of the Recorded Crime Data. *In GEOProcessing 2011: The Third International Conference on Advanced geographic Information Systems, Applications, and Services*, February 23 – 28, 2011, Gosier, Guadeloupe, France. Wilmington (DE): International Academy, Research, and Industry Association (IARIA) [CD ROM], 40-45.
- Trotta, M. (2012). New Hypotheses on serial offender's spatial behavior. *In H., Pundt & L., Bernard (Eds.), Proceedings 1st AGILE PhD School*, Aachen, Germany. Germany: Shaker Verlag, 1-9.
- Triola, M.M., Triola, M.F., Hunault, G. & Desdevises, Y. (2012). *Biostatistique pour les sciences de la vie et de la santé*. Montreuil : Pearson, 367 p.
- Vanraes, N., Cornélis, B. & Donnay, J.P. (1998). Decision Support for Improving Public Transport Network. *In Eindhoven University of Technology (Ed.), Proceedings of the 4th Design and Decision Support Systems in Architecture and Urban Planning*, Eindhoven, Pays-Bas. Eindhoven : Proceedings of the 4th Design and Decision Support Systems in Architecture and Urban Planning, 17.
- Warshall, S. (1962). A Theorem on Boolean Matrices. *Journal of the ACM*, 9, 11-12.
DOI : [10.1145/321105.321107](https://doi.org/10.1145/321105.321107).
- Weber, A. & Friedrich, C.J. (1929). Theory of the Location of Industries. *Economic geography*, 7(1), 106.

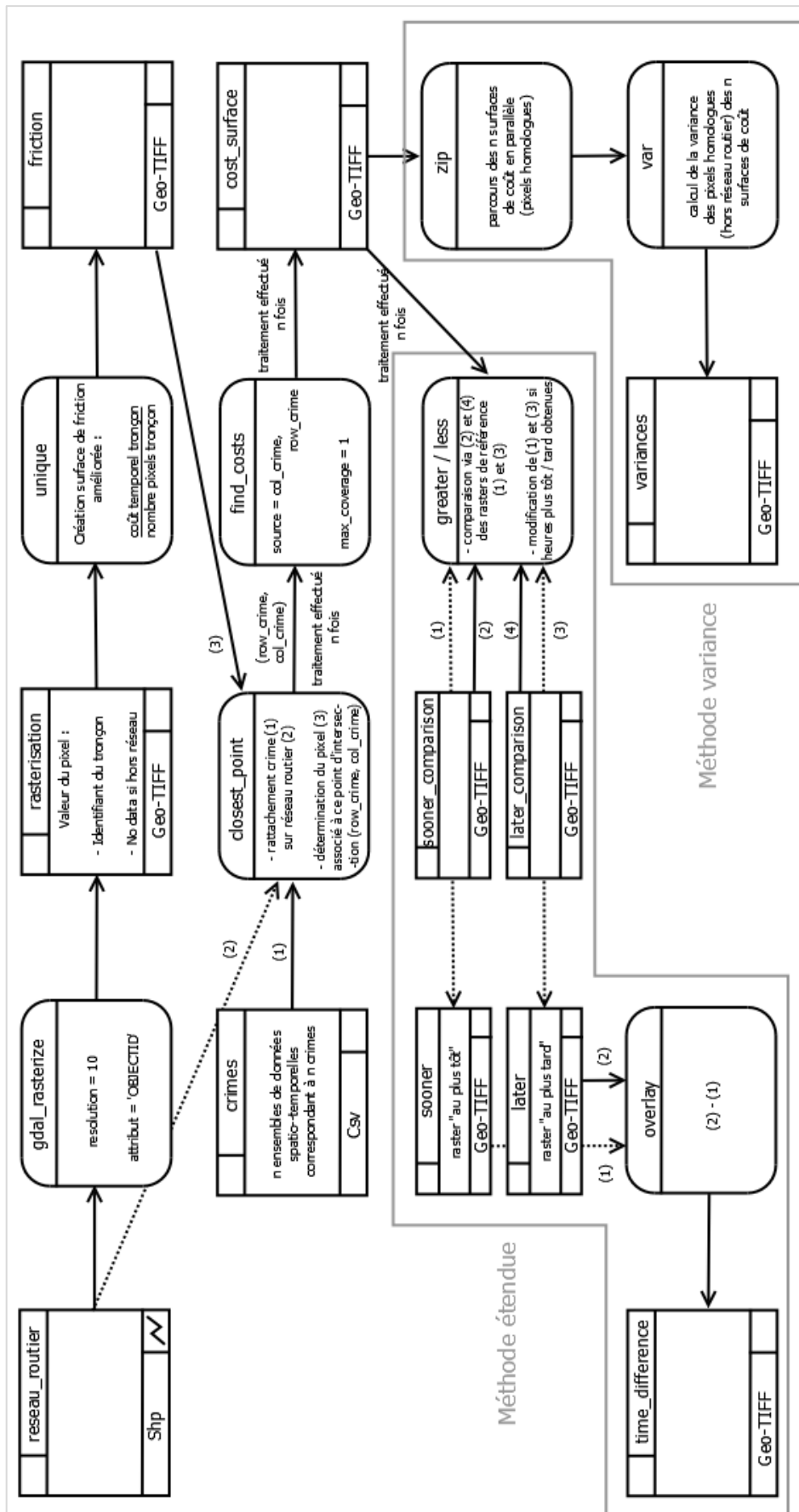
Liste des annexes

Annexe 1 : exemple de fonctionnement des trois premières itérations de l'algorithme de surface de coût dans le cas d'un raster 3x3	84
Annexe 2 : <i>workflow</i> du processus.....	85
Annexe 3 : code source de la fonction <i>closest_point</i>	86
Annexe 4 : code source de la fonction <i>proj_to_segment</i>	87
Annexe 5 : code source de la fonction <i>coord_to_pixel</i>	87
Annexe 6 : code source de la fonction <i>pixel_to_coord</i>	88
Annexe 7 : code source de l'algorithme de détermination des mouvements non permis.....	88
Annexe 8 : code source de l'algorithme amélioré de détermination des intersections comprenant un pont ou un tunnel.....	90
Annexe 9 : code source du fichier de lignes de commandes (batch) afin de mettre en place les liaisons entre Qt Creator et Python (et donc QGIS)	90

Annexe 1 : exemple de fonctionnement des trois premières itérations de l'algorithme de surface de coût dans le cas d'un raster 3x3



Annexe 2 : workflow du processus



Annexe 3 : code source de la fonction *closest_point*

```

def closest_point(lon, lat):
    # création du point au format GDAL
    point = ogr.Geometry(ogr.wkbPoint)
    # définition des coordonnées
    point.AddPoint(lon, lat)
    # initialisation du rayon du buffer
    buffer_radius = 0
    # intitialisation d'un booléen pour sortir de la boucle while
    intersection = False
    # tant qu'un tronçon routier n'a pas été intersecté, on continue le
    # processus
    while intersection == False:
        # augmentation de la taille du buffer (rayon)
        buffer_radius += 100
        # création du buffer autour du point (crime)
        buffer = point.Buffer(buffer_radius)
        # initialisation d'une distance de référence supérieure au buffer
        distance = buffer_radius + 1
        # remise à zéro de la lecture
        source_layer.ResetReading()
        # parcours des tronçons routier du réseau routier
        for feature in source_layer:
            # récupération de la géométrie du tronçon routier
            geometry = feature.GetGeometryRef().Clone()
            # test d'intersection entre le tronçon routier (feature) et
            # le buffer
            if (geometry.Intersects(buffer) == True):
                # s'il y a intersection d'au moins une feature, on pourra
                # sortir de
                # la boucle while
                intersection = True
                # si la distance minimale entre le point et ce tronçon
                # routier est inférieure à la distance minimale de
                # référence
                if (geometry.Distance(point) < distance):
                    # on modifie la distance avec sa nouvelle valeur
                    distance = geometry.Distance(point)
                    linestring_geom = geometry
        # définition d'une distance de référence très grande
        nearest_point_distance = 1000
        # récupération des coordonnées du crime
        X_point = point.GetX()
        Y_point = point.GetY()
        # boucle pour parcourir tous les segments (X1,Y1 et X2,Y2) du
        # tronçon routier (polyligne)
        for i in range(0, linestring_geom.GetPointCount()-1, 1):
            X1 = linestring_geom.GetPoint(i)[0]
            Y1 = linestring_geom.GetPoint(i)[1]
            X2 = linestring_geom.GetPoint(i+1)[0]
            Y2 = linestring_geom.GetPoint(i + 1)[1]
            # projection orthogonale pour trouver l'intersection
            x_intersect, y_intersect = \
                proj_to_segment(X1, Y1, X2, Y2, X_point, Y_point)
            # si distance trouvée est inférieure à celle de référence, on
            # affecte les coordonnées et la distance du nouveau point le plus
            # proche
            if dist_eucl(x_intersect, y_intersect, X_point, Y_point) < \
                nearest_point_distance:
                nearest_point_distance = dist_eucl(x_intersect, y_intersect,

```

```

                                X_point,Y_point)
        closest_pt_x = x_intersect
        closest_pt_y = y_intersect
    # appel de la fonction "coord_to_pixel" afin de renvoyer les
    coordonnées -
    # images correspondant à ce point d'intersection en coordonnées
    cartographiques
    return coord_to_pixel(closest_pt_x,closest_pt_y)

```

Annexe 4 : code source de la fonction *proj_to_segment*

```

def proj_to_segment(x1,y1,x2,y2,x_point,y_point):
    # calcul du produit scalaire entre les vecteurs formés par
    # x1,y1 et x2,y2 d'une part et d'autre part par x1,y1 et
    # x_point, y_point
    scalar_product = (x_point - x1) * (x2 - x1) + \
                    (y_point - y1) * (y2 - y1)
    norm = dist_eucl(x1, y1, x2, y2)
    # calcul de la projection orthogonale
    ratio = scalar_product / pow(norm, 2)
    # projection orthogonale entre 0 et 1 : intersecte le segment
    if 0 <= ratio <= 1:
        x_intersect = x1 + ratio * (x2 - x1)
        y_intersect = y1 + ratio * (y2 - y1)
    # projection orthogonale négative : point x1, y1 est le
    # point le plus proche
    elif ratio < 0:
        x_intersect = x1
        y_intersect = y1
    # projection orthogonale strictement supérieure à 1 : point
    # x2, y2 est le point le plus proche
    elif ratio > 1:
        x_intersect = x2
        y_intersect = y2

    return x_intersect,y_intersect

```

Annexe 5 : code source de la fonction *coord_to_pixel*

```

def coord_to_pixel(mx,my):
    # récupération des informations propres au raster
    (upper_left_x, x_size, x_rotation, upper_left_y, y_rotation, y_size) \
    = target_ds2.GetGeoTransform()
    # transformation des coordonnées vectorielles du crime, en
    # coordonnées - image (lignes - rangées)
    px = int((mx - upper_left_x) / x_size)
    py = int((my - upper_left_y) / y_size)
    # récupération de la bande du raster
    band = target_ds2.GetRasterBand(1)
    # lecture de cette bande sous forme d'un array numpy
    np_array = band.ReadAsArray()
    # test si le pixel obtenu ci-dessus fait bien partie du réseau
    # routier. Si ce n'est pas le cas (valeur de -99), on tente de
    # trouver le pixel voisin appartenant au réseau routier ET qui
    # minimise la distance euclidienne au réseau routier
    if np_array[py, px] == -99:
        classement = []
        # double boucles pour parcourir les pixels voisins en Y et en X

```



```

for i in range(py - 1, py + 2, 1):
    for j in range(px - 1, px + 2, 1):
        # si le pixel voisin est un pixel du réseau routier
        if (np_array[i, j] != -99):
            # on fait la transformation inverse pour obtenir ses
            # coordonnées Lambert afin de calculer la distance
            # qui le sépare du réseau routier
            mx2, my2 = pixel_to_coord(j, i, upper_left_x,
                                      upper_left_y, resolution,
                                      resolution)

            # Calcul de la distance euclidienne
            dist = dist_eucl(mx, my, mx2, my2)
            classement.append([dist, i, j])

# tri ordonné de la liste en fonction de la distance
classement.sort()
# récupération des coordonnées - image correspondant à la
# distance minimale (classement[0][0] - haut de la liste)
y = classement[0][1]
x = classement[0][2]

# si le pixel obtenu fait bien partie du réseau routier, on retourne
# les coordonnées - image directement
else:
    y = py
    x = px

# on retourne les coordonnées - image
return y, x

```

Annexe 6 : code source de la fonction *pixel_to_coord*

```

def pixel_to_coord(px, py, upper_left_x, upper_left_y, x_size, y_size):
    mx = px * x_size + upper_left_x
    my = py * y_size + upper_left_y
    return mx, my

```

Annexe 7 : code source de l'algorithme de détermination des mouvements non permis

```

"""Création d'un dictionnaire dont la clé est l'identifiant des tronçons
routiers. Ainsi, il est possible de lier toutes les informations voulues
à cette clé, ce qui est nettement plus simple afin de récupérer ces infos
portant sur le sens de circulation et le caractère ou non de pont et de
tunnel."""
att = {}
for feature in source_layer:
    att[str(feature.GetField("OBJECTID"))] = {
        'direction': str(feature.GetField("DIR_TRAVEL")),
        'REF': int(feature.GetField("REF_IN_ID")),
        'NREF': int(feature.GetField("NREF_IN_ID")),
        'BRIDGE': str(feature.GetField("BRIDGE")),
        'TUNNEL': str(feature.GetField("TUNNEL"))}

"""Ensuite, on parcourt tous les pixels de l'image via les incréments 'r'
et 'c'. Dès qu'un pixel routier est obtenu, on considère ses voisins qui
appartiennent eux aussi au réseau routier via les incréments i et j."""
for r in range(1, row-1, 1):
    for c in range(1, col-1, 1):
        if datas[r, c] != NoData_value:
            for i in range(r - 1, r + 2, 1):
                for j in range(c - 1, c + 2, 1):
                    id_courant = str(int(datas[r, c]))
                    id_voisin = str(int(datas[i, j]))
                    if id_voisin != id_courant and id_voisin != \

```

```

    str(int(NoData_value)):
    """Enfin, on teste les 8 cas possible induisant une
    impossibilité de se déplacer du pixel courant au
    pixel voisin. A chaque fois que les conditions
    d'un cas sont respectées, on agrandit la liste
    "restrictions" avec les coordonnées - images du
    pixel courant puis du pixel voisin."""
    # CAS 1
    if (att[id_courant]['direction'] != 'B' and
        att[id_voisin]['direction'] != 'B') and \
        (att[id_courant]['direction'] ==
         att[id_voisin]['direction']) and \
        (att[id_courant]['REF'] ==
         att[id_voisin]['REF']
         or att[id_courant]['NREF'] ==
         att[id_voisin]['NREF']):
        restrictions.append([c, r, j, i])
    # CAS 2
    if (att[id_courant]['direction'] != 'B' and
        att[id_voisin]['direction'] != 'B') and \
        (att[id_courant]['direction'] !=
         att[id_voisin]['direction']) and \
        (att[id_courant]['REF'] ==
         att[id_voisin]['NREF'] or
         att[id_courant]['NREF'] ==
         att[id_voisin]['REF']):
        restrictions.append([c, r, j, i])
    # CAS 3
    if att[id_courant]['direction'] == 'F' and \
        att[id_voisin]['direction'] == 'F' and \
        att[id_courant]['REF'] == \
        att[id_voisin]['NREF']:
        restrictions.append([c, r, j, i])
    # CAS 4
    if att[id_courant]['direction'] == 'T' and \
        att[id_voisin]['direction'] == 'T' and \
        att[id_courant]['NREF'] == \
        att[id_voisin]['REF']:
        restrictions.append([c, r, j, i])
    # CAS 5
    if att[id_courant]['direction'] == 'T' and \
        att[id_voisin]['direction'] == 'F' and \
        att[id_courant]['NREF'] == \
        att[id_voisin]['NREF']:
        restrictions.append([c, r, j, i])
    # CAS 6
    if att[id_courant]['direction'] == 'F' and \
        att[id_voisin]['direction'] == 'T' and \
        att[id_courant]['REF'] == \
        att[id_voisin]['REF']:
        restrictions.append([c, r, j, i])
    # CAS 7
    if att[id_courant]['direction'] == 'B' and \
        att[id_voisin]['direction'] == 'T' and \
        att[id_courant]['REF'] == \
        att[id_voisin]['REF']:
        restrictions.append([c, r, j, i])
    # CAS 8
    if att[id_courant]['direction'] == 'B' and \
        att[id_voisin]['direction'] == 'F' and \
        att[id_courant]['NREF'] == \

```

```
att[id_voisin]['NREF']:  
restrictions.append([c, r, j, i])
```

Annexe 8 : code source de l'algorithme amélioré de détermination des intersections comprenant un pont ou un tunnel

```
"""Cas unique à considérer ici."""  
if(att[id_courant]['BRIDGE'] != att[id_voisin]['BRIDGE'] or  
att[id_courant]['TUNNEL'] != att[id_voisin]['TUNNEL']) and (  
att[id_courant]['REF'] != att[id_voisin]['REF'] and  
att[id_courant]['REF'] != att[id_voisin]['NREF'] and  
att[id_courant]['NREF'] != att[id_voisin]['REF'] and  
att[id_courant]['NREF'] != att[id_voisin]['NREF']):  
    normal_issue = True  
    """Parcours des voisins du pixel empêchant la  
    propagation via les incréments m et n. On  
    vérifie alors si un de ces voisins fait partie  
    du même tronçon ou d'un tronçon contigus."""  
    for m in range(i - 1, i + 2, 1):  
        for n in range(j - 1, j + 2, 1):  
            id_voisin_voisin = str(int(datas[m,n]))  
  
            if (id_voisin_voisin != id_voisin and int(id_voisin_voisin) !=  
                NoData_value) and (r != m or c != n)  
                and (att[id_courant]['REF'] ==  
                    att[id_voisin_voisin]['REF']  
                    or att[id_courant]['REF'] ==  
                    att[id_voisin_voisin]['NREF'] or  
                    att[id_courant]['NREF'] ==  
                    att[id_voisin_voisin]['REF']  
                    or att[id_courant]['NREF'] ==  
                    att[id_voisin_voisin]['NREF']):  
                restrictions.append([c, r, j, i, n, m])  
                normal_issue = False  
if normal_issue:  
    restrictions.append([c, r, j, i, 0, 0])
```

Annexe 9 : code source du fichier de lignes de commandes (batch) afin de mettre en place les liaisons entre Qt Creator et Python (et donc QGIS)

```
@echo off  
  
call "C:\Program Files\QGIS 3.4\bin\o4w_env.bat"  
  
call "C:\Program Files\QGIS 3.4\bin\qt5_env.bat"  
  
call "C:\Program Files\QGIS 3.4\bin\py3_env.bat"  
  
@echo on  
  
pyrcc5 -o resources.py resources.qrc
```