Rapport d'élève ingénieur
Projet de 2ème année
Filière : F3 - Systèmes d'Information et Aide à la Décision

Description du chemin entre les carrefours

Présenté par Maxime Guillaumin et Shisong Wang

Tuteurs du projet : M. Jean-Marie Favreau et M. Jérémy Kalsron
Référent ISIMA : Mme. Myoung-Ah Kang
Date de la soutenance : 17/03/2023
Durée du projet : 4 mois
Campus des Cézeaux. 1 rue de la Chebarde. TSA,60125. 63178 Aubière CEDEX

Remerciements

Nous aimerions, en premier lieu, remercier notre tuteur de projet M. Jean-Marie Favreau, enseignant-chercheur au LIMOS (Laboratoire d'informatique, de modélisation et d'optimisation des systèmes) pour sa disponibilité, son suivi régulier et sa pédagogie tout au long du projet.

Nous remercions également M. Jérémy Kalsron, étudiant doctorant au LIMOS et cotuteur de notre projet pour sa très grande disponibilité et son aide particulièrement précieuse.

Nous remercions enfin, Mme. Myoung-Ah Kang, notre professeure référente ISIMA.

Table des figures

Résumé

L'objectif de notre projet est de générer automatiquement une description textuelle des rues entre les intersections à partir de données géographiques. Il s'inscrit dans un projet plus gobal : le projet **ACTIVmap** qui s'intéresse à la mobilité des personnes en situation de déficience visuelle. Ces descriptions, qui peuvent être lues par synthèse vocale, ont pour but de faciliter et sécuriser leurs déplacements notamment en milieu urbain.

Ce projet relève du domaine de la **géomatique** (contraction des termes géographie et informatique). Nous utilisons des données géographiques collaboratives issues d'**OpenStreetMap** (OSM), que nous manipulons en langage **Python** à l'aide des bibliothèque **OSMnx** et **NetworkX**.

Le développement a été réalisé sous Anaconda (distribution Python) avec la création de Jupyter Notebooks qui permettent de créer des documents contenant des portions de codes exécutables de manière interactives et des explications textuelles.

Nous avons rendu un module Python documenté, compatible et pouvant être intégré aux travaux déjà réalisés pour la génération de sémantique à partir de zones géographiques. Ce module permet notamment une structuration de la donnée géographique, sous la forme d'un **graphe** accompagné de sémantique directement exploitable et ouvrant de nombreuses perspectives pour la description d'un itinéraire entier. Nous fournissons également un autre petit module Python produisant une génération simple de texte à partir de ce graphe.

Mots-clés : ACTIVmap, OpenStreetMap, Géomatique, Python, OSMnx, NetworkX, Graphe

Abstract

The goal of our project is to automatically generate a textual description of streets between intersections from geographic data. It is part of a larger project : the **ACTIVmap** project which focuses on the mobility of visually impaired people. The aim of these descriptions is to facilitate and secure their movements, especially in urban areas, using, for example, audio descriptions.

This project is in the field of **geomatics**. We use collaborative geographic data from **OpenStreetMap** (OSM), and we manipulate them in **Python** using the **OSMnx** and **NetworkX** libraries.

The development was done under Anaconda (Python distribution) with the creation of Jupyter Notebooks that allow to integrate Python code, execution results and textual explanations.

We made a documented Python module that is compatible and can be integrated with the work already done for the generation of semantics from geographical areas. This module allows in particular a structuring of the geographical data, in the form of a **graph** accompanied by directly exploitable semantics and opening many perspectives for the description of an entire route. We also provide another small Python module producing a simple text generation from this graph.

Keywords : ACTIVmap, OpenStreetMap, Geomatics, Python, OSMnx, NetworkX, Graph

Table des matières

Introduction

Ce projet nous a été confié par le LIMOS et s'inscrit dans le projet ACTIVmap[mettre référence vers le site web ici] (Assistance à la conception de cartes pour déficients Visuels). Il est mené par le LIMOS conjointement avec l'équipe GeoVIS de l'UMR LASTIG (IGN), l'Institut de Recherche en Informatique de Toulouse, et l'entreprise FeelObject. Il vise la conception d'outils informatiques pouvant traiter et structurer des données géographiques de plus en plus disponibles et permettant la création de cartes adaptées à la diversité des handicaps.

Nous nous concentrerons ici sur la production de texte à partir de données géographiques pouvant être utilisées en audiodescription par des personnes en situation de déficience visuelle. Cette production de sémantique comprend la description d'intersections en milieu urbain, et du chemin entre ces différentes intersections pour faciliter le déplacement des personnes d'un point A vers un point B.

L'enjeu est donc la structuration et l'exploitation de données à la fois informatiques (on veut une génération à grande échelle, et donc automatisée, de textes descriptifs) mais aussi topologiques (on cherche une version simplifiée de l'espace mais permettant de s'orienter) et sémantique(en conservant les données textuelles disponibles).

Nous nous baserons sur les travaux de Jean-Marie Favreau et Jérémy Kalsron qui ont déjà implémenté des modèles permettant une représentation simplifiée des intersections et la génération de texte sur ces carrefours afin de faciliter leur traversée.

Il nous a donc fallu, pour ce projet, comprendre et utiliser les données brutes issues d'OpenStreetMap mais aussi les modèles déjà implémentés et respecter les choix de modélisation pour pouvoir produire un module Python cohérent et intégrable fournissant de nouveaux modèles permettant de décrire à la fois les intersections et les rues entre ces intersections.

Nous donnerons dans un premier temps le contexte global du projet puis nous détaillerons ce qu'est OpenStreetMap et comment sont structurées les données que nous utilisons. Après une présentation des travaux de M. Favreau et Kalsron, nous présenterons nos choix de conception et d'implémentation et justifierons notamment notre approche utilisant la théorie des graphes pour compléter les modèles existants et amener de nouvelles perspectives que nous aborderons dans le bilan du projet.

1 Contexte du projet

1.1 Le LIMOS

Le LIMOS (Laboratoire d'Informatique, Modélisation et Optimisation des Systèmes) est un laboratoire de recherche situé à Clermont Ferrand, France. Il est rattaché au Centre National de la Recherche Scientifique (CNRS), à l'Université Clermont Auvergne (UCA) et à l'Institut des Sciences et Systèmes de l'Ingénieur (INSIS).

Les activités de recherche du LIMOS sont centrées sur l'informatique, la modélisation et l'optimisation des systèmes complexes. Plus précisément, le laboratoire travaille sur un large éventail de sujets, notamment les systèmes distribués, l'intelligence artificielle, le génie logiciel, l'apprentissage automatique, l'exploration de données, l'optimisation, la simulation, etc. Les recherches menées au LIMOS comportent à la fois des aspects théoriques et appliqués et visent à contribuer à l'avancement des connaissances dans ces domaines ainsi qu'au développement de solutions innovantes à des problèmes du monde réel.[1]

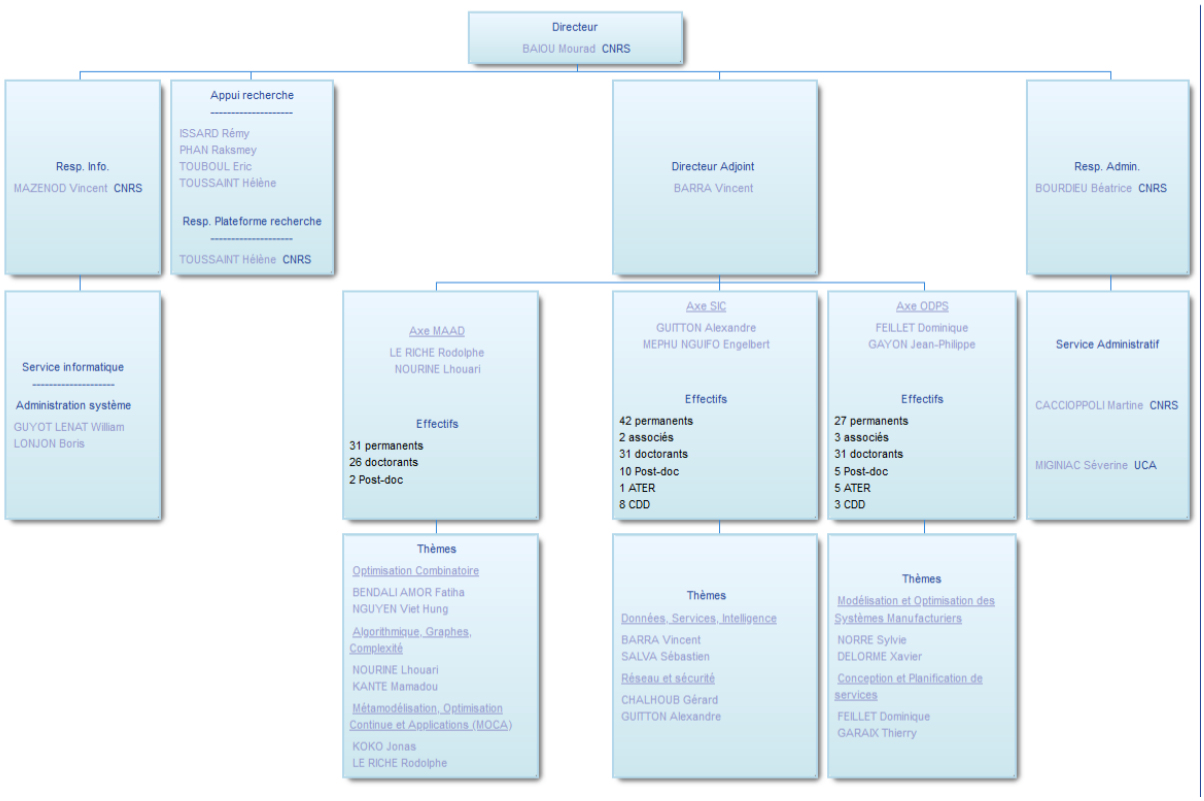

FIGURE 1 – Organigramme du LIMOS
Source : https ://limos.fr/organigramme

Le tuteur de notre projet, Jean-Marie FAVREAU, a principalement participé à l'Axe SIC du LIMOS. Les travaux de l'axe Systèmes d'Information et de Communication (SIC) portent sur les questions fondamentales et appliquées liées à l'acquisition de données via les réseaux de capteurs sans fil et à leur sécurité, gestion et communication. L'analyse de grandes masses de données ainsi que l'analyse des systèmes (qualité, interopérabilité), notamment au travers des services web et des processus métiers.[2]

La direction de recherche de l'axe SIC est constituée de deux thématiques : "Données, Services, Intelligence" et "Réseau et sécurité". Notre projet s'inscrit dans le premier thème.

1.2 OpenstreetMap

OpenStreetMap (OSM*) est un projet libre et collaboratif de production cartographiques visant la création d'une base de données géographiques complète et accessible. Les données OSM peuvent être créées et éditées en ligne par tous les utilisateurs. Elles proviennent de contributeurs bénévoles utilisant des appareils GPS, des images aériennes et des informations locales. Les données cartographiques qui en résultent sont gratuites et ouvertes, ce qui signifie que tout le monde peut les utiliser, les partager et les améliorer. Le projet compte maintenant plus de 7 millions d'utilisateurs enregistrés et ayant fourni des données à OSM.

L'un des avantages d'OSM est le niveau d'information très détaillée, on peut avoir par exemple l'emplacement et la forme de bâtiments individuels, le type de végétation dans un parc ou encore la limite de vitesse sur une route particulière. Ce niveau de détail permet une exploitation des données pour une analyse et une prise de décision plus précise dans divers domaines. OSM permet également aux contributeurs d'ajouter des attributs spécifiques à la carte, par exemple, un contributeur peut ajouter une étiquette pour indiquer qu'une certaine route est une route à péage, ou qu'un certain bâtiment est un hôpital[3].

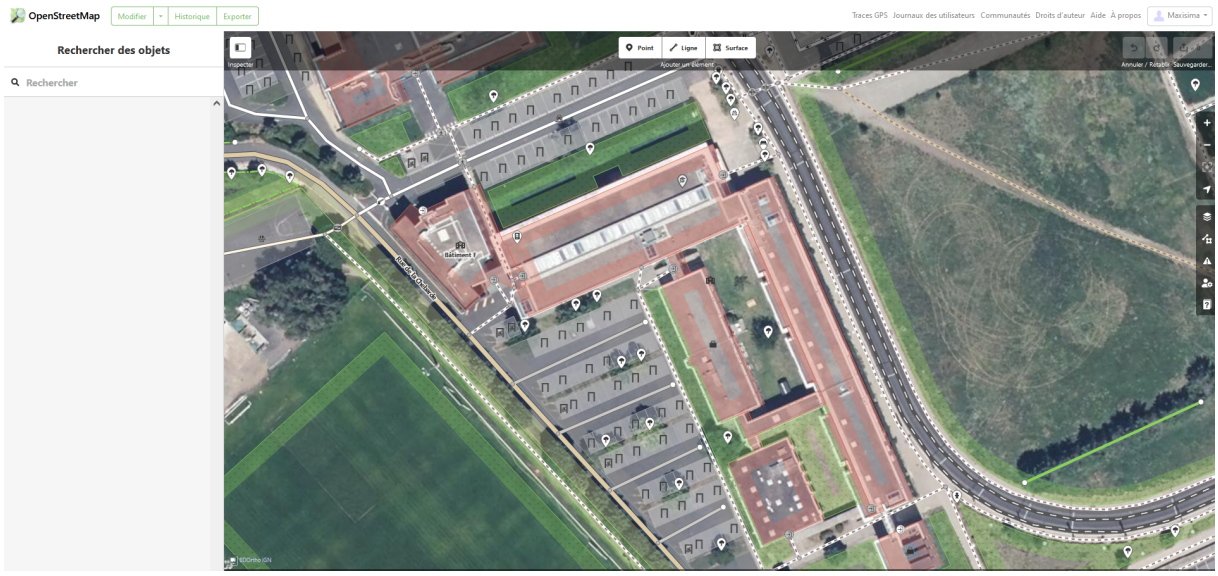

FIGURE 2 – Vue OSM de l'ISIMA

On peut voir sur cette figure le niveau de détail de la donnée OSM avec notamment,

les différents bâtiments de l'ISIMA géométriquement réprésentés et précisément délimités, les routes, les voies de tramway, les différentes places de parking, le stade de foot ou encore les arbres présents sur le parking de l'ISIMA.

Les données OSM sont structurées en éléments fondamentaux qui peuvent être des nœuds (points), des chemins (lignes) ou des relations (groupes d'éléments). Chaque élément possède un identifiant unique (OSMid) et peut posséder des attributs sous la forme clé :valeur décrivant sémantiquement l'élément.

FIGURE 3 – Attributs OSM du bâtiment F de l'ISIMA

Nous avons ici cliqué sur le bâtiment F de l'ISIMA. On voit que cet élément possède 4 attributs. Par exemple **building** est une clé associée à la valeur **university**. Ce qui signifie que ce bâtiment est classé comme un bâtiment universitaire. La valeur 2 associée à la clef **building :levels** signifie que ce bâtiment possède 2 étages.

Tous ces éléments étant inter-connectés entre eux par d'autres éléments de type "ligne" pouvant être par exemple une route, une rue piétonne etc... Nous pouvons voir la donnée OSM comme un graphe. C'est à dire une structure de données composées de noeuds reliés entre eux par des arêtes et complétés par de la sémantique (qui sont les attributs associés à chaque noeud ou arête).

Les données OSM* d'un périmètre géographique donné sont téléchargeables et exportables dans un fichier au format .XML* ou .JSON*. Ce fichier contient l'ensemble des éléments OSM avec leur identifiant et leurs attributs.

1.3 La segmentation des intersections

Un point important à expliquer est le modèle qui a été implémenté et qui permet la détection et la segmentation des carrefours. Le model consiste à considérer un carrefour dans sa globalité composé d'un coeur de carrefour avec plusieurs branches qui peuvent contenir plusieurs segments. [4].

Le but de cette modélisation est d'adopter un point de vue piéton est de voir une intersection comment un élément très particulier que l'on doit traverser et qui possédent plusieurs surlesquelles on peut se diriger.[5]. La délimitation de ses intersections se fait à l'aide d'éléments géométriques, topologiques mais aussi principalement de sémantique : en utilisant notamment les passages piétons ou encore les feux tricolores. .

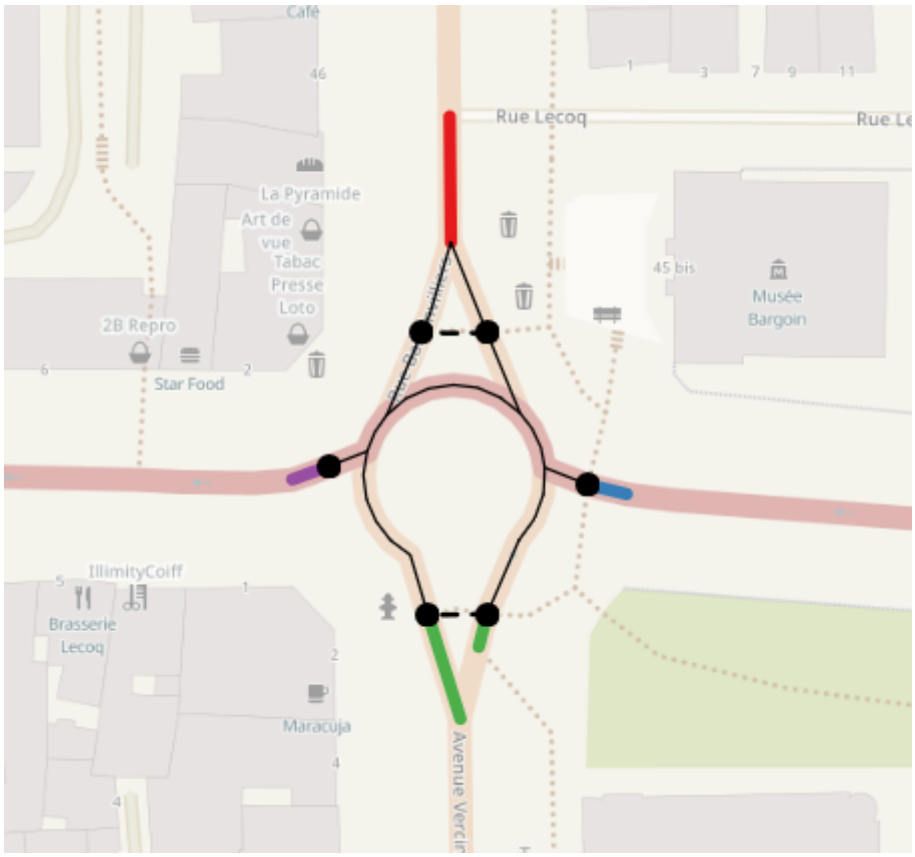

FIGURE 4 – Une Intersection segmentée à 4 branches
Source : https ://carrefour.anatidaepho.be/

L'intersection ci-dessus est un carrefour possèdant 4 branches (une de chaque couleur). Il n'y a qu'une seule branche au nord et une seule autre branche au sud même si elles sont séparées par un îlot au bord de l'intersection, ce n'est bien qu'une seule et même direction (les voies fusionnent après). En revanche, il y a plusieurs arêtes internes à l'intersection (ce sont les traits noirs). Les points noirs correspondent à des noeuds OSM représente des passages piétons. L'avantage de cette modélisation d'intersection segmentée est que l'on peut décrire comment se traverse l'intersection.

Le modèle détecte donc des intersections et construit un objet segmentation contenant une région (objet) intersection et des régions branches. La frontière délimitant un coeur d'intersection d'une branche se trouve d'un point de vue piéton toujours au niveau des passages piétons (ou feux tricolores s'il n'y a pas de passage piéton)[5]. En distinguant deux types de régions (branches et coeur de carrefour) on peut classer deux types de **noeud OSM** :

— Les **inner_nodes**.
— Les **border_nodes**.

Les inner_nodes sont tous les noeuds OSM qui font partie d'arêtes internes au coeur de carrefour. Ils sont ajoutés à l'objet Crossroad avec leurs différents attributs.

Les border_nodes sont les noeuds qui font partie à la fois d'arête interne au carrefour et externe. Ce sont souvent le début des branches : ils marquent la frontière entre ce qui

fait partie de l'intersection et ce qui fait partie de la branche. Ils sont présents dans les objets crossroad et branche avec leur liste d'attributs.

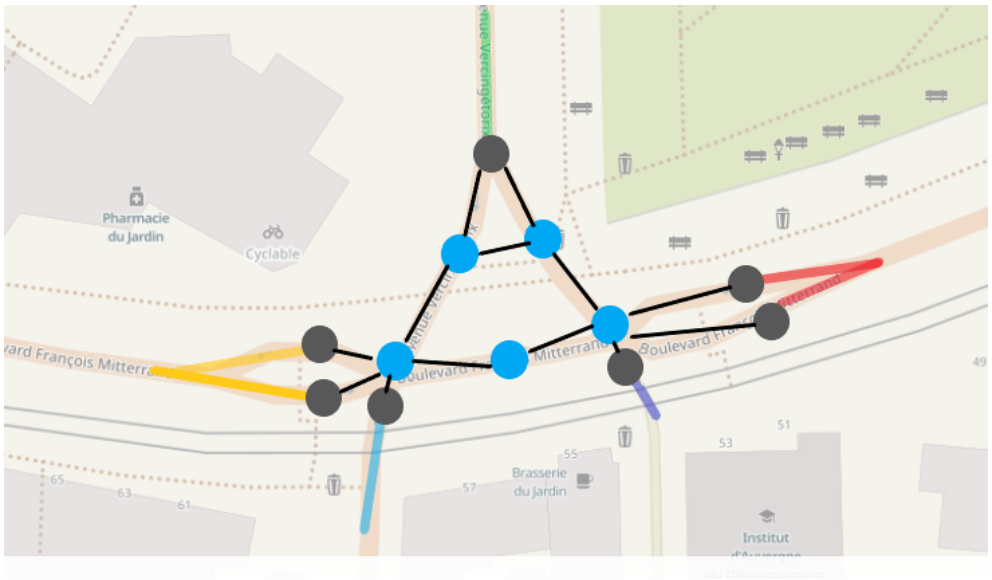

FIGURE 5 – Carrefour segmenté à 5 branches

Sur cet exemple, on a un carrefour à 5 branches, les inner_nodes sont représentés en bleu et les border_nodes en gris. Les arêtes ou segments internes en noir.

1.4 Analyse du problème

Le modèle de segmentation développé produit un ensemble d'objets structurés (coeurs de carrefour et branches) permettant de décrire assez aisément les intersections situées dans une zone géographique donnée. La limite de ce modèle est que l'on ne connaît pas la topologie externe de ces intersections. En effet, on ne peut pas savoir à priori si les intersections sont reliées entre elles ou non par des routes. Si oui, sont-elles reliées directement par une voie ou bien faut-il traverser une autre intersection ? À quelle distance, d'un point de vue géographique, se situent-elles les unes des autres ?
La problématique est donc d'étendre ce modèle et lui ajouter un topologie qui corresponde à la réalité géographique. Le modèle doit rester simple d'un point de vue piéton pour permettre la description automatique. Il est également nécessaire de conserver la dimension sémantique des intersections tout en ajoutant de la sémantique permettant de décrire une voie entre deux intersections.

2 Solution apportée

2.1 Étude du problème, angle d'approche

Il y avait plusieurs angles d'approche pour apporter une solution à notre problème. Le premier était d'utiliser principalement la sémantique et les propriétés géométriques et/ou géographiques des éléments OSM pour établir un nouveau modèle contenant les relations entre les intersections. Cette approche s'appuie sur une manipulation de dataframes* avec des données géographiques associées en utilisant des outils tel que QGIS* qui s'appuient

sur des modélisations par couche.

Nous n'avons pas retenu cette approche car nous nous sentions moins à l'aise sur ces notions qui relèvent un peu plus de l'approche géographique mais aussi parce que cette modélisation ne facilite pas une exploration topologique des données.

L'approche que nous avons choisie, est beaucoup plus proche de l'informatique et relève de la théorie des graphes. L'idée est de considérer les intersections segmentées dans leur globabilité (noeuds internes et toutes ses branches) comme un seul noeud d'un nouveau graphe que l'on va construire. Ainsi, si deux intersections sont voisines (reliées par une route) alors on considère une arête entre les deux noeuds correspondants.

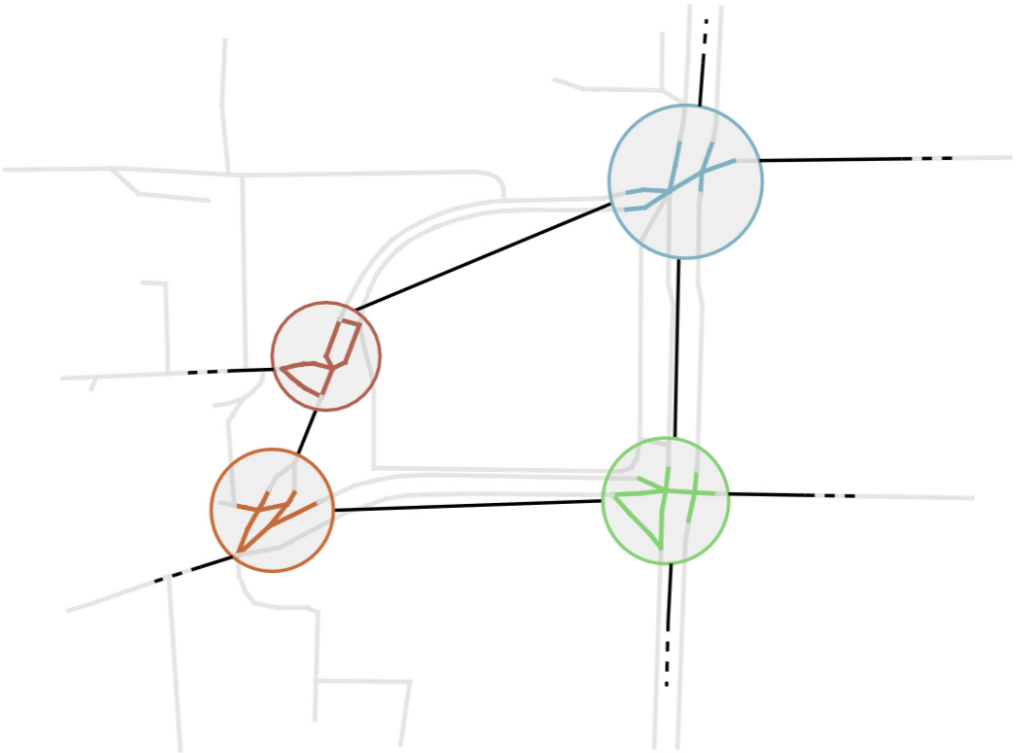

FIGURE 6 – Graphe conceptuel des relations entre les carrefours
Source : Jérémy Kalsron, données contributeurs OpenStreetMap.

Voici un exemple du type de structure que l'on souhaiterait obtenir. Les intersections segmentées ici, de différentes couleurs, sont les noeuds de notre graphe et sont reliées entre elle par des arêtes (traits noirs) si une route existe réellement entre deux intersections. L'avantage avec cette structure de données est qu'elle est assez générique. Elle garde assez d'information topologiques et géométriques pour notre besoin et permet d'utiliser des algorithmes théoriques efficaces et robustes pour la parcourir par exemple.

2.2 Quelques outils très utiles

Les principaux outils utilisés dans ce projet sont OSMnx et NetworkX. OSMnx est un package Python qui permet aux utilisateurs de télécharger et d'analyser les données

OpenStreetMap. OSMnx permet le traitement des données cartographiques OSM en Python. On peut, utilisant OSMnx, télécharger des données du réseau routier et urbain de n'importe quelle ville du monde à partir d'OpenStreetMap puis analyser et visualiser ces réseaux (sous la forme de graphes).

Les fonctionnalités d'**OSMnx** incluent : le téléchargement de données routières pour des villes arbitraires à partir d'OSM, la construction de réseaux routiers sous forme de NetworkX MultiDiGraphs[6], le calcul de diverses métriques de réseau telles que des calculs de distance, de plus courts chemins et l'enregistrement de réseaux routiers sous forme de fichiers de formes ou de fichiers GraphML.[7] OSMnx est principalement utilisé pour traiter des données OSM avec Python pour des domaines d'application tels que l'urbanisme, l'analyse du trafic et la science des données spatiales. Nous utiliserons cette bibliothèque pour extraire des données OSM (éléments OpenStreetMap que l'on a vu au dessus) et les stocker dans un graphe. Les éléments OSM sont des noeuds (par exemple un feu tricolore) ou des arêtes de ce graphes(routes etc...). L'ensemble des attributs sémantiques des éléments OSM (paire clé :valeur) sont stockées en tant qu'attribut de noeud, ou attribut d'arête. Chaque noeud du graphe OSMnx, contient un dictionnaire python qui stocke l'identifiant unique OSMid de l'élément et ses attributs.

NetworkX est une bibliothèque Python permettant l'étude et l'analyse des réseaux complexes. C'est un outil pour créer, manipuler et analyser des graphes et des réseaux, et des algorithmes pour mesurer les propriétés et les caractéristiques de ces réseaux.[8] Les principales fonctions de NetworkX sont les suivantes : fournir une variété de méthodes pour créer et manipuler des graphes, y compris l'ajout de nœuds et d'arêtes et la génération de graphes aléatoires. Il permet à l'utilisateur de modifier le graphe en ajoutant ou en supprimant des nœuds et des arêtes et en fusionnant ou en divisant le graphe. NetworkX comprend également des algorithmes pour calculer diverses propriétés de graphes, telles que les chemins les plus courts. NetworkX fournit également une variété de méthodes de visualisation graphique. NetworkX peut être utilisé avec d'autres bibliothèques Python telles que NumPy, Pandas.

2.3 Conception théorique

Les objets python produits par la segmentation et utilisés par le modèle actuel de description des carrefours ne permettent pas un parcours efficace pour reconstruire la donnée topologique (en connectant les différents carrefours sous la forme d'un graphe conceptuel identifié dans l'étude du problème) et ils ne contiennent pas l'ensemble de la donnée sémantique (on a seulement l'identifiant OSM des éléments situés dans une intersection). Ces objets sont cependant créés à partir d'un graphe OSMnx sur lequel on effectue des traitements pour réaliser la segmentation. Avant segmentation, ce graphe que l'on notera **G**, contient toute l'information sémantique (l'ensemble des noeuds,arêtes et attributs des éléments OSM de la zone requêtée) et topologique. L'idée est alors de se servir de ce graphe **G** qui peut être facilement parcouru mais aussi de l'objet python **segmentation** pour construire notre nouveau graphe NetWorkX que l'on appellera **G'**. Nous exportons les données de la segmentation dans un fichier .JSON (avec une fonction **toJson()** fournie dans le module python de la segmentation) à partir duquel on pourra recréer nos propres structures de données facilitant la création de notre graphe **G'**.

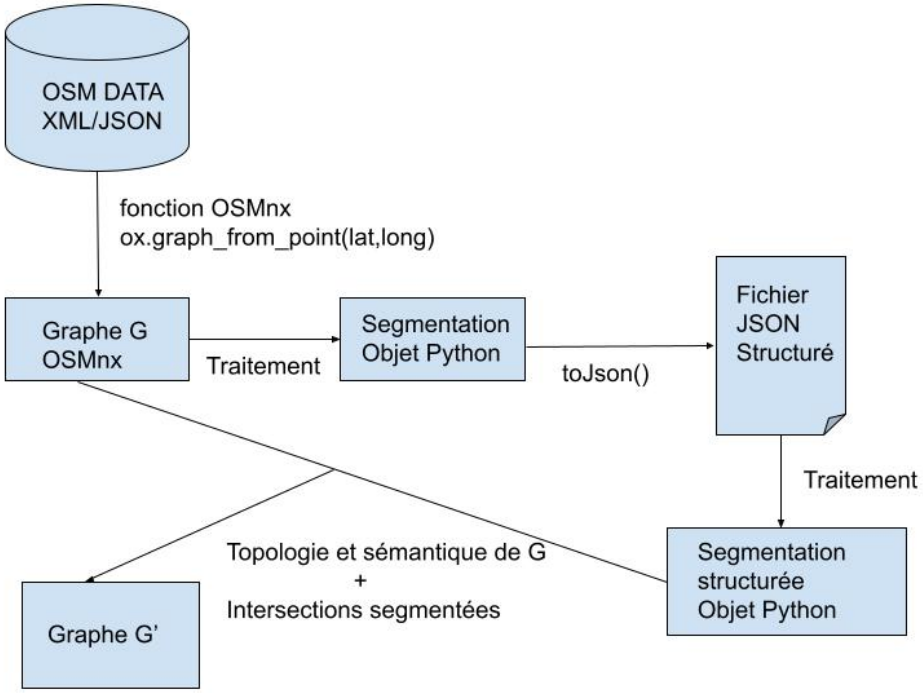

FIGURE 7 – Vue d'ensemble du processus pour la création de G'

La phase de traitement pour construire une nouvelle structure de données Segmentation à partir du fichier JSON consiste à créer et initialiser des objets de type **Intersection** pour chaque intersection segmentée présente dans le fichier JSON. L'objet **Intersection** contient un identifiant **id** unique, une liste des noeuds OSM intérieurs de l'intersection : les **inner_nodes**, une liste des noeuds OSM situés en bordure de l'intersection : les **border_nodes**, une liste d'objet python **Branche** qui contient les différentes branches de l'intersection. Chaque objet **Branche** contient aussi une liste de **inner_nodes** et de **border_nodes** et un identifiant **id**. Les données JSON à exploiter et issues de la segmentation se présente sous cette forme (c'est une liste de dictionnaires pythons qui contiennent les différentes valeurs) :


```
[[{"type": "crossroad", "nodes": {"inner": [], "border": [482036644, 1627306561]}, "edges_by_nodes": [[482036644, 1627306561]],
"coordinates": {"482036644": {"x": 3.0876005, "y": 45.7754085}, "1627306561": {"x": 3.0876006, "y": 45.7754413}}}, {"type": "branch",
"nodes": {"inner": [], "border": [482036643, 482036644]}, "edges_by_nodes": [[482036643, 482036644]], "coordinates": {"482036643": {"x":
3.0881687, "y": 45.7753816}, "482036644": {"x": 3.0876005, "y": 45.7754085}}}, {"type": "branch", "nodes": {"inner": [], "border":
[482036644, 3164636373]}, "edges_by_nodes": [[3164636373, 482036644]], "coordinates": {"482036644": {"x": 3.0876005, "y": 45.7754085},
"3164636373": {"x": 3.0869944, "y": 45.7753949}}}, {"type": "branch", "nodes": {"inner": [], "border": [482036644, 482036653]},
"edges_by_nodes": [[482036653, 482036644]], "coordinates": {"482036644": {"x": 3.0876005, "y": 45.7754085}, "482036653": {"x": 3.0875867,
"y": 45.7747076}}}, {"type": "branch", "nodes": {"inner": [], "border": [1627306561, 482036650]}, "edges_by_nodes": [[1627306561,
482036650]], "coordinates": {"1627306561": {"x": 3.0876006, "y": 45.7754413}, "482036650": {"x": 3.0875967, "y": 45.7758528}}}]]
```


FIGURE 8 – Exemple de quelques lignes d'un fichier JSON produit par la segmentation

La première étape de création du graphe **G'** consiste à ajouter un nouveau noeud pour chaque **intersection**. Nous allons ensuite nous servir du graphe G pour ajouter des arêtes entre les noeuds de G' si les intersections correspondantes sont connectées par une rue. Comme nous l'avons vu dans la partie de présentation d'OSMnx, nos différents noeuds possèdent un identifiant OSM unique. On peut donc, à partir de cet idenfiant, savoir si un noeud du graphe G appartient ou non à une intersection. Il suffit de regarder

si le noeud du graphe G apparaît dans les **inner_nodes** ou **border_nodes** d'une **intersection**.Si c'est le cas, on peut même savoir de quelle intersection il s'agit grâce aux **identifiants** de chaque intersection. Un moyen d'ajouter les arêtes de G' est de prendre un **border_node** d'une intersection comme point de départ dans G est de continuer le long d'une branche en explorant les composantes connexes du noeud et en faisant bien attention d'explorer les voisins du noeud qui ne font pas partie de la même intersection que le noeud de départ. On continue ce parcours jusqu'à tomber sur un voisin qui fait partie d'une autre intersection et on en déduit donc que cette intersection est connectée à l'intersection de départ et nous ajoutons l'arête correspondante dans notre graphe G'.

Voici l'algorithme en pseudo code permettant la création du graphe G'.

Algorithm 1 Create G'

```
for all crossroad in Segmentation do
  add_node(intersection,G')
end for
for all crossroad in Segmentation do
  for all branche in crossroad do
    start ← select_start(G,branche)
    next_crossroad ← BFS(start,G)
    add_edge(G',crossroad,next_crossroad)
  end for
end for
return G
```

Comme nous pouvons le voir dans cet algorithme, nous devons parcourir le graphe G à partir d'un point de départ avec **BFS*** et détecter un noeud qui fait parti d'une autre intersection. La fonction est un **parcours en largeur** du graphe G. Son principe est le suivant :

1. Choisir un noeud de départ et l'ajouter à une file.
2. Défiler le premier élément de la file et ajouter tous ses voisins non coloriés à la file.
3. Colorier le noeud qui vient d'être traité.
4. Retourner à l'étape 2 tant que la file n'est pas vide.

Ce type de parcours est très classique en théorie des graphes et il est particulièrement pertinent dans notre contexte. En effet, il suffit d'ajouter les conditions suivantes :

— Colorier tous les noeuds qui font partie de l'intersection de départ avant de lancer le BFS

— Arrêter le parcours lorsque le noeud que l'on traite fait partie d'une intersection

— Renvoyer le dernier noeud traité

pour garantir qu'on se déplace bien vers l'extérieur du carrefour et que l'on renvoie un noeud qui fait partie d'un carrefour voisin de notre carrefour de départ.

FIGURE 9 – Exemple de configuration du graphe OSM G

Si nous prenons par exemple cette illustration du graphe G que l'on va parcourir : il y a deux intersections voisines délimitées en pointillé. Si nous traitons la branche de droite de l'intersection rouge en prenant comme point de départ le noeud vert, en exécutant le **parcours en largeur** avec les conditions décrites précédemment on aura la configuration suivante :

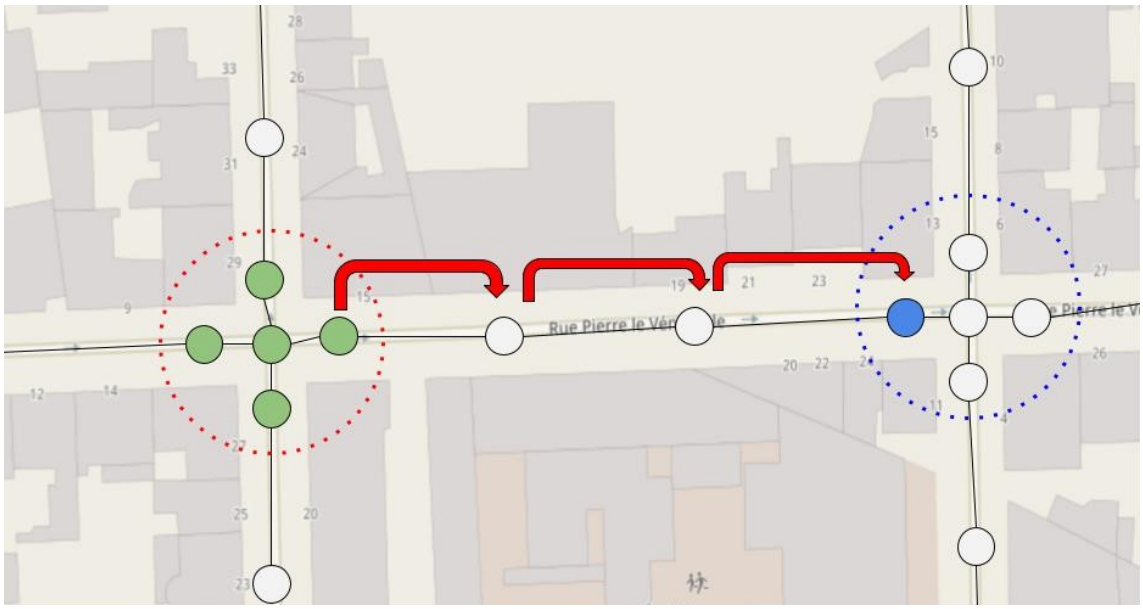

FIGURE 10 – Ordre de visite des noeuds de G sur la configuration précèdente

Les noeuds du carrefour de départ sont tous coloriés (en vert ici) donc lors de l'exploration des voisins, on suivra forcément le chemin rouge et on s'arrêtera sur le noeud bleu car on détectera qu'il fait parti d'une intersection. Il ne reste plus qu'à ajouter l'arête dans le graphe G' (dont les noeuds sont les intersections segmentées) entre l'intersection rouge et bleue qui sont bien des intersections voisines.

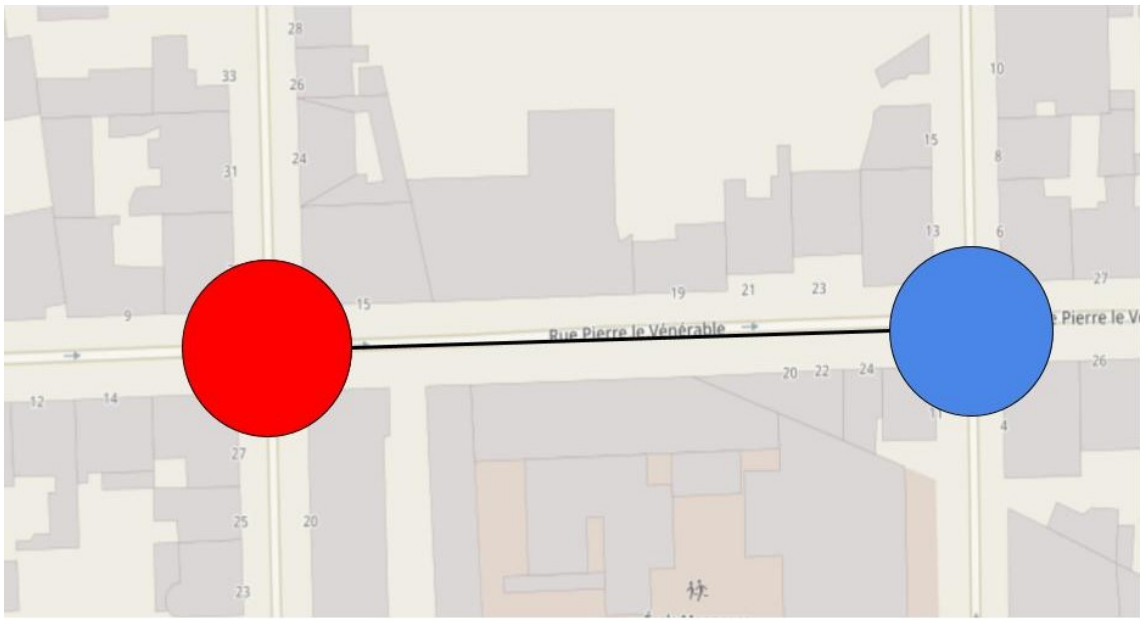

FIGURE 11 – Graphe G' à la fin du traitement de la branche de départ

2.4 Détails d'implémentation

Une manière efficace et pratique d'implémenter la solution précédente est d'effectuer une phase de prétraitement du graphe G. Ce graphe est un objet **OSMnx**, on peut donc ajouter des attributs supplémentaires pour chacun de ses noeuds. Nous avons ajouté les trois attributs suivants :

- cid
- branch
- color

Ils sont tous initialisés avec la valeur 0. Lors du parcours de notre objet **Segmentation** et des différents noeuds (**border_nodes et inner_nodes**) qui composent les intersections que nous ajoutons à G', on modifie en parallèle les attributs **cid** et **branch** de ces mêmes noeuds dans le graphe G (correspondance avec l'**id** OSM). On pourra ainsi très facilement, lors du **BFS**, savoir si le noeud que l'on traite est dans une intersection ou non en regardant la valeur de son attribut **cid** ou **branch**. S'il est égal à 0 alors on continue le parcours sinon on peut ajouter dans G' l'arête entre l'intersection de départ et l'intersection identifiée par l'attribut **cid**. Lorsqu'un noeud est visité, son attribut **color** prend la valeur 1, on continue l'exploration en visitant seulement les noeuds voisins qui ont la valeur 0 pour cet attribut.

On peut également ajouter des attributs d'arête à nos graphes python (graphe NetWorkX), ce qui est très utile. Il est primordial de conserver la sémantique des rues que l'on va décrire. Cette sémantique se trouve au niveau des noeuds du graphe G que l'on parcourt (ils possèdent l'ensemble des paires clé :valeur décrivant l'élément OSM sous la forme d'un dictionnaire python). On initialise donc une liste **data** au début du **BFS**. Chaque noeud visité est ajouté à cette liste et à la fin du parcours, la liste est ajoutée en tant qu'attribut de la nouvelle arête de G'. Ainsi, le graphe G' nous donne bien accès à l'information :

— topologique : un chemin entre deux carrefours est modélisé par une arête entre les deux noeuds correspondants.

— sémantique : ce chemin peut être décrit avec les données stockées en attribut d'arête.

Pour gagner en efficacité, pour chaque intersection, on choisit un point de départ parmi les **border_nodes** d'une branche non marquée. On part d'une de ces branches et dès que l'on arrive vers l'intersection voisine, on marque la branche de l'intersection voisine par laquelle on est arrivée. Cela permet de ne pas ajouter 2 fois la même arête en passant 2 fois dans la même rue. Notre graphe n'est pas orienté, si on a ajouté une arête de A vers B, inutile d'ajouter une arête de B vers A. Si on se plaçait d'un point de vue automobiliste cela aurait du sens, mais d'un point de vue piéton les rues sont toutes à double sens de circulation.

2.5 Génération de texte

À partir du graphe G' créé, la génération de textes descriptifs est plutôt aisée. Nous avons implémenté des petites fonctions python renvoyant des chaînes de caractères qui décrivent un chemin donné. Il faut d'abord déterminer ce que l'on souhaite décrire et quel point de vue on adopte. Nous avons fait le choix de décrire, étant donnée une position dans un carrefour, l'itinéraire piéton le plus court pour se rendre à un autre carrefour. L'avantage et la puissance de notre modélisation par ce graphe G' est que l'on peut utiliser des algorithmes classiques de théorie des graphes comme l'algorithme de Dijkstra qui calcule le plus court chemin entre deux sommets dans un graphe pondéré. Nous avons dû pour cela, ajouter un nouvel attribut distance pour chaque arête de G' et calculer toutes les distances entre les intersections voisines. Le calcul de ces distances se fait grâce à des fonctions OSMnx permettant de calculer une distance entre deux points connaissant ses coordonnées géographiques (latitude et longitude). Cette sémantique étant stockée dans notre graphe G', cette étape a été facilement réalisée.

Pour produire du texte, nous avons écrit des phrases génériques avec des paramètres qui changent selon la donnée sémantique récupérée. Cette donnée est l'ensemble des paires clé :valeur contenues dans G'. Si par exemple on veut décrire les passages piétons d'un chemin comme on l'a fait, on regarde si ce chemin contient un noeud possédant comme donnée sémantique la paire **highways :crossing** qui correspond, pour un noeud OSM, à un passage piéton. Si on a un tel noeud, on peut accéder à ses autres attributs dont sa latitude et longitude et donc calculer une distance par rapport au début du chemin et ainsi générer automatiquement une phrase du type "Il y a un passage piéton à X mètres pour traverser la voie", X étant le paramètre calculé à partir des coordonnées du noeud.

Un point qui peut être interessant est d'intégrée la description interne des carrefours, qui a déjà été implémentée, pour décrire un chemin complet et pas seulement le chemin entre les intersections. Nous avons donc commencé à intégrer (partiellement) cette description dans nos fonction de la manière suivante : nos itinéraires calculés sont des listes de noeuds du graphe G' et donc d'intersections. On peut récupérer le texte descriptif d'une intersection en appliquant le modèle de description (correspondance avec l'identifiant OSM) pour l'intégrer directement à notre texte.

2.6 Méthodes de travail

Nous avons fonctionné en méthode **AGILE*** en programmant régulièrement (environ une fois par semaine) des réunions avec nos tuteurs de projet. Le but était de se fixer des objectifs à court terme que nous devions réaliser pour la semaine suivante pour pouvoir parler et présenter un peu ce qu'on a réalisé et surtout identifier et essayer de résoudre les points sur lesquels nous bloquions.

FIGURE 12 – Diagramme de Gantt prévisionnel

Nous avions estimé que ce qui allait nous prendre le plus de temps était la conception et l'implémentation des fonctions qui génèrent du texte. Nous avions assez bien compris comment modéliser et concevoir le graphe, cependant la réalité fut assez différente, notamment au niveau de l'implémentation.

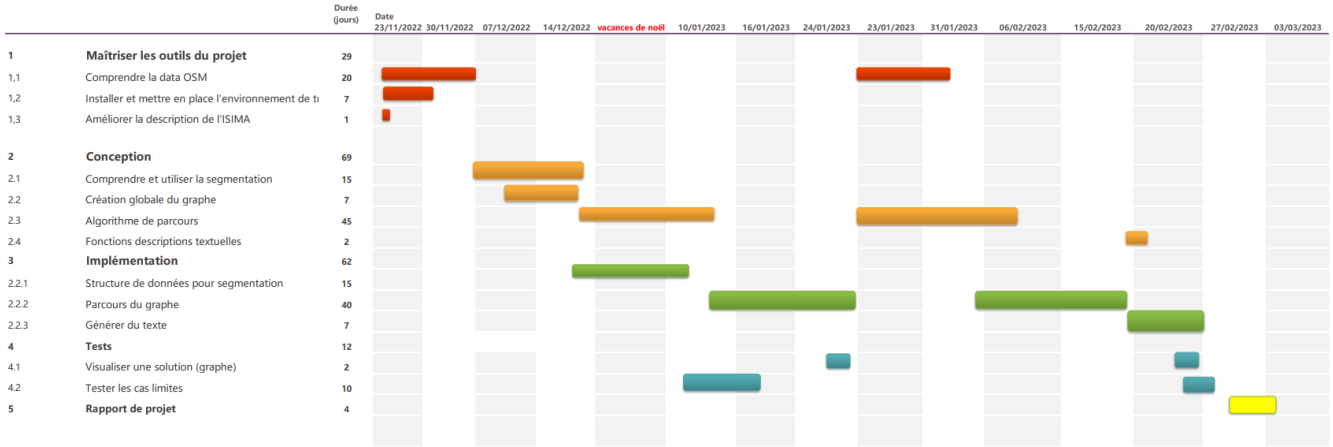

FIGURE 13 – Diagramme de Gantt réel

Maîtriser les outils et les données OSM fut plus compliqué que prévu. On identifie très bien sur le diagramme de Gantt réel deux phases de conceptions et deux phases d'implémentation pour le parcours du graphe G. En effet, notre première idée pour le parcours de G n'était pas d'utiliser le parcours en largeur. Nous avons conçu et implémenté une solution basée sur l'orientation des arêtes du graphe G. En effet, par défaut, le graphe OSM obtenu est un graphe orienté, c'est à dire que les arêtes entre les noeuds ne sont pas

forcément symétriques (on peut avoir une arête de A vers B, mais pas de B vers A).

OSMnx, en plus de fournir des fonctions pour avoir les voisins d'un noeud, permet également de connaître le successeur et le prédécesseur d'un noeud si le graphe est orienté. En sachant que l'orientation du graphe semblait suivre une certaine logique (les arêtes étaient toutes orientées dans le même sens lorsqu'elles font parties de la même rue), nous voulions nous servir de ces propriétés pour parcourir le graphe G et notamment ne pas revenir vers l'intérieur du carrefour lorsqu'on part d'une branche. Nous avons implémenté cette solution, elle semblait assez bien marcher, notamment à Barcelone où l'on a effectué nos premiers tests mais on a rapidement identifié des cas où ça ne marchait pas, avec des algorithmes qui tournent à l'infini.

Nous avons perdu pas mal de temps à implémenter une solution qui sur le plan théorique semblait très bonne, mais qui dans la réalité ne fonctionne pas. Nous avons identifié certains cas qui étaient assez fréquent et posaient problème notamment lorsque les noeuds du graphe OSM sont orientés de cette manière :

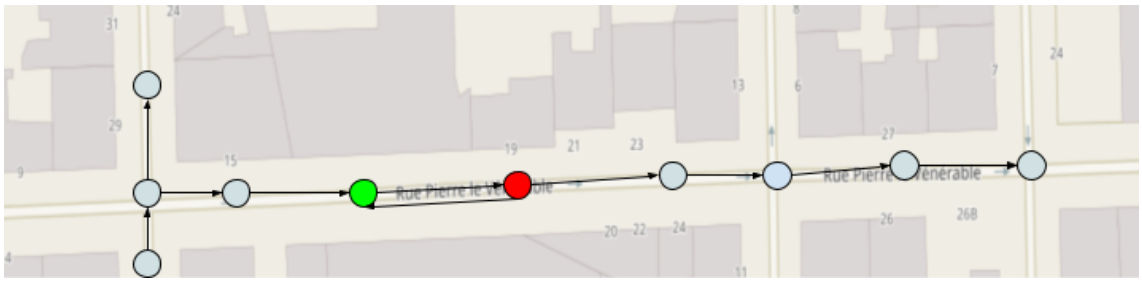

FIGURE 14 – Cas rencontré sur un graphe orienté

On peut voir que les arêtes entre le noeud rouge et le noeud vert posent problème. Le noeud rouge est le successeur du noeud vert et le noeud vert est le successeur du noeud rouge. On ne peut donc pas explorer la rue en utilisant la fonction successeur pour parcourir le graphe car on se retrouve bloqué dans une boucle. Cette orientation du graphe OSM est en fait dû à la manière dont sont créer les données sur OSM. La plupart sont saisies manuellement par des contributeurs bénévoles, la manière de lier les éléments entre eux n'est donc pas 100% homogène. Cela nous a fait prendre conscience que la théorie n'est pas la réalité et que nous sommes dépendants des données. On a donc repensé la manière de parcourir le graphe OSM en travaillant sur le graphe non orienté pour arriver à un parcours en largeur beaucoup plus robustes et beaucoup moins sensible aux multiples configurations différentes des données.

3 Résultats

3.1 Exemples de graphes obtenus et de descriptions générées

Pour bien comprendre et valider notre modèle, on peut afficher le graphe OSM G d'une zone géographique avec des fonctions d'affichage fournies par OSMnx puis le comparer avec le graphe G' construit à partir de G et des intersections segmentées.

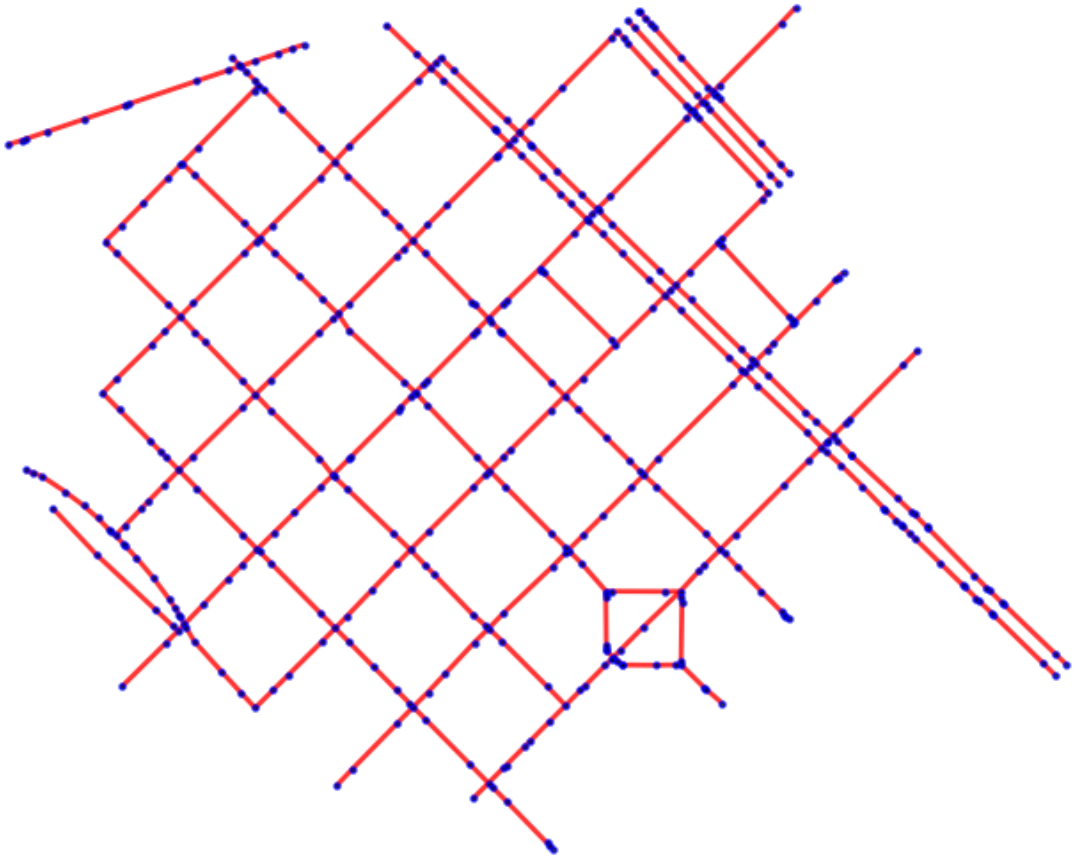

FIGURE 15 – Graphe OSMnx avant segmentation

Nous reconnaissons ici le réseau urbain de Barcelone qui est pertinent pour bien visualiser les différences entre les deux graphes. Nous avons extrait ces données avec la fonction OSMnx graphe_from_point(latitude, longitude), en prenant des coordonnées au centre de Barcelone et un rayon de 700 mètres autour ce point. Chaque point bleu est un éléments OSM et un noeud du graphe, les arêtes, en rouge, correspondent aux rues.

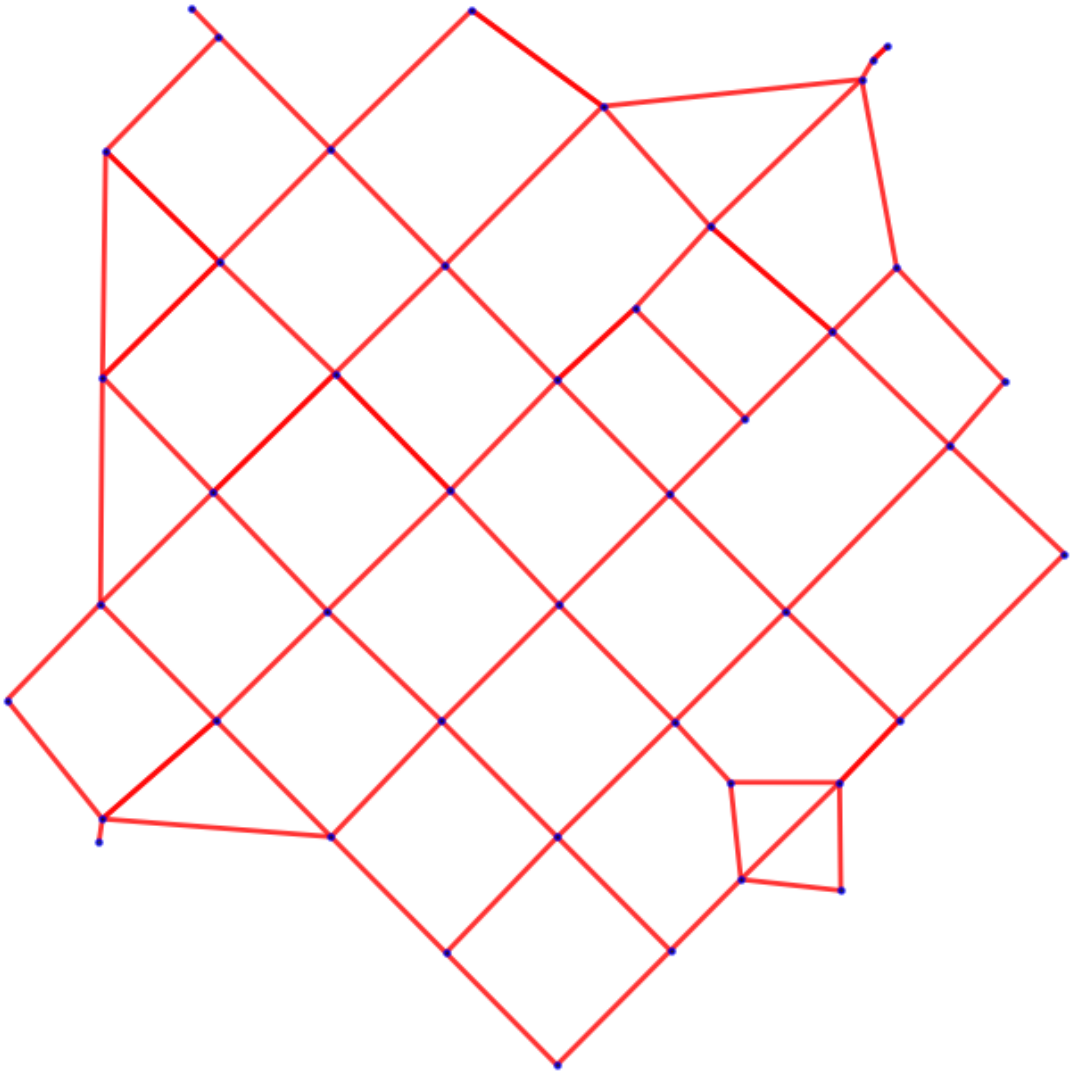

FIGURE 16 – Notre modèle appliqué à ce graphe

Si on exclue les bords du graphes (nous en reparlerons dans la partie sur les limites du modèle), on a bien la modélisation attendue : les noeuds du nouveau graphe correspondent seulement aux intersections segmentées et les noeuds OSM sont "fusionnés" s'ils font partie de la même intersection. Notre nouveau graphe conserve également la topologie de la zone géographique : il n'y a pas d'arête entre deux carrefours qui ne sont pas connectés par une rue et réciproquement, comme dans le graphe de base, une rue entre deux carrefours correspond à une arête dans le graphe.

Voyons maintenant avec un autre exemple proche des cézeaux de génération d'un itinéraire étape par étape :

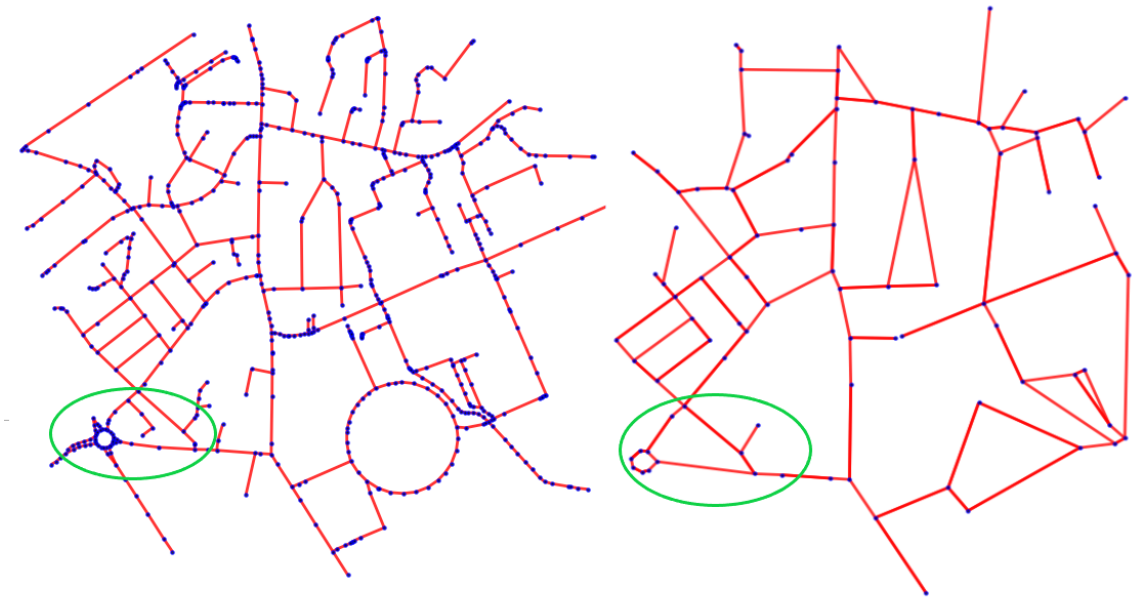

FIGURE 17 – Graphe OSMnx à gauche autour des Cézeaux et graphe segmenté à droite

Les intersections du petit itinéraire que l'on va décrire se situent dans les cercle verts.

FIGURE 18 – Carte du chemin que nous cherchons à décrire

Nous avons rentré 5 et 8 comme paramètre de notre fonction, ce qui veut dire que l'on part de 5 et on souhaite décrire le chemin le plus court pour aller jusqu'à l'intersection qui a pour id 8.

```
In [306]: print(Generate_description(G2,G1,5,8))

[5, 29, 8]
Vous êtes sur  Le carrefour à l'intersection de la rue des Rivaux, de la rue Pasteur et de la rue des Cézeaux qui est un c
arrefour à 4 branches.
Vous devez continuer sur la Rue des Cézeaux pendant 103 mètres la portion est limitée à 30 kilomètres par heure.
Il n'y a pas de passage piéton pour traverser la voie.
Vous arriverez sur Le carrefour à l'intersection de la rue Pasteur qui est un carrefour à 3 branches.
Continuez sur la Rue des Cézeaux pendant 36 mètres la portion est limitée à 30 kilomètres par heure.
Il n'y a pas de passage piéton pour traverser la voie.
Vous arriverez sur Le carrefour à l'intersection de la rue Pasteur qui est un carrefour à 3 branches..
Il y a 1 passage piéton pour traverser cette voie.
Il se situe à 8 mètres de l'intersection
```

FIGURE 19 – Exemple de ce que l'on est capable de générer grâce au graphe

Cette génération fait appel au modèle de description de carrefour, en ajoutant des informations sur le nom de la voie que l'on doit emprunter, la présence ou non de passage piéton entre les intersections et la limite de vitesse. C'est un début d'intégration entre la desccription des carrefours et celle entre les carrefours mais les deux modules ne sont pas encore complètement intégrés pour produire une description très précise : il reste encore à ajouter la description de la branche que l'on emprunte lorsque l'on arrive sur un carrefour.

3.2 Limites de ce modèle

Nous avons rapidement identifié plusieurs cas limites que nous devons prendre en compte. Le premier est la gestion des bordures du graphe. Nous travaillons sur des données OSM téléchargées et présentes sous la forme d'un graphe OSMnx d'une zone délimitée. Nous pouvons choisir un rayon pour la zone sur laquelle nous travaillons mais on ne peut pas savoir si les bords coïncideront sur des intersections (peu probable) ou si la rue sera juste coupée. Le parcours en largeur prend en compte le cas général et ne pose pas de problème d'éxécution même en bordure de graphe : si on arrive au bord du graphe en ne trouvant aucun noeud qui appartient à une intersection alors tous les voisins auront été traités et coloriés car on est en bord de graphe, le parcours s'arrêtera et on ne fera rien (pas d'ajout d'arête dans G'). Ce qui est bien le comportement que l'on souhaite. En revanche, la manière dont la segmentation des intersections est implémentée peut causer des erreurs dans certains cas.

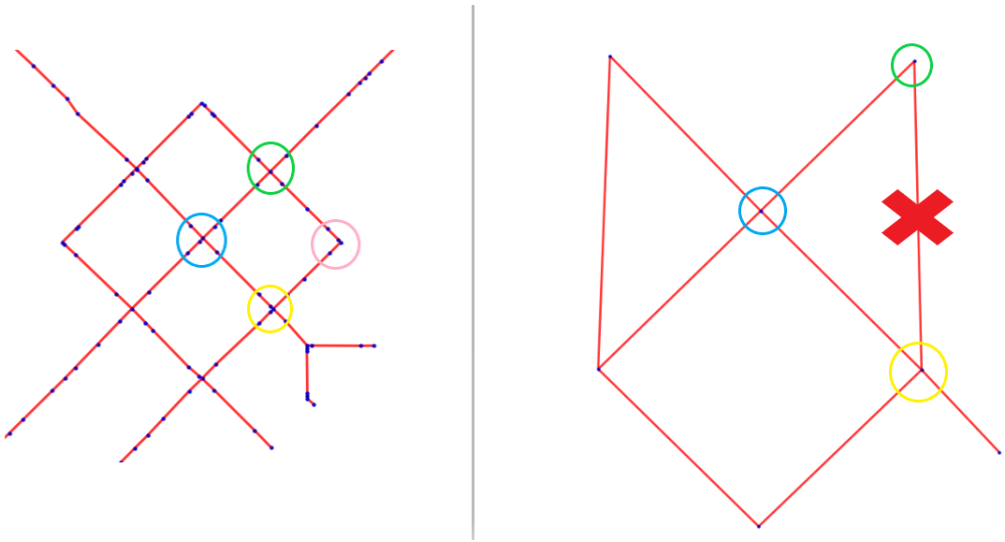

FIGURE 20 – bordure de graphe OSM (à gauche) et segmenté (à droite)

Si on observe le cas ci-dessus, on remarque qu'une arête est ajoutée dans le graphe G' entre l'intersection verte et l'intersection jaune, ce qui ne reflète pas la topologie réelle puisque ces deux carrefours ne sont pas connectés (on a le cas symétrique à l'ouest du carrefour bleu). Cela se produit à cause du carrefour en rose, certains noeuds qui composent ce carrefour sont présents dans le graphe G mais comme l'intersection est en bordure de graphe, il manque des informations et ce carrefour n'est pas détecté comme tel par la segmentation et n'apparaît donc pas dans le graphe G'. Il existe pourtant bien un chemin dans G entre l'intersection verte et l'intersection jaune, le **BFS** devrait s'arreter au niveau de l'intersection rose mais comme elle n'existe pas, la parcours continue jusqu'à la prochaine intersection et ajoute l'arête.

Nous ne pouvons pas vraiment régler ce problème car nous nous basons sur la segmentation des intersections. On peut toutefois ne pas considérer les bordures et pour générer la description par exemple, établir une zone avec un rayon (plus petit que celui du graphe G) autour du point central, pour laquelle on est sûr que les données ne contiennent pas d'errueur. Les algorithmes étant implémentés de manière efficace, on peut créer un graphe G à partir des intersections segmentées sur une très grande zone (plus de deux kilomètres) en moins de 2 secondes.

Un autre cas limite est lorsqu'une voie se sépare en deux. On a modélisé une route par une arête en considérant qu'une arête relie toujours deux et uniquement deux intersections. Ce qui est vrai pour au moins 90% des cas.

FIGURE 21 – Cas de la division d'une voie

Dans le cas ci-dessus, on remarque la séparation du cours Sablon en deux voies dis-

tinctes. Dans cette configuration, le graphe G' produit sera comme affiché sur le fond de carte : on aura deux arêtes différentes qui relient l'intersection sud aux deux autres. La topologie du graphe ne reflète pas vraiment la réalité car d'un point de vue piéton, même si l'on a deux arêtes, on est sur la même voie qui se divise plus tard. On perd donc de l'information avec cette modélisation.

L'une des possibilités est de détecter ce cas lors de la segmentation des intersections et donc au lieu d'avoir un seul noeud pour représenter l'intersection, on aurait un double noeud (pour les deux voies). L'autre piste évoquée était de créer une intersection factice, qui n'existerait que dans le graphe G' (comme le losange ci dessous) et ainsi mieux correspondre à la topologie réelle. Nous avons ajouté des informations supplémentaires lors de la création du graphe G' pour avoir en tant qu'attribut d'arête l'id des branches qui composent les deux extrémités de l'arête. Ainsi, nous avons soumis des idées pour détecter ce pattern et ajouter l'intersection factice en postproduction. L'idée est d'utiliser une petite fonction qui regarde pour chaque intersection, toutes ses arêtes et si on a deux arêtes qui partent de la même branche, alors nous avons cette configuration. Nous n'avons pas implémenté cette fonction par manque de temps.

3.3 Perspectives et avenir du modèle

Ce projet a été effectué dans un contexte de recherche scientifique. Il s'inscrit dans la continuité de ce qui avait déjà été réalisé tout en offrant de nouvelles perspectives. Nos fonctions de génération de description sont assez basiques et peu évoluées. Mais les possibilités de réutiliser notre modèle pour générer des descriptions beaucoup plus poussées sont nombreuses. En effet, le graphe que nous avons produit contient énormément d'informations et cette structure permet l'utilisation d'outils puissants (nous avons par exemple, utilisé l'algorithme de Dijkstra).

Nous nous sommes limités, pour la description, aux noeuds OSM qui composent les voies. Mais il est possible, à partir des noeuds qui composent une rue, de calculer ce qu'on appelle des buffers. Ce sont des fonctions permettant à partir d'un objet de définir des polygones par proximité géospatiale[9] et ainsi capter les objets proches en les sélectionnant selon un attribut par exemple. On pourrait avec ce principe, décrire les objets de part et d'autres d'une rue, notamment les bâtiments.

Conclusion

L'objectif de ce sujet était de concevoir et implémenter un nouveau modèle pour permettre une génération automatique de description d'éléments géographiques à destination des personnes déficientes visuelles. L'enjeu était de taille puisqu'il fallait trouver un modèle qui conserve les aspects géométrique, topologique et sémantique des données et permettre l'exploitation de celles-ci. Nous avons, dans un contexte de recherche scientifique, proposé une solution qui, jusqu'alors, n'existait pas malgré un besoin présent. Nous avons effecté quelques démonstrations non exhaustives de ce que l'on peut générer à partir de ce modèle mais il est destiné à être intégré et réutilisé par d'autres modèles plus poussés.

Nous avons découvert l'univers de la géomatique, qui est très intéressant mais nécessite une bonne imprégnation du fonctionnement des outils et des structures de données utilisées. Il était particulièrement agréable de pouvoir utiliser certaines connaissances en informatique théorique (notamment sur les graphes) sur un cas très appliqué. Nous avons aussi appris à nous adapter en manipulant des données concrètes et donc imparfaites et parfois incomplètes nous rappelant que la théorie est différente de la réalité.

Les perspectives d'amélioration sont extrêmements nombreuses comme nous l'avons vu. L'enjeu est de pouvoir décrire automatiquement et précisément tout une zone urbaine, faire de la détection d'obstacles etc... Le modèle que nous avons développé pourrait également d'autres applications en géomatique. Il pourrait notamment être réutilisé pour faire de la simplification de carte géographique en supprimant certains détails de la forme des routes tout en conservant une topologie globale d'un réseau routier.

Références

[1] B. Mourad, "Présentation du limos," 2022. https ://limos.fr/presentation/, date du dernier accès :01/03/2023.

[2] A. Guitton, "Présentation d'axe sic de limos," 2022. https ://limos.fr/AxeSic/, date du dernier accès :01/03/2023.

[3] M. Haklay and P. Weber, "Openstreetmap : User-generated street maps," *IEEE Pervasive computing*, vol. 7, no. 4, pp. 12–18, 2008.

[4] J. Kalsron, J.-M. Favreau, and G. Touya, "CrossroadsDescriber – Automatic Textual Description of OpenStreetMap Intersections," *AGILE : GIScience Series*, vol. 3, p. 40, June 2022.

[5] J.-M. Favreau and J. Kalsron, "What are intersections for pedestrian users ?," *AGILE : GIScience Series*, vol. 3, pp. 1–15, 2022.

[6] G. Boeing, "Osmnx : New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017.

[7] G. Boeing, "Osmnx : A python package to work with graph-theoretic openstreetmap street networks," *Journal of Open Source Software*, vol. 2, no. 12, 2017.

[8] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[9] V. Pászto and J. Pánek, "Spationomy simulation game 16," *Spationomy*, p. 305, 2020.

Glossaire

AGILE La méthode AGILE est une façon de gérer des projets qui s'adapte aux besoins et aux changements du client, plutôt que de suivre un plan rigide.

BFS L'algorithme de parcours en largeur (BFS) est un algorithme de recherche d'une structure de données arborescente pour un nœud qui satisfait une propriété donnée.

Dataframes Dataframe est une structure de données bidimensionnelle semblable à un tableau qui est couramment utilisée dans l'analyse de données et la science des données..

JSON JavaScript Object Notation (JSON) est un format de données textuel dérivé de la notation des objets du langage JavaScript.

NetworkX NetworkX est un package Python pour la création, la manipulation et l'étude de réseaux ou de graphes complexes.

OSM OpenStreetMap(OSM) est un projet collaboratif de cartographie en ligne.

OSMnx OSMnx est un outil Python pour l'analyse des réseaux routiers.

QGIS QGIS est une application de système d'information géographique (SIG) de bureau multiplateforme gratuite et open source.

XML Extensible Markup Language (XML) est un langage de balisage et un format de fichier pour stocker, transmettre et reconstruire des données arbitraires.