



HAL
open science

H2Network: un outil pour la modélisation et l'analyse de graphes dans le Système d'Information Géographique OrbisGIS

Erwan Bocher, Gwendall Petit, Mireille Lecoeuvre

► To cite this version:

Erwan Bocher, Gwendall Petit, Mireille Lecoeuvre. H2Network: un outil pour la modélisation et l'analyse de graphes dans le Système d'Information Géographique OrbisGIS. [Rapport de recherche] IRSTV FR CNRS 2488; IFSTTAR. 2014. halshs-01133333

HAL Id: halshs-01133333

<https://shs.hal.science/halshs-01133333>

Submitted on 19 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License



IFSTTAR

INSTITUT FRANÇAIS DES SCIENCES ET TECHNOLOGIES
DES TRANSPORTS, DE L'AMÉNAGEMENT ET DES RÉSEAUX



Réseau GEBD Grand Équipement Bases de Données J11-06

Lot B2 :

**H2Network : un outil pour la modélisation et
l'analyse de graphes dans le Système
d'Information Géographique OrbisGIS**

Convention : 10-AST-1-CVS-058

Programme : 0190 THUR-BSAF

Action 13 sous-action 10

Octobre 2014

HUBERT Jean-Paul

Directeur de recherches

DEST

Téléphone : +33 (1) 81 66 86 02

Jean-paul.hubert@ifsttar.fr

Auteurs du rapport : BOCHER Erwan (erwan.bocher@ec-nantes.fr)
PETIT Gwendall (gwendall.petit@ec-nantes.fr)
LECOEUVRE Mireille

Développements :

GOUGE Adam - IRSTV / IFSTTAR
LECOEUVRE Mireille - IRSTV / IFSTTAR
BOCHER Erwan – IRSTV / CNRS
PETIT Gwendall – IRSTV / École Centrale de Nantes

Responsables administratifs :

BONIN Olivier - LVMT / IFSTTAR
HUBERT Jean-Paul - DEST / IFSTTAR
HEGRON Gérard - AME / IFSTTAR

IRSTV

Institut de Recherche en Sciences et Techniques de la Ville – FR CNRS 2488

www.irstv.fr

École Centrale de Nantes
1 rue de la Noë
BP 92101
44321 Nantes Cedex 3



Projet collaboratif LCPC, INRETS et IGN subventionné par le MEDDE/DRI

Service destinataire : mission des Pôles Scientifiques et Techniques

MEDDE / CGDD / DRI / AST3

Table des matières

Introduction.....	4
1. Contexte.....	5
2. La théorie des graphes.....	8
2.1 Définitions.....	8
2.1.1 Graphe.....	8
2.1.2 Chemins.....	8
2.1.3 Plus courts chemins.....	9
2.1.4 Composantes (fortement) connexes.....	9
2.1.5 Centralité d'intermédiarité et centralité de proximité.....	9
2.2 Algorithmes.....	10
2.2.1 Parcours en largeur et en profondeur.....	10
2.2.2 L'algorithme de Dijkstra-Moore.....	10
2.2.3 La méthode des contractions hiérarchiques.....	11
3. Les outils.....	12
3.1 Critères d'analyse.....	12
3.2 Les outils existants.....	14
3.2.1 OpenTripPlanner (OTP).....	14
3.2.2 Graphhopper.....	15
3.2.3 JGraphT.....	16
3.2.4 JGraphT-SNA.....	17
3.2.5 Jung.....	18
Conclusion.....	19
4. H2Network.....	20
4.1 Architecture et fonctionnalités.....	20
4.2 Description des fonctions SQL.....	23
4.2.1 Fonction de construction du graphe.....	23
4.2.2 Fonctions d'analyse.....	25
4.2.2.1 Paramètres communs aux fonctions d'analyses.....	25
4.2.2.2 ST_ConnectedComponents.....	26
4.2.2.3 ST_ShortestPath.....	28
4.2.2.4 ST_ShortestPathLength.....	30
4.2.2.5 ST_Accessibility.....	33
4.2.2.6 ST_GraphAnalysis.....	36
5. Application.....	44
5.1 Préparation du réseau routier.....	44
5.1.1 Analyse du réseau routier.....	44
5.1.2 Corrections géométriques du réseau routier.....	48
5.1.3 Pondération des tronçons du réseau routier.....	55
5.1.4 Raccordement de nouveaux points de navigation au réseau routier.....	61
5.2 Calcul des temps de trajet domicile-travail sur la France métropolitaine.....	66
5.2.1 Introduction.....	66
5.2.2 Données d'entrées et pré-traitements.....	66
5.2.3 Importation des données dans OrbisGIS.....	67
5.2.3.1 Préambule.....	67
5.2.3.2 Importation des données.....	68

5.2.4 Pondération des tronçons routiers.....	72
5.2.5 Correction géométrique des tronçons routiers.....	77
5.2.6 Raccordement des points de navigation aux tronçons routiers.....	78
5.2.7 Création du graphe routier.....	81
5.2.8 Calcul du distancier.....	82
5.2.8.1 Création des destinations.....	82
5.2.8.2 Création du distancier.....	83
5.2.9 Analyse succincte des résultats.....	85
Conclusion et perspectives.....	88
Ressources en ligne.....	89
Références.....	90
Glossaire.....	92
Acronyme.....	92
Index des figures.....	93

Introduction

Dans le cadre du projet Belgrand - Grand Équipement Bases de Données (GEBD) une bibliothèque pour manipuler des graphes a été développée. Cette bibliothèque appelée H2Network comprend un ensemble de fonctionnalités telles que le calcul d'itinéraires, le calcul de distances ou encore d'indices de centralité de proximité ou d'intermediarité. Ces fonctionnalités permettent par exemple de mesurer l'accessibilité de certains nœuds dans un réseau routier et d'automatiser le calcul de distances entre un ensemble de nœuds, par exemple de communes (distancier). Combiner avec des données démographiques, l'utilisateur peut mettre en perspective la distribution des équipements sur un territoire par rapport à celle de la population.

Nous présentons dans ce rapport l'architecture et les fonctionnalités d'H2Network (*Partie 4*) ainsi que des méthodes pour manipuler des graphes routiers conjointement avec des données géographiques. Ces méthodes sont appliquées pour étudier les distances domicile travail sur la France (*Partie 5*). Ces deux parties sont précédées de 3 parties. Dans la première partie, nous donnons des éléments de contexte qui ont conduit à la mise en place de ce travail. Dans la seconde partie, le lecteur trouvera des définitions et une description des principaux algorithmes utilisés en théorie des graphes. Enfin, la troisième partie présente une liste non exhaustive des outils open source disponibles. Nous analysons ces outils au regard des besoins du projet.

1. Contexte

Les besoins en distancier ou en mesures d'accessibilités sont récurrents dans les études de mobilité ou de développement local. Ces distanciers peuvent être obtenus par des calculs d'itinéraires, généralement effectués à l'aide de logiciels spécialisés. Cependant, les outils du commerce ou disponibles sur l'Internet satisfont rarement les besoins des chercheurs, qui voudraient :

- pouvoir travailler sur n'importe quel réseau, que ce soient des réseaux actuels, du passé, ou encore des projections dans le futur ; un distancier permet alors de traduire en gain de temps et d'accessibilité le développement du réseau ;
- paramétrer librement les poids utilisés sur les tronçons du réseau, en fonction de la nature des tronçons, de leur appartenance à des zones géographiques (aires urbaines par exemple), ou encore de la topographie ;
- produire un distancier à partir de contraintes de localisation de nœuds de calcul. par exemple les chefs-lieux des communes de France ou encore la localisation d'équipements publics sur un territoire;
- analyser les particularités des distances par itinéraires, les vitesses pratiquées (vitesse moyenne, maximale, minimale), les profils topographiques (pentes, sinuosité) ;
- conditionner les calculs de distance sur des itinéraires établis à d'autres données, comme la charge du réseau, les points accidentogènes ;
- contraindre le calculs d'itinéraire, comme par exemple le choix de l'itinéraire qui s'écarte le moins du vol d'oiseau, ou une répartition probabiliste entre le plus court chemin et les chemins alternatifs (affectation probit).

Les produits du commerce appartiennent globalement à deux familles : les services d'usages gratuits sur internet, comme Mappy¹, ViaMichelin², Google Maps³ qui ne sont pas conçus pour réaliser des calculs en masse, et les outils métiers, plutôt destinés aux concepteurs et exploitants de réseau de transport, comme TransCAD⁴ ou Cube⁵. Les SIG proposent de plus en plus des modules de calcul d'itinéraires et de méthodes de représentation des résultats (isochrones, grille d'interpolation des distances...) , comme ChronoMap⁶ pour MapInfo⁷ et le Network Analyst⁸ de ArcGIS. Les bases de données spatiales comme PostGreSQL⁹ et son cartouche PostGIS¹⁰ proposent également ce type d'analyses avec l'extension pgRouting¹¹.

1 <http://fr.mappy.com> consulté en octobre 2014.

2 <http://www.viamichelin.fr/> consulté en octobre 2014.

3 <https://www.google.com/maps> consulté en octobre 2014.

4 <http://www.caliper.com/tcovu.htm> consulté en octobre 2014.

5 <http://www.citilabs.com/products/cube> consulté en octobre 2014.

6 <http://www.opti-time.com/fr/calcul-zone-chalandise-logiciel-Chronomap> consulté en octobre 2014.

7 <http://www.mapinfo.com/> consulté en octobre 2014.

8 http://www.esrifrance.fr/Network_Analyst.aspx consulté en octobre 2014.

9 <http://www.postgresql.org/> consulté en octobre 2014.

10 <http://postgis.org/> consulté en octobre 2014.

11 <http://pgrouting.org/> consulté en octobre 2014.

D'autres produits, de diffusion plus restreinte, se rapprochent d'une partie des besoins que nous avons évoqués. Ils sont parfois développés à partir de questions de la recherche, comme OdoMatrix¹². Cependant, ils présentent l'inconvénient d'être fermés, tant en ce qui concerne les données d'entrée (réseaux utilisés, points d'intérêt) que les algorithmes et paramétrages qui sont utilisés. Leurs usages sont donc limités pour développer de nouvelles questions de recherche sur l'analyse des réseaux.

Les distanciers commerciaux étaient, et sont encore, des produits onéreux. Lors de leur mise au point, il y a une dizaine d'années, le coût de la licence couvrait à la fois le développement du logiciel et le droit d'utilisation des données du réseau, qui restait enfermé dans la boîte noire du logiciel.

La diffusion par l'IGN¹³ de réseaux navigables France entière (Route 120^{®14} et Route 500^{®15}), de fichiers contenant les coordonnées des chefs-lieux de commune et centroïdes de commune (GeoFLA^{®16}), ainsi que la multiplication des projets open source (PGRouting, OpenTripPlanner, ...), le développement de l'open data communautaire (OpenStreetMap¹⁷) et institutionnelle (OpenDataFrance¹⁸, Data.gouv.fr¹⁹) rend d'autant plus intéressante la possibilité d'utiliser plusieurs réseaux, et incite à se concentrer sur les possibilités de paramétrage des réseaux et les algorithmes de calcul.

Dans le cadre du projet réseau-GEBD, le présent développement souhaite trouver un juste milieu entre les outils destinés aux professionnels du transport et aux géomaticiens, souvent difficiles à prendre en main et relativement fermés, et les distanciers prêts à l'emploi qui offrent peu ou pas de possibilité de variantes. L'objectif est donc, parallèlement à la bibliothèque de calcul d'itinéraires proprement dite, de proposer des patrons d'application (*templates*), sous la forme de scripts SQL, qui détaillent l'ensemble des opérations de préparation des données : calcul et paramétrage du réseau, choix des points d'intérêt, et enfin analyse et export des résultats. Le langage SQL présente l'avantage d'être largement utilisé en dehors du domaine de la géomatique, et donc d'être moins déroutant pour un utilisateur non informaticien qu'un ensemble d'opérations à piloter via des menus, ou une macro à exécuter dans un langage propriétaire.

La plateforme OrbisGIS²⁰ est un Système d'Information Géographique Open Source²¹ dont l'une de particularités majeures, outre de disposer de l'ensemble des fonctions de bases (affichage, cartographie, édition de données) est d'utiliser un système de gestion de données spatiales relationnelles interne pour modéliser et questionner des données géographiques et attributaires (Bocher et Petit, 2012). Ce système s'appuie sur la base de données Open Source H2 Database²² pour laquelle un cartouche d'opérateurs et de prédicats spatiaux a été développé : H2GIS²³ (Gouge *et al*, 2014). Intégré au cœur de la plate-forme OrbisGIS, le couple "H2 Database - H2GIS" offre un environnement idéal pour

12 <http://www2.dijon.inra.fr/cesaer/en/membres/mohamed-hilal/#odomatrix> consulté en 2014.

13 <http://www.ign.fr/> consulté en octobre 2014.

14 <http://professionnels.ign.fr/route120> consulté en octobre 2014.

15 <http://professionnels.ign.fr/route500> consulté en octobre 2014.

16 <http://professionnels.ign.fr/geofla> consulté en octobre 2014.

17 <http://www.openstreetmap.org/> consulté en octobre 2014.

18 <http://opendatafrance.net/> consulté en octobre 2014.

19 <https://www.data.gouv.fr/fr/> consulté en octobre 2014.

20 <http://www.orbisgis.org/> consulté en octobre 2014.

21 OrbisGIS est diffusé sous la licence GPL version 3.

22 <http://www.h2database.com/> consulté en octobre 2014.

23 <http://www.h2gis.org/> consulté en octobre 2014.

construire des analyses spatiales, modéliser les données d'un réseau ou encore constituer des bibliothèques de scripts paramétrables qui lient traitements de l'information et représentation des résultats. De plus, construite sur la base d'une architecture extensible dont chaque composant peut évoluer au gré des besoins des utilisateurs, la plate-forme OrbisGIS est un réceptacle idéal pour construire de nouveaux outils pour la recherche. Pour cela, l'utilisateur dispose de plusieurs points d'entrée :

- possibilité d'étendre la grammaire SQL via l'ajout de fonctions sur mesure,
- possibilité d'écrire des scripts avancés via le langage de programmation Groovy,
- possibilité de développer des extensions (plug-ins) via l'utilisation des API de la plate-forme.

Ces avantages nous ont amené à considérer la plate-forme OrbisGIS pour développer une bibliothèque de calcul de plus courts chemins, nécessaire à la mise au point d'un distancier. Ce développement ouvre la porte à d'autres analyses de graphes, notamment la très grande famille des indicateurs de réseaux classiques en géographie, et plus généralement en analyse de réseaux : *closeness centrality*, *betweenness centrality*, *modularity*, *eigenvector centrality*, ainsi que les calculs de composantes connexes, de diamètre, etc. Ce domaine, stimulé par les analyses de réseaux sociaux et de données sur le Web, est aujourd'hui en plein essor.

Enfin, soulignons que le choix de l'open source n'est pas simplement le choix de la gratuité, mais bien de l'ouverture : transparence sur les algorithmes, sur leur implémentation, possibilité pour tous d'étendre les fonctionnalités ou de corriger des erreurs. Ce choix permet également à d'autres plates-formes SIG de pouvoir bénéficier des fonctionnalités de la bibliothèque, puisqu'elle reprend des éléments standards dans sa conception et se prête ainsi à l'interopérabilité.

Le distancier ouvert prend tout son intérêt lorsqu'on s'intéresse à l'évolution de l'accessibilité due au développement des réseaux au cours du temps. Cet outil est très complémentaire des autres lots méthodologiques du projet.

Dans le cadre de plusieurs projets de recherche menés par des membres du réseau Belgrand, les réseaux ferrés et autoroutiers depuis 1975 ont été saisis à partir de cartes scannées et mises à disposition par l'IGN, et seront prochainement publiés. Une plate-forme de saisie collaborative permettra la montée en charge pour la saisie du réseau routier dans son intégralité. Le distancier est un élément clé de ces projets.

Pour conclure, notons que dans le cadre des réflexions sur la statistique publique, le distancier Belgrand peut être une des réponses à la demande du CNIS²⁴ :

Mesure de l'accessibilité physique

Afin de pouvoir partager entre acteurs les diagnostics sur l'accessibilité physique et favoriser l'usage des données géoréférencées mises à disposition par le service statistique public, il serait utile de mettre à disposition des utilisateurs un distancier qui aurait de plus la qualité de pouvoir rendre compte de situations qui nécessitent un positionnement à l'infracommunal.

Rapport du Cnis – Moyen terme 2014-2018 Actes des rencontres et entretiens, avril 2014, p.169

24 http://www.cnis.fr/files/content/sites/Cnis/files/Fichiers/publications/rapports/2014/RAP_2014_137_actes_rencontres_entretiens.pdf

2. La théorie des graphes

2.1 Définitions

2.1.1 Graphe

Un graphe est un objet mathématique comprenant des points (appelés nœuds ou sommets) dont certaines paires sont reliées par un ou plusieurs lien(s) (appelés arcs ou arêtes). Si l'arc du nœud v vers le nœud w peut être distingué de l'arc de w vers v , le graphe est *orienté* ; sinon il est *non orienté*. Si un graphe ne contient ni des boucles (un arc d'un nœud à lui-même) ni des arcs multiples (plusieurs arcs qui relient la même paire de nœuds), il est *simple*.

Mathématiquement, un *graphe simple non orienté* est un couple $G=(V, E)$, où V est l'ensemble de nœuds du graphe, et

$$E \subset \{\{v_1, v_2\} : v_1 \neq v_2, v_i \in V\}$$

est l'ensemble des arcs.

Dans le cas d'un *graphe simple orienté*, on écrit

$$E \subset \{(v_1, v_2) : v_1 \neq v_2, v_i \in V\},$$

car l'ordre du couple (v_1, v_2) est important.

Dans le cas d'un graphe général (qui permet par exemple des boucles et/ou des arcs multiples) on écrit $G=(V, E, \gamma)$, où

$$\gamma : E \rightarrow V \times V$$

est une fonction d'incidence qui associe à chaque arc une paire (ou un couple, si le graphe est orienté) de nœuds.

Si chaque arc est associé à un poids, le graphe est *pondéré*. Cette association est exprimée par la fonction du poids

$$w : E \rightarrow R$$

et on écrit $G=(V, E, w)$. Dans le cas d'un graphe *non pondéré*, on attribue le même poids à tous les arcs : on a $w=1$, c'est-à-dire $w(e)=1$ quel que soit $e \in E$.

2.1.2 Chemins

Soit $G=(V, E)$ un graphe non orienté et $v, w \in V$. Une *chaîne* $p(v, w)$ de v à w est une suite finie d'arcs consécutifs reliant v à w :

$$p(v, w) = \{v_0=v, v_1\}, \{v_1, v_2\}, \dots, \{v_{n-1}, v_n=w\}$$

Si G est orienté, on appelle la suite

$$p(v, w) = (v_0=v, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n=w)$$

un *chemin*. Par simplicité, on fera souvent la substitution de *chemin* pour *chaîne*.

2.1.3 Plus courts chemins

La longueur (ou coût) du chemin $p(v, w)$ est la somme des poids de chaque arc :

$$c(p) = \sum_{i=1}^n w(v_{i-1}, v_i).$$

Si la valeur de $c(p)$ est minimale parmi tous les chemins de v à w , p est dit un *plus court chemin* de v à w . Dans ce cas, on définit la distance de v à w :

$$d(v, w) = \min\{c(p) : p \text{ est un chemin de } v \text{ à } w\}.$$

Les plus courts chemins ne sont pas uniques en général.

2.1.4 Composantes (fortement) connexes

Un graphe $G=(V, E)$ non orienté est dit *connexe* si pour toute paire $v, w \in V$ de nœuds, il existe une chaîne de v à w . Un sous-graphe connexe maximal d'un graphe non orienté quelconque est dit une *composante connexe* de ce graphe.

Si G est orienté, il est dit *fortement connexe* si pour toute paire $v, w \in V$ de nœuds, il existe un chemin de v à w et un chemin de w à v . Un sous-graphe fortement connexe maximal d'un graphe orienté quelconque est dit une *composante fortement connexe* de ce graphe.

2.1.5 Centralité d'intermédierité et centralité de proximité

Soit $d(s, t)$ la distance de $s \in V$ à $t \in V$, c'est-à-dire la longueur minimale de tous les chemins de s à t . Par convention, on a $d(s, s) = 0$ quel que soit $s \in V$.

Soit σ_{st} le nombre de plus courts chemins de $s \in V$ à $t \in V$. Soit $\sigma_{ss} = 1$ (par convention). Soit $\sigma_{st}(v)$ le nombre de plus courts chemins de s à t qui contiennent $v \in V$ comme nœud intermédiaire.

Quel que soit $v \in V$, nous définissons :

$$C_C(v) = \left(\sum_{t \in V} d(v, t) \right)^{-1} \quad \text{centralité de proximité (closeness centrality)}$$

$$C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad \text{centralité d'intermédierité (betweenness centrality)}$$

La définition de la centralité d'intermédierité pour les arcs est similaire.

Si $C_C(v), v \in V$ est grand, v est relativement proche aux autres nœuds du graphe. Si $C_B(v), v \in V$ est grand, v se situe sur un nombre relativement grand de plus courts chemins.

2.2 Algorithmes

Une multitude d'algorithmes existent pour parcourir un graphe. Nous en présentons les principaux.

2.2.1 Parcours en largeur et en profondeur

Le parcours en profondeur (Depth-First Search) consiste à visiter tous les sommets et tous les arcs d'un graphe G . Il s'agit à partir d'un sommet qui n'a pas encore été visité, de le marquer pour signifier qu'il est en cours de visite et de continuer le parcours vers le premier sommet accessible et ainsi de suite jusqu'à ce qu'il n'y ait plus un seul sommet qui n'a pas été visité. L'algorithme de parcours en profondeur peut être étendu pour trouver un chemin entre 2 sommets ou trouver un cycle dans un graphe. La complexité en temps de l'algorithme est de $O(n+m)$ où n est le nombre de noeuds du graphe et m le nombre d'arcs.

Le parcours en largeur (Breadth-First Search) consiste à visiter d'abord les sommets d'un graphe à distance 1 puis ceux à distance 2, etc. La distance s'entend en nombre d'arcs parcourus en partant du sommet initial. Chaque sommet est marqué pour s'assurer qu'on n'y passe qu'une seule fois. Pour réaliser ce parcours, il faut également garder en mémoire les sommets que l'on devra visiter "plus tard", c'est-à-dire ceux qu'il faudra visiter une fois que l'on aura terminé la visite de tous les sommets à la même distance. L'algorithme de parcours en profondeur peut être étendu pour trouver le plus court chemin entre 2 sommets ou trouver un cycle simple dans un graphe. La complexité en temps de l'algorithme est de $O(n+m)$.

2.2.2 L'algorithme de Dijkstra-Moore

L'algorithme de Dijkstra (1956), Moore (1957) permet d'obtenir le plus court chemin d'un sommet 1 à tous les autres. Le graphe doit être connexe (orienté ou non) et le poids lié aux arcs peut-être positif ou nul. L'algorithme de Dijkstra est souvent qualifié d'algorithme glouton puisque l'ensemble des chemins d'un sommet vers les autres sommets du graphe sont calculés. Il reste néanmoins très efficace et fiable à condition de disposer de la puissance matériel nécessaire (mémoire) à contrario de l'algorithme A^* qui bien que plus rapide et moins gourmand en ressource ne donne pas des résultats toujours exacts (la première solution trouvée doit être l'une des meilleures).

L'algorithme de Dijkstra a prouvé son utilité dans le calcul des itinéraires routiers. Le poids des arcs pouvant être la distance (pour le trajet le plus court), le temps estimé (pour le trajet le plus rapide)...

La complexité en temps de l'algorithme est de $O(n^2)$. Cette complexité peut être réduite à $O(m+n\log(n))$ en utilisant un tas de Fibonnaci (Fredman, Tarjan, 1987).

2.2.3 La méthode des contractions hiérarchiques

La méthode des contractions hiérarchiques est une méthode développée pour optimiser les calculs de plus court chemin (Geisberger *et al*, 2008) . Elle a pour objectif de réduire le temps de parcours d'un graphe en établissant des relations directes entre ses sommets, contrairement à l'approche *Highway Hierarchies* qui classe les arcs (Sanders, Schultes, 2005) . Prenons un graphe G composé de 3 nœuds A, B, C , ordonnés avec les tuples {A -> B}, {B -> C} dont les poids respectifs sont 1 et 3. Le chemin pour aller de A à C est représenté par le parcours {A, B, C} dont la distance sera $1 + 3 = 4$. La méthode des contractions hiérarchiques consiste à établir un lien entre A et C en ajoutant le raccourci {A -> C} dans le graphe. Le calcul des raccourcis est basé sur une hiérarchisation des nœuds qui sont classés par ordre d'importance. Le lecteur trouvera une description complète de l'algorithme dans Geisberger *et al*, 2008²⁵.

Notons que si cette méthode permet de réduire sensiblement les temps de parcours d'un graphe, une recherche de plus court chemin est 5 fois plus rapide qu'avec l'algorithme de Dijkstra y compris avec une approche bidirectionnelle, en revanche l'approche souffre de quelques contraintes.

- L'étape de hiérarchisation des nœuds nécessite un pré-traitement de l'ensemble des sommets du graphe et les contractions ajoutées (raccourcis) augmentent la taille du graphe. Pour des graphes volumineux, il est donc nécessaire de mettre en place des stratégies pour délester la mémoire au profit d'un stockage sur disque.
- Si le graphe est modifié (ajout, suppression d'un nœud ou d'un arc), la hiérarchisation des nœuds doit être reconstruite pour établir les nouvelles contractions. Néanmoins, dans de nombreuses applications, les analyses sur les graphes sont répétées en modifiant le poids des arcs. C'est le cas d'un calcul de plus court chemin avec prise en compte des distances ou du temps de trajet (Protsenko, 2010).

Cette rapide synthèse montre qu'il existe plusieurs approches pour calculer le plus court chemin. Nous retiendrons de la littérature les éléments suivants :

- L'algorithme de Dijkstra bien que simple à mettre en œuvre n'est valable tant que les coûts sont positifs.
- Pour être performant, il est indispensable d'utiliser d'autres structures de données tel que le Tas de Fibonacci.
- Pour prendre en compte, les cycles dit absorbants (boucle dont le poids total est négatif), il est nécessaire d'utiliser d'autres approches comme le propose Bellman-Ford (Grondran, Minoux, 1986).
- De meilleures performances sont obtenues avec de nouvelles approches comme les Contraction Hiérarchiques.

25 <http://algo2.iti.kit.edu/download/presentation.pdf> consulté en septembre 2014.

3. Les outils

Il existe de nombreux logiciels de calcul de parcours de graphe, aussi bien open-source que propriétaires. L'un des objectifs de cette tâche étant de doter le Système d'Information Géographique open source OrbisGIS de capacités d'analyse de graphe, nous avons concentré notre étude sur les outils open source existants. L'utilisation d'une plateforme open source est motivée pour deux raisons :

- maîtriser l'intégralité du modèle et des algorithmes utilisés pour les adapter aux questions de recherche développées par les acteurs du projets,
- fournir une plateforme libre d'utilisation, de distribution qui puisse en outre servir une communauté scientifique mais également s'intégrer dans les dispositifs d'enseignements des partenaires du projet et au-delà.

3.1 Critères d'analyse

Les critères d'analyse et de sélection des outils existants sont contraints par l'environnement et l'architecture de la plate-forme OrbisGIS.

Le Système d'Information Géographique OrbisGIS est développé en langage JAVA et son architecture repose sur l'utilisation du standard OSGi²⁶. L'ajout d'un composant (ou plugin) dans OrbisGIS doit respecter le canevas imposé par ce standard. Chaque développement est déclaré comme un service propre à la plateforme. Ce service dispose des propriétés suivantes :

- des métadonnées pour qualifier le composant : auteur, description, dernière date de mise à jour, information sur la licence...
- des "activateurs" pour gérer le statut du composant dans la plateforme (arrêt, redémarrage, mise à jour).
- un système automatique de résolution des dépendances. Si le composant nécessite des bibliothèques tierces pour fonctionner, le système vérifie leur présence et si possible les télécharge sur un serveur. L'équipe de développement d'OrbisGIS utilise à cet effet une plateforme de gestion centralisée de ses composants, visible à l'adresse <http://nexus.orbisgis.org>.

OrbisGIS utilise la base de données relationnelle H2 Database comme moteur d'accès et de manipulation des données. Cette base de données a été étendue par l'équipe d'OrbisGIS pour stocker et interroger des géométries. Un nouveau cartouche appelé H2GIS permet d'ajouter l'équivalent de PostGIS en terme de fonction d'analyses spatiales. La bibliothèque d'analyse de graphes utilisée dans ce projet devra pouvoir s'interfacer avec H2GIS afin de tirer profit d'une part des fonctions existantes (triangulation et extraction de contour pour construire des iso-surfaces), d'autre part de la puissance du langage SQL pour automatiser des traitements mais également des avantages des structures relationnelles pour modéliser des données.

²⁶ <http://www.osgi.org/Technology/WhatIsOSGi> consulté en septembre 2014

En complémentarité de ces critères de plateforme de développement, nous avons retenus les éléments suivants :

1. **Fonctionnalités de bases** La bibliothèque doit fournir des fonctionnalités de bases pour analyser les réseaux tels que des algorithmes de calcul de plus court chemin, des indices pour qualifier un graphe (centralité, proximité, connectivité).

2. **Extensibilité.** La bibliothèque doit être extensible tant du point de vue de l'ajout de nouveaux algorithmes d'analyse que des possibilités d'adaptation du modèle de données du graphe. Les arcs et nœuds qui composent le graphe doivent par exemple pouvoir porter des propriétés autre que le poids.

3. **Généricité.** La bibliothèque doit tirer profit des capacités SIG et ainsi s'interfacer avec différents formats de données géographiques. Son utilisation doit être paramétrable.

4. **Rapidité des calculs et capacité à traiter de gros volumes de données.** Les calculs doivent pouvoir se faire sur des gros volumes de données et dans des temps raisonnables : moins d'une seconde pour calculer un plus court chemin d'un nœud à un autre nœud sur le réseau routier français Route 120[®] serait acceptable.

5. **Dépendances.** L'intégration d'une bibliothèque externe ajoute forcément des dépendances, qui augmentent la taille de téléchargement d'OrbisGIS et/ou de ses plugins. Les dépendances ajoutées doivent être minimales aussi bien en taille qu'en nombre.

6. **Interface de programmation (API), couverture de code par des tests unitaires, qualité de la documentation.** Les fonctionnalités de la bibliothèque, le modèle de graphe doivent pouvoir être manipulés via une interface programmatique documentée. Les codes sont validés systématiquement par des tests unitaires qui reprennent les conditions d'application et de validation des algorithmes exprimées dans la littérature scientifique .

7. **Méthodologie de développement.** Le développement de la bibliothèque doit suivre un cadre normé. L'ensemble des codes sont accessibles à la communauté via une plateforme de gestion. Les codes sont développés en utilisant les règles d'un projet Maven²⁷. A l'issue d'une modification majeure, la bibliothèque est compilée et publiée dans un entrepôt accessible en ligne, idéalement Maven Central Repository²⁸.

27 <http://maven.apache.org/> consulté en septembre 2014.

28 <http://search.maven.org> consulté en septembre 2014.

3.2 Les outils existants

Il existe une multitude de projets open source autour des outils de parcours de graphes. Ces projets se sont multipliés ces dernières années avec notamment la montée en puissance du référentiel routier communautaire OpenStreetMap²⁹. Cette base de données fournit une information spatialisée sur l'offre de transport routier et cyclable. Y sont notamment décrits le type de voie, la vitesse, le sens de circulation. La communauté open source s'est très rapidement emparé de cette source d'informations pour mettre en œuvre de nouvelles techniques plus performantes (cf. section 1.2). C'est le cas des projets Graphhopper³⁰ ou encore OSRM³¹ qui utilisent l'approche par contraction pour optimiser les recherches de plus court chemin.

Nous présentons une sélection des outils existants.

3.2.1 OpenTripPlanner (OTP)

OTP^{32,33} est un service web en JAVA destiné à calculer des itinéraires multi-modaux (en transport, en voiture, à pied, à vélo, et en fauteuil roulant). Le code de l'application est disponible sur la plateforme GitHub à l'adresse suivante : <https://github.com/opentripplanner/OpenTripPlanner>.

Caractéristiques générales
<ul style="list-style-type: none"> • Temps de calcul : à peu près 100ms à l'échelle de la ville. • Prise en compte de : <ul style="list-style-type: none"> ◦ l'orientation des routes ◦ pistes cyclables ◦ données d'altitude • Licence : LGPL 3
Données d'entrée
<ul style="list-style-type: none"> • Import de données <ul style="list-style-type: none"> ◦ TFS ◦ Shapefiles ◦ OpenStreetMap ◦ National Elevation Dataset • Standards

29 http://wiki.openstreetmap.org/wiki/Routing/online_routers consulté en septembre

30 <https://graphhopper.com/> consulté en septembre 2014.

31 <http://project-osrm.org/> consulté en septembre 2014.

32 <http://www.opentripplanner.org> consulté en septembre 2014.

33 http://belgrand-gebd.ifsttar.fr/fileadmin/seminaire/_9/byrd-presentation.pdf consulté en septembre 2014.

<ul style="list-style-type: none"> • General Transit Feed Specification (GTFS) pour les horaires des transports <ul style="list-style-type: none"> ◦ GTFS-Realtime pour le positionnement et retard des transports, ainsi que des alertes
Fonctionnalités - Algorithmes
<ul style="list-style-type: none"> • Calcul d'isochrones et d'indicateurs d'accessibilité • Requêtes <ul style="list-style-type: none"> ◦ d'une seule origine à plusieurs destinations, ◦ de plusieurs origines à plusieurs destinations.
Statut du projet
<ul style="list-style-type: none"> • Année de démarrage : 2009 • Développement actif. Mise à jour hebdomadaire du code. • Nouvelle architecture est en cours. • Dernière version : 0.11.0 le 01/09/2014

3.2.2 Graphhopper

Graphhopper est une bibliothèque java de calcul d'itinéraires (Peter, 2014). Sa particularité est d'utiliser la méthode des contractions hiérarchiques. Le code source est disponible à l'adresse <https://github.com/graphhopper/graphhopper>.

Caractéristiques générales
<ul style="list-style-type: none"> • Calcul d'itinéraires en voiture, à vélo, à pied, etc. (mais pas de multi-modal par défaut) • Structures de données et algorithmes peu gourmands en mémoire vive ; utilisable sur des appareils avec moins de 32MB de mémoire vive. Utilise des primitives objets de la bibliothèque trove4j³⁴. • Licence : Apache License 2.0
Données d'entrée
<ul style="list-style-type: none"> • Intégration avec données OpenStreetMap (osm, pbf) comprenant <ul style="list-style-type: none"> ◦ le type de route, ◦ le matériau de construction ("surface"), ◦ les barrières, ◦ les accès restreints,

³⁴ <http://trove.starlight-systems.com/> consulté en septembre 2014.

<ul style="list-style-type: none"> ◦ les ferries, ◦ etc. • Prise en compte de données d'altitude
Fonctionnalités - Algorithmes
<ul style="list-style-type: none"> • Utilisation de "Contraction Hierarchies" pour des calculs très rapides, • Algorithmes A*/A* bidirectionnel, Dijkstra/Dijkstra bidirectionnel, • Calcul de chemin de nœud à nœud, • Calcul de chemin de n-nœuds à n-nœuds, • Statistiques sur un graphe, • Recherche de points d'intérêt par rapport à un nœud.
Statut du projet
<ul style="list-style-type: none"> • Année de démarrage : 2013 • Développement actif. Mise à jour hebdomadaire du code • Dernière version : 0.3 le 13/05/2014

3.2.3 JGraphT

JGraphT³⁵ est l'une des plus ancienne bibliothèque en JAVA pour la manipulation de graphes, la première version datant de 2002. Le code source de l'application est disponible à l'adresse <https://github.com/jgrapht/jgrapht>.

Caractéristiques générales
<ul style="list-style-type: none"> • Structure de données interne en mémoire pour stocker un graphe, • Prise en compte de graphes orientés (ou non), pondérés (ou non), de multigraphes et de pseudographes, • Licence : LGPL 2.1 et EPL 1.0.
Données d'entrée
<ul style="list-style-type: none"> • Modèle de graphe interne stocké en mémoire dans des tables de hachage. Le modèle de graphe est implémentée via des interfaces non-typées (génériques) ce qui permet de l'étendre pour utiliser d'autres structures de stockage (fichier plat ou base de données relationnelles), • Export d'un graphe au format GraphML³⁶ ou DOT³⁷.

³⁵ <http://trove.starlight-systems.com/> consulté en septembre 2014.

³⁶ <http://en.wikipedia.org/wiki/GraphML> consulté en septembre 2014.

³⁷ http://en.wikipedia.org/wiki/DOT_language consulté en septembre 2014.

Fonctionnalités - Algorithmes
<ul style="list-style-type: none"> • Calcul de chemin de nœud à nœud, • Calcul de chemin de n-nœuds à n-nœuds, • Multitudes d'algorithmes de parcours : <ul style="list-style-type: none"> ◦ Bellman-Ford, Dijkstra, Edmonds Blossom Shrinking, Floyd-Warshall, Hamilton Path Problem ..., ◦ Algorithme de parcours optimisé (tas de Fibonacci). • Utilitaires pour les graphes : recherche de cycle, extraction de sous-graphes...
Statut du projet
<ul style="list-style-type: none"> • Année de démarrage : 2003, • Développement actif. Mise à jour hebdomadaire du code, • Très importante communauté d'utilisateurs, • Dernière version : 0.9.0 le 06/12/2013.

3.2.4 JGraphT-SNA

JGraphT-SNA³⁸ est une extension de la bibliothèque JGraphT dédiée à l'analyse des réseaux sociaux (Atmakuri-Davidsen and Padmavathamma, 2014). Elle apporte des fonctionnalités manquantes à JGraphT notamment en ce qui concerne les mesures de centralités : degré de centralité, centralité intermédiaire, centralité de proximité, centralité de vecteur propre (*eigenvector centrality*)... Ces fonctionnalités fournissent des indicateurs structurels sur le graphe et permettent de qualifier le degré d'importance d'un nœud ou d'un arc dans le graphe. Par exemple, la centralité intermédiaire qui mesure la fréquence à laquelle un nœud se trouve sur le chemin liant deux autres nœuds quelconques du graphe peut être utilisée pour identifier l'importance d'un franchissement, pont dans un réseau routier. Celui-ci peut s'avérer un point de passage stratégique. Notons ici que l'implémentation de cet indice dans SNA ne permet pas de prendre en compte les graphes pondérés : Les nœuds du graphe étant estimés à une distance égale à 1.

Statut du projet
<ul style="list-style-type: none"> • Année de démarrage : 2011, • Développement arrêté en août 2012, • Dernière version : 1.2.0 le 08/08/2012.

38 <https://bitbucket.org/sorend/jgraph-t-sna> consulté en octobre 2014.

3.2.5 Jung

JUNG³⁹ (*Java Universal Network/Graph Framework*) est une bibliothèque qui offre la possibilité de modéliser, interroger et visualiser des graphes (O'Madadhain *et al*, 2005).

Caractéristiques générales
<ul style="list-style-type: none">• Structure de données interne en mémoire pour stocker un graphe,• Prise en compte de tous types de graphe (orienté, pondéré, hypergraphe...),• Licence : BSD.
Données d'entrée
<ul style="list-style-type: none">• Import et export d'un graphe au format GraphML,• Modèle de graphe extensible. Les objets nœuds et arcs qui composent la structure du modèle de données peuvent être étendu via des interfaces.
Fonctionnalités - Algorithmes
<ul style="list-style-type: none">• Algorithmes de rendu de graphes (Fruchterman-Rheingold, Kamada-Kawai, Self-Organizing Maps...),• Interfaces graphiques de visualisation en Swing,• Mesures de centralité (Markov et intermédiaire),• Calcul de distances,• Analyse de connectivités,• Mécanismes d'annotation.
Statut du projet
<ul style="list-style-type: none">• Année de démarrage : 2003,• Développement arrêté depuis 2010,• Dernière version : 2.0.1 le 24/01/2010.

39 <http://jung.sourceforge.net/> consulté en septembre 2014.

Conclusion

Les bibliothèques de manipulation de graphe sont largement répandus dans le domaine des logiciels open source. Nous avons présenté une liste des projets les plus abouti et programmés en langage JAVA et ce pour des questions de compatibilité avec la plate-forme OrbisGIS. Cette liste est loin d'être exhaustive. En effet, nous pouvons citer entre autre :

- le projet GraphStream⁴⁰ , bibliothèque dédiée aux graphes dynamiques,
- GRPH⁴¹, une bibliothèque et une plate-forme pour manipuler des graphes. Son développement est motivé par des questions de performance tant en terme de temps de traitement mais également de capacité à manipuler de gros volumes de données.

Au regard, des contraintes de développement et des fonctionnalités attendues, nous avons décidé de retenir la bibliothèque JGraphT. Cette bibliothèque offre plusieurs atouts :

- possibilité de créer des graphes orientés, multiples, pondérés, directs...,
- possibilité d'adapter les objets manipulés par un graphe (nœuds et arcs) grâce à l'utilisation des génériques java,
- disponibilités des principaux algorithmes de parcours de graphe (Dijkstra, Bellman-Ford...),
- forte communauté d'utilisateurs et développeurs : nombreux exemples d'utilisation, nombreuses applications dans la communauté scientifique,
- disponibilité du code source mis à jour régulièrement sur une plate-forme communautaire qui offre également la possibilité de contribuer au projet.

Combinée avec JGraphT-SNA, la bibliothèque JGraphT offre une couverture quasi-complète des fonctionnalités requises par ce projet pour analyser les réseaux routiers : calcul de distance de nœuds à nœuds, indices de centralité. Malgré tout, JGraphT-SNA ne sera pas utilisée car les indices de centralité ne tiennent pas compte des graphes pondérés. Or dans l'analyse d'un réseau routier, les notions de distance et de temps de trajet sont des propriétés indispensables. Pour terminer, ajoutons que le code source du projet n'a pas été amélioré, modifié depuis septembre 2012 et qu'aucune communauté n'est active autour de ce projet.

40 <http://graphstream-project.org/> consulté en septembre 2014.

41 <http://www.i3s.unice.fr/~hogie/grph/> consulté en septembre 2014.

4. H2Network

H2Network est un ensemble de fonctions SQL développées dans le cadre de ce projet pour construire et manipuler des graphes à partir de données géographiques. Dans cette section, nous présentons son architecture et nous décrivons ses fonctions.

4.1 Architecture et fonctionnalités

H2Network est un module SQL pour la base de données H2 Database. Ce module est lié au cartouche spatial H2GIS qui comprend des opérateurs et prédicats spatiaux ainsi que des pilotes pour lire et écrire des données géographiques au format : ShapeFile⁴², GeoJson⁴³, GPX⁴⁴.

H2Network propose à l'utilisateur des fonctions SQL pour les graphes. L'intérêt du SQL est de disposer d'un langage standardisé pour interroger, créer et mettre à jour des données. L'utilisateur peut construire via un ensemble d'instruction des chaînes de traitements répétitives et complexes notamment liées à la préparation des données, la construction d'un graphe et ensuite son exploitation (indice de centralité, calcul de distances). Ces chaînes peuvent ensuite être transformées en script paramétrable pour automatiser des analyses.

Les algorithmes de parcours de graphes sont implémentés dans la bibliothèque Java Network Analyser (JNA) qui tire profit de l'interface programmatique de JGraphT. Cette séparation permet d'utiliser les algorithmes de JNA avec d'autres systèmes de gestion de données (figure 1). Il est par exemple, possible d'établir un lien avec une base de données PostGIS via le standard JDBC⁴⁵ et le langage Groovy⁴⁶.

Les fonctions SQL d'H2Network sont exécutées dans OrbisGIS via la console SQL (figure 2). Cette console permet d'interagir avec les données (tables) qui sont chargées dans OrbisGIS. L'utilisateur dispose d'un panneau sur la droite qui liste l'ensemble des fonctions disponibles. Ce panneau fournit également une documentation pour chacune des fonctions (info-bulle). Les résultats des commandes SQL sont visibles dans le catalogue de données (GeoCatalogue) lorsqu'il s'agit d'une table ou dans une fenêtre de sortie lorsqu'il s'agit d'un résultat textuel ou d'une valeur. Lorsque le résultat d'une instruction SQL produit une table, l'utilisateur peut exploiter les outils de l'interface graphique d'OrbisGIS pour par exemple afficher le compte-tenu de la table ou encore visualiser géographiquement des objets géométriques sous la forme d'une couche dans la fenêtre TOC. Il dispose ainsi des fonctions de bases d'un SIG pour naviguer, interroger et cartographier.

42 <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> consulté en septembre 2014.

43 <http://geojson.org/> consulté en septembre 2014.

44 <http://www.topografix.com/gpx.asp> consulté en septembre 2014.

45 <http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/> consulté en septembre 2014.

46 <http://groovy.codehaus.org/> consulté en septembre 2014.

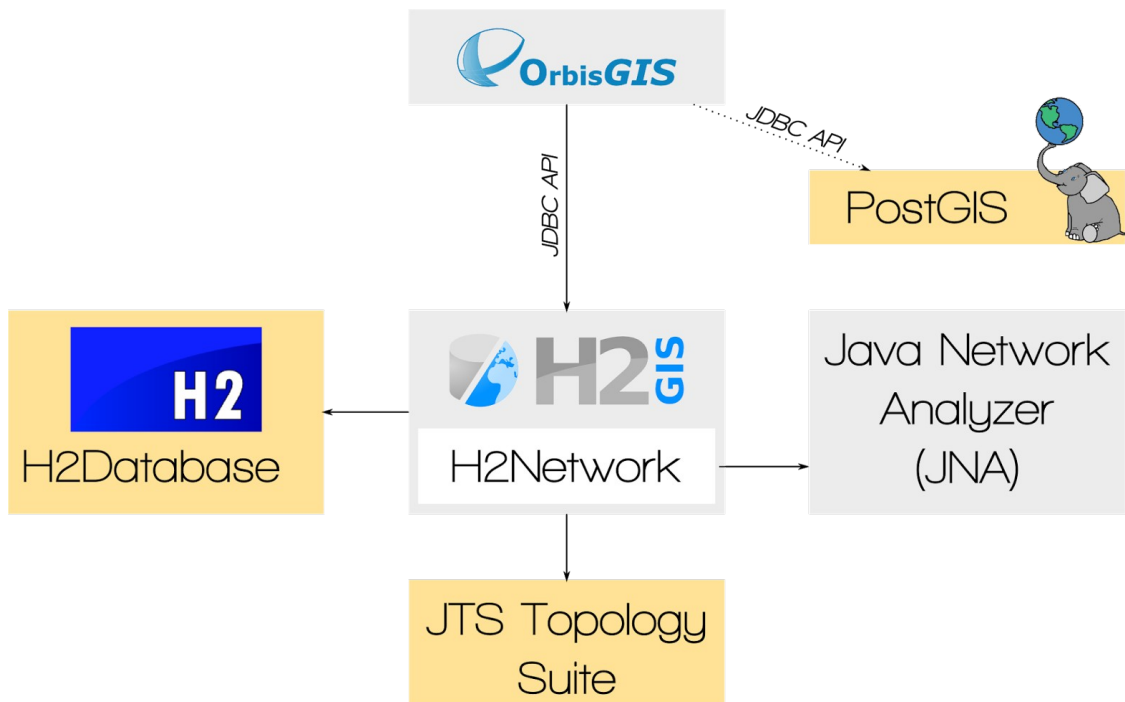


Figure 1: Positionnement de la bibliothèque H2Network dans l'architecture d'OrbisGIS.

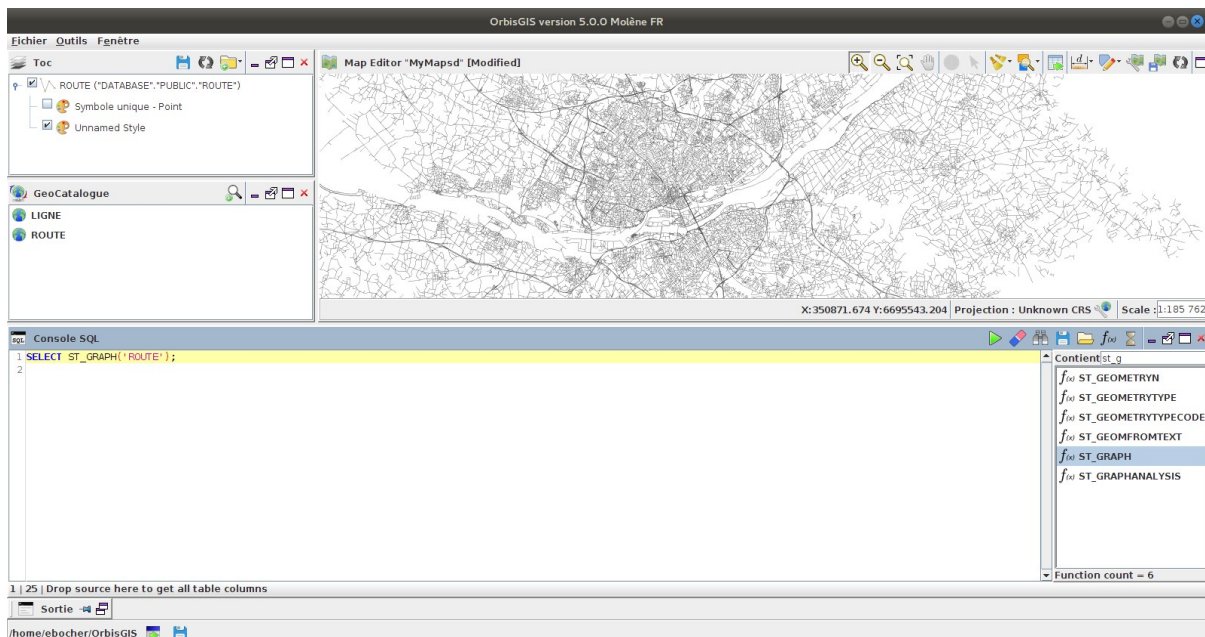


Figure 2: Environnement graphique d'OrbisGIS et console SQL

H2Network contient 7 fonctions SQL dont l'une ST_Graph est dédiée à la construction du graphe et les autres ST_Accessibility, ST_ConnectedComponents, ST_GraphAnalysis, ST_ShortestPath, ST_ShortestPathLength et ST_ShortestPathTree à son analyse (figure 3). Nous présentons en détails ces fonctions dans la partie suivante. Le lecteur qui souhaite obtenir des détails sur l'implémentation des fonctions dans H2Network et des algorithmes dans JNA pourra consulter les documentations en ligne disponibles sur les liens suivants :

- http://javadoc.orbisgis.org/latest/h2gis/md_readme.html
- http://javadoc.orbisgis.org/latest/jna/md_readme.html .

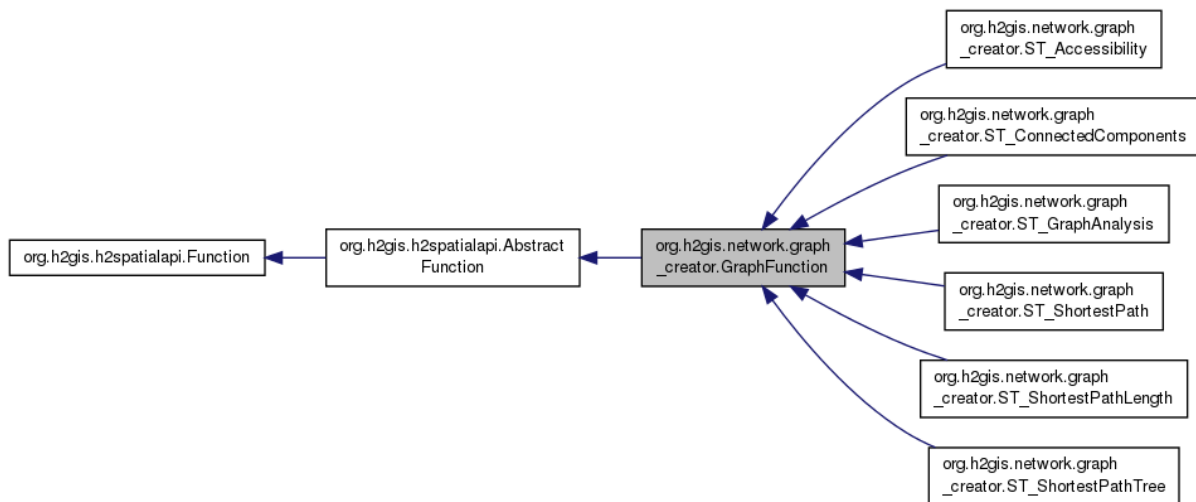


Figure 3: Implémentation des fonctions d'analyses de graphe

4.2 Description des fonctions SQL

H2Network permet de construire un graphe à partir de données géographiques et de réaliser des analyses à partir du langage SQL de la base de données H2 : plus court chemin, matrices de distances, connectivité, accessibilité, centralité, etc.

Toutes les fonctions, sauf `ST_Graph`, utilisent JGraphT comme modèle de graphe.

4.2.1 Fonction de construction du graphe

La fonction `ST_Graph` est le point de départ de tous les traitements. Elle permet de créer un graphe mathématique à partir d'une table contenant des `LINestring` ou des `MULTILINESTRINGs`.

Signature

La fonction comprend 4 signatures :

```
BOOLEAN ST_Graph('INPUT');
BOOLEAN ST_Graph('INPUT', 'THE_GEOM');
BOOLEAN ST_Graph('INPUT', 'THE_GEOM', tolerance);
BOOLEAN ST_Graph('INPUT', 'THE_GEOM', tolerance, orientBySlope);
```

Le paramètre `INPUT` précise le nom de la table d'entrée. La table d'entrée doit contenir une colonne de type `GEOMETRY`, dont on peut préciser le nom par le deuxième paramètre `THE_GEOM`. Par défaut, `ST_Graph` utilise la première colonne de type `GEOMETRY`. La table d'entrée doit également contenir une clé primaire. Cette clé primaire est utilisée pour faire le lien avec la table de sortie `INPUT_EDGES`.

Le paramètre `tolerance` précise une tolérance d'accrochage des nœuds du graphe, ce qui permet dans certains cas de corriger de façon automatique des erreurs dans les données d'entrée. La valeur par défaut est `0.0`.

Le paramètre `orientBySlope` permet d'orienter les arcs du graphe en fonction des valeurs `z` de la première et de la dernière coordonnée de chaque géométrie. En générale, la valeur `z` concerne l'altitude du point, mais elle peut au besoin qualifier autre chose (un poids, une vitesse, ...). Si `orientBySlope` est `TRUE`, l'arc est orienté dans le sens descendant de la pente : de la coordonnée avec une valeur `z` plus grande à la coordonnée avec une valeur `z` plus petite. La valeur par défaut est `FALSE`. Dans ce cas, les arcs sont orientés de la première coordonnée de la géométrie à la dernière coordonnée de la géométrie.

Données de sortie

La fonction `ST_Graph` produit deux tables stockées dans la base de données H2.

1. Une table `INPUT_NODES` avec les colonnes [`NODE_ID INT PRIMARY KEY`, `THE_GEOM GEOMETRY`].

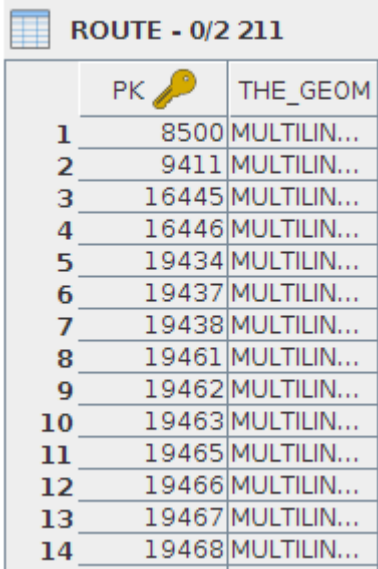
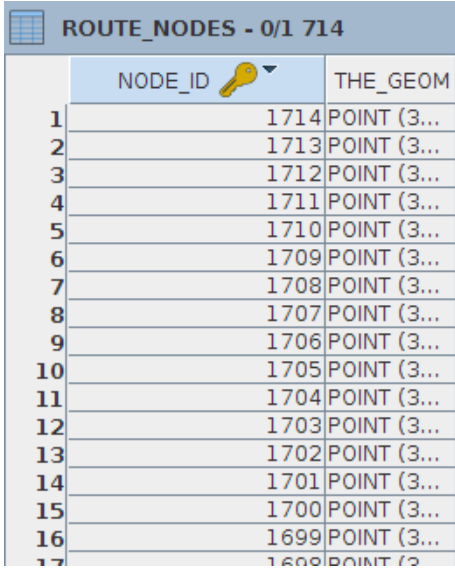
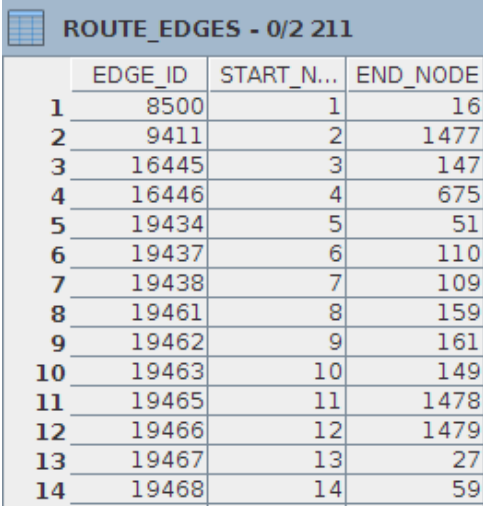
Elle représente les nœuds du graphe avec :

- NODE_ID = identifiant unique du nœud,
- THE_GEOM = géométrie de type POINT.

2. Une table INPUT_EDGES avec les colonnes [EDGE_ID INT PRIMARY KEY, START_NODE INT, END_NODE INT]. Cette table représente les arcs du graphe avec :

- EDGE_ID = identifiant unique de l'arc,
- START_NODE et END_NODE correspondent aux identifiants NODE_ID dans la table INPUT_NODES.

Exemple d'utilisation

Données d'entrée	 <table border="1"> <thead> <tr> <th></th> <th>PK</th> <th>THE_GEOM</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>MULTILIN...</td></tr> <tr><td>2</td><td>9411</td><td>MULTILIN...</td></tr> <tr><td>3</td><td>16445</td><td>MULTILIN...</td></tr> <tr><td>4</td><td>16446</td><td>MULTILIN...</td></tr> <tr><td>5</td><td>19434</td><td>MULTILIN...</td></tr> <tr><td>6</td><td>19437</td><td>MULTILIN...</td></tr> <tr><td>7</td><td>19438</td><td>MULTILIN...</td></tr> <tr><td>8</td><td>19461</td><td>MULTILIN...</td></tr> <tr><td>9</td><td>19462</td><td>MULTILIN...</td></tr> <tr><td>10</td><td>19463</td><td>MULTILIN...</td></tr> <tr><td>11</td><td>19465</td><td>MULTILIN...</td></tr> <tr><td>12</td><td>19466</td><td>MULTILIN...</td></tr> <tr><td>13</td><td>19467</td><td>MULTILIN...</td></tr> <tr><td>14</td><td>19468</td><td>MULTILIN...</td></tr> </tbody> </table>		PK	THE_GEOM	1	8500	MULTILIN...	2	9411	MULTILIN...	3	16445	MULTILIN...	4	16446	MULTILIN...	5	19434	MULTILIN...	6	19437	MULTILIN...	7	19438	MULTILIN...	8	19461	MULTILIN...	9	19462	MULTILIN...	10	19463	MULTILIN...	11	19465	MULTILIN...	12	19466	MULTILIN...	13	19467	MULTILIN...	14	19468	MULTILIN...																																																																					
	PK	THE_GEOM																																																																																																																	
1	8500	MULTILIN...																																																																																																																	
2	9411	MULTILIN...																																																																																																																	
3	16445	MULTILIN...																																																																																																																	
4	16446	MULTILIN...																																																																																																																	
5	19434	MULTILIN...																																																																																																																	
6	19437	MULTILIN...																																																																																																																	
7	19438	MULTILIN...																																																																																																																	
8	19461	MULTILIN...																																																																																																																	
9	19462	MULTILIN...																																																																																																																	
10	19463	MULTILIN...																																																																																																																	
11	19465	MULTILIN...																																																																																																																	
12	19466	MULTILIN...																																																																																																																	
13	19467	MULTILIN...																																																																																																																	
14	19468	MULTILIN...																																																																																																																	
Instruction	<code>SELECT ST_GRAPH('ROUTE_LIMIT');</code>																																																																																																																		
Résultats	<div style="display: flex; justify-content: space-around;"> <div data-bbox="395 1422 852 1989">  <table border="1"> <thead> <tr> <th></th> <th>NODE_ID</th> <th>THE_GEOM</th> </tr> </thead> <tbody> <tr><td>1</td><td>1714</td><td>POINT (3...</td></tr> <tr><td>2</td><td>1713</td><td>POINT (3...</td></tr> <tr><td>3</td><td>1712</td><td>POINT (3...</td></tr> <tr><td>4</td><td>1711</td><td>POINT (3...</td></tr> <tr><td>5</td><td>1710</td><td>POINT (3...</td></tr> <tr><td>6</td><td>1709</td><td>POINT (3...</td></tr> <tr><td>7</td><td>1708</td><td>POINT (3...</td></tr> <tr><td>8</td><td>1707</td><td>POINT (3...</td></tr> <tr><td>9</td><td>1706</td><td>POINT (3...</td></tr> <tr><td>10</td><td>1705</td><td>POINT (3...</td></tr> <tr><td>11</td><td>1704</td><td>POINT (3...</td></tr> <tr><td>12</td><td>1703</td><td>POINT (3...</td></tr> <tr><td>13</td><td>1702</td><td>POINT (3...</td></tr> <tr><td>14</td><td>1701</td><td>POINT (3...</td></tr> <tr><td>15</td><td>1700</td><td>POINT (3...</td></tr> <tr><td>16</td><td>1699</td><td>POINT (3...</td></tr> <tr><td>17</td><td>1698</td><td>POINT (3...</td></tr> </tbody> </table> </div> <div data-bbox="900 1422 1385 1928">  <table border="1"> <thead> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table> </div> </div>		NODE_ID	THE_GEOM	1	1714	POINT (3...	2	1713	POINT (3...	3	1712	POINT (3...	4	1711	POINT (3...	5	1710	POINT (3...	6	1709	POINT (3...	7	1708	POINT (3...	8	1707	POINT (3...	9	1706	POINT (3...	10	1705	POINT (3...	11	1704	POINT (3...	12	1703	POINT (3...	13	1702	POINT (3...	14	1701	POINT (3...	15	1700	POINT (3...	16	1699	POINT (3...	17	1698	POINT (3...		EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59
	NODE_ID	THE_GEOM																																																																																																																	
1	1714	POINT (3...																																																																																																																	
2	1713	POINT (3...																																																																																																																	
3	1712	POINT (3...																																																																																																																	
4	1711	POINT (3...																																																																																																																	
5	1710	POINT (3...																																																																																																																	
6	1709	POINT (3...																																																																																																																	
7	1708	POINT (3...																																																																																																																	
8	1707	POINT (3...																																																																																																																	
9	1706	POINT (3...																																																																																																																	
10	1705	POINT (3...																																																																																																																	
11	1704	POINT (3...																																																																																																																	
12	1703	POINT (3...																																																																																																																	
13	1702	POINT (3...																																																																																																																	
14	1701	POINT (3...																																																																																																																	
15	1700	POINT (3...																																																																																																																	
16	1699	POINT (3...																																																																																																																	
17	1698	POINT (3...																																																																																																																	
	EDGE_ID	START_N...	END_NODE																																																																																																																
1	8500	1	16																																																																																																																
2	9411	2	1477																																																																																																																
3	16445	3	147																																																																																																																
4	16446	4	675																																																																																																																
5	19434	5	51																																																																																																																
6	19437	6	110																																																																																																																
7	19438	7	109																																																																																																																
8	19461	8	159																																																																																																																
9	19462	9	161																																																																																																																
10	19463	10	149																																																																																																																
11	19465	11	1478																																																																																																																
12	19466	12	1479																																																																																																																
13	19467	13	27																																																																																																																
14	19468	14	59																																																																																																																

Implémentation

Cette fonction est implémentée uniquement en SQL. Le code source est disponible à l'adresse:

https://github.com/irstv/H2GIS/blob/master/h2network/src/main/java/org/h2gis/network/graph_creator/GraphCreator.java

4.2.2 Fonctions d'analyse

Les fonctions d'analyses de graphe partagent des paramètres communs afin d'une part de simplifier leur usage mais également leur implémentation.

4.2.2.1 Paramètres communs aux fonctions d'analyses

Toutes les fonctions d'analyses de graphe sont composées des éléments de signatures suivant :

```
NOM_FONCTION('INPUT_EDGES', 'O[ - EO]'[, 'W'], ...);
```

ou

1. 'INPUT_EDGES' représente la table d'entrée. Cette table est produite à partir de la fonction ST_Graph. Néanmoins, il est également possible de la créer manuellement.
2. 'O[- EO]'. Le string o précise l'orientation globale du graphe. Il prend les valeurs suivantes :
 - directed (orienté),
 - reversed (orienté avec l'orientation de chaque arc renversé),
 - undirected (non orienté).

Le string EO indique l'orientation de chaque arc individuel. Il est requis si o a la valeur directed ou reversed et facultatif si o a la valeur undirected.

EO correspond au nom de la colonne précisant les orientations des arcs. Le nom de la colonne est libre néanmoins les arcs doivent être codés avec des entiers :

- 1 si l'arc doit être orienté dans le sens (START_NODE, END_NODE),
- -1 si l'arc doit être orienté dans le sens (END_NODE, START_NODE),
- 0 si l'arc doit être orienté dans les deux sens.

3. 'W'. Le string w indique le nom d'une colonne indiquant le poids de chaque arc. Le poids est exprimé avec des valeurs DOUBLE.

4.2.2.2 ST_ConnectedComponents

L'analyse d'un graphe impose d'analyser préalablement la qualité de celui-ci, notamment sous l'angle de la connectivité. En effet, il n'est pas rare qu'un graphe soit composé de sous-ensembles, non connectés entre eux. Cette absence de connexion est problématique car elle empêche, par exemple, un calcul de distance entre deux points qui se trouveraient chacun dans un sous-graphe.

Afin d'évaluer la connectivité d'un graphe, nous avons développé la fonction `ST_ConnectedComponents`. Elle permet de comptabiliser le nombre d'arcs reliés entre eux et par conséquent les sous-graphes.

La figure 4 illustre l'identification de 3 sous-graphes comportant 4, 3 et 9 arcs.

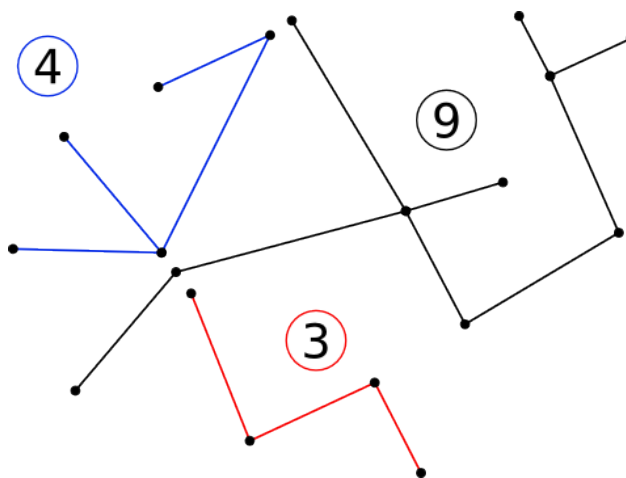


Figure 4: Principe de l'analyse de connectivité d'un graphe

Signature

```
ST_ConnectedComponents('INPUT_EDGES', 'O[ - EO]');
```

Données de sortie

Cette fonction produit deux tables : `INPUT_EDGES_NODE_CC` et `INPUT_EDGES_EDGE_CC` ou `INPUT_EDGES` correspond au nom de la table passée en paramètre de la fonction. La table `INPUT_EDGES_NODE_CC` comprend deux colonnes `NODE_ID` et `CONNECTED_COMPONENT`. La colonne `CONNECTED_COMPONENT` stocke l'identifiant du sous-graphe auquel le nœud appartient. Cet identifiant est également stocké dans la table `INPUT_EDGES_EDGE_CC` ce qui offre à l'utilisateur la possibilité de visualiser les arcs qui composent un sous-graphe. A noter : Si un arc a un nœud de départ et un nœud d'arrivée qui appartiennent à deux sous-graphes différents alors une valeur de `-1` est assignée dans la colonne `CONNECTED_COMPONENT`.

Exemple d'utilisation

Données d'entrée	<table border="1"> <thead> <tr> <th colspan="4">ROUTE_EDGES - 0/2 211</th> </tr> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table>	ROUTE_EDGES - 0/2 211					EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59																																												
ROUTE_EDGES - 0/2 211																																																																																																													
	EDGE_ID	START_N...	END_NODE																																																																																																										
1	8500	1	16																																																																																																										
2	9411	2	1477																																																																																																										
3	16445	3	147																																																																																																										
4	16446	4	675																																																																																																										
5	19434	5	51																																																																																																										
6	19437	6	110																																																																																																										
7	19438	7	109																																																																																																										
8	19461	8	159																																																																																																										
9	19462	9	161																																																																																																										
10	19463	10	149																																																																																																										
11	19465	11	1478																																																																																																										
12	19466	12	1479																																																																																																										
13	19467	13	27																																																																																																										
14	19468	14	59																																																																																																										
Instruction	<pre>SELECT ST_CONNECTEDCOMPONENTS('ROUTE_EDGES', 'undirected');</pre>																																																																																																												
Résultats	<table border="1"> <thead> <tr> <th colspan="3">ROUTE_EDGES_NODE_CC - 0/1 714</th> </tr> <tr> <th></th> <th>NODE_ID</th> <th>CONNECTED_COMPONENT</th> </tr> </thead> <tbody> <tr><td>1</td><td>1</td><td>837</td></tr> <tr><td>2</td><td>2</td><td>588</td></tr> <tr><td>3</td><td>3</td><td>975</td></tr> <tr><td>4</td><td>4</td><td>87</td></tr> <tr><td>5</td><td>5</td><td>974</td></tr> <tr><td>6</td><td>6</td><td>605</td></tr> <tr><td>7</td><td>7</td><td>345</td></tr> <tr><td>8</td><td>8</td><td>82</td></tr> <tr><td>9</td><td>9</td><td>410</td></tr> <tr><td>10</td><td>10</td><td>408</td></tr> <tr><td>11</td><td>11</td><td>688</td></tr> <tr><td>12</td><td>12</td><td>693</td></tr> <tr><td>13</td><td>13</td><td>307</td></tr> <tr><td>14</td><td>14</td><td>610</td></tr> <tr><td>15</td><td>15</td><td>837</td></tr> <tr><td>16</td><td>16</td><td>837</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="3">ROUTE_EDGES_EDGE_CC - 0/2 211</th> </tr> <tr> <th></th> <th>EDGE_ID</th> <th>CONNECTED_COMPONENT</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>837</td></tr> <tr><td>2</td><td>9411</td><td>-1</td></tr> <tr><td>3</td><td>16445</td><td>975</td></tr> <tr><td>4</td><td>16446</td><td>-1</td></tr> <tr><td>5</td><td>19434</td><td>-1</td></tr> <tr><td>6</td><td>19437</td><td>605</td></tr> <tr><td>7</td><td>19438</td><td>345</td></tr> <tr><td>8</td><td>19461</td><td>-1</td></tr> <tr><td>9</td><td>19462</td><td>-1</td></tr> <tr><td>10</td><td>19463</td><td>-1</td></tr> <tr><td>11</td><td>19465</td><td>-1</td></tr> <tr><td>12</td><td>19466</td><td>-1</td></tr> <tr><td>13</td><td>19467</td><td>-1</td></tr> <tr><td>14</td><td>19468</td><td>-1</td></tr> <tr><td>15</td><td>19481</td><td>837</td></tr> <tr><td>16</td><td>19482</td><td>837</td></tr> </tbody> </table>	ROUTE_EDGES_NODE_CC - 0/1 714				NODE_ID	CONNECTED_COMPONENT	1	1	837	2	2	588	3	3	975	4	4	87	5	5	974	6	6	605	7	7	345	8	8	82	9	9	410	10	10	408	11	11	688	12	12	693	13	13	307	14	14	610	15	15	837	16	16	837	ROUTE_EDGES_EDGE_CC - 0/2 211				EDGE_ID	CONNECTED_COMPONENT	1	8500	837	2	9411	-1	3	16445	975	4	16446	-1	5	19434	-1	6	19437	605	7	19438	345	8	19461	-1	9	19462	-1	10	19463	-1	11	19465	-1	12	19466	-1	13	19467	-1	14	19468	-1	15	19481	837	16	19482	837
ROUTE_EDGES_NODE_CC - 0/1 714																																																																																																													
	NODE_ID	CONNECTED_COMPONENT																																																																																																											
1	1	837																																																																																																											
2	2	588																																																																																																											
3	3	975																																																																																																											
4	4	87																																																																																																											
5	5	974																																																																																																											
6	6	605																																																																																																											
7	7	345																																																																																																											
8	8	82																																																																																																											
9	9	410																																																																																																											
10	10	408																																																																																																											
11	11	688																																																																																																											
12	12	693																																																																																																											
13	13	307																																																																																																											
14	14	610																																																																																																											
15	15	837																																																																																																											
16	16	837																																																																																																											
ROUTE_EDGES_EDGE_CC - 0/2 211																																																																																																													
	EDGE_ID	CONNECTED_COMPONENT																																																																																																											
1	8500	837																																																																																																											
2	9411	-1																																																																																																											
3	16445	975																																																																																																											
4	16446	-1																																																																																																											
5	19434	-1																																																																																																											
6	19437	605																																																																																																											
7	19438	345																																																																																																											
8	19461	-1																																																																																																											
9	19462	-1																																																																																																											
10	19463	-1																																																																																																											
11	19465	-1																																																																																																											
12	19466	-1																																																																																																											
13	19467	-1																																																																																																											
14	19468	-1																																																																																																											
15	19481	837																																																																																																											
16	19482	837																																																																																																											

Implémentation

La fonction `ST_ConnectedComponents` utilise deux classes de la bibliothèque JGraphT :

- `ConnectivityInspector` pour des graphes non-orientés,
- `StrongConnectivityInspector` pour des graphes orientés.

Le code source est disponible à l'adresse :

https://github.com/irstv/H2GIS/blob/master/h2network/src/main/java/org/h2gis/network/graph_creator/ST_ConnectedComponents.java

4.2.2.3 ST_ShortestPath

ST_ShortestPath permet de calculer le plus court chemin entre deux nœuds d'un graphe.

Signature

```
ST_ShortestPath('INPUT_EDGES', 'O[ - EO]'[, 'W'], s INT, d INT);
```

ou

- s représente l'identifiant du nœud de départ.
- d représente l'identifiant du nœud d'arrivée.
- w est un paramètre optionnel utilisé pour préciser le nom de la colonne qui contient le poids des arcs.

Données de sortie




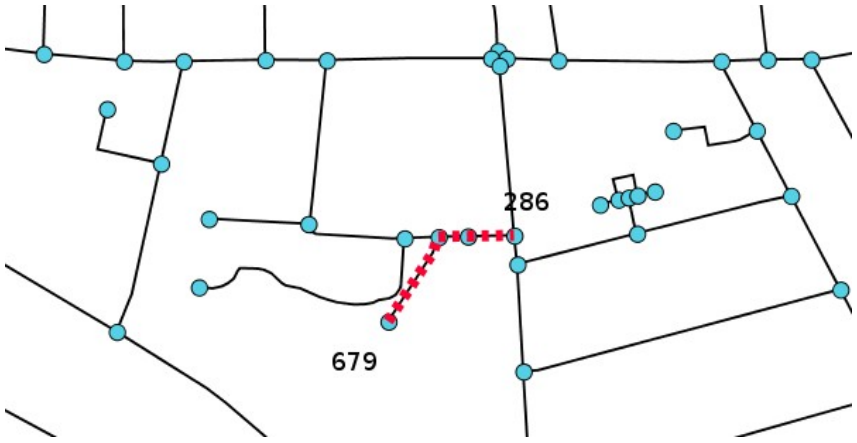
La fonction ST_ShortestPath produit une table qui représente le ou les plus court(s) chemin(s) entre deux nœuds. Elle a la forme suivante :

```
TABLE[ [THE_GEOM GEOMETRY, ]  
        EDGE_ID INT, PATH_ID INT, PATH_EDGE_ID INT,  
        SOURCE INT, DESTINATION INT, WEIGHT DOUBLE ]
```

- THE_GEOM. La géométrie de cet arc. La table de sortie contient cette colonne si et seulement si INPUT_EDGES la contient également.
- EDGE_ID. Identifiant de la table d'entrée INPUT_EDGES.
- PATH_ID. S'il existe plus d'un seul plus court chemin, on les distingue grâce à cet identifiant.
- PATH_EDGE_ID. Cet identifiant sert à dénombrer les arcs d'un seul plus court chemin.
- SOURCE. Identifiant du nœud de départ.
- DESTINATION. Identifiant du nœud d'arrivée.
- WEIGH. Poids de l'arc.

S'il n'existe pas de chemin de s à d, la table de sortie sera vide.

Exemple d'utilisation

Donnée d'entrée	<div style="display: flex; justify-content: space-around;"> <table border="1" data-bbox="443 344 932 855"> <caption>ROUTE_EDGES - 0/2 211</caption> <thead> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table> <table border="1" data-bbox="983 344 1326 855"> <caption>ROUTE - 0/2 211</caption> <thead> <tr> <th></th> <th>PK </th> <th>THE_GEOM</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>MULTILIN...</td></tr> <tr><td>2</td><td>9411</td><td>MULTILIN...</td></tr> <tr><td>3</td><td>16445</td><td>MULTILIN...</td></tr> <tr><td>4</td><td>16446</td><td>MULTILIN...</td></tr> <tr><td>5</td><td>19434</td><td>MULTILIN...</td></tr> <tr><td>6</td><td>19437</td><td>MULTILIN...</td></tr> <tr><td>7</td><td>19438</td><td>MULTILIN...</td></tr> <tr><td>8</td><td>19461</td><td>MULTILIN...</td></tr> <tr><td>9</td><td>19462</td><td>MULTILIN...</td></tr> <tr><td>10</td><td>19463</td><td>MULTILIN...</td></tr> <tr><td>11</td><td>19465</td><td>MULTILIN...</td></tr> <tr><td>12</td><td>19466</td><td>MULTILIN...</td></tr> <tr><td>13</td><td>19467</td><td>MULTILIN...</td></tr> <tr><td>14</td><td>19468</td><td>MULTILIN...</td></tr> </tbody> </table> </div>		EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59		PK 	THE_GEOM	1	8500	MULTILIN...	2	9411	MULTILIN...	3	16445	MULTILIN...	4	16446	MULTILIN...	5	19434	MULTILIN...	6	19437	MULTILIN...	7	19438	MULTILIN...	8	19461	MULTILIN...	9	19462	MULTILIN...	10	19463	MULTILIN...	11	19465	MULTILIN...	12	19466	MULTILIN...	13	19467	MULTILIN...	14	19468	MULTILIN...
	EDGE_ID	START_N...	END_NODE																																																																																																							
1	8500	1	16																																																																																																							
2	9411	2	1477																																																																																																							
3	16445	3	147																																																																																																							
4	16446	4	675																																																																																																							
5	19434	5	51																																																																																																							
6	19437	6	110																																																																																																							
7	19438	7	109																																																																																																							
8	19461	8	159																																																																																																							
9	19462	9	161																																																																																																							
10	19463	10	149																																																																																																							
11	19465	11	1478																																																																																																							
12	19466	12	1479																																																																																																							
13	19467	13	27																																																																																																							
14	19468	14	59																																																																																																							
	PK 	THE_GEOM																																																																																																								
1	8500	MULTILIN...																																																																																																								
2	9411	MULTILIN...																																																																																																								
3	16445	MULTILIN...																																																																																																								
4	16446	MULTILIN...																																																																																																								
5	19434	MULTILIN...																																																																																																								
6	19437	MULTILIN...																																																																																																								
7	19438	MULTILIN...																																																																																																								
8	19461	MULTILIN...																																																																																																								
9	19462	MULTILIN...																																																																																																								
10	19463	MULTILIN...																																																																																																								
11	19465	MULTILIN...																																																																																																								
12	19466	MULTILIN...																																																																																																								
13	19467	MULTILIN...																																																																																																								
14	19468	MULTILIN...																																																																																																								
Instruction	<pre> SELECT A.THE_GEOM, B.EDGE_ID, B.PATH_ID, B.PATH_EDGE_ID, B.SOURCE, B.DESTINATION, B.WEIGHT FROM ROUTE A, SELECT * FROM ST_SHORTESTPATH('ROUTE_EDGES', 'undirected', 679, 286) B WHERE A.PK=ABS(B.EDGE_ID); </pre>																																																																																																									
Résultats	<table border="1" data-bbox="411 1122 1358 1301"> <caption>PATH - 0/3</caption> <thead> <tr> <th></th> <th>THE_GEOM</th> <th>EDGE_ID</th> <th>PATH_ID</th> <th>PATH_ED...</th> <th>SOURCE</th> <th>DESTINAT...</th> <th>WEIGHT</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>MULTILIN...</td> <td>73341</td> <td>1</td> <td>1</td> <td>464</td> <td>286</td> <td>1</td> </tr> <tr> <td>2</td> <td>MULTILIN...</td> <td>104597</td> <td>1</td> <td>2</td> <td>1421</td> <td>464</td> <td>1</td> </tr> <tr> <td>3</td> <td>MULTILIN...</td> <td>74713</td> <td>1</td> <td>3</td> <td>679</td> <td>1421</td> <td>1</td> </tr> </tbody> </table> <div style="text-align: center;">  <p>The graph shows a network of nodes (blue dots) and edges (black lines). A specific path is highlighted in red dashed lines, starting at node 679 and ending at node 286. The path consists of three segments: 679 to 464, 464 to 286, and 286 to 1421.</p> </div> <p>Le plus court chemin est ici représenté en pointillé rouge.</p>		THE_GEOM	EDGE_ID	PATH_ID	PATH_ED...	SOURCE	DESTINAT...	WEIGHT	1	MULTILIN...	73341	1	1	464	286	1	2	MULTILIN...	104597	1	2	1421	464	1	3	MULTILIN...	74713	1	3	679	1421	1																																																																									
	THE_GEOM	EDGE_ID	PATH_ID	PATH_ED...	SOURCE	DESTINAT...	WEIGHT																																																																																																			
1	MULTILIN...	73341	1	1	464	286	1																																																																																																			
2	MULTILIN...	104597	1	2	1421	464	1																																																																																																			
3	MULTILIN...	74713	1	3	679	1421	1																																																																																																			

Implémentation

La fonction `ST_ShortestPath` utilise l'algorithme de Dijkstra pour le calcul de plus courts chemins. Cet algorithme est implémenté dans la classe `Dijkstra` de JNA. Il fait appel à la méthode :

```
public double oneToOne(V source, final V target).
```

L'algorithme est récursif et démarre la recherche du plus court chemin en partant du nœud de départ. La recherche est suspendue dès que l'indice du nœud source est identifié. Au cours de la recherche des nœuds candidats, les valeurs de `PATH_ID` et `PATH_EDGE_ID` sont incrémentées de telle sorte que :

- `PATH_ID` soit un identifiant unique pour chaque plus court chemin,
- `PATH_EDGE_ID` indique le nombre d'arcs qui reste avant d'arriver au nœud d'arrivée.

A noter que si la table d'entrée contient une colonne de type `geometry`, cette dernière est conservée dans la table résultat.

Le code source est visible à l'adresse :

<https://github.com/irstv/Java-Network-Analyzer/blob/master/src/main/java/org/javaneetworkanalyzer/alg/Dijkstra.java>

4.2.2.4 `ST_ShortestPathLength`

La fonction `ST_ShortestPathLength` sert à calculer des distances de nœuds à nœuds.

Signature

Quatre signatures sont disponibles pour cette fonction.

```
ST_ShortestPathLength('INPUT_EDGES', 'O[ - EO]'[, 'W'],
                      s INT, d INT);      -- 1-1
```

permet de calculer la distance d'un nœud de départ à un nœud d'arrivée. `s` et `d` correspondent aux identifiants des nœuds de départ et d'arrivée.

```
ST_ShortestPathLength('INPUT_EDGES', 'O[ - EO]'[, 'W'],
                      s INT, 'ds');      -- 1-M
```

permet de calculer la distance d'un nœud de départ à un ensemble de `M` nœuds d'arrivée. L'ensemble de nœuds d'arrivée est spécifié par le *string* `ds` qui correspond à une liste d'identifiants de nœuds séparés par des virgules.

```
ST_ShortestPathLength('INPUT_EDGES', 'O[ - EO]'[, 'W'],
                      s INT);            -- 1-Tous
```

permet de calculer la distance d'un nœud de départ à tous les autres nœuds joignables du graphe.


```
ST_ShortestPathLength('INPUT_EDGES', 'O[ - EO]',[, 'W'],
                    'SDT');           -- N-M
```

permet de calculer les distances d'un ensemble de N nœuds de départ à un ensemble de M nœuds d'arrivée (matrices de distance). Les couples $N-M$ sont spécifiés dans le *string* SDT indiquant le nom d'une table de paires origine-destination.

La table SDT doit avoir la forme suivante :

```
SDT[SOURCE INT, DESTINATION INT, ...]
```

Données de sortie

La fonction ST_ShortestPathLength produit une table de la forme suivante :

```
TABLE[SOURCE INT, DESTINATION INT, DISTANCE DOUBLE]
```

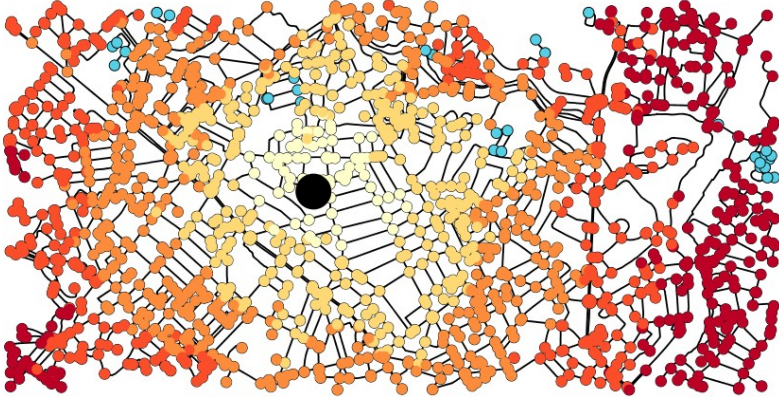
- SOURCE. Identifiant du nœud de départ,
- DESTINATION. Identifiant du nœud d'arrivée,
- DISTANCE. Distance du nœud de départ au nœud d'arrivée.

Exemple d'utilisation

Donnée d'entrée	<div style="display: flex; justify-content: space-around;"> <table border="1" data-bbox="448 1055 876 1500"> <caption>ROUTE_EDGES - 0/2 211</caption> <thead> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table> <table border="1" data-bbox="922 1055 1323 1547"> <caption>ROUTE_NODES - 0/1 714</caption> <thead> <tr> <th></th> <th>NODE_ID</th> <th>THE_GEOM</th> </tr> </thead> <tbody> <tr><td>1</td><td>1714</td><td>POINT (3...</td></tr> <tr><td>2</td><td>1713</td><td>POINT (3...</td></tr> <tr><td>3</td><td>1712</td><td>POINT (3...</td></tr> <tr><td>4</td><td>1711</td><td>POINT (3...</td></tr> <tr><td>5</td><td>1710</td><td>POINT (3...</td></tr> <tr><td>6</td><td>1709</td><td>POINT (3...</td></tr> <tr><td>7</td><td>1708</td><td>POINT (3...</td></tr> <tr><td>8</td><td>1707</td><td>POINT (3...</td></tr> <tr><td>9</td><td>1706</td><td>POINT (3...</td></tr> <tr><td>10</td><td>1705</td><td>POINT (3...</td></tr> <tr><td>11</td><td>1704</td><td>POINT (3...</td></tr> <tr><td>12</td><td>1703</td><td>POINT (3...</td></tr> <tr><td>13</td><td>1702</td><td>POINT (3...</td></tr> <tr><td>14</td><td>1701</td><td>POINT (3...</td></tr> <tr><td>15</td><td>1700</td><td>POINT (3...</td></tr> <tr><td>16</td><td>1699</td><td>POINT (3...</td></tr> <tr><td>17</td><td>1698</td><td>POINT (3...</td></tr> </tbody> </table> </div>		EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59		NODE_ID	THE_GEOM	1	1714	POINT (3...	2	1713	POINT (3...	3	1712	POINT (3...	4	1711	POINT (3...	5	1710	POINT (3...	6	1709	POINT (3...	7	1708	POINT (3...	8	1707	POINT (3...	9	1706	POINT (3...	10	1705	POINT (3...	11	1704	POINT (3...	12	1703	POINT (3...	13	1702	POINT (3...	14	1701	POINT (3...	15	1700	POINT (3...	16	1699	POINT (3...	17	1698	POINT (3...
	EDGE_ID	START_N...	END_NODE																																																																																																																
1	8500	1	16																																																																																																																
2	9411	2	1477																																																																																																																
3	16445	3	147																																																																																																																
4	16446	4	675																																																																																																																
5	19434	5	51																																																																																																																
6	19437	6	110																																																																																																																
7	19438	7	109																																																																																																																
8	19461	8	159																																																																																																																
9	19462	9	161																																																																																																																
10	19463	10	149																																																																																																																
11	19465	11	1478																																																																																																																
12	19466	12	1479																																																																																																																
13	19467	13	27																																																																																																																
14	19468	14	59																																																																																																																
	NODE_ID	THE_GEOM																																																																																																																	
1	1714	POINT (3...																																																																																																																	
2	1713	POINT (3...																																																																																																																	
3	1712	POINT (3...																																																																																																																	
4	1711	POINT (3...																																																																																																																	
5	1710	POINT (3...																																																																																																																	
6	1709	POINT (3...																																																																																																																	
7	1708	POINT (3...																																																																																																																	
8	1707	POINT (3...																																																																																																																	
9	1706	POINT (3...																																																																																																																	
10	1705	POINT (3...																																																																																																																	
11	1704	POINT (3...																																																																																																																	
12	1703	POINT (3...																																																																																																																	
13	1702	POINT (3...																																																																																																																	
14	1701	POINT (3...																																																																																																																	
15	1700	POINT (3...																																																																																																																	
16	1699	POINT (3...																																																																																																																	
17	1698	POINT (3...																																																																																																																	
Instruction	<pre>CREATE TABLE DISTANCE AS SELECT a.THE_GEOM, a.NODE_ID, b.DISTANCE FROM ROUTE_NODES a, (SELECT * FROM ST_ShortestPathLength('ROUTE_EDGES', 'undirected', 679)) b WHERE a.NODE_ID=b.DESTINATION;</pre>																																																																																																																		

SHORTESTPATHLENGTH - 0/1 682			
	SOURCE	DESTINATION ▼	DISTANCE
1	679	1713	14
2	679	1712	15
3	679	1711	28
4	679	1710	23
5	679	1709	23
6	679	1708	44
7	679	1707	16
8	679	1706	16
9	679	1705	30
10	679	1704	16
11	679	1703	30
12	679	1702	49
13	679	1701	29
14	679	1700	24
15	679	1699	16
16	679	1698	29
17	679	1697	28
18	679	1696	17
19	679	1695	24
20	679	1694	24
21	679	1693	17
22	679	1692	19
23	679	1691	14
24	679	1690	24

Résultats



En noir le nœud de départ dans le réseau. Le dégradé du blanc vers le rouge représente les distances du plus proche au plus lointain.

Les points bleus représentent les nœuds non connectés au réseau et qui ne peuvent donc pas être atteint à partir du point noir.

Implémentation

Cette fonction utilise la classe Dijkstra de JNA pour le calcul de plus courts chemins.

Pour calculer la distance d'un nœud à un autre nœud, on utilise la méthode

```
public double oneToOne(V source, final V target)
```

Cette méthode arrête la recherche une fois que *target* a été trouvé.

Pour calculer la distance d'un nœud à un ensemble de M nœuds ou bien à tous les autres nœuds, on utilise la méthode

```
public Map<V, Double> oneToMany(V source, final Set<V> targets)
```

Cette méthode arrête la recherche une fois que tous les nœuds dans *targets* ont été trouvés.

Pour calculer la distance d'un ensemble de N nœuds à un ensemble de M nœuds, on utilise d'abord la méthode

```
private static Map<VDijkstra, Set<VDijkstra>>
    prepareSourceDestinationMap
```

de `ST_ShortestPathLength` pour construire, à partir des colonnes `SOURCE` et `DESTINATION` de la table `SDT` un `Map<V, Set<V>>` qui associe chaque nœud de départ s à un ensemble de nœuds d'arrivées pour lesquels on aimerait connaître la distance depuis s . Ensuite on parcourt ce `map`, et pour chaque nœud de départ s , on appelle `oneToMany(s, targets)` où `targets` est l'ensemble de destinations associé à s . Cette technique peut réduire le temps de calcul si l'on demande plusieurs destinations pour une seule source.

Le code source est visible à l'adresse :

<https://github.com/irstv/Java-Network-Analyzer/blob/master/src/main/java/org/javanetworkanalyzer/alg/Dijkstra.java>

4.2.2.5 ST_Accessibility

La fonction `ST_Accessibility` est dédiée au calcul d'indicateurs d'accessibilité. Elle prend en entrée une liste de destinations possible parmi les nœuds d'un graphe. Elle calcule, pour chaque nœud du graphe, la destination la plus proche, ainsi que la distance envers cette destination.

Signature

Deux signatures sont possibles.

```
ST_Accessibility('INPUT_EDGES', 'O[ - EO]'[, 'W'], 'ds');
```

permet de mesurer l'accessibilité de tous les nœuds du graphe vers une liste de nœuds d'arrivée (`ds`) séparés par des virgules.

```
ST_Accessibility('INPUT_EDGES', 'O[ - EO]'[, 'W'], 'DT');
```

permet de mesurer l'accessibilité de tous les nœuds du graphe vers une liste de nœuds d'arrivée. Cette liste est exprimée dans une table (`DT`) dont le nom est donné en paramètre.

La table `DT` doit avoir la forme suivante :

```
DT[DESTINATION INT, ...]
```

Données de sortie

La fonction ST_Accessibility produit une table de la forme suivante :

TABLE[**SOURCE** INT, CLOSEST_DEST INT, DISTANCE DOUBLE]

- SOURCE. Identifiant du nœud de départ,
- CLOSEST_DEST. Identifiant de la destination la plus proche,
- DISTANCE. Distance du nœud de départ à la destination la plus proche.

Exemple d'utilisation

Donnée d'entrée	<div style="display: flex; justify-content: space-around;"> <table border="1" data-bbox="411 689 861 1164"> <caption>ROUTE_EDGES - 0/2 211</caption> <thead> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table> <table border="1" data-bbox="909 689 1364 1254"> <caption>ROUTE_NODES - 0/1 714</caption> <thead> <tr> <th></th> <th>NODE_ID</th> <th>THE_GEOM</th> </tr> </thead> <tbody> <tr><td>1</td><td>1714</td><td>POINT (3...</td></tr> <tr><td>2</td><td>1713</td><td>POINT (3...</td></tr> <tr><td>3</td><td>1712</td><td>POINT (3...</td></tr> <tr><td>4</td><td>1711</td><td>POINT (3...</td></tr> <tr><td>5</td><td>1710</td><td>POINT (3...</td></tr> <tr><td>6</td><td>1709</td><td>POINT (3...</td></tr> <tr><td>7</td><td>1708</td><td>POINT (3...</td></tr> <tr><td>8</td><td>1707</td><td>POINT (3...</td></tr> <tr><td>9</td><td>1706</td><td>POINT (3...</td></tr> <tr><td>10</td><td>1705</td><td>POINT (3...</td></tr> <tr><td>11</td><td>1704</td><td>POINT (3...</td></tr> <tr><td>12</td><td>1703</td><td>POINT (3...</td></tr> <tr><td>13</td><td>1702</td><td>POINT (3...</td></tr> <tr><td>14</td><td>1701</td><td>POINT (3...</td></tr> <tr><td>15</td><td>1700</td><td>POINT (3...</td></tr> <tr><td>16</td><td>1699</td><td>POINT (3...</td></tr> <tr><td>17</td><td>1698</td><td>POINT (3...</td></tr> </tbody> </table> </div>		EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59		NODE_ID	THE_GEOM	1	1714	POINT (3...	2	1713	POINT (3...	3	1712	POINT (3...	4	1711	POINT (3...	5	1710	POINT (3...	6	1709	POINT (3...	7	1708	POINT (3...	8	1707	POINT (3...	9	1706	POINT (3...	10	1705	POINT (3...	11	1704	POINT (3...	12	1703	POINT (3...	13	1702	POINT (3...	14	1701	POINT (3...	15	1700	POINT (3...	16	1699	POINT (3...	17	1698	POINT (3...
	EDGE_ID	START_N...	END_NODE																																																																																																																
1	8500	1	16																																																																																																																
2	9411	2	1477																																																																																																																
3	16445	3	147																																																																																																																
4	16446	4	675																																																																																																																
5	19434	5	51																																																																																																																
6	19437	6	110																																																																																																																
7	19438	7	109																																																																																																																
8	19461	8	159																																																																																																																
9	19462	9	161																																																																																																																
10	19463	10	149																																																																																																																
11	19465	11	1478																																																																																																																
12	19466	12	1479																																																																																																																
13	19467	13	27																																																																																																																
14	19468	14	59																																																																																																																
	NODE_ID	THE_GEOM																																																																																																																	
1	1714	POINT (3...																																																																																																																	
2	1713	POINT (3...																																																																																																																	
3	1712	POINT (3...																																																																																																																	
4	1711	POINT (3...																																																																																																																	
5	1710	POINT (3...																																																																																																																	
6	1709	POINT (3...																																																																																																																	
7	1708	POINT (3...																																																																																																																	
8	1707	POINT (3...																																																																																																																	
9	1706	POINT (3...																																																																																																																	
10	1705	POINT (3...																																																																																																																	
11	1704	POINT (3...																																																																																																																	
12	1703	POINT (3...																																																																																																																	
13	1702	POINT (3...																																																																																																																	
14	1701	POINT (3...																																																																																																																	
15	1700	POINT (3...																																																																																																																	
16	1699	POINT (3...																																																																																																																	
17	1698	POINT (3...																																																																																																																	
Instruction	<pre>CREATE TABLE DISTANCE_ACCESSIBILITY AS SELECT A.THE_GEOM, A.NODE_ID, B.DISTANCE FROM ROUTE_NODES A, (SELECT * FROM ST_ACCESSIBILITY('ROUTE_EDGES', 'undirected', '1168,694,635,986')) B WHERE A.NODE_ID=B.SOURCE;</pre>																																																																																																																		

Résultats

ACCESSIBILITY_RESULT - 0/1 714			
	SOURCE	CLOSEST_DEST	DISTANCE
1	16	1168	10
2	1	1168	10
3	1477	1168	10
4	2	1168	9
5	147	635	19
6	3	635	18
7	675	694	9
8	4	694	10
9	51	635	17
10	5	635	16
11	110	1168	15
12	6	1168	16
13	109	1168	18
14	7	1168	17
15	159	635	11
16	8	635	12
17	161	635	11
18	9	635	12
19	149	635	10
20	10	635	11

Les croix noires localisent les “nœuds d’arrivée”.

Le dégradé de couleur (du blanc au rouge) représente la distance d’un nœud du graphe par rapport au nœud d’arrivée

Implémentation

Cette fonction utilise la classe `AccessibilityAnalyzer` de JNA, qui elle utilise la classe `DijkstraForAccessibility` de JNA pour le calcul de plus courts chemins. Cette classe comprend une version augmentée de l’algorithme de Dijkstra où la destination la plus proche et la distance envers cette destination sont enregistrées.

Au lieu d’exécuter l’algorithme de Dijkstra pour chaque nœud du graphe et choisir la destination la plus proche, on renverse plutôt l’orientation de chacun des arcs, puis on exécute l’algorithme de Dijkstra à pour chaque destination. Ainsi la distance de la destination envers chaque nœud dans le graphe avec orientations renversées est égale à la distance de chaque nœud envers la destination dans le graphe initial. L’algorithme de Dijkstra est donc exécuté autant de fois qu’il y a de destinations, au lieu de $|V|$ fois.

Le code source est visible à l’adresse : <https://github.com/irstv/Java-Network-Analyzer/blob/master/src/main/java/org/javanetworkanalyzer/analyzer/AccessibilityAnalyzer.java>

4.2.2.6 ST_GraphAnalysis

La fonction `ST_GraphAnalysis` calcule deux mesures de centralité : la centralité d'intermédiarité (*betweenness centrality* en anglais) et la centralité de proximité (*closeness centrality* en anglais). Ces deux mesures sont réalisées en une seule passe.

Signature

```
ST_GraphAnalysis('INPUT_EDGES', 'O[ - EO]'[, 'W']);
```

Données de sortie

La fonction `ST_GraphAnalysis` produit deux tables indiquant les indices de centralité des nœuds et des arcs. La structure des tables est la suivante :

```
INPUT_EDGES_NODE_CENT[NODE_ID INT PRIMARY KEY,  
                       BETWEENNESS DOUBLE, CLOSENESS DOUBLE]
```

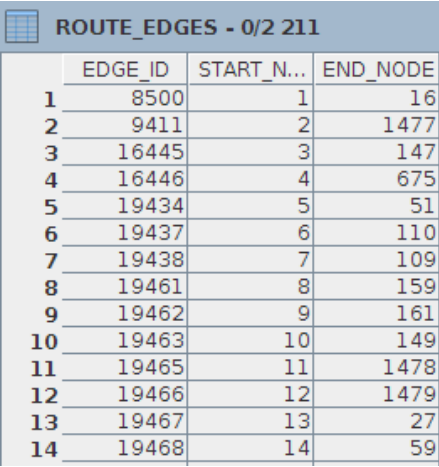
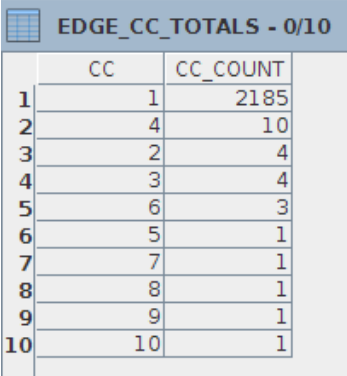
```
INPUT_EDGES_EDGE_CENT[EDGE_ID INT PRIMARY KEY,  
                      BETWEENNESS DOUBLE]
```

A noter que les résultats ne seront pas justes si le graphe :

- contient des arcs dupliqués (ayant les mêmes origine, destination et poids)
- n'est pas connecté. Si $C_c(v)=0$ quel que soit $v \in V$.

Les indices de centralité sont normalisés, mais cette normalisation dépend de la connectivité du graphe. Pour s'assurer que l'analyse est réalisée sur un graphe connecté, il est conseillé d'utiliser la fonction `ST_ConnectedComponents`.

Exemple d'utilisation

<p>Donnée d'entrée</p>	 <table border="1"> <thead> <tr> <th></th> <th>EDGE_ID</th> <th>START_N...</th> <th>END_NODE</th> </tr> </thead> <tbody> <tr><td>1</td><td>8500</td><td>1</td><td>16</td></tr> <tr><td>2</td><td>9411</td><td>2</td><td>1477</td></tr> <tr><td>3</td><td>16445</td><td>3</td><td>147</td></tr> <tr><td>4</td><td>16446</td><td>4</td><td>675</td></tr> <tr><td>5</td><td>19434</td><td>5</td><td>51</td></tr> <tr><td>6</td><td>19437</td><td>6</td><td>110</td></tr> <tr><td>7</td><td>19438</td><td>7</td><td>109</td></tr> <tr><td>8</td><td>19461</td><td>8</td><td>159</td></tr> <tr><td>9</td><td>19462</td><td>9</td><td>161</td></tr> <tr><td>10</td><td>19463</td><td>10</td><td>149</td></tr> <tr><td>11</td><td>19465</td><td>11</td><td>1478</td></tr> <tr><td>12</td><td>19466</td><td>12</td><td>1479</td></tr> <tr><td>13</td><td>19467</td><td>13</td><td>27</td></tr> <tr><td>14</td><td>19468</td><td>14</td><td>59</td></tr> </tbody> </table>		EDGE_ID	START_N...	END_NODE	1	8500	1	16	2	9411	2	1477	3	16445	3	147	4	16446	4	675	5	19434	5	51	6	19437	6	110	7	19438	7	109	8	19461	8	159	9	19462	9	161	10	19463	10	149	11	19465	11	1478	12	19466	12	1479	13	19467	13	27	14	19468	14	59
	EDGE_ID	START_N...	END_NODE																																																										
1	8500	1	16																																																										
2	9411	2	1477																																																										
3	16445	3	147																																																										
4	16446	4	675																																																										
5	19434	5	51																																																										
6	19437	6	110																																																										
7	19438	7	109																																																										
8	19461	8	159																																																										
9	19462	9	161																																																										
10	19463	10	149																																																										
11	19465	11	1478																																																										
12	19466	12	1479																																																										
13	19467	13	27																																																										
14	19468	14	59																																																										
<p>Instructions</p>	<pre>-- Étape 1 : Identification des sous-graphes fortement connectés CALL ST_ConnectedComponents('ROUTE_EDGES', 'undirected'); -- Étape 2 : Classement des sous-graphes selon le nombre d'arcs qu'ils contiennent. DROP TABLE IF EXISTS EDGE_CC_TOTALS; CREATE TABLE EDGE_CC_TOTALS AS SELECT CONNECTED_COMPONENT CC, COUNT(CONNECTED_COMPONENT) CC_COUNT FROM ROUTE_EDGES_EDGE_CC GROUP BY CC ORDER BY CC_COUNT DESC; -- Étape 3 : Sélection des arcs du plus grand sous-graphe DROP TABLE IF EXISTS ROUTE_EDGES_LARGEST; CREATE TABLE ROUTE_EDGES_LARGEST AS SELECT A.*, B.CONNECTED_COMPONENT CC FROM ROUTE_EDGES A, ROUTE_EDGES_EDGE_CC B WHERE A.EDGE_ID=B.EDGE_ID AND B.CONNECTED_COMPONENT=(SELECT CC FROM EDGE_CC_TOTALS LIMIT 1); -- Étape 4 : Représentation cartographique du plus grand sous-graphe CREATE TABLE ROUTE_EDGES_LARGEST_GEOM AS SELECT A.THE_GEOM, B.* FROM ROUTE A, ROUTE_EDGES_LARGEST B WHERE A.PK=B.EDGE_ID; -- Étape 5 : Calcul des indices de centralité CALL ST_GRAPHANALYSIS('ROUTE_EDGES_LARGEST', 'undirected');</pre>																																																												
<p>Résultats</p>	<p>Étape 1 : voir ST_ConnectedComponents</p> <p>Étape 2 :</p>  <table border="1"> <thead> <tr> <th></th> <th>CC</th> <th>CC_COUNT</th> </tr> </thead> <tbody> <tr><td>1</td><td>1</td><td>2185</td></tr> <tr><td>2</td><td>4</td><td>10</td></tr> <tr><td>3</td><td>2</td><td>4</td></tr> <tr><td>4</td><td>3</td><td>4</td></tr> <tr><td>5</td><td>6</td><td>3</td></tr> <tr><td>6</td><td>5</td><td>1</td></tr> <tr><td>7</td><td>7</td><td>1</td></tr> <tr><td>8</td><td>8</td><td>1</td></tr> <tr><td>9</td><td>9</td><td>1</td></tr> <tr><td>10</td><td>10</td><td>1</td></tr> </tbody> </table>		CC	CC_COUNT	1	1	2185	2	4	10	3	2	4	4	3	4	5	6	3	6	5	1	7	7	1	8	8	1	9	9	1	10	10	1																											
	CC	CC_COUNT																																																											
1	1	2185																																																											
2	4	10																																																											
3	2	4																																																											
4	3	4																																																											
5	6	3																																																											
6	5	1																																																											
7	7	1																																																											
8	8	1																																																											
9	9	1																																																											
10	10	1																																																											

Étape 3 :


ROUTE_EDGES_LARGEST - 0/2 185			
	EDGE_ID	START_N...	END_NODE
1	8500	1	16
2	9411	2	1477
3	16445	3	147
4	16446	4	675
5	19434	5	51
6	19437	6	110
7	19438	7	109
8	19461	8	159
9	19462	9	161
10	19463	10	149
11	19465	11	1478
12	19466	12	1479
13	19467	13	27
14	19468	14	59
15	19481	15	1
16	19482	16	19
17	19483	17	20
18	19484	18	17
19	19485	19	18


Étape 4 :



En rouge le plus grand sous-graphe. En noir les arcs non connectés.

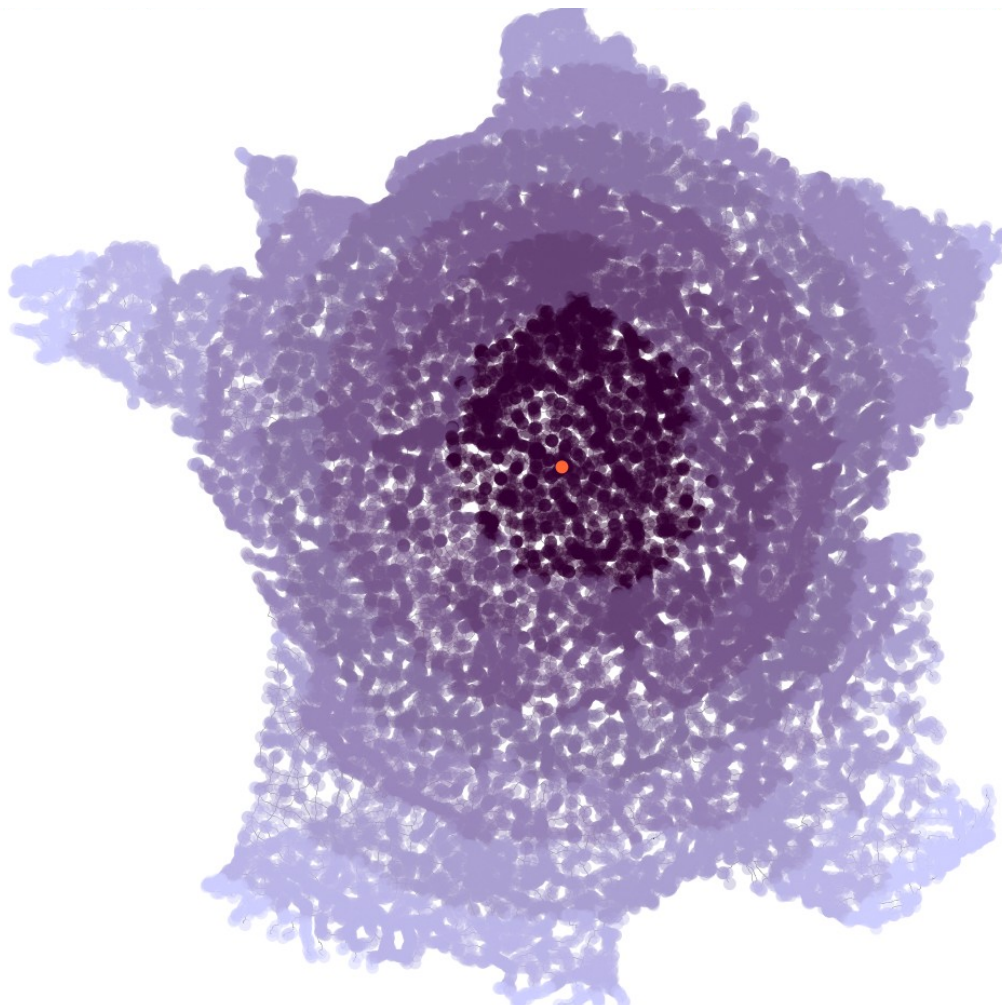
Étape 5 :

ROUTE_EDGES_LARGEST_NODE_CENT - 0/1 682			
	NODE_ID 	BETWEENNESS	CLOSENESS
1	1	0,27545469387...	0,034520998049081
2	2	0,22023737397...	0,0384782658456749
3	3	0,01756122540...	0,0297006961376727
4	4	0	0,0358521551816068
5	5	0,06348278950...	0,0315479318369492
6	6	0,01119230259...	0,0411193463956361
7	7	0,00273702614...	0,0381170495000113
8	8	0,00036183540...	0,0331166272655634
9	9	0,00544345801...	0,0339369713120546
10	10	0,00519738436...	0,0342488081170286
11	11	0,01580472812...	0,0343530950483314
12	12	0,00824563119...	0,0334853887372762
13	13	0,00757150990...	0,0317163827097602
14	14	0,00924669834...	0,0324298254075432
15	15	0,28341910879...	0,0348986879255938
16	16	0,24792530046...	0,034531635168447
17	17	0,63450597869...	0,0361699838622916
18	18	0,36084366539...	0,0356295040271301

ROUTE_EDGES_LARGEST_EDGE_CENT - 0/2 185		
	EDGE_ID 	BETWEENNESS
1	8500	0,0004920515209101
2	9411	0,0056789333064643
3	16445	0,018086751654991
4	16446	0,0056789333064643
5	19434	0,057471500263714
6	19437	0,0081759292244566
7	19438	0,0038841008217164
8	19461	0,0016214198530609
9	19462	0,0032991190950624
10	19463	0,0075123934713481
11	19465	0,0056789333064643
12	19466	0,0056789333064643
13	19467	0,0133700684452107
14	19468	0,0146369400919728
15	19481	0,2876492987684263
16	19482	0,258234551983708
17	19483	0,2798717127879135
18	19484	0,3767902875347868

Discussion sur la représentation des indices de centralité

Les indices de centralité de proximité sont généralement représentés sur les nœuds du graphe à l'exemple des figures 5 et 6.



*Figure 5: Centralité de proximité des nœuds du réseau Route 120® de l'IGN
en rouge le nœud avec la centralité égale à 1*

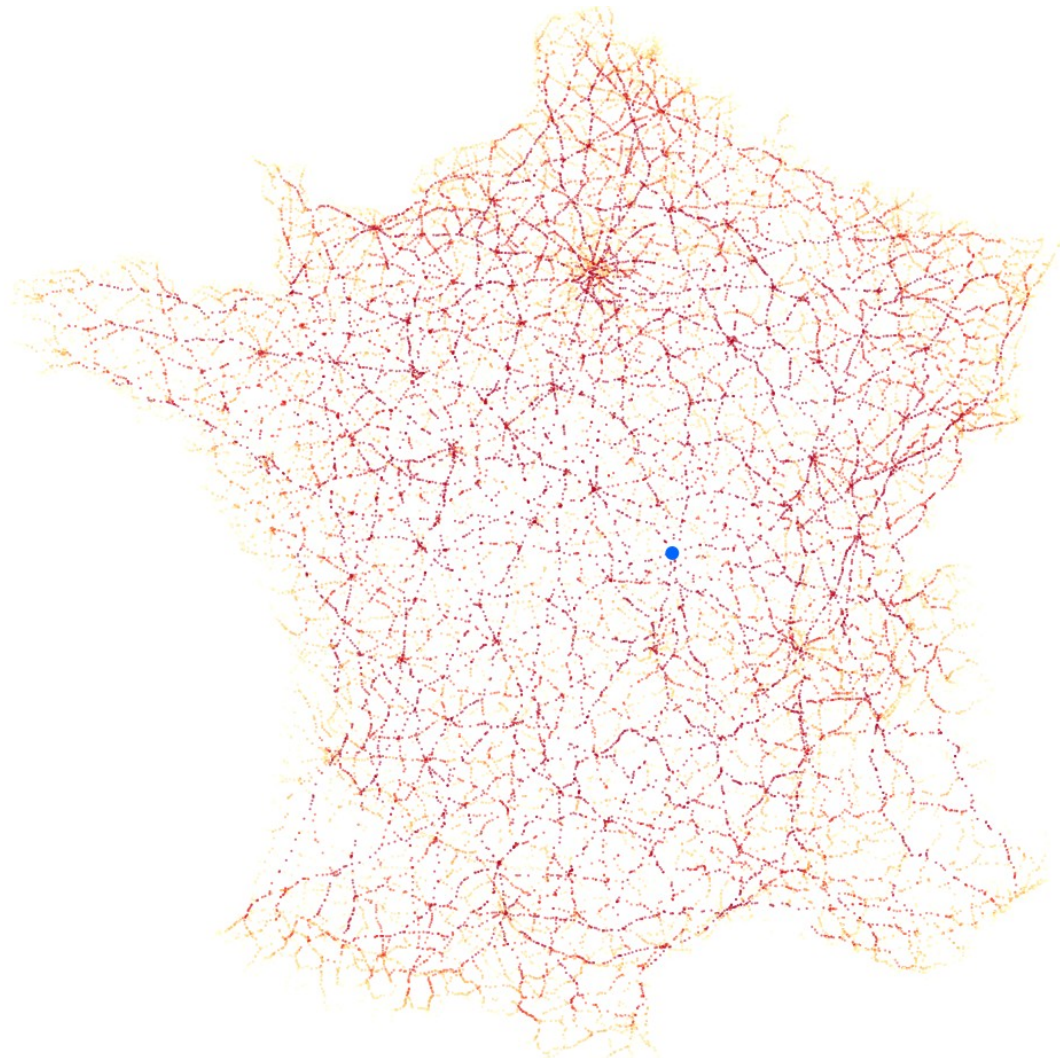


Figure 6: Centralité d'intermédiarité des nœuds du réseau routier Route 120® de l'IGN en bleu le nœud avec la centralité égale à 1

Ces indices sont plus délicats à représenter sur les arcs notamment lors qu'ils sont à double sens. En effet, partageant la même géométrie, une méthode simple consiste à cartographier séparément les indices pour les arcs ayant un identifiant positif puis les indices ayant un identifiant négatif. Néanmoins cette approche ne permet pas de traduire spatialement l'importance de l'arc dans le réseau. Aussi, une autre solution consiste à représenter la moyenne de l'indice pour les arcs (figure 7).

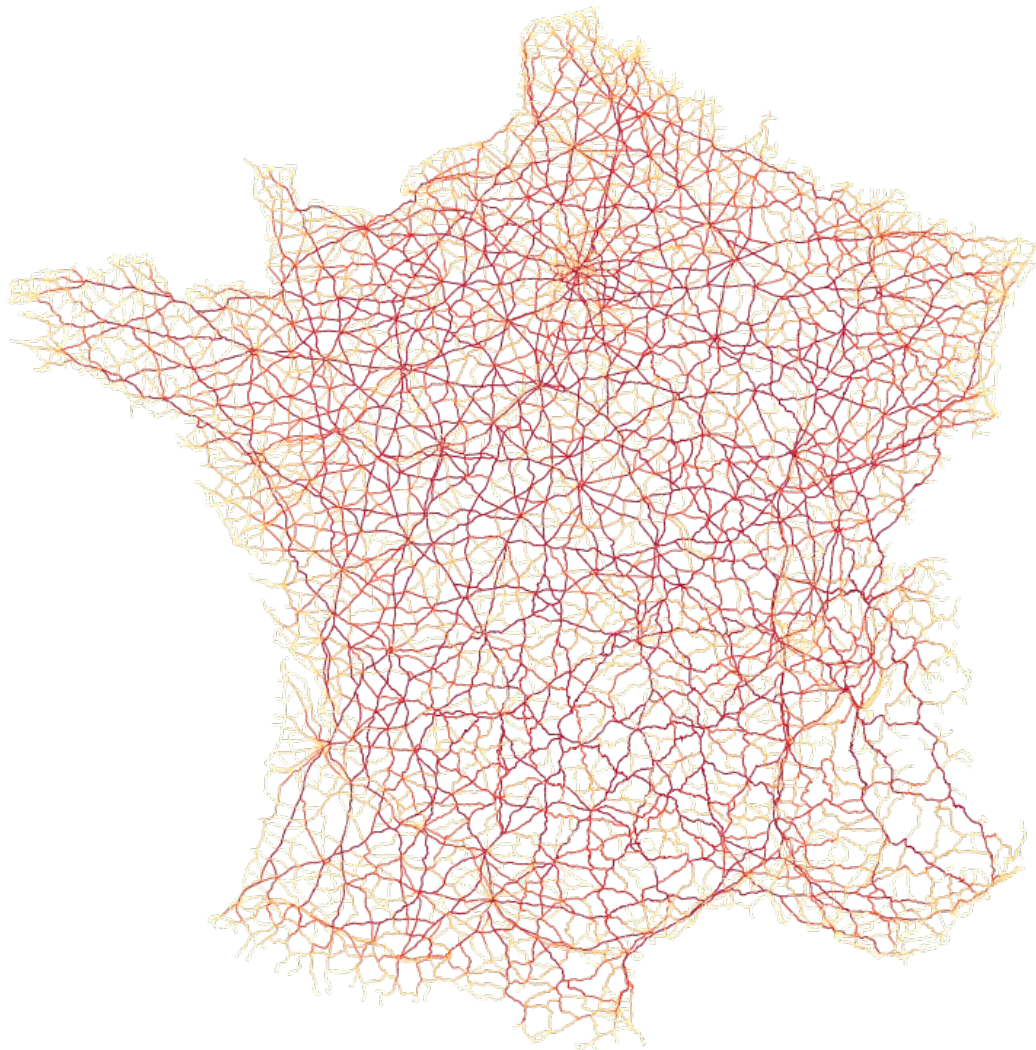


Figure 7: "Moyenne" de la centralité d'intermédiation pour les arcs du réseau routier Route 120®

Implémentation

La fonction `ST_GraphAnalysis` utilise la classe `GraphAnalyzer` de JNA, qui implémente l'algorithme de Brandes pour calculer la centralité d'intermédiarité (Brandes, 2001). Cet algorithme est plus rapide qu'une approche naïve mais nécessite néanmoins l'exécution d'une version optimisée de l'algorithme de Dijkstra pour exécuter l'analyse une fois par nœud. Le temps de calcul peut donc être assez long.

Les valeurs de centralité d'intermédiarité sont normalisées selon l'équation

$$\underline{C_B(v)} = \frac{C_B(v) - b}{B - b},$$

où

$$b = \min_{v \in V} \{C_B(v)\}, B = \max_{v \in V} \{C_B(v)\}.$$

Ainsi la valeur minimale est toujours 0 et la valeur maximale est toujours 1.

Les valeurs de centralité de proximité sont normalisées selon l'équation

$$\underline{C_C(v)} = |V| C_C(v).$$

Le code source est visible à l'adresse :

<https://github.com/irstv/Java-Network-Analyzer/blob/master/src/main/java/org/javanetwork/analyzer/analyzers/GraphAnalyzer.java>

5. Application

La partie suivante présente un cas d'utilisation de la bibliothèque H2Network. Il concerne la construction d'un distancier de temps de trajet domicile- travail sur la France. Cet exemple est précédé d'une section méthodologique sur la préparation des données. En effet, afin que les analyses de graphe s'effectuent de façon efficiente, il est essentiel de mettre en forme la donnée d'entrée qui constituera le graphe.

Pour des questions de lisibilité, nous appelons dans la suite de ce document, réseau routier un jeu de données représentant un linéaire de routes. Ce linéaire est composé de lignes ou de multilignes (au sens du standard *Simple Feature Access*⁴⁷). Une ligne ou multiligne, associée d'attributs de description (largeur, vitesse, longueur, ...), est appelée tronçon routier.

5.1 Préparation du réseau routier

5.1.1 Analyse du réseau routier

Analyse de la structuration topologique du réseau routier

L'utilisation d'un réseau routier impose d'analyser préalablement la qualité de celui-ci, notamment sous l'angle de la connectivité. En effet, il n'est pas rare qu'un réseau soit composé de sous-ensembles, non connectés entre eux. Cette absence de connexion peut être problématique car elle empêche, par exemple, le calcul de distance entre deux points qui se trouveraient potentiellement dans deux sous-graphes.

Afin d'évaluer la connectivité d'un réseau, nous appliquons une analyse qui permet de comptabiliser le nombre d'arcs reliés entre eux. Plus un sous-réseau comporte d'arcs, plus il est connecté et donc propice à l'utilisation pour du parcours de réseaux. Cette analyse est réalisée à l'aide de la fonction `ST_ConnectedComponents` présentée dans la section 4.2.2.2. Notons que cette analyse nécessite la construction au préalable d'un graphe (fonction `ST_Graph`). Le réseau routier de l'IGN Route 120[®] est utilisé pour cette démonstration.

47 <http://www.opengeospatial.org/standards/sfa> consulté en septembre 2014.

La chaîne d'instructions SQL suivante est appliquée :

```

-- Étape 1 : Calcul du graphe
DROP TABLE IF EXISTS TRONCON_ROUTE_NODES, TRONCON_ROUTE_EDGES;
CALL ST_GRAPH('TRONCON_ROUTE');

-- Étape 2 : Identification des sous-graphes
DROP TABLE IF EXISTS TRONCON_ROUTE_EDGES_NODE_CC, TRONCON_ROUTE_EDGES_EDGE_CC;
CALL ST_ConnectedComponents('TRONCON_ROUTE_EDGES', 'undirected');

-- Étape 3 : Dénombrement du nombre de sous-graphes
DROP TABLE IF EXISTS EDGE_CC_TOTALS;
CREATE TABLE EDGE_CC_TOTALS AS
SELECT CONNECTED_COMPONENT CC,
COUNT(CONNECTED_COMPONENT) CC_COUNT
FROM TRONCON_ROUTE_EDGES_EDGE_CC
GROUP BY CC
ORDER BY CC_COUNT DESC;

-- Étape 4 : Représentation cartographique des sous-graphes
DROP TABLE IF EXISTS SOUS_GRAPHES_GEOM;
CREATE INDEX ON TRONCON_ROUTE_EDGES_EDGE_CC(EDGE_ID);
CREATE TABLE SOUS_GRAPHES_GEOM AS SELECT a.THE_GEOM, b.*
FROM TRONCON_ROUTE a, TRONCON_ROUTE_EDGES_EDGE_CC b
WHERE a.PK=b.EDGE_ID;
    
```

Le réseau routier Route 120[®] comporte 42 013 tronçons. La construction du graphe produit 35 194 nœuds. L'analyse des sous-graphes permet d'identifier 8 sous-ensembles fortement connectés. Le tableau 1 donne le nombre d'arcs pour chaque sous-ensembles.

Identifiant du sous-graphe	Nombre d'arcs
1	41715
2	2
3	1
4	1
5	4
6	2
7	5
8	283

Tableau 1 : Dénombrement du nombre d'arcs par sous-graphe pour le réseau routier Route 120[®].

La figure 8 illustre la représentation spatiale de ces sous-graphes. Le lecteur observera que le plus grand graphe (en bleu) représente le réseau viaire continental.

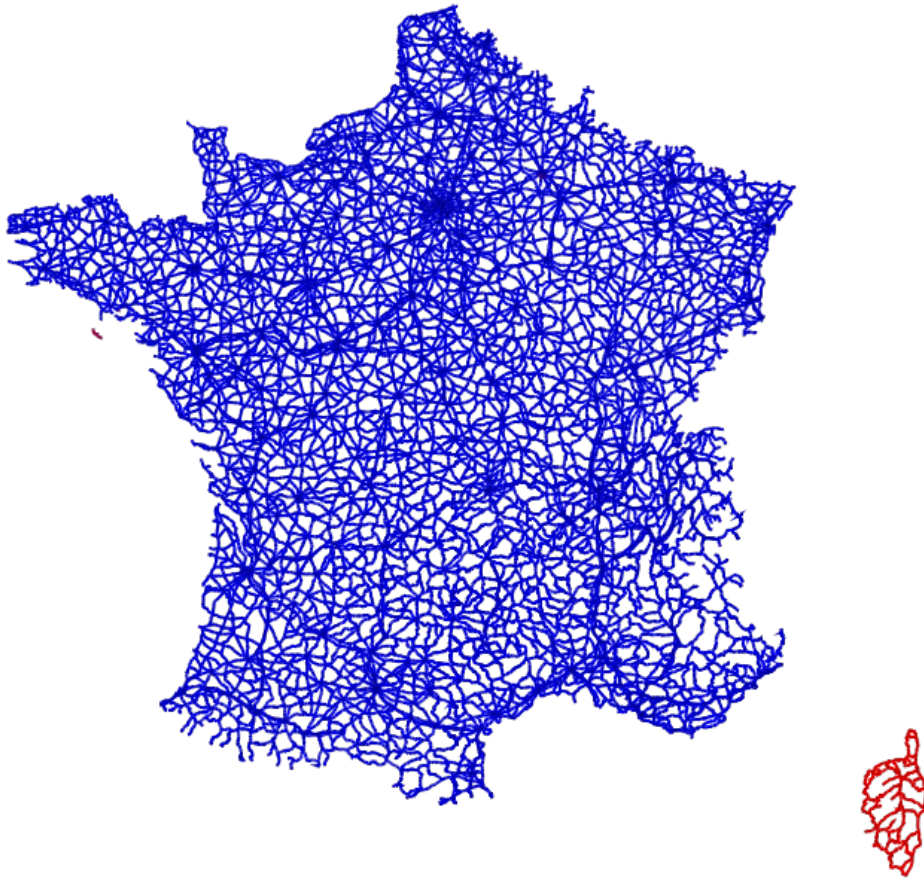


Figure 8: Représentation des sous-graphes du réseau routier Route 120®

Analyse sémantique des tronçons

L'analyse sémantique des données consiste à utiliser les descripteurs du réseau routier pour qualifier les intersections entre les tronçons. Il s'agit d'identifier les erreurs potentielles liées à l'absence ou la présence de nœuds aux intersections du réseau routier. Les vignettes de la figure 9 illustrent deux exemples communs :

1. dans le premier couple de vignettes, un tronçon routier est franchi par un pont. Il n'y a donc pas de nœud marquant l'intersection entre ces deux éléments. La représentation du réseau est correcte.
2. dans le second couple de vignettes, le réseau routier ne matérialise qu'une seule intersection alors que la photographie aérienne du site montre qu'il en existe deux. Il sera donc nécessaire de corriger le réseau pour associer un nouveau nœud et décomposer les tronçons en sous-ensembles.

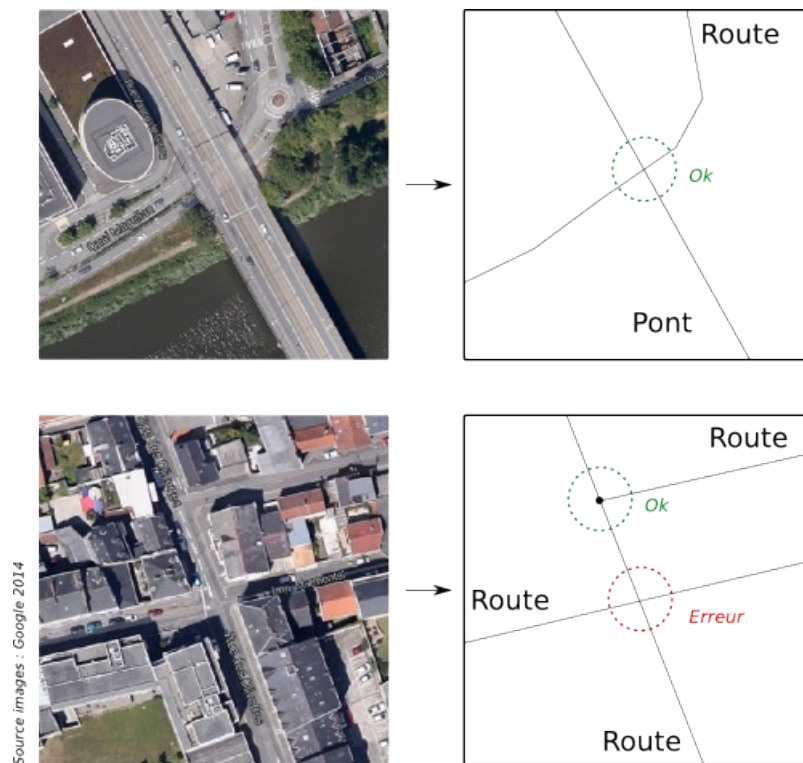


Figure 9: Exemples de croisements

La qualification des intersections est réalisée pour un réseau routier à partir d'une instruction SQL. Posons les conditions suivantes :

- soit ROUTES le nom de la table contenant les tronçons,
- soit THE_GEOM, la colonne contenant les géométries,
- soit TYPE, la colonne qualifiant la nature des tronçons avec les valeurs ROUTE et FRANCHISSEMENT,
- soit GID un identifiant unique pour les tronçons.

L'instruction SQL qui permettrait d'isoler les tronçons routiers à corriger serait :

```
CREATE TABLE ROUTES_A_CORRIGER
AS SELECT A.THE_GEOM, A.TYPE, A.GID
FROM ROUTES A, ROUTES B
WHERE A.GID!=B.GID AND A.TYPE='ROUTE'
AND B.TYPE='ROUTE' AND A.THE_GEOM && B.THE_GEOM;
```

Il est important de noter que cette qualification des intersections devra s'adapter en fonction des descripteurs disponibles dans le jeu de données.

5.1.2 Corrections géométriques du réseau routier

Dans la section précédente, nous avons proposé de qualifier le réseau routier notamment les incohérences géométriques (structurelles). Nous exposons maintenant deux méthodes pour traiter d'une part le cas des tronçons non-intersectants et d'autre part le cas des tronçons dits pendants (nœud d'ordre 1 dans un graphe).

Cas des tronçons non-intersectants

Nous appelons tronçons non-intersectants lorsqu'il y a absence de point pour matérialiser une intersection entre n géométries. La figure 10 présente 3 cas.

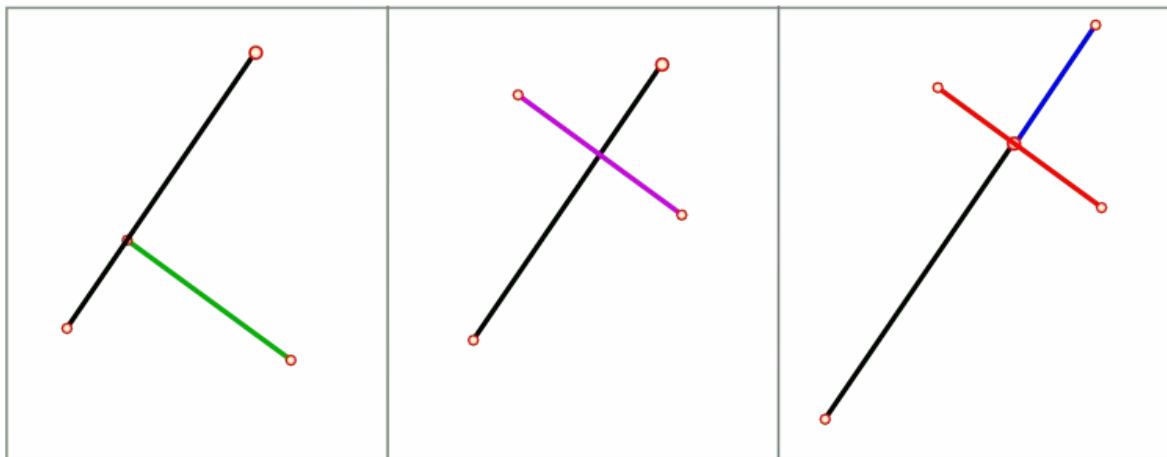


Figure 10: Exemples de tronçons non-intersectants

Vignette de gauche :

L'une des extrémités du tronçon vert se superpose avec le tronçon noir. Par conséquent, le tronçon noir devrait être décomposé en deux parties.

Vignette centrale :

Deux tronçons se croisent sans intersection. La modélisation géométrique attendue serait 4 tronçons segmentés au niveau du croisement.

Vignette de droite :

Le tronçon bleu est jointif au tronçon noir. Le tronçon rouge est d'un seul tenant et ne matérialise pas l'intersection au point de jonction. L'arc rouge devrait ici être décomposé en 2 parties avec comme nœud commun le nœud déjà existant entre les arcs bleu et noir.

A l'exemple de l'analyse sémantique des tronçons routiers (section 5.1.1) nous proposons un script SQL qui permet de décomposer les tronçons non-intersectants.

- soit ROUTES le nom de la table contenant les tronçons,
- soit THE_GEOM, la colonne contenant les géométries,
- soit GID un identifiant unique pour les tronçons.

L'instruction SQL qui permet de segmenter les tronçons est la suivante :

```
CREATE SPATIAL INDEX ON ROUTES (THE_GEOM);
CREATE TABLE DECOUPE_TRONCONS
AS SELECT A.GID,
CASEWHEN (
ST_DIMENSION(ST_INTERSECTION(A.THE_GEOM,ST_UNION(ST_ACCUM(B.THE_GEOM))))=0,
ST_DIFFERENCE(A.THE_GEOM,ST_UNION(ST_ACCUM(B.THE_GEOM))),A.THE_GEOM)
THE_GEOM
FROM ROUTES A , ROUTES B
WHERE A.GID != B.GID AND A.THE_GEOM && B.THE_GEOM
AND ST_INTERSECTS(A.THE_GEOM, B.THE_GEOM)
GROUP BY A.GID, A.THE_GEOM;

CREATE TABLE TRONCONS_DECOUPES
AS SELECT *
FROM ST_EXPLODE('DECOUPE_TRONCONS');
```

Pour une géométrie donnée dans la table ROUTES (A.THE_GEOM) , les géométries qui l'intersectent sont sélectionnées (ST_ACCUM(B.THE_GEOM)). Cet ensemble de géométrie est unifié (ST_UNION()). La fonction CASEWHEN permet ensuite d'adapter le traitement à réaliser. Ainsi, si l'intersection entre la géométrie donnée et les géométries intersectantes correspond à des points (nœuds) alors nous appliquons la fonction ST_DIFFERENCE qui décomposera la géométrie donnée aux niveaux des nœuds identifiés (figure 11) ; sinon nous retournons une copie de la géométrie donnée. L'ultime instruction, ST_Explode⁴⁸, est appliquée pour décomposer en objets simples les géométries qui auraient été découpées. A noter que la décomposition des géométries conservent les attributs associés (e.g la vitesse, la largeur, ...) en les dupliquant pour chaque sous-ensemble de géométries.

48 http://www.h2gis.org/docs/dev/ST_Explode/ consulté en septembre 2014

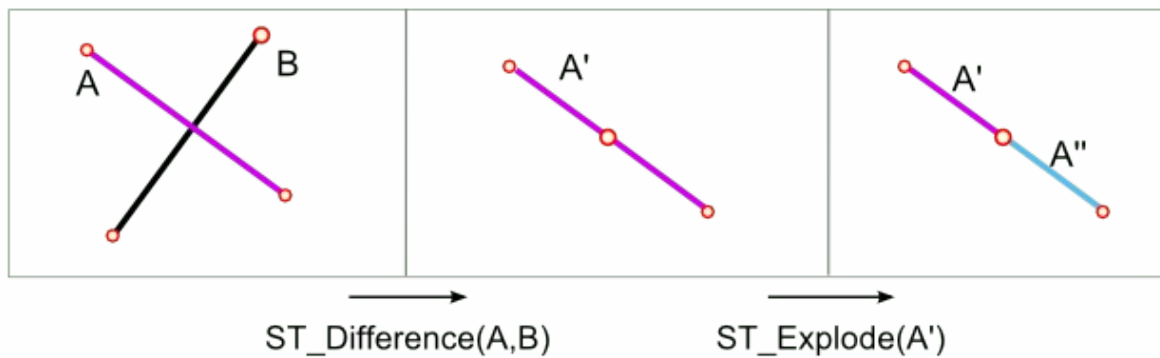


Figure 11: Étapes pour le découpage des arcs se croisant

A l'issue de ces traitements, l'utilisateur dispose d'un jeu de données dans lequel un point est placé à toutes les intersections des tronçons. La figure 12 illustre les corrections ainsi apportées.

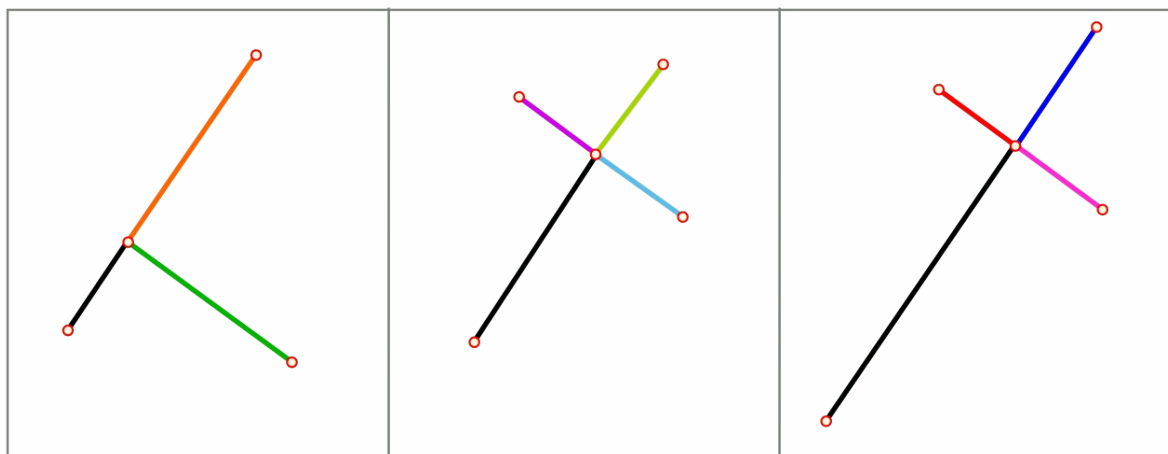


Figure 12: Correction des erreurs topologiques aux croisements

Cas des tronçons pendants

Nous appelons tronçon « pendant » une géométrie dont l'un des points extrêmes (départ et arrivée) est d'ordre 1. Le calcul de cet ordre est réalisé à partir de la structuration du réseau routier sous la forme d'un graphe. Cette mesure consiste à évaluer le nombre d'arcs entrants et sortants pour un nœud, aussi appelée degré. L'identification des tronçons pendants permet de localiser des erreurs topologiques potentielles sur un réseau routier (figure 13) .

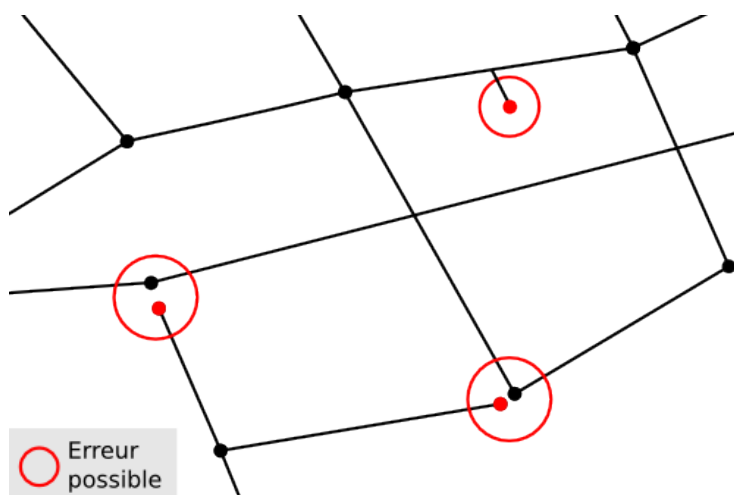


Figure 13: Cas possible d'erreur sur les sommets pendants

La qualification des tronçons pendants dans un réseau routier nécessite une bonne connaissance de la nature des tronçons et de l'organisation du réseau. En effet, un tronçon pendent ne peut systématiquement être interprété comme une erreur de précision géométrique. L'identification mathématique doit être confirmée ou infirmée par des informations sémantiques qui permettent de recontextualiser le tronçon : doit-il être raccroché ou non aux géométries qui lui sont proches ? Si oui lesquelles ?

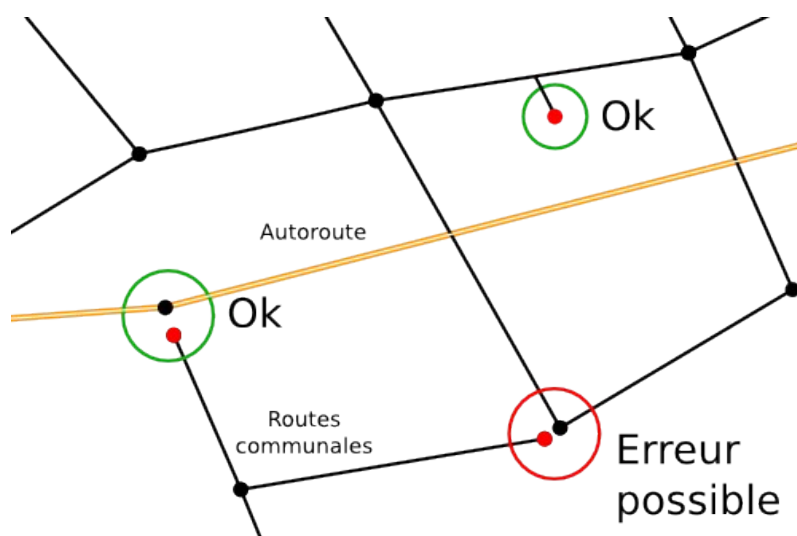


Figure 14: Distinction des cas possibles d'erreur sur les sommets pendants en fonction de la sémantique

Dans le cadre de ce projet, nous avons opté pour deux approches :

1. une identification puis une cartographie des tronçons pendants qui peuvent ensuite être corrigés manuellement au cas par cas (figure 14);
2. une correction globale de tous les tronçons pendants d'un réseau.

Identification et cartographie des tronçons pendants

L'identification des tronçons pendants repose sur l'analyse des nœuds et des arcs du réseau routier calculés à partir de la fonction `ST_Graph`. La requête "NODES_DEGREE" (présentée ci-dessous) est mise en place pour dénombrer le nombre total d'arcs entrants et sortants pour chaque nœud du graphe. La table de sortie conserve pour chaque nœuds du réseau routier, l'identifiant du nœud, sa géométrie et son degré.

```
SELECT ST_GRAPH('TRONCON_ROUTE');
CREATE INDEX ON TRONCON_ROUTE_EDGES(START_NODE);
CREATE INDEX ON TRONCON_ROUTE_EDGES(END_NODE);

DROP TABLE IF EXISTS NODES_DEGREE;
CREATE TABLE NODES_DEGREE
  AS SELECT NODES.NODE_ID, NODES.THE_GEOM, COUNT(*) AS DEGREE
  FROM TRONCON_ROUTE_NODES NODES
  JOIN TRONCON_ROUTE_EDGES EDGES
  ON NODES.NODE_ID IN( EDGES.START_NODE, EDGES.END_NODE)
  GROUP BY NODES.NODE_ID
  ORDER BY NODES.NODE_ID;
```

La figure 15 présente une cartographie des nœuds pendants (en rouge) sur le réseau routier Route 120®. Nous remarquerons d'une part qu'il existe très peu de nœuds pendants sur ce jeu de données et d'autre part que ces nœuds qualifient des "finistères" routiers (tronçons qui se terminent vers la mer pour l'essentiel).

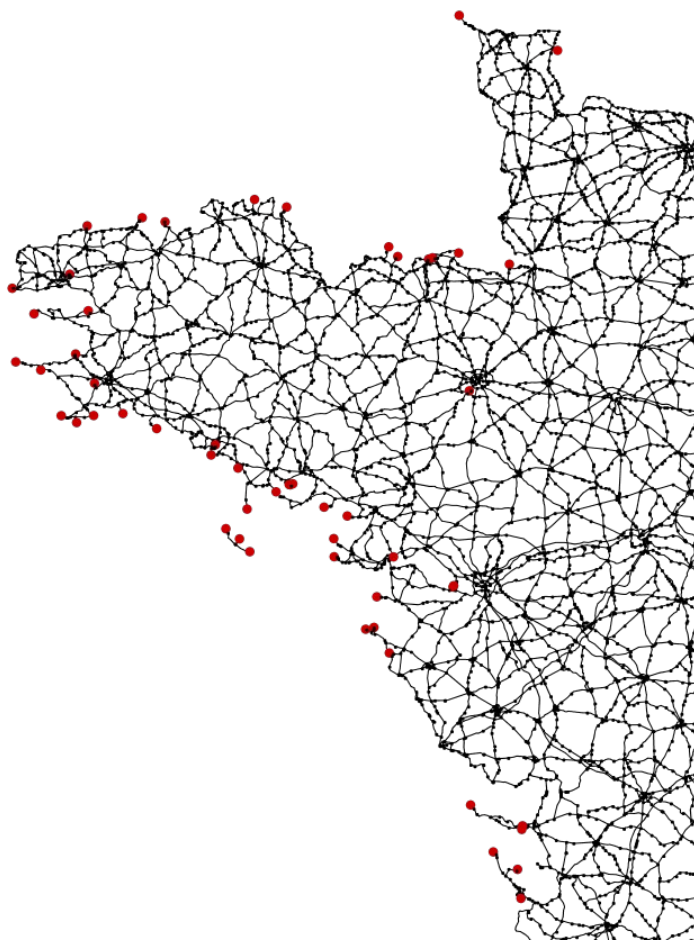


Figure 15: Cartographie des nœuds pendants (points rouge) -
Extrait sur le réseau routier ROUTE 120®

Dans l'hypothèse, ou l'utilisateur se trouve en présence d'une multitude de tronçons pendants, le calcul de la distance entre le nœud pendant et le point le plus proche dans le réseau routier peut constituer une première étape pour filtrer les tronçons qui seront à corriger. Ainsi, si le sommet pendant est à moins de x mètres d'un autre sommet, alors nous pouvons considérer qu'il est suspect et qu'il mérite d'être analysé plus en détail. A l'inverse, si la distance est suffisamment grande entre les sommets, alors nous considérons qu'il s'agit bien de l'extrémité d'un graphe. La requête SQL ci-dessous calcule cette distance.

```
SELECT D.NODE_ID, ST_DISTANCE(D.THE_GEOM, ST_CLOSESTPOINT(R.THE_GEOM, D.THE_GEOM))
AS DISTANCE
FROM ROADS R, NODES_DEGREE D
WHERE ST_INTERSECTS(ST_EXPAND(D.THE_GEOM, 50, 50), R.THE_GEOM)
ORDER BY ST_DISTANCE(D.THE_GEOM, R.THE_GEOM) LIMIT 1;
```

Cette analyse peut être affinée en intégrant, comme nous l'avons mentionné précédemment, les caractéristiques des tronçons. Par exemple, si un tronçon pendant a des propriétés similaires au tronçon potentiel de raccordement ou bien encore si des tronçons pendants proches décrivent les mêmes objets, alors un raccordement géométrique peut être opéré. La figure 16 illustre ce cas.

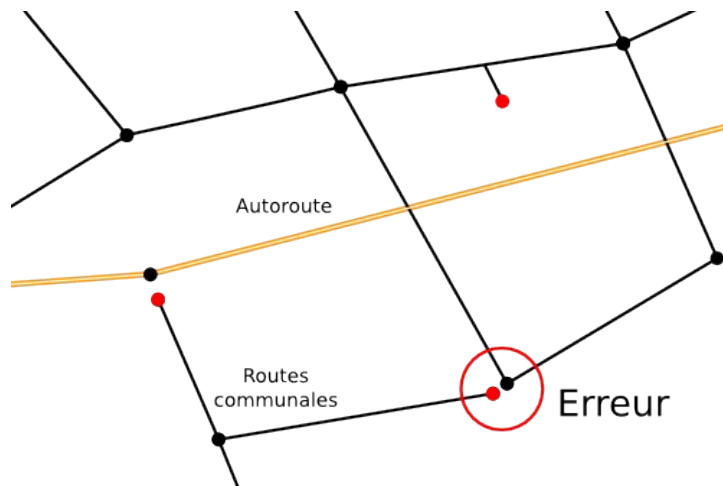


Figure 16: Mise en évidence des sommets pendants à corriger

Correction global des tronçons pendants

La correction global des tronçons pendants consiste à faire l'hypothèse que l'ensemble des nœuds pendants du réseau routier doivent être raccordé entre eux (pas de distinction sémantique). Ce traitement est intégré dans la fonction `ST_Graph`. Pour l'activer l'utilisateur doit spécifier une tolérance de raccrochage, exprimée dans l'unité de la géométrie.

La figure 17 ci-dessous schématise les différents cas qui seront traités.

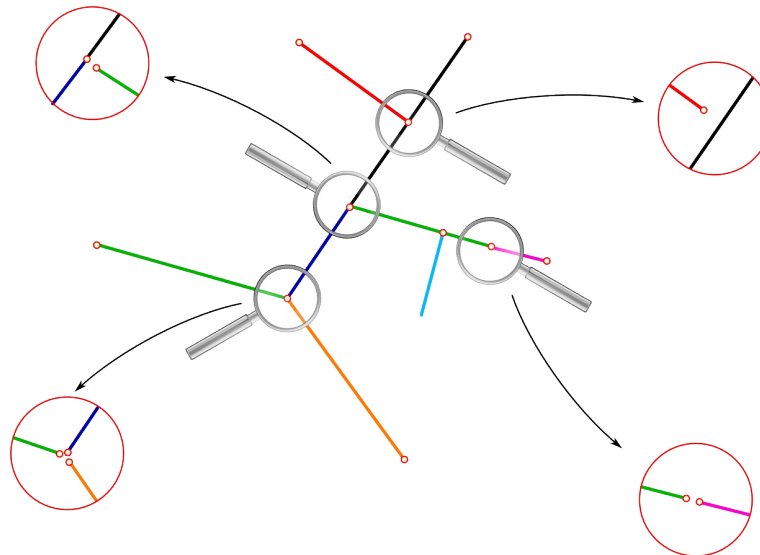


Figure 17: Schématisation des différents cas de tronçons pendants corrigés

5.1.3 Pondération des tronçons du réseau routier

La pondération des tronçons consiste à enrichir le réseau routier initial avec de nouveaux attributs permettant, par la suite, une analyse plus fine du graphe. Pour la plupart des applications, il est nécessaire de valuer le graphe, c'est-à-dire d'affecter un coût (aussi appelé poids) aux tronçons. Usuellement, il s'agit de la distance (longueur du tronçon) et du temps de trajet (vitesse moyenne, maximale). La littérature foisonne d'exemples à ce sujet. Les approches les plus simples consistent à pondérer le graphe sur la base des attributs disponibles dans les bases de données routières. Ainsi Di Salvo (2006), propose une affectation des vitesses selon la vocation des tronçons pour les réseaux routiers IGN-BDCarto® et TéléAtlas-Multinet® (Tableau 2).

IGN - BDCarto® Attribut "vocation"	TéléAtlas - Multinet® Attribut "FRC" (Functional Road Class)	Vitesse (en km/h)
Pas présent	Other road	5
Piste cyclable	Pas présent	10
Liaison locale	- Local Road of Minor Importance - Local Road - Local Road of High importance - Local Connecting Road	20
Bretelle	Pas présent	40
Liaison régionale	Secondary Road	50
Liaison principale	Other Major Road	60
Type autoroutier	- Motorway, Freeway or Other Major Road - Major Road less Important than a Motor Way	110

Tableau 2 : Correspondance entre les attributs des tronçons routiers de IGN-BDCarto® et TéléAtlas-Multinet® et des vitesses de circulation (source : Di Salvo, 2006)

Le coût est ensuite formulé comme étant le rapport de la distance sur la vitesse.

D'autres approches (Hilal (2008), Palmier (2011)) proposent de codifier les tronçons routiers et d'attribuer les vitesses de circulation en fonction de paramètres environnant. Par exemple, la vitesse pourra être plus ou moins élevée si le tronçon est situé ou non à l'intérieur du boulevard périphérique d'une grande ville. De même, s'il est situé dans une zone fortement peuplée (e.g une grande agglomération), la vitesse moyenne constatée sera plus faible que la vitesse légale (ou moyenne) située en campagne. Ces approches nécessitent l'utilisation de données complémentaires qui sont croisées avec la localisation des tronçons. Le tableau 3 donne un récapitulatif des vitesses appliquées par Odomatrix en fonction des trois paramètres : l'unité administrative (environnement), la population et le type de tronçon routier.

Tableau 4 : vitesses de circulation aux heures creuses et aux heures de pointe selon l'environnement géographique et le type de voie

Environnement	Population pôle urbain	Type de voie	Vitesse en heure	
			creuse (HC)	de pointe (HP)
Ville centre des aires urbaines	plus de 200 000 habitants	autoroute	65	35
		2x2 voies	30	16
		principale et régionale	25	14
		locale à 1 voie	20	11
		bretelle	60	42
	entre 100 000 et 200 000 habitants	autoroute	65	41
		2x2 voies	30	19
		principale et régionale	25	16
		locale à 1 voie	20	13
		bretelle	60	47

Tableau 3: extrait de l'attribution des vitesses selon l'environnement géographique et le type de voie selon Odomatrix (source : Hilal 2008)

Nous présentons ci-dessous une méthodologie pour qualifier les vitesses des tronçons du réseau routier ROUTE 120[®] de l'IGN en combinant la nature des tronçons et les aires urbaines qu'ils traversent.

La chaîne de traitement des données repose sur une approche "top - down" qui consiste à partir du général (le réseau routier) pour aller vers le particulier (qualification des vitesses en fonction des unités spatiales).

Étape 1

Les vitesses des tronçons sont déterminées à partir des attributs "VOCATION" et "ACCES"⁴⁹. Le tableau 4 les synthétise

Catégorie de tronçon	Vitesse retenue (km/h)
Réseau autoroutier à péage	130
Réseau autoroutier libre	110
Liaisons principales	90
Liaisons régionales	90
Liaisons locales	90
Bretelle	50
Autres	50

Tableau 4: Vitesses de circulation retenues sur le réseau routier Route 120[®]

49 http://professionnels.ign.fr/sites/default/files/DC_ROUTE120-1-1.pdf consulté en octobre 2014

La requête SQL ci-dessous permet de réaliser l'affectation des vitesses sur la table "ROADS" qui représente le réseau Route 120®.

```
ALTER TABLE ROADS ADD COLUMN VITESSE INT DEFAULT 50;
UPDATE ROADS SET VITESSE=130 WHERE VOCATION='Type autoroutier';
UPDATE ROADS SET VITESSE=110 WHERE ACCES='A péage' AND VOCATION='Type autoroutier';
UPDATE ROADS SET VITESSE=90 WHERE VOCATION='Liaison locale'
OR VOCATION='Liaison principale'
OR VOCATION='Liaison régionale';
UPDATE ROADS SET VITESSE=50 WHERE VOCATION='Bretelle';
```

Étape 2

Nous affinons les vitesses en fonction des aires urbaines. Cette information est fournie à l'échelle des communes, sur l'ensemble du territoire français. Les aires urbaines sont listées dans un tableau fourni par l'INSEE⁵⁰ (livrée au format .xls). Ce tableau contient notamment les colonnes CATAEU2010 et TAU2010 (tableaux 5 et 6). Le code INSEE des communes est utilisé comme identifiant unique.

CATAEU2010 : Catégorie de la commune dans le zonage en aires urbaines 2010.

Codification et signification :

- 111 : Commune appartenant à un grand pôle (10 000 emplois ou plus)
- 112 : Commune appartenant à la couronne d'un grand pôle
- 120 : Commune multipolarisée des grandes aires urbaines
- 211 : Commune appartenant à un moyen pôle (5 000 à moins de 10 000 emplois)
- 212 : Commune appartenant à la couronne d'un moyen pôle
- 221 : Commune appartenant à un petit pôle (de 1 500 à moins de 5 000 emplois)
- 222 : Commune appartenant à la couronne d'un petit pôle
- 300 : Autre commune multipolarisée
- 400 : Commune isolée hors influence des pôles

Tableau 5 : Catégorie de la commune au sein du découpage en aires urbaines pour la Région Pays de la Loire.

Source : http://www.insee.fr/fr/insee_regions/pays-de-la-loire/themes/etudes/etudes98/zau2010_pdl.xls

TAU2010 : Tranche d'aire urbaine 2010.

Codification et signification :

- 00 Commune hors aire urbaine
- 01 Commune appartenant à une aire urbaine de moins de 15 000 habitants
- 02 Commune appartenant à une aire urbaine de 15 000 à 19 999 habitants
- 03 Commune appartenant à une aire urbaine de 20 000 à 24 999 habitants
- 04 Commune appartenant à une aire urbaine de 25 000 à 34 999 habitants

⁵⁰ http://www.insee.fr/fr/methodes/default.asp?page=zonages/aires_urbaines.htm consulté en septembre 2014

- 05 Commune appartenant à une aire urbaine de 35 000 à 49 999 habitants
- 06 Commune appartenant à une aire urbaine de 50 000 à 99 999 habitants
- 07 Commune appartenant à une aire urbaine de 100 000 à 199 999 habitants
- 08 Commune appartenant à une aire urbaine de 200 000 à 499 999 habitants
- 09 Commune appartenant à une aire urbaine de 500 000 à 9 999 999 habitants
- 10 Commune appartenant à l'aire urbaine de Paris

Tableau 6 : Tranche de taille de l'aire urbaine à laquelle appartient la commune au recensement de la population 2008 pour la Région Pays de la Loire.

Source : http://www.insee.fr/fr/insee_regions/pays-de-la-loire/themes/etudes/etudes98/zau2010_pdl.xls

Le tableau des aires urbaines est importé dans la base de données d'OrbisGIS sous le nom AIRES_URBAINES. Le jeu de données "Communes" de France disponible dans la base GeoFLA® de l'IGN est également importé sous le nom COMMUNES. Une nouvelle table des communes est produite afin de permettre l'identification géographique des communes en fonction de leur catégorie d'aire urbaine. Cette table est le résultat de la jointure ci-dessous :

```
CREATE TABLE COMMUNES_AIRES_URBAINES
AS SELECT a.*, b.*
FROM COMMUNES a
LEFT JOIN AIRES_URBAINES b ON b.CODGEO=a.INSEE_COM;
```

Note: dans la base GEOFLA®, les villes de Lyon, Marseille et Paris sont découpées en arrondissements ; ce qui n'est pas le cas de la base communale des aires urbaines de l'INSEE ou seules les limites communales sont considérées.

Lors de la jointure des deux tables, il faudra donc veiller à conserver l'ensemble des "communes GEOFLA" en réalisant une "jointure à gauche". Ainsi pour les trois cas cités préalablement, la valeur "null" sera affectée dans les champs relatifs aux données INSEE (ex : les informations relatives aux populations, ...). L'opérateur devra ensuite faire les mises à jour qui s'imposent pour ces cas particuliers.

Les unités communales, enrichies des informations INSEE (table : COMMUNES_AIRES_URBAINES), sont utilisées pour découper les tronçons routier. Lors de ce découpage l'attribut CATAEU2010 est conservé. La chaîne d'instructions SQL suivante permet de réaliser cette opération.

```
CREATE TABLE ROADS_AIRES_URBAINES (THE_GEOM GEOMETRY, GID SERIAL, ID_RTE120
CHAR(15), VOCATION CHAR(18), NB_CHAUSSE CHAR(11), NB_VOIES CHAR(26), ACCES
CHAR(10), SENS CHAR(12), ONEWAY INT, LENGTH DOUBLE, VITESSE INT, CATAEU2010
CHAR(5))
AS SELECT ST_Intersection(a.the_geom, b.the_geom) AS THE_GEOM, null,
a.ID_RTE120, a.VOCATION, a.NB_CHAUSSE, a.NB_VOIES, a.ACCES, a.SENS, a.ONEWAY,
a.LENGTH, a.VITESSE, b.CATAEU2010
FROM ROADS a, COMMUNE_AU b
WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersects(a.THE_GEOM, b.THE_GEOM);
```

Les deux vignettes de la figure 18 illustrent cette opération spatiale.

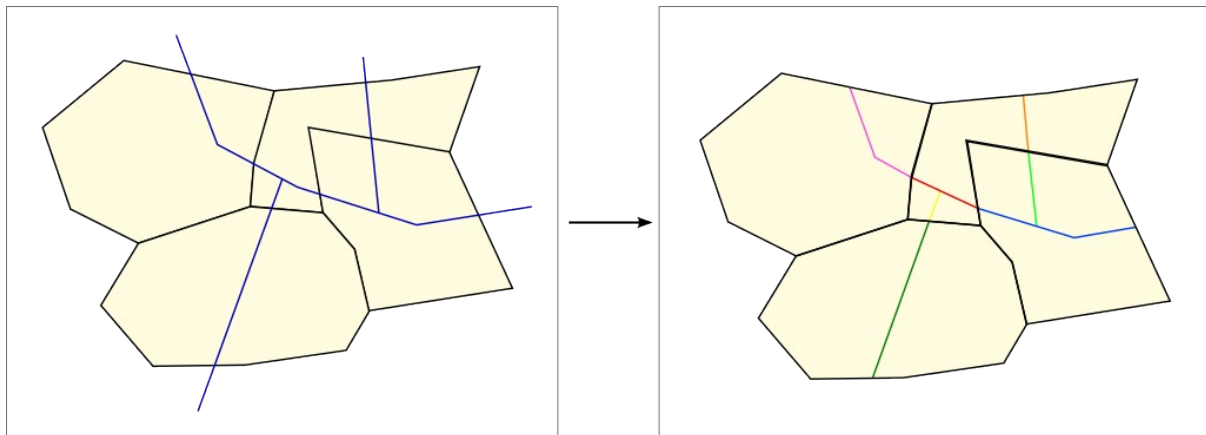


Figure 18: Découpage des tronçons routiers à partir des unités communales

Selon la précision planimétrique des jeux de données, ce traitement des tronçons peut être partiel. En effet, comme le montre la figure 19, il n'est pas omis que des tronçons soient en dehors des limites communales notamment sur les zones frontalières ou bien littorales .

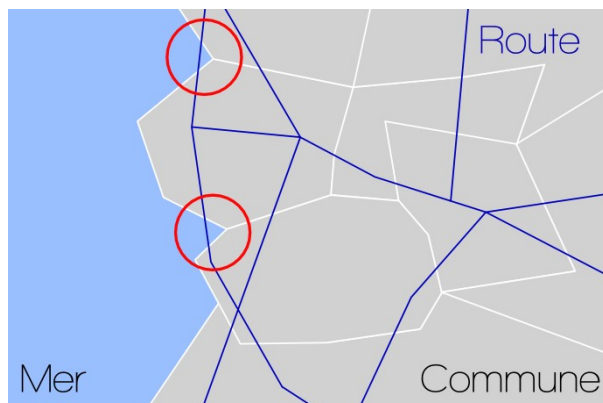


Figure 19: Illustration du problème des tronçons littoraux

Pour éviter que ces objets ne soient exclus des analyses et induisent ainsi des ruptures de continuités dans le réseau, nous ajoutons une étape supplémentaire de traitements (figure 20).

- les communes de France sont fusionnées pour produire une seule géométrie,
- une zone tampon de 2 000 km est réalisée autour des contours de la France,
- une différence géométrique entre la zone tampon et l'emprise des contours des communes de France est effectuée,
- les tronçons routiers sont découpés à partir de la zone tampon des contours.

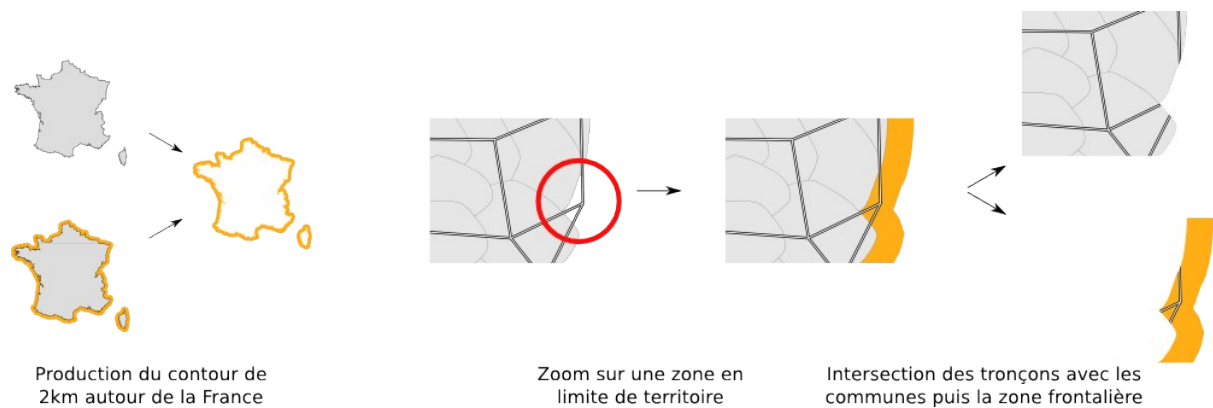


Figure 20: Méthode de découpage des tronçons en limite de zone d'étude

```

CREATE TABLE FUSION_COMMUNES
AS SELECT ST_UNION(ST_ACCUM(THE_GEOM)) AS THE_GEOM FROM COMMUNE;

CREATE TABLE BUFFER_FRANCE
AS SELECT ST_BUFFER(THE_GEOM, 2000) AS THE_GEOM FROM FUSION_COMMUNES ;

CREATE TABLE RING_FRANCE
AS SELECT ST_DIFFERENCE(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM
FROM FUSION_COMMUNES AS a, BUFFER_FRANCE AS b;

CREATE TABLE ROADS_RING(THE_GEOM GEOMETRY, GID SERIAL, ID_RTE120 CHAR(15), VOCATION
CHAR(18), NB_CHAUSSE CHAR(11), NB_VOIES CHAR(26), ACCES CHAR(10), SENS CHAR(12),
ONEWAY INT, LENGTH DOUBLE, VITESSE INT, CATAEU2010 CHAR(5))
AS SELECT ST_Intersection(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM, null,
a.ID_RTE120, a.VOCATION, a.NB_CHAUSSE, a.NB_VOIES, a.ACCES, a.SENS,
a.ONEWAY, a.LENGTH, a.VITESSE, -1
FROM ROADS AS a, RING_FRANCE AS b
WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersects(a.THE_GEOM, b.THE_GEOM);
    
```

Finalement, les deux tables "ROADS_AIRES_URBAINES" et "ROADS_RING" sont réunies.

```

CREATE TABLE FINAL_ROADS AS
SELECT * FROM ROADS_AIRES_URBAINES
UNION
SELECT * FROM ROADS_RING;
    
```

Étape 3

L'ultime étape consiste à mettre à jour la colonne VITESSE de la table unique des routes (FINAL_ROADS) à partir des informations sur la catégorie d'aires urbaines, la vocation, le nombre de chaussées et le type d'accès pour chaque tronçon.

```

UPDATE FINAL_ROADS SET vitesse=130 WHERE (CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée'));

UPDATE FINAL_ROADS SET vitesse=85 WHERE (CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (nb_chausse='2 chaussées' AND acces<>'A péage');

UPDATE FINAL_ROADS SET vitesse=70 WHERE ((CATAEU2010='112' OR CATAEU2010='212' OR
CATAEU2010='222') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée')) OR ((CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (vocation='Liaison principale' OR vocation='Liaison
régionale') AND nb_chausse='1 chaussée'));

UPDATE FINAL_ROADS SET vitesse=65 WHERE (CATAEU2010='111' OR CATAEU2010='211' OR
CATAEU2010='221') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée'));

UPDATE FINAL_ROADS SET vitesse=60 WHERE ((CATAEU2010<>'112' OR CATAEU2010<>'212' OR
CATAEU2010<>'222') AND vocation='Bretelle') OR ((CATAEU2010='120' OR
CATAEU2010='300' OR CATAEU2010='400') AND vocation='Liaison locale' AND
nb_chausse='1 chaussée');

UPDATE FINAL_ROADS SET vitesse=40 WHERE (CATAEU2010='112' OR CATAEU2010='212' OR
CATAEU2010='222') AND (nb_chausse='2 chaussées' AND acces<>'A péage');

UPDATE FINAL_ROADS SET vitesse=30 WHERE ((CATAEU2010='111' OR CATAEU2010='211' OR
CATAEU2010='221') AND (nb_chausse='2 chaussées' AND acces<>'A péage')) OR
((CATAEU2010='112' OR CATAEU2010='212' OR CATAEU2010='222') AND (vocation='Liaison
principale' OR vocation='Liaison régionale') AND nb_chausse='1 chaussée');

UPDATE FINAL_ROADS SET vitesse=25 WHERE (CATAEU2010='111' OR CATAEU2010='211' OR
CATAEU2010='221') AND (vocation='Liaison principale' OR vocation='Liaison
régionale') AND nb_chausse='1 chaussée';

UPDATE FINAL_ROADS SET vitesse=20 WHERE ((CATAEU2010<>'120' AND CATAEU2010<>'300'
AND CATAEU2010<>'400') AND vocation='Liaison locale' AND nb_chausse='1 chaussée')
OR ((CATAEU2010='112' OR CATAEU2010='212' OR CATAEU2010='222') AND
vocation='Bretelle');

```

5.1.4 Raccordement de nouveaux points de navigation au réseau routier

Les calculs de plus courts chemins, de distances sur un graphe se basent sur un (ou plusieurs) point(s) de départ vers une (ou plusieurs) destination(s). Quelques soit ces points, que nous appellerons ici “*points de navigation*”, il est impératif qu’ils fassent partie du réseau routier.

Usuellement, lorsqu’un point de navigation n’est pas présent dans le graphe, celui-ci est raccordé au nœud le plus proche dans le graphe. Les critères de voisinage, de distance sont alors utilisés. Cette approche a l’inconvénient de dépendre de la structure topologique du réseau. Elle peut induire des raccords incohérents. Par exemple, un point de navigation peut être relié à un nœud éloigné dans le graphe dans le cas d’une distance de recherche élevée ou inversement ne pas trouver de nœud candidat dans la cas d’une faible distance (figure 21).

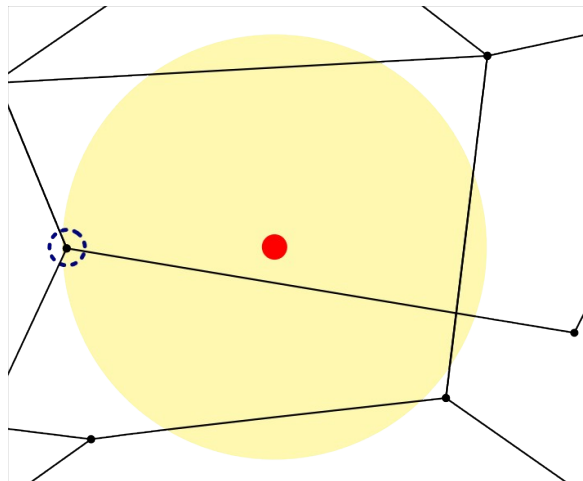


Figure 21: Raccordement d'un point de navigation à un point existant dans un réseau routier

Pour s'affranchir de cette difficulté, nous proposons une méthode de raccordement des nœuds de navigation donc l'objectif est d'enrichir avec un nouveau tronçon le réseau routier. Cette méthode est décomposée en 4 étapes.

Étape 1 : Recherche des tronçons les plus proches pour un point de navigation

Pour un point de navigation donné, nous recherchons l'ensemble des tronçons du réseau routier qui s'intersectent avec une zone de sélection rectangulaire (figure 22). Parmi les candidats retenus celui qui est situé à la distance la plus faible du point de navigation est conservé.

Dans cette figure (n°22) :

- le point rouge représente le point de navigation que l'on étudie,
- t_1, t_2, t_3, \dots , correspondent aux identifiants des tronçons,
- le carré gris correspond à la zone de recherche,
- les tronçons rouges (t_2, t_3, t_7 et t_9) intersectent la zone de recherche,
- les lignes en pointillés bleu correspondent aux distances (notées d_1, d_2, \dots) entre le point de navigation et le tronçon correspondant.

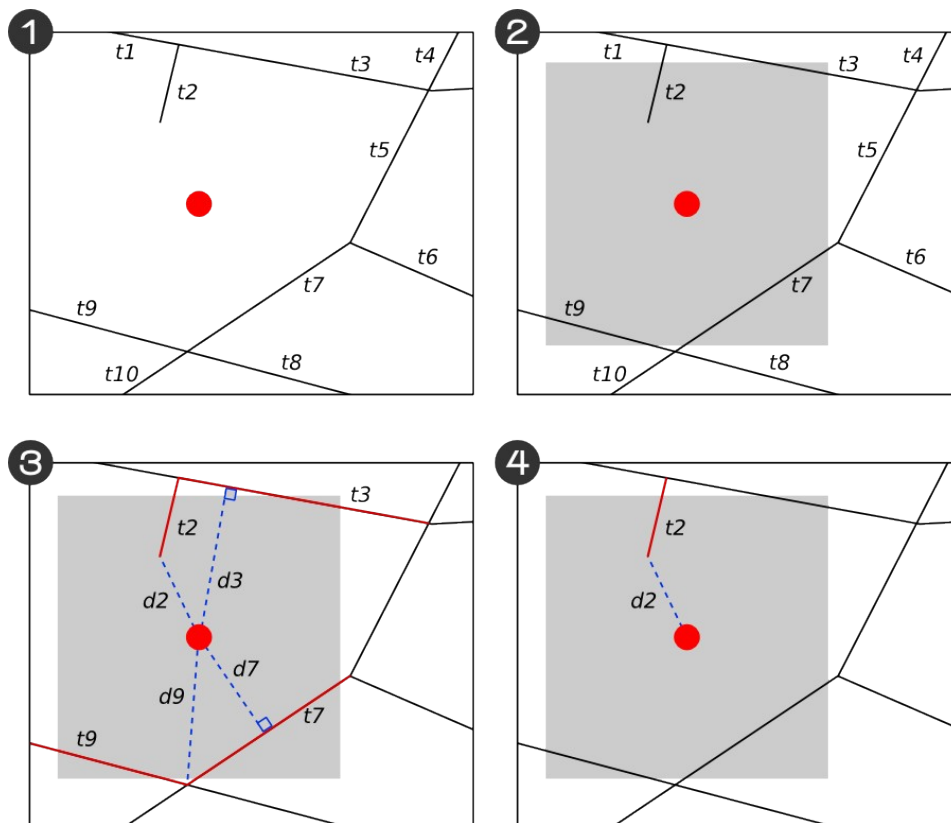


Figure 22: Méthode pour l'identification du tronçon le plus proche d'un nœud de navigation

Pour une table POINTS_NAVIGATION localisant les points de navigation et une table ROADS représentant les tronçons routiers, la requête suivante est appliquée :

```
CREATE TABLE NEAREST_ROADS AS SELECT DT.GID AS NAV_ID, DT.THE_GEOM,
(SELECT R.ID ID
FROM ROADS R, POINTS_NAVIGATION D
WHERE D.GID = DT.GID
AND ST_EXPAND(D.THE_GEOM, 20000, 20000) && R.THE_GEOM
ORDER BY ST_DISTANCE(D.THE_GEOM, R.THE_GEOM) LIMIT 1) AS ROADS_ID
FROM POINTS_NAVIGATION DT;
```

Le résultat de la requête produit une table avec deux colonnes : l'identifiant du nœud (GID qui est renommé en NAV_ID) ainsi que l'identifiant du tronçon le plus proche (ID renommé en ROADS_ID). A noter que :

- LIMIT 1 permet de ne retenir que le candidat le plus proche (après avoir fait un tri sur les distances).
- 20000 correspond aux valeurs x et y (exprimées en mètre) requises dans la fonction ST_Expand. Il s'agit ici de la largeur et de la hauteur du rectangle de la zone de sélection.

Remarque

La fonction *ST_Distance* permet de déterminer la distance la plus faible qui existe entre deux géométries (dans notre cas, entre un point et une ligne). Cette distance correspond à la distance projetée si la projection du nœud sur la ligne se trouve bien sur la ligne. Si jamais ce n'est pas le cas, alors la distance représente la distance entre le nœud et l'extrémité de la ligne la plus proche. A titre d'illustration, dans la figure 23 ci-dessous, les distances retenues pour les tronçons *t1* et *t2* sont représentées en bleu.

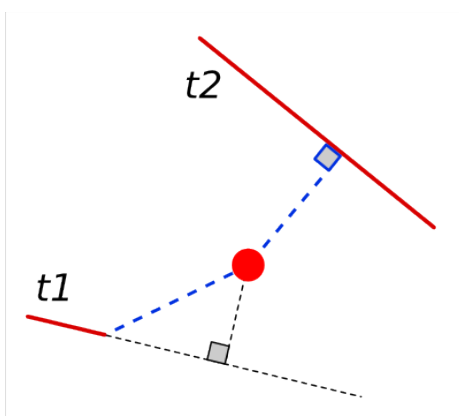


Figure 23: Calcul de la distance la plus courte entre un point et des lignes

Étape 2 : Connexion entre le tronçon et le point de navigation

La connexion entre le tronçon et le point de navigation est matérialisée par une nouvelle ligne. Cette nouvelle ligne comprend le point de navigation et le point le plus proche identifié sur le tronçon candidat. Pour déterminer ce point, la fonction *ST_ClosestPoint*⁵¹ est utilisée. Celle-ci permet de générer le point de la géométrie *A* qui est le plus proche de la géométrie *B*. Une fois ce point identifié, il ne reste plus qu'à générer la ligne à l'aide de la fonction *ST_MakeLine*⁵².

Ces deux étapes sont imbriquées au sein d'une même instruction SQL, présentée ci-dessous :

```
SELECT N.NAV_ID, N.ROADS_ID,
ST_MAKELINE(N.THE_GEOM, ST_CLOSESTPOINT(R.THE_GEOM, N.THE_GEOM)) AS THE_GEOM
FROM NEAREST_ROADS N, ROADS R
WHERE N.ROADS_ID = R.ID;
```

Étape 3 : Découpage du tronçons et affectation des attributs

Cette étape consiste à découper le tronçon sur lequel est raccordé le nœud de navigation (figure 24). La méthode utilisée est identique à celle présentée dans la section « 5.1.2 Correction géométrique du réseau routier ».

51 http://www.h2gis.org/docs/dev/ST_ClosestPoint/ consulté en septembre 2014.

52 http://www.h2gis.org/docs/dev/ST_MakeLine/ consulté en septembre 2014.

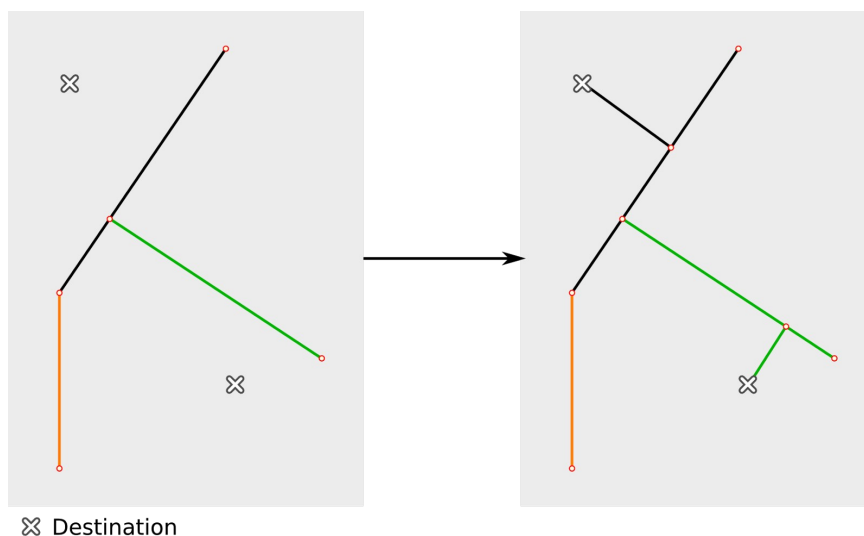


Figure 24: Raccordement des points de navigation et découpage des tronçons routiers

Remarque : L'utilisateur devra veiller à mettre à jour également les attributs nécessaires pour naviguer dans le graphe comme la vitesse, le sens de circulation ou encore la longueur des tronçons

Conclusion

Dans cette partie, nous avons présenté un ensemble de méthodes pour préparer un réseau routier. Ces méthodes passent par une série d'analyses et de traitements : évaluation de la qualité du réseau, correction des intersections, pondération des tronçons, enrichissement local des données... Chacune de ces étapes doit être adaptées aux données d'entrée tant d'un point de vue de leur richesse sémantique que de leur précision géométrique. Dans la section, suivante nous présentons une application complète d'H2Network qui inclue une étape de préparation des données, la construction du graphe et son exploitation.

5.2 Calcul des temps de trajet domicile-travail sur la France métropolitaine

5.2.1 Introduction

Le temps de trajet domicile-travail est un indicateur de développement des territoires. Il exprime le temps de déplacement des actifs entre leur domicile et leur lieu de travail. Sa représentation cartographique permet de matérialiser les disparités spatiales dans les mobilités quotidiennes. Les nombreux travaux sur ce sujet ont ainsi montré que les individus étaient inégaux face à la maîtrise des mobilités notamment au regard de la hiérarchie sociale.

Nous proposons une méthodologie simplifiée pour calculer les durées de trajet routiers entre des couples de points de navigation qui représentent les mobilités professionnelles en France. Ces couples sont issus de la base de données des flux de mobilités professionnelles (domicile - travail) de l'INSEE⁵³. Précisons que les résultats ne peuvent être utilisés à des fins d'analyses de par le nombre limité de paramètres mis en jeu (absence de prise en compte de données démographiques, de la sinuosité d'un tronçon ou encore de ses propriétés topographique ...). L'application a un objectif démonstratif pour présenter les possibilités de la bibliothèque H2Network. Néanmoins, l'ensemble des étapes nécessaires à la mise en œuvre d'analyses sur un réseau routier sont présentes. Elles permettent à un utilisateur averti de construire une chaîne de traitements plus poussés et opérationnels.

5.2.2 Données d'entrées et pré-traitements

Pour calculer les durées de trajet à l'échelle de la France métropolitaine, quatre jeux de données sont utilisés :

- les limites communales de la base de donnée GEOFLA[®] stockées sous la forme de polygones, fichier COMMUNES.SHP,
- le réseau routier de la base de données ROUTE 120[®], représenté par des lignes, fichier TRONCON_ROUTE.SHP,
- les flux de mobilités professionnelles (domicile - travail) par communes pour l'année 2001, fichier Base-flux-mobilite-domicile-lieu-travail-2011.xls ,
- la base communale des aires urbaines de l'INSEE⁵⁴, fichier AU2010.xls.

53 http://www.insee.fr/fr/themes/detail.asp?reg_id=99&ref_id=mobilite-professionnelle-11 consulté en septembre 2014.

54 http://www.insee.fr/fr/methodes/default.asp?page=zonages/aires_urbaines.htm consulté en septembre 2014.

Ces données sont mises en forme et pré-traitées :

- Les limites communales sont unies géométriquement pour produire les limites surfaciques de la France Métropolitaine, fichier `FRANCE.SHP`,
- Le fichier `Base-flux-mobilite-domicile-lieu-travail-2011.xls` est filtré afin de ne conserver que les flux supérieur à 100 individus actifs de 15 ans ou plus ayant un emploi. Cet extrait est enregistré dans le fichier `couple-domicile-lieu-travail-2011.csv`,
- La feuille Composition communale qui regroupe les informations sur la tranche d'aire urbaine et la catégorie de la commune dans le zonage en aires urbaines pour 2010 est enregistrée dans le fichier `AU2010.csv`.

Pour faciliter la démonstration qui va suivre, nous plaçons l'ensemble de ces données au sein d'un dossier que nous appelons "Data_distancier" (figure 25).

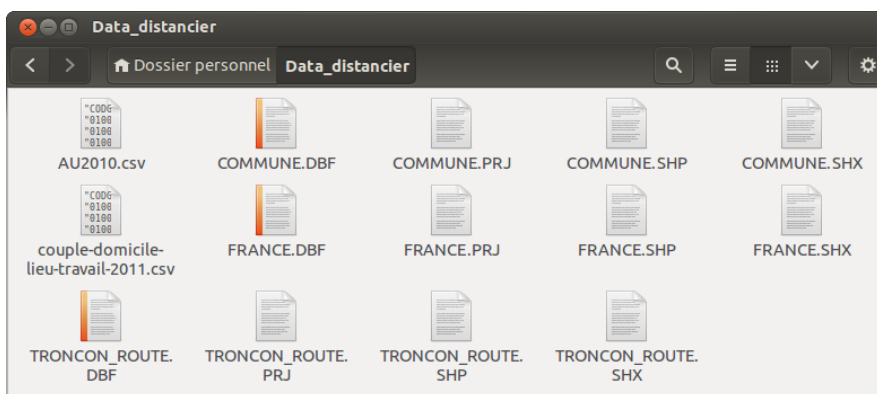


Figure 25: Capture d'écran du dossier dans lequel sont enregistrées les données

5.2.3 Importation des données dans OrbisGIS

5.2.3.1 Préambule

Dans les sections suivantes, plusieurs scripts SQL seront présentés. Afin d'accompagner le lecteur dans la compréhension de ces scripts des commentaires seront systématiquement ajoutés et mis en évidence comme suit :

```
-- Ceci est un commentaire  
--  
-- Le commentaire n'est pas pris en compte lors de l'exécution de l'instruction SQL
```

Les instructions SQL sont exécutées dans la console SQL d'OrbisGIS (figure 26). La visualisation des résultats est effectuée dans la fenêtre Map Editor. Cette fenêtre permet à l'utilisateur de naviguer dans ses données mais également d'accès aux outils de cartographie via la fenêtre de gestion des couches géographiques (TOC).

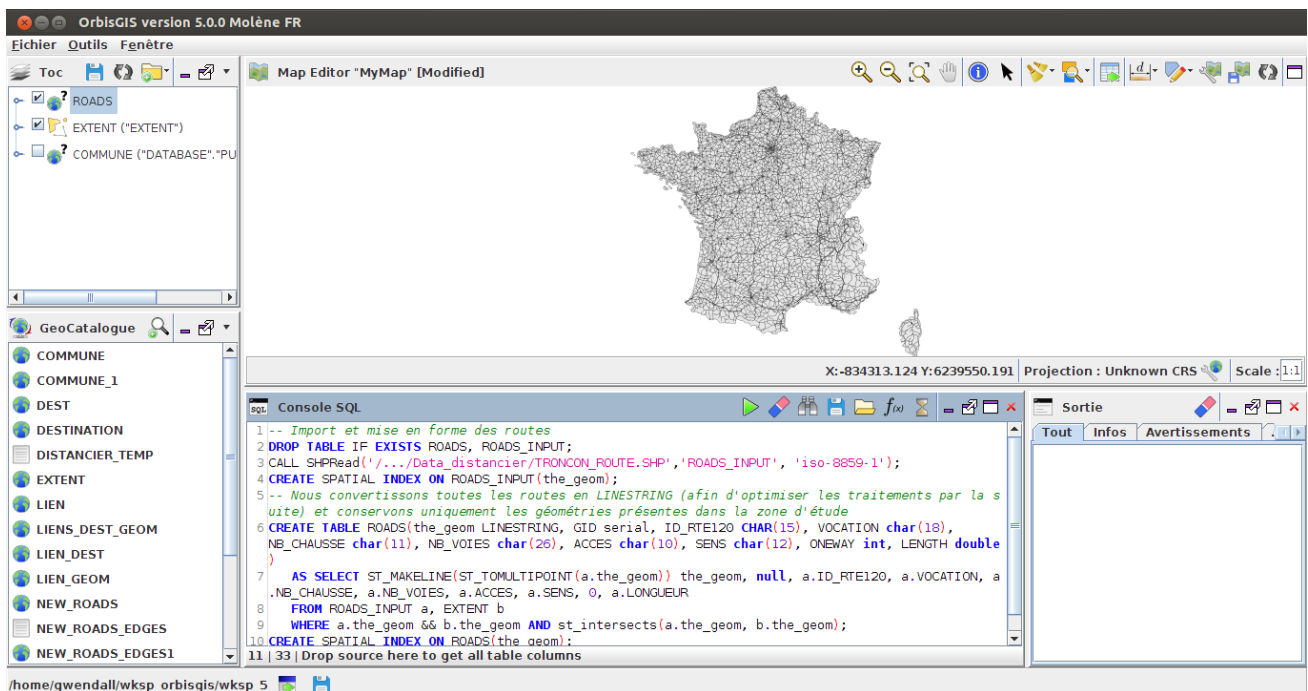


Figure 26: Présentation de l'interface d'OrbisGIS et de la Console SQL

5.2.3.2 Importation des données

L'importation des données dans la base de données consiste à copier le contenu des fichiers dans celle-ci. Elle est réalisée à l'aide des fonctions "SHPRead⁵⁵" et "CSVRead⁵⁶".

Afin de contraindre le domaine d'étude, nous limitons celui-ci à une zone tampon de 2km autour des limites de la France Métropolitaine.

```

-- Suppression des tables existantes
DROP TABLE IF EXISTS ZONE_ETUDE, ZONE_ETUDE_TEMP;
-- Import du fichier FRANCE.SHP qui est dans le dossier "Data_distancier" dans une
table nommée "ZONE_ETUDE_TEMP"
CALL SHPRead('../Data_distancier/FRANCE.SHP', 'ZONE_ETUDE_TEMP');
-- Calcul du domaine d'étude avec une zone tampon de 2km
CREATE TABLE ZONE_ETUDE
AS SELECT ST_BUFFER(THE_GEOM, 2000) AS THE_GEOM
FROM ZONE_ETUDE_TEMP;
--Suppression de la table qui ne servira plus
DROP TABLE ZONE_ETUDE_TEMP;

```

55 <http://www.h2gis.org/docs/dev/SHPRead/> consulté en septembre 2014.

56 <http://www.h2gis.org/docs/dev/CSVRead/> consulté en septembre 2014.

Le réseau routier ROUTE 120® est importé et filtré spatialement afin de ne conserver que les tronçons qui s'intersectent avec le domaine.

Afin d'optimiser le temps de traitement de cette étape, nous découpons le domaine d'étude à l'aide d'une grille (ici avec un pas de 150km). Les cellules ainsi générées nous servent à sélectionner les tronçons routiers les intersectant.

```
-- Import de ROUTE 120®
DROP TABLE IF EXISTS ROUTES, TRONCON_ROUTE;
CALL SHPRead('/.../Data_distancier/TRONCON_ROUTE.SHP', 'TRONCON_ROUTE',
'iso-8859-1');
-- Création d'un index spatial afin d'optimiser la recherche des tronçons
CREATE SPATIAL INDEX ON TRONCON_ROUTE (THE_GEOM);

-- Filtrage spatial des tronçons à partir du domaine
DROP TABLE IF EXISTS GRILLE_FR, ZONE_COUPE;
-- Génération d'une grille sur la France, avec un pas de cellule de 150km
CREATE TABLE GRILLE_FR
  AS SELECT NODE_GEOM as THE_GEOM
  FROM ST_MAKEGRID('ZONE_ETUDE', 150000, 150000) ;
CREATE SPATIAL INDEX ON GRILLE_FR (THE_GEOM);

-- Découpage de la France avec les cellules de la grille précédemment créées
CREATE TABLE ZONE_COUPE
  AS SELECT ST_INTERSECTION(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM
  FROM ZONE_ETUDE a, GRILLE_FR b
  WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersects(a.THE_GEOM, b.THE_GEOM);
CREATE SPATIAL INDEX ON ZONE_COUPE (THE_GEOM);

-- Sélection des tronçons uniques qui intersectent un des éléments de ZONE_COUPE
DROP TABLE IF EXISTS ROUTES;
CREATE TABLE ROUTES
  AS SELECT DISTINCT a.*
  FROM TRONCON_ROUTE a, ZONE_COUPE b
  WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersects(a.THE_GEOM, b.THE_GEOM);
CREATE SPATIAL INDEX ON ROUTES (THE_GEOM);
DROP TABLE GRILLE_FR, ZONE_COUPE, TRONCON_ROUTE;
```

La figure 27 représente le résultat de ces traitements.

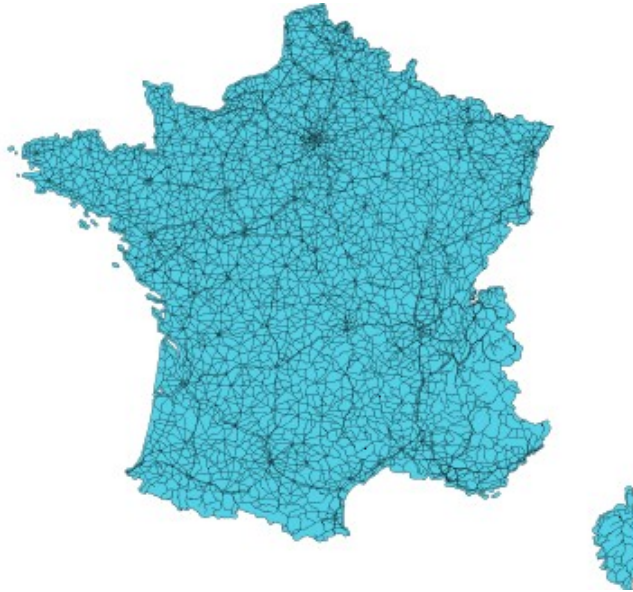


Figure 27: Représentation des couches “ZONE_ETUDE” (en bleu) et “ROUTES” (en noir)

Les communes sont importées ainsi que les couples domicile-travail. Ces couples sont localisés au centre administratif de la commune à laquelle ils appartiennent. Cette étape permet de construire la table des points de navigation à partir de laquelle seront calculées les distances.

```
-- Import des communes
DROP TABLE IF EXISTS COMMUNE;
CALL SHPRead('/.../Data_distancier/COMMUNE.SHP','commune','iso-8859-1');
-- Création des index spatiaux (THE_GEOM) et alphanumérique (INSEE_COM)
CREATE SPATIAL INDEX ON COMMUNE (THE_GEOM);
CREATE INDEX ON COMMUNE (INSEE_COM);
-- Import des couples domicile - travail
DROP TABLE COUPLE_DT IF EXISTS;
CREATE TABLE COUPLE_DT AS SELECT *
  FROM CSVRead('/.../Data_distancier/couple-domicile-lieu-travail-2011.csv');
CREATE INDEX ON COUPLE_DT (CODGEO);
CREATE INDEX ON COUPLE_DT (DCLT);

/* Filtrage des couples domicile-travail afin de construire une liste unique de
communes
INSEE_COM = code INSEE de la commune
CODGEO = code géographique de la commune (ou de l'arrondissement municipal) de
résidence
DCLT : Code géographique de la commune (ou de l'arrondissement municipal) du lieu
de travail
*/
DROP TABLE IF EXISTS COUPLE_DT_FILTRE;
CREATE TABLE COUPLE_DT_FILTRE AS SELECT DISTINCT INSEE_COM FROM
  (SELECT a.INSEE_COM FROM COMMUNE a, COUPLE_DT b
   WHERE a.INSEE_COM=b.CODGEO
  UNION ALL
   SELECT a.INSEE_COM FROM COMMUNE a, COUPLE_DT b
```



```

WHERE a.INSEE_COM=b.DCLT);
CREATE INDEX ON COUPLE_DT_FILTRE (INSEE_COM);

/* Localisation des points de navigation
Les points de navigation sont basés sur les coordonnées du chef lieu des communes
(champs X_CHF_LIEU et Y_CHF_LIEU) (le centre administratif). A noter que pour ces
deux champs, les coordonnées sont arrondies à 10-2. Aussi pour récupérer les bonnes
valeurs, nous ajoutons deux zéros ('00')
*/
DROP TABLE IF EXISTS POINTS_NAV;
CREATE TABLE POINTS_NAV (THE_GEOM POINT, GID SERIAL, ID_GEOFLA INT, NOM_COMM
CHAR(80), INSEE_COM CHAR(7))
AS SELECT ST_MakePoint(CONCAT(CAST(a.X_CHF_LIEU AS VARCHAR(5)), '00'),
CONCAT(CAST(a.Y_CHF_LIEU AS VARCHAR(5)), '00')) AS THE_GEOM, null, a.ID_GEOFLA,
a.NOM_COMM, a.INSEE_COM
FROM COMMUNE a, COUPLE_DT_FILTRE b
WHERE a.INSEE_COM=b.INSEE_COM;
CREATE SPATIAL INDEX ON POINTS_NAV (THE_GEOM);
CREATE INDEX ON POINTS_NAV (GID);
CREATE INDEX ON POINTS_NAV (INSEE_COM);
DROP TABLE COUPLE_DT_FILTRE;

```

	CODGEO	LIBGEO	DCLT	L_DCLT	NBFLUX_C11_ACTOCC15P
1	01004	Ambérieu-en-Bugey	01053	Bourg-en-Bresse	210
2	01004	Ambérieu-en-Bugey	01202	Lagnieu	125
3	01004	Ambérieu-en-Bugey	01262	Montluel	103
4	01004	Ambérieu-en-Bugey	01390	Saint-Vulbas	506
5	01004	Ambérieu-en-Bugey	69383	Lyon 3e Arrondissement	183
6	01007	Ambronay	01004	Ambérieu-en-Bugey	216
7	01008	Ambutrix	01004	Ambérieu-en-Bugey	122
8	01014	Arbent	01031	Bellignat	113

Figure 28: Extrait de la table “COUPLE_DT”

OU

- LIBGEO : Commune (ou arrondissement municipal) de résidence,
- L_DCLT : Commune (ou arrondissement municipal) du lieu de travail,
- NBFLUX_C11_ACTOCC15P : Flux d’actifs de 15 ans ou plus ayant un emploi.

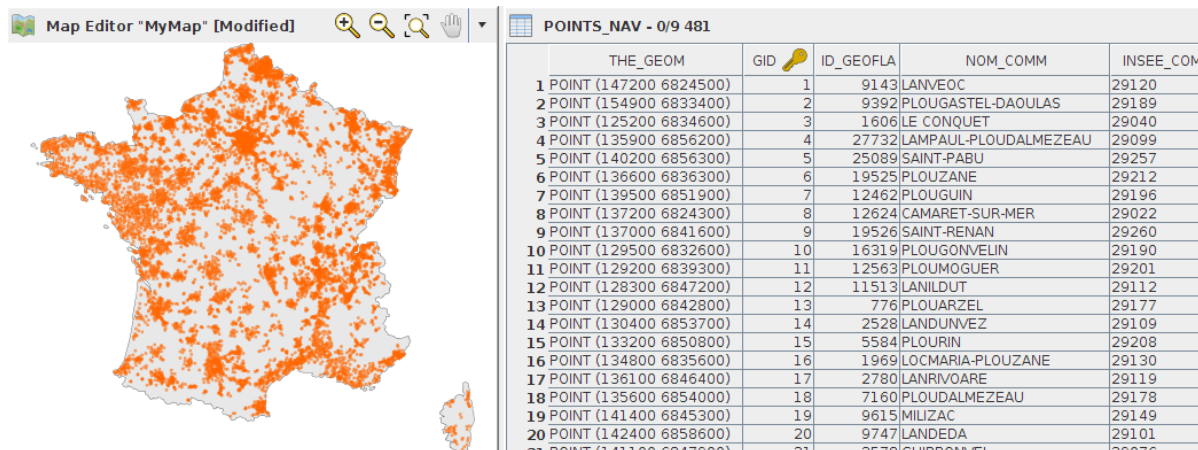


Figure 29: Représentation cartographique (points orange) et attributive de la table "POINTS_NAV"

5.2.4 Pondération des tronçons routiers

L'étape de pondération des tronçons routiers consiste à valuer les géométries en fonction de leur nature et de leur appartenance à une catégorie d'aires urbaines .

Les vitesses des tronçons sont initialisées à partir des catégories disponibles dans la colonne VOCATION. Les valeurs sont stockées dans la colonne VITESSE.

```
ALTER TABLE ROUTES ADD COLUMN VITESSE INT DEFAULT 50;
UPDATE ROUTES SET VITESSE=130 WHERE VOCATION='Type autoroutier';
UPDATE ROUTES SET VITESSE=110 WHERE ACCES='A péage' AND VOCATION='Type
autoroutier';
UPDATE ROUTES SET VITESSE=90 WHERE VOCATION='Liaison locale' OR VOCATION='Liaison
principale' OR VOCATION='Liaison régionale';
UPDATE ROUTES SET VITESSE=50 WHERE VOCATION='Bretelle';
```

Les vitesses sont re-qualifiées à l'aide des catégories d'aires urbaines issues des données de l'INSEE.

```
--Import des aires urbaines de l'INSEE
DROP TABLE IF EXISTS AIRES_URBAINES;
CREATE TABLE AIRES_URBAINES AS SELECT *
FROM CSVRead ('/.../Data_distancier/AU2010.csv');
CREATE INDEX ON AIRES_URBAINES (CODGEO);
CREATE INDEX ON AIRES_URBAINES (CATAEU2010);

-- Récupération des données des aires urbaines pour chaque commune
DROP TABLE IF EXISTS COMMUNE_AU;
CREATE TABLE COMMUNE_AU (THE_GEOM MULTIPOLYGON, GID SERIAL, ID_GEOFLA INT, INSEE_COM
CHAR (5), NOM_COMM CHAR (80), STATUT CHAR (20), SUPERFICIE INT, POPULATION DOUBLE,
CATAEU2010 CHAR (5))
AS SELECT a.THE_GEOM, null, a.ID_GEOFLA, a.INSEE_COM, a.NOM_COMM, a.STATUT,
a.SUPERFICIE, a.POPULATION, b.CATAEU2010
FROM COMMUNE a LEFT JOIN AIRES_URBAINES b ON b.CODGEO=a.INSEE_COM;
```

```
-- Traitement des cas de Marseille, Lyon et Paris qui sont découpés en
arrondissement dans les communes GEOFLA et en ville dans les données INSEE
UPDATE COMMUNE_AU SET CATAEU2010='111'
  WHERE CATAEU2010 IS null AND (nom_comm LIKE 'LYON%'
                                OR nom_comm LIKE 'MARSEILLE%');
UPDATE COMMUNE_AU SET CATAEU2010='111'
  WHERE CATAEU2010 IS null AND nom_comm LIKE 'PARIS%';
CREATE SPATIAL INDEX ON COMMUNE_AU (THE_GEOM);
DROP TABLE COMMUNE, AIRES_URBAINES;
```

	THE_GEOM	GID	ID_GEOFLA	INSEE_COM	NOM_COMM	STATUT	SUPERFICIE	POPULATION	CATAEU2010
1	MULTIPOLYGON (((854602 66258...	1	1	71398	SAINT-CHRISTOPHE-EN-BRESSE	Commune simple	2049	0,9112	
2	MULTIPOLYGON (((855077 66596...	2	2	21708	VILLY-LE-MOUTIER	Commune simple	2017	0,3120	
3	MULTIPOLYGON (((825085 64710...	3	3	42067	COLOMBIER	Commune simple	1785	0,3120	
4	MULTIPOLYGON (((679501 62230...	4	4	41132	FABREZAN	Commune simple	2893	1,2300	
5	MULTIPOLYGON (((473094 62008...	5	5	65123	CAMPAN	Chef-lieu canton	9734	1,5211	
6	MULTIPOLYGON (((425675 62488...	6	6	64445	PAU	Préfecture	3152	84111	
7	MULTIPOLYGON (((507632 67940...	7	7	72102	COURCNVAL	Commune simple	894	0,1120	
8	MULTIPOLYGON (((670110 60707...	8	8	80136	BRAY-SUR-ORNE	Chef-lieu canton	1703	1,3300	

Figure 30: Extrait de la table “COMMUNE_AU”

La re-qualification des vitesses est alimentée par une méthode de jointure spatiale qui consiste dans un premier temps à stocker dans une table (ROUTES_AU) l'intersection des tronçons avec les communes puis dans un deuxième temps à construire le négatif des intersections (table ROUTES_NEG). C'est à dire les routes qui se trouvent en dehors des communes dans une zone tampon de 2km. Ces deux tables sont in-fine unifiées.

```
-- Découpage du réseau routier avec les limites des communes.
DROP TABLE IF EXISTS ROUTES_AU;
CREATE TABLE ROUTES_AU (THE_GEOM GEOMETRY, ID_RTE120 CHAR(15), VOCATION CHAR(18),
NB_CHAUSSE CHAR(11), NB_VOIES CHAR(26), ACCES CHAR(10), SENS CHAR(12), LONGUEUR
DOUBLE, VITESSE INT, INSEE_COM CHAR(5), NOM_COMM CHAR(80), CATAEU2010 CHAR(5))
AS SELECT ST_Intersection(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM, a.ID_RTE120,
a.VOCATION, a.NB_CHAUSSE, a.NB_VOIES, a.ACCES, a.SENS, a.LONGUEUR, a.VITESSE,
b.insee_com, b.nom_comm, b.CATAEU2010
FROM ROUTES a, COMMUNE_AU b
WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersects(a.THE_GEOM, b.THE_GEOM);
```

	THE_GEOM	ID_RTE120	VOCATION	NB_CHAUSSE	NB_VOIES	ACCES	SENS	LONGUEUR	VITESSE	INSEE_COM	NOM_COMM	CATAEU2010
1	LINestring (1...	13193	Liaison régionale	1 chaussée	1 voie ou...	Libre	Double sens	5,1	90	29120	LANVEOC	400
2	LINestring (1...	13179	Liaison régionale	1 chaussée	1 voie ou...	Libre	Double sens	0,3	90	29120	LANVEOC	400
3	LINestring (1...	13178	Liaison régionale	1 chaussée	1 voie ou...	Libre	Double sens	5,8	90	29120	LANVEOC	400
4	LINestring (1...	12267	Type autoroutier	2 chaussées	Sans objet	Libre	Double sens	8,9	130	29189	PLOUGAST...	111
5	LINestring (1...	11967	Liaison régionale	1 chaussée	1 voie ou...	Libre	Double sens	5,5	90	29040	LE CONQUET	112
6	LINestring (1...	12425	Liaison régionale	1 chaussée	1 voie ou...	Libre	Double sens	0,7	90	29040	LE CONQUET	112
7	LINestring (1...	12458	Liaison principale	1 chaussée	2 voies	Libre	Double sens	11	90	29040	LE CONQUET	112

Figure 31: Extrait de la table “ROUTE_AU”

A ce stade nous devons réaliser un traitement afin de prendre en compte les communes frontalières ou bien littorales (*voir section 5.1.3 – Étape 2*).

```
-- Différence entre la zone d'étude et les communes de France
DROP TABLE IF EXISTS ZONE_ETUDE_NEG;
CREATE TABLE ZONE_ETUDE_NEG AS
  SELECT ST_Difference(b.THE_GEOM,ST_UNION(ST_ACCUM(a.THE_GEOM))) AS THE_GEOM
  FROM COMMUNE_AU a, ZONE_ETUDE b;
CREATE SPATIAL INDEX ON ZONE_ETUDE_NEG (THE_GEOM);

-- Intersections des routes comprises dans la zone frontalière à l'aide d'une
grille de 50km
DROP TABLE IF EXISTS GRILLE;
CREATE TABLE GRILLE AS SELECT NODE_GEOM as THE_GEOM, ID
  FROM ST_MAKEGRID('ZONE_ETUDE_NEG', 50000, 50000) ;
CREATE SPATIAL INDEX ON GRILLE (THE_GEOM);

-- Nous décomposons la zone frontalière en lignes
DROP TABLE IF EXISTS NEG_LIGNE, NEG_LIGNE_EXPL;
CREATE TABLE NEG_LIGNE AS
  SELECT ST_TOMULTILINE(THE_GEOM) AS THE_GEOM
  FROM ZONE_ETUDE_NEG;
CREATE TABLE NEG_LIGNE_EXPL AS SELECT * FROM ST_EXPLODE('NEG_LIGNE');
CREATE SPATIAL INDEX ON NEG_LIGNE_EXPL (THE_GEOM);

-- Puis nous isolons les cellules de la grille qui intersectent la zone frontalière

DROP TABLE IF EXISTS NEG_SELECT, NEG_CUT;
CREATE TABLE NEG_SELECT AS SELECT a.*
  FROM GRILLE a, NEG_LIGNE_EXPL b
  WHERE a.THE_GEOM && b.THE_GEOM AND ST_INTERSECTS(a.THE_GEOM, b.THE_GEOM);
CREATE SPATIAL INDEX ON NEG_SELECT (THE_GEOM);

CREATE TABLE NEG_CUT AS
  SELECT ST_INTERSECTION(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM
  FROM ZONE_ETUDE_NEG a, NEG_SELECT b
  WHERE a.THE_GEOM && b.THE_GEOM AND ST_INTERSECTS(a.THE_GEOM, b.THE_GEOM);
CREATE SPATIAL INDEX ON NEG_CUT (THE_GEOM);

-- Enfin nous découpons les tronçons routiers sur la base des cellules
sélectionnées juste avant
DROP TABLE IF EXISTS ROUTES_COUPE, ROUTES_COUPE_UNION;
CREATE TABLE ROUTES_COUPE (THE_GEOM GEOMETRY, ID_RTE120 CHAR(15)) AS
  SELECT ST_Intersection(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM, a.ID_RTE120
  FROM ROUTES a, NEG_CUT b
  WHERE a.THE_GEOM && b.THE_GEOM AND ST_Intersection(a.THE_GEOM, b.THE_GEOM);

-- Pour finir, nous unifions les morceaux de routes qui font référence au même
tronçon initial
CREATE TABLE ROUTES_COUPE_UNION AS
  SELECT ST_UNION(ST_ACCUM(THE_GEOM)) AS THE_GEOM, ID_RTE120
  FROM ROUTES_COUPE
  GROUP BY ID_RTE120;
```

```
-- Et nous associons à cette table les informations relatives aux tronçons
DROP TABLE IF EXISTS ROUTES_NEG;
CREATE TABLE ROUTES_NEG (THE_GEOM GEOMETRY, ID_RTE120 CHAR(15), VOCATION CHAR(18),
NB_CHAUSSE CHAR(11), NB_VOIES CHAR(26), ACCES CHAR(10), SENS CHAR(12), LONGUEUR
DOUBLE, VITESSE INT, INSEE_COM CHAR(5), NOM_COMM CHAR(80), CATAEU2010 CHAR(5))
AS SELECT a.THE_GEOM, a.ID_RTE120, b.VOCATION, b.NB_CHAUSSE, b.NB_VOIES,
b.ACCES, b.SENS, b.LONGUEUR, b.VITESSE, null, null, null
FROM ROUTES_COUPE_UNION a, ROUTES b
WHERE a.ID_RTE120=b.ID_RTE120;
```

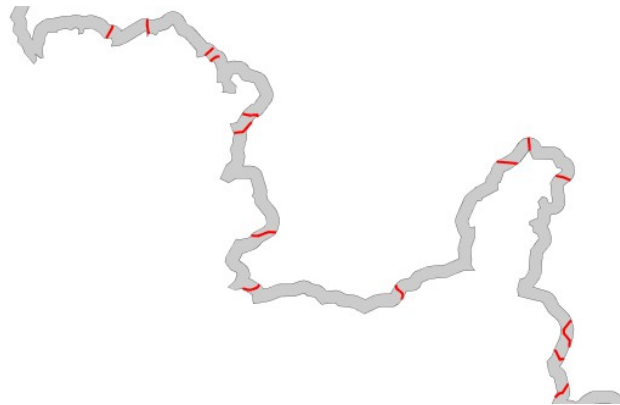


Figure 32: Représentation d'une partie des couches "ZONE_ETUDE_NEG" (en gris) et "ROUTES_NEG" (en rouge)

```
-- Union des tables ROUTES_AU et ROUTES_NEG
DROP TABLE IF EXISTS ROUTES_UNION;
CREATE TABLE ROUTES_UNION AS
SELECT * FROM ROUTES_NEG
UNION ALL
SELECT * FROM ROUTES_AU;

-- Simplification des géométries du réseau routier uni
DROP TABLE IF EXISTS ROUTES_FINAL;
CREATE TABLE ROUTES_FINAL AS SELECT *
FROM ST_EXPLODE('ROUTES_UNION')
WHERE ST_DIMENSION(THE_GEOM)=1;
ALTER TABLE ROUTES_FINAL ADD COLUMN GID SERIAL;
CREATE SPATIAL INDEX ON ROUTES_FINAL(THE_GEOM);
DROP TABLE ROUTES_UNION, ROUTES_NEG, ROUTES_COUPE_UNION;
```

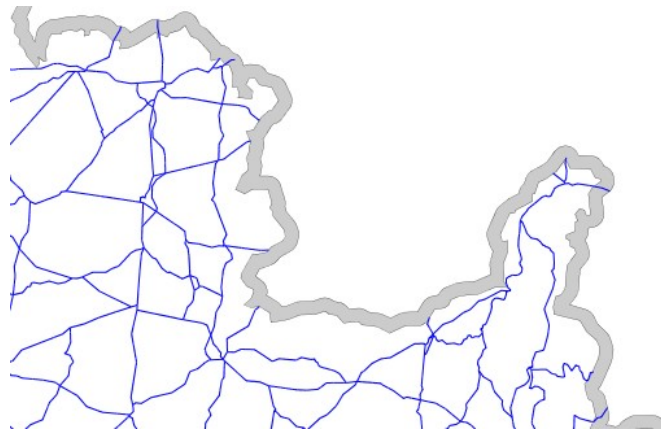


Figure 33: Extrait de la couche "ROUTES_FINAL"

THE_GEOM	ID_RTE120	VOCATION	NB_CHAU...	NB_VOIES	ACCES	SENS	LONGUEUR	VITESSE	INSEE_COM	NOM_COMM	CATAEU2010	EXPLOD_ID	GID
508 LINESTRING (...)	4910	Liaison principale	1 chauss...	2 voies	Libre	Double sens	6,3	90 null	null	null		1	508
509 LINESTRING (...)	5355	Liaison principale	1 chauss...	2 voies	Libre	Double sens	3	90 null	null	null		1	509
510 LINESTRING (...)	5535	Liaison principale	1 chauss...	2 voies	Libre	Double sens	4,6	90 null	null	null		1	510
511 LINESTRING (...)	7554	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	4,1	90 null	null	null		1	511
512 LINESTRING (...)	13193	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	5,1	90 29120	LANVEOC	400		1	512
513 LINESTRING (...)	13179	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	0,3	90 29120	LANVEOC	400		1	513
514 LINESTRING (...)	13178	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	5,8	90 29120	LANVEOC	400		1	514

Figure 34: Illustration de la table "ROUTES_FINAL"

L'ultime chaîne d'instructions consiste à mettre à jour les valeurs de vitesse selon la catégorie de l'aire urbaine et les qualifications des tronçons (nombre de chaussées, vocation). Les classes de vitesses retenues sont inspirées de celles utilisées par Odomatrix.

```

UPDATE ROUTES_FINAL SET VITESSE=130 WHERE (CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée'));
UPDATE ROUTES_FINAL SET VITESSE=85 WHERE (CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (nb_chausse='2 chaussées' AND acces<>'A péage');
UPDATE ROUTES_FINAL SET VITESSE=70 WHERE ((CATAEU2010='112' OR CATAEU2010='212' OR
CATAEU2010='222') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée'))) OR ((CATAEU2010='120' OR CATAEU2010='300' OR
CATAEU2010='400') AND (vocation='Liaison principale' OR vocation='Liaison
régionale') AND nb_chausse='1 chaussée');
UPDATE ROUTES_FINAL SET VITESSE=65 WHERE (CATAEU2010='111' OR CATAEU2010='211' OR
CATAEU2010='221') AND (acces='A péage' OR (vocation='Type autoroutier' AND
nb_chausse='1 chaussée'));
UPDATE ROUTES_FINAL SET VITESSE=60 WHERE ((CATAEU2010<>'112' OR CATAEU2010<>'212'
OR CATAEU2010<>'222') AND vocation='Bretelle') OR ((CATAEU2010='120' OR
CATAEU2010='300' OR CATAEU2010='400') AND vocation='Liaison locale' AND
nb_chausse='1 chaussée');
UPDATE ROUTES_FINAL SET VITESSE=40 WHERE (CATAEU2010='112' OR CATAEU2010='212' OR
CATAEU2010='222') AND (nb_chausse='2 chaussées' AND acces<>'A péage');
UPDATE ROUTES_FINAL SET VITESSE=30 WHERE ((CATAEU2010='111' OR CATAEU2010='211' OR

```

```

CATAEU2010='221') AND (nb_chausse='2 chaussées' AND acces<>'A péage')) OR
((CATAEU2010='112' OR CATAEU2010='212' OR CATAEU2010='222') AND (vocation='Liaison
principale' OR vocation='Liaison régionale') AND nb_chausse='1 chaussée');
UPDATE ROUTES_FINAL SET VITESSE=25 WHERE (CATAEU2010='111' OR CATAEU2010='211' OR
CATAEU2010='221') AND (vocation='Liaison principale' OR vocation='Liaison
régionale') AND nb_chausse='1 chaussée';
UPDATE ROUTES_FINAL SET VITESSE=20 WHERE ((CATAEU2010<>'120' AND CATAEU2010<>'300'
AND CATAEU2010<>'400') AND vocation='Liaison locale' AND nb_chausse='1 chaussée')
OR ((CATAEU2010='112' or CATAEU2010='212' OR CATAEU2010='222') AND
vocation='Bretelle');

```

A l'issue de ce traitement, la vitesse de chaque tronçon est bien mise à jour. La figure 35 ci-dessous en donne un aperçu (colonne VITESSE).

THE_GEOM	ID_RT120	VOCATION	NB_CHAU...	NB_VOIES	ACCES	SENS	LONGUEUR	VITESSE	INSEE_COM	NOM_COMM	CATAEU2010	EXPLOD_ID	GID
508 LINESTRING (...)	4910	Liaison principale	1 chauss...	2 voies	Libre	Double sens	6,3	90 null	null	null	null	1	508
509 LINESTRING (...)	5355	Liaison principale	1 chauss...	2 voies	Libre	Double sens	3	90 null	null	null	null	1	509
510 LINESTRING (...)	5535	Liaison principale	1 chauss...	2 voies	Libre	Double sens	4,6	90 null	null	null	null	1	510
511 LINESTRING (...)	7554	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	4,1	90 null	null	null	null	1	511
512 LINESTRING (...)	13193	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	5,1	70 29120	LANVEOC	400	400	1	512
513 LINESTRING (...)	13179	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	0,3	70 29120	LANVEOC	400	400	1	513
514 LINESTRING (...)	13178	Liaison régionale	1 chauss...	1 voie ou 2 voies...	Libre	Double sens	5,8	70 29120	LANVEOC	400	400	1	514

Figure 35: Mise à jour de la colonne "VITESSE" dans la table "ROUTES_FINAL"

5.2.5 Correction géométrique des tronçons routiers

Il s'agit ici de décomposer tous les tronçon qui présentent une intersection avec un ou plusieurs autres tronçons (cf. section 5.1.2).

Remarque : Ici, les tunnels et les ponts ne sont pas pris en compte car les informations ne sont pas présentes sur la couche "TRONCON_ROUTE" issue de ROUTE 120®.

```

-- Découpage des tronçons
DROP TABLE IF EXISTS ROUTES_DECOUP;
CREATE TABLE ROUTES_DECOUP
AS SELECT CASEWHEN(
ST_Dimension(ST_Intersection(a.THE_GEOM, ST_Union(ST_Accum(b.THE_GEOM)))) = 0,
ST_Difference(a.THE_GEOM,ST_Union(ST_Accum(b.THE_GEOM))),a.THE_GEOM)
AS THE_GEOM,A.VITESSE, A.SENS
FROM ROUTES_FINAL a , ROUTES_FINAL b
WHERE a.GID != b.GID AND a.THE_GEOM && b.THE_GEOM
AND ST_Intersects(a.THE_GEOM, b.THE_GEOM)
GROUP BY a.GID, a.THE_GEOM;

-- Décomposition des géométries multiples en géométries simples
DROP TABLE IF EXISTS ROUTES_CORRECT;
CREATE TABLE ROUTES_CORRECT
AS SELECT *
FROM ST_EXPLODE('ROUTES_DECOUP');
ALTER TABLE ROUTES_CORRECT ADD COLUMN GID SERIAL;
CREATE SPATIAL INDEX ON ROUTES_CORRECT (THE_GEOM);
CREATE INDEX ON ROUTES_CORRECT (GID);
DROP TABLE ROUTES_DECOUP;

```

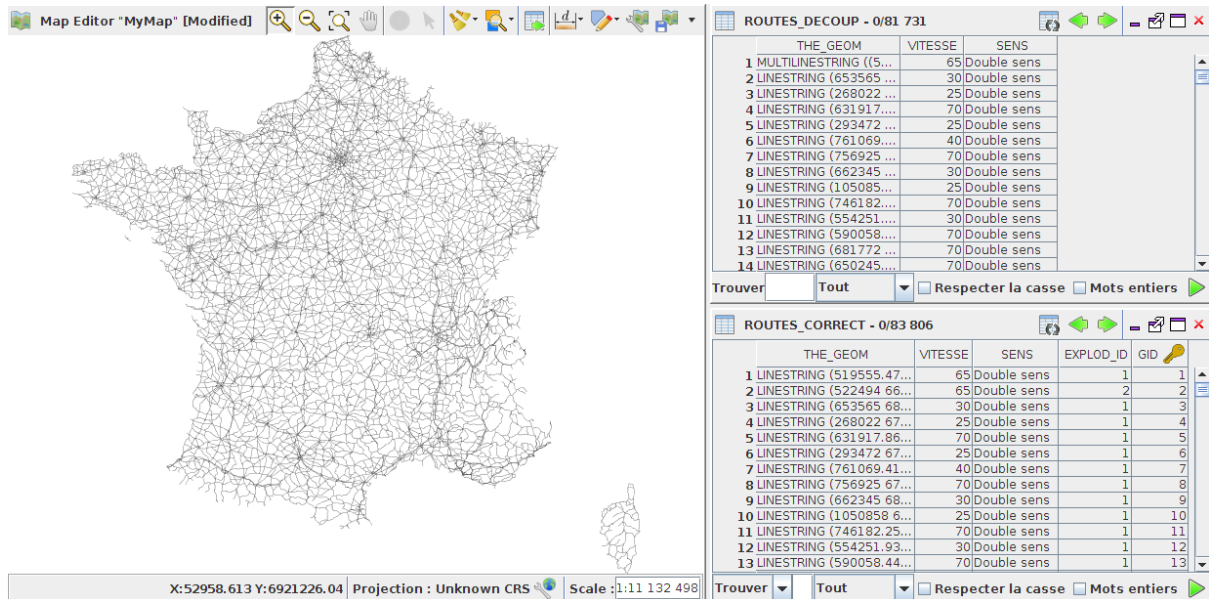



Figure 36: Représentation cartographique et attributaire de la table “*ROUTES_CORRECT*”, ainsi qu’un extrait de la table “*ROUTES_DECROUP*”

5.2.6 Raccordement des points de navigation aux tronçons routiers

A présent, nous raccordons les points de navigation (les points représentant les couples “domicile” et “travail”) au réseau routier. Cette procédure est décrite dans la section 5.1.4.

```
-- Récupération de l'identifiant du tronçon routier le plus proche du point de navigation
DROP TABLE IF EXISTS ROUTES_PROCHE;
CREATE TABLE ROUTES_PROCHE AS SELECT DT.GID AS NAV_GID, DT.THE_GEOM,
  (SELECT R.GID GID FROM ROUTES_CORRECT R, POINTS_NAV D WHERE D.GID = DT.GID
  AND ST_EXPAND(D.THE_GEOM, 20000, 20000) && R.THE_GEOM
  ORDER BY ST_DISTANCE(D.THE_GEOM, R.THE_GEOM) LIMIT 1) AS ROUTES_GID
FROM POINTS_NAV DT;
CREATE INDEX ON ROUTES_PROCHE (ROUTES_GID);

-- Création des liens entre les points de navigation et le tronçon routier le plus proche
DROP TABLE IF EXISTS ROUTES_LIEN;
CREATE TABLE ROUTES_LIEN AS SELECT N.NAV_GID, N.ROUTES_GID,
  ST_MAKELINE(N.THE_GEOM, ST_CLOSESTPOINT(R.THE_GEOM, N.THE_GEOM)) AS THE_GEOM
FROM ROUTES_PROCHE N, ROUTES_CORRECT R
WHERE N.ROUTES_GID = R.GID;
CREATE INDEX ON ROUTES_LIEN (ROUTES_GID);
```



```

-- Découpage des tronçons à partir des liens de navigation
DROP TABLE IF EXISTS ROUTES_LIENS_DECOUPEES, ROUTES_GRAPHE, ROUTES_TEMP;
CREATE TABLE ROUTES_LIENS_DECOUPEES AS
  SELECT R.GID, R.VITESSE, R.SENS,
         ST_LineIntersector(R.THE_GEOM,
                             ST_ACCUM(ST_MAKELINE(ST_STARTPOINT(L.THE_GEOM),
                                                  ST_ENDPOINT(L.THE_GEOM), ST_LOCATEALONG(L.THE_GEOM, 1.1, 0)))
                            ) AS THE_GEOM
  FROM ROUTES_CORRECT R LEFT JOIN ROUTES_LIEN L ON (R.GID = L.ROUTES_GID)
  GROUP BY R.GID, R.THE_GEOM;

-- Union des liens de navigation et des tronçons découpés
-- Une valeur de 50 km/h est affectée pour les liens de navigation
CREATE TABLE ROUTES_TEMP AS
  SELECT THE_GEOM, VITESSE, SENS FROM ROUTES_LIENS_DECOUPEES
  UNION ALL
  SELECT THE_GEOM, 50, null FROM ROUTES_LIEN;

-- Décomposition pour obtenir des géométries simples
CREATE TABLE ROUTES_GRAPHE AS SELECT * FROM ST_EXPLODE('ROUTES_TEMP');

-- Ajout de la colonne longueur
ALTER TABLE ROUTES_GRAPHE ADD COLUMN LONGUEUR DOUBLE;
UPDATE ROUTES_GRAPHE SET LONGUEUR = ST_Length(THE_GEOM);

-- Ajout de la colonne ORIENTATION qui précise l'orientation des tronçons.
ALTER TABLE ROUTES_GRAPHE ADD COLUMN ORIENTATION INT DEFAULT 0;
UPDATE ROUTES_GRAPHE SET ORIENTATION=1 WHERE SENS='Sens unique';
UPDATE ROUTES_GRAPHE SET ORIENTATION=-1 WHERE SENS='Sens inverse';

-- Ajout de la colonne TEMPS.
ALTER TABLE ROUTES_GRAPHE ADD COLUMN TEMPS DOUBLE;
UPDATE ROUTES_GRAPHE SET TEMPS=((LONGUEUR/1000)/VITESSE)*60;

CREATE SPATIAL INDEX ON ROUTES_GRAPHE(THE_GEOM);
-- Ajout d'un identifiant unique nécessaire au calcul du graphe
ALTER TABLE ROUTES_GRAPHE ADD COLUMN GID SERIAL;

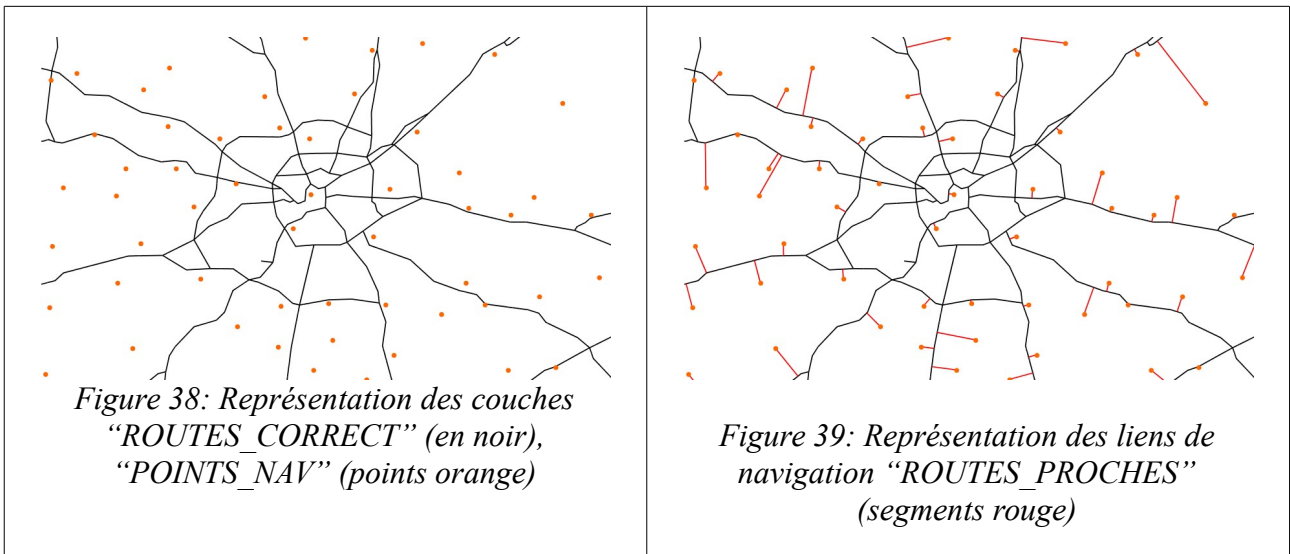
```

La table “ROUTES_PROCHE” présente trois champs (*figure 37 ci-dessous*) :

- NAV_GID : identifiant du point de navigation que l'on souhaite raccorder au tronçon,
- THE_GEOM : géométrie qui fait le lien entre le point de navigation et le tronçon le plus proche,
- ROUTES_GID : identifiant du tronçon le plus proche du point de navigation.

ROUTES_PROCHE - 0/9 481			
	NAV_GID	THE_GEOM	ROUTES_GID
1	1	POINT (147200 6824500)	26268
2	2	POINT (154900 6833400)	494
3	3	POINT (125200 6834600)	52601
4	4	POINT (135900 6856200)	19549
5	5	POINT (140200 6856300)	22622
6	6	POINT (136600 6836300)	20928
7	7	POINT (139500 6851900)	72561
8	8	POINT (137200 6824300)	17828

Figure 37: Extrait de la table "ROUTES_PROCHES"



A l'issue, nous obtenons un réseau routier complet (la table "ROUTES_GRAPHE" illustrée ci-dessous), comportant les liens géométriques entre les points de navigation et les tronçons initiaux.

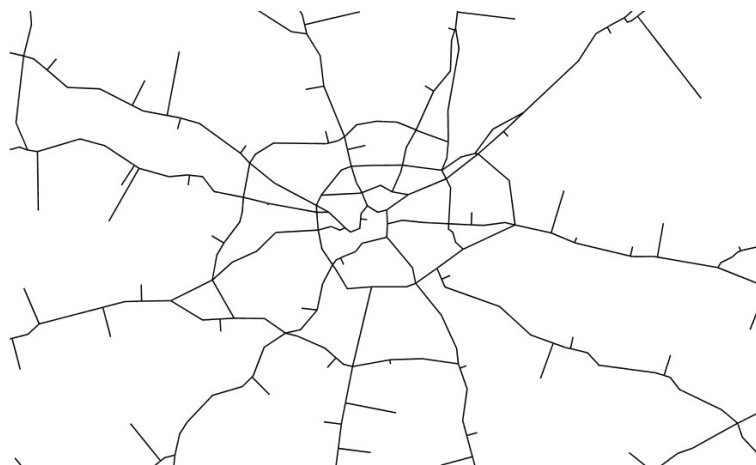


Figure 40: Extrait de la couche "ROUTES_GRAPHE"

5.2.7 Création du graphe routier

Les traitements précédemment ont permis de préparer le réseau routier ROUTE 120® : corrections géométriques, appariement de points, pondération. Nous procédons maintenant au calcul du graphe de données qui sera utilisé pour construire le distancier.

```
-- Création du graphe de données
DROP TABLE IF EXISTS ROUTES_GRAPHE_EDGES, ROUTES_GRAPHE_NODES,
ROUTES_GRAPHE_EDGES_PONDEREES;
SELECT ST_Graph('ROUTES_GRAPHE', 'THE_GEOM', 0.01);
CREATE INDEX ON ROUTES_GRAPHE_EDGES(EDGE_ID);

-- Affectation des attributs TEMPS et ORIENTATION nécessaires à l'orientation des
arcs du graphe et à leur pondération.
CREATE TABLE ROUTES_GRAPHE_EDGES_PONDEREES AS
SELECT a.THE_GEOM, b.*, a.ORIENTATION, a.TEMPS
FROM ROUTES_GRAPHE AS a, ROUTES_GRAPHE_EDGES AS b
WHERE a.GID=b.EDGE_ID;
CREATE SPATIAL INDEX ON ROUTES_GRAPHE_NODES(THE_GEOM);
```

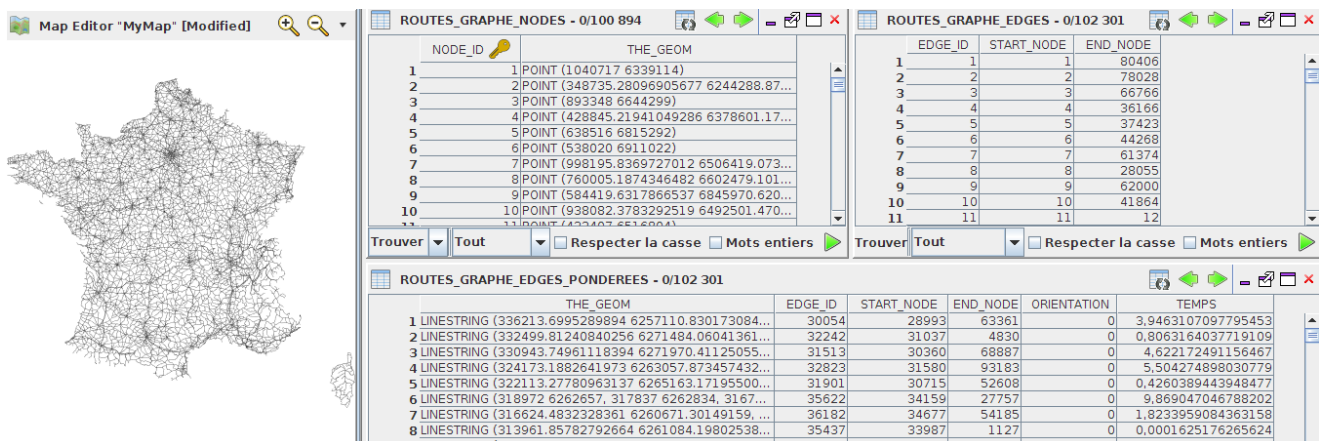


Figure 41: Visualisation de la couche “ROUTES_GRAPHE_EDGES_PONDEREES” (à gauche) et extraits des tables issues de la production du graphe (à droite)

5.2.8 Calcul du distancier

5.2.8.1 Création des destinations

Le calcul du distancier nécessite de construire une table contenant les identifiants des nœuds source et destination. Cette table (SOURCE_DEST) établie une correspondance entre les couples domicile - travail et les identifiants des nœuds du graphe produit à l'étape précédente (ROUTES_GRAPHE_NODES).

```

DROP TABLE IF EXISTS DESTINATIONS;
CREATE TABLE DESTINATIONS
  AS SELECT a.*, b.NODE_ID
  FROM POINTS_NAV a, ROUTES_GRAPHE_NODES b
  WHERE ST_EXPAND(a.THE_GEOM, 0.01, 0.01) && b.THE_GEOM;
CREATE INDEX ON DESTINATIONS (NODE_ID);
CREATE INDEX ON DESTINATIONS (INSEE_COM);

DROP TABLE IF EXISTS SOURCE_DEST;
CREATE TABLE SOURCE_DEST(SOURCE INT, INSEE_SOURCE CHAR(5), COMM_SOURCE CHAR(80),
DESTINATION INT, INSEE_DEST CHAR(5), COMM_DEST CHAR(80))
  AS SELECT a.NODE_ID SOURCE, b.CODGEO INSEE_SOURCE, b.LIBGEO COMM_SOURCE,
c.NODE_ID DESTINATION, b.DCLT INSEE_DEST, b.L_DCLT COMM_DEST
  FROM DESTINATIONS a, COUPLE_DT b, DESTINATIONS c
  WHERE a.INSEE_COM=b.CODGEO AND b.DCLT=c.INSEE_COM
  ORDER BY COMM_SOURCE, COMM_DEST;

CREATE INDEX ON SOURCE_DEST (INSEE_SOURCE);
CREATE INDEX ON SOURCE_DEST (INSEE_DEST);
DROP TABLE COUPLE_DT;
    
```

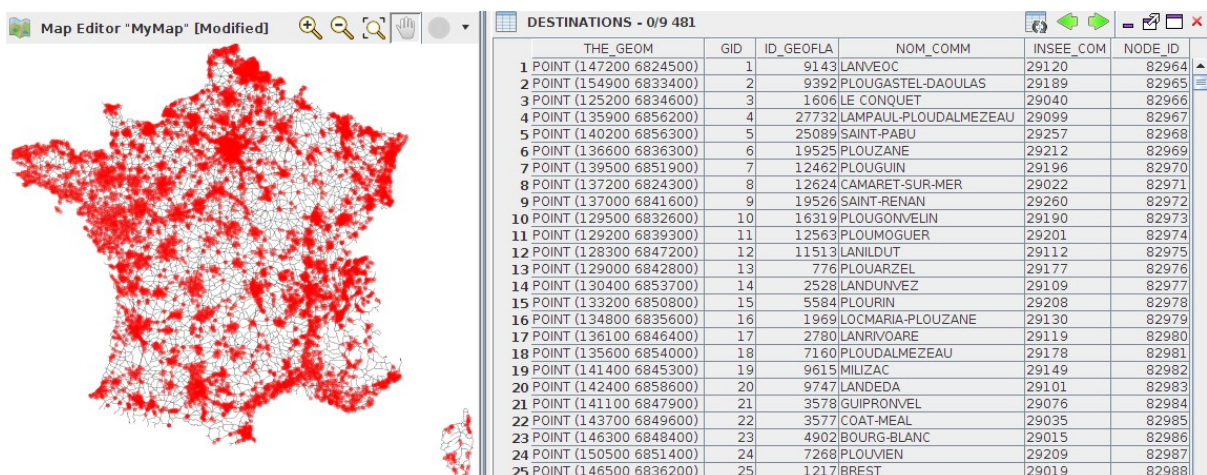


Figure 42: Visualisation de la table "DESTINATIONS" (points rouge)

La table résultante "SOURCE_DEST" présente 6 colonnes (figure 43) :

- SOURCE : l'identifiant du nœud du graphe qui servira de point de départ,
- INSEE_SOURCE : le code Insee de la commune de départ,
- COMM_SOURCE : le nom de la commune de départ,
- DESTINATION : l'identifiant du nœud du graphe qui servira de point d'arrivée,
- INSEE_DEST : le code Insee de la commune d'arrivée,
- COMM_DEST : le nom de la commune d'arrivée.

	SOURCE	INSEE_SOURCE	COMM_SOURCE	DESTINATION	INSEE_DEST	COMM_DEST
1	88357	80001	Abbeville	88027	80021	Amiens
2	88357	80001	Abbeville	88358	80318	Flixecourt
3	87813	60003	Abbeville-Saint-Lucien	88327	60057	Beauvais
4	89379	34001	Abeilhan	88744	34032	Béziers
5	86691	62001	Ablain-Saint-Nazaire	86794	62498	Lens
6	83469	78003	Ablis	83606	78517	Rambouillet
7	87706	14001	Ablon	87557	14333	Honfleur
8	83694	94001	Ablon-sur-Seine	83678	94054	Orly

Figure 43: Extrait de la table "SOURCE_DEST"

5.2.8.2 Création du distancier

Le calcul du distancier est réalisé avec la fonction ST_ShortestPathLength, décrite dans la section 4.2.2.4.

```
-- Calcule du distancier
DROP TABLE IF EXISTS PLUS_COURT_CHEMIN;
CREATE TABLE PLUS_COURT_CHEMIN
  AS SELECT *
  FROM ST_ShortestPathLength('ROUTES_GRAPHE_EDGES_PONDEREES', 'DIRECTED -
ORIENTATION', 'TEMPS', 'SOURCE_DEST');
CREATE INDEX ON PLUS_COURT_CHEMIN(SOURCE);
CREATE INDEX ON PLUS_COURT_CHEMIN(DESTINATION);

-- Reconstruction d'une table de synthèse des distances par couples
domicile-travail
-- La valeur de distance est arrondie à deux décimales
DROP TABLE IF EXISTS DISTANCIER;
CREATE TABLE DISTANCIER
  AS SELECT b.NOM_COMM AS COM_SOURCE, b.INSEE_COM AS INSEE_SOURCE, c.NOM_COMM AS
COM_DEST, c.INSEE_COM AS INSEE_DEST, ROUND(a.DISTANCE, 2) AS TEMPS_TRAJET
  FROM PLUS_COURT_CHEMIN a, DESTINATIONS b, DESTINATIONS c
  WHERE a.SOURCE=b.NODE_ID AND a.DESTINATION=c.NODE_ID
  ORDER BY INSEE_SOURCE, INSEE_DEST;
DROP TABLE PLUS_COURT_CHEMIN;
```

```

-- Export du distancier dans un fichier CSV avec un encodage de type utf-8
CALL CSVWrite('../Data_distancier/distancier_FR_r120_dom_travail.csv', 'SELECT *
FROM DISTANCIER', 'charSET=UTF-8');
DROP TABLE IF EXISTS SOURCE_DEST;

-- Nettoyage des tables qui ne servent plus
DROP TABLE IF EXISTS GRILLE, NEG_CUT, NEG_LIGNE, NEG_LIGNE_EXPL, NEG_SELECT,
ROUTES, ROUTES_CORRECT, ROUTES_AU, ROUTES_COUPE, ROUTES_COUPE_UNION, ROUTES_FINAL,
ROUTES_LIEN, ROUTES_LIENS_DECOUPEES, ROUTES_PROCHE, ROUTES_TEMP, ZONE_ETUDE_NEG;
    
```

Le résultat final est un tableau constitué de cinq champs :

- COM_SOURCE : nom de la commune de départ,
- INSEE_SOURCE : code Insee de la commune de départ,
- COM_DEST : nom de la commune d'arrivée,
- INSEE_DEST : code Insee de la commune d'arrivée,
- TEMPS_TRAJET : le temps de trajet associé.

DISTANCIER - 0/25 484					
	COM_SOURCE	INSEE_SOURCE	COM_DEST	INSEE_DEST	TEMPS_TRAJET
1	ABBEVILLE	80001	AMIENS	80021	51,33
2	ABBEVILLE	80001	FLIXECOURT	80318	26,65
3	ABBEVILLE-SAINT-LUCIEN	60003	BEAUVAIS	60057	21,76
4	ABEILHAN	34001	BEZIERS	34032	39,19
5	ABLAIN-SAINT-NAZAIRE	62001	LENS	62498	39,19
6	ABLIS	78003	RAMBOUILLET	78517	31,23
7	ABLON	14001	HONFLEUR	14333	16,99
8	ABLON-SUR-SEINE	91001	ORLY	91054	25,81

Figure 44: Extrait du distancier “Domicile-Travail” réalisé sur la France à partir de la base de données Route 120®

5.2.9 Analyse succincte des résultats

Le distancier contient 25 484⁵⁷ couples de flux domicile-travail. 9 089 communes sont référencées comme des lieux d'habitation (domicile) contre 3 218 identifiées comme des lieux de travail. Ces deux chiffres témoignent de l'attractivité de certains pôles géographiques qui concentrent les lieux d'emploi. Ils sont matérialisés avec la figure 45.

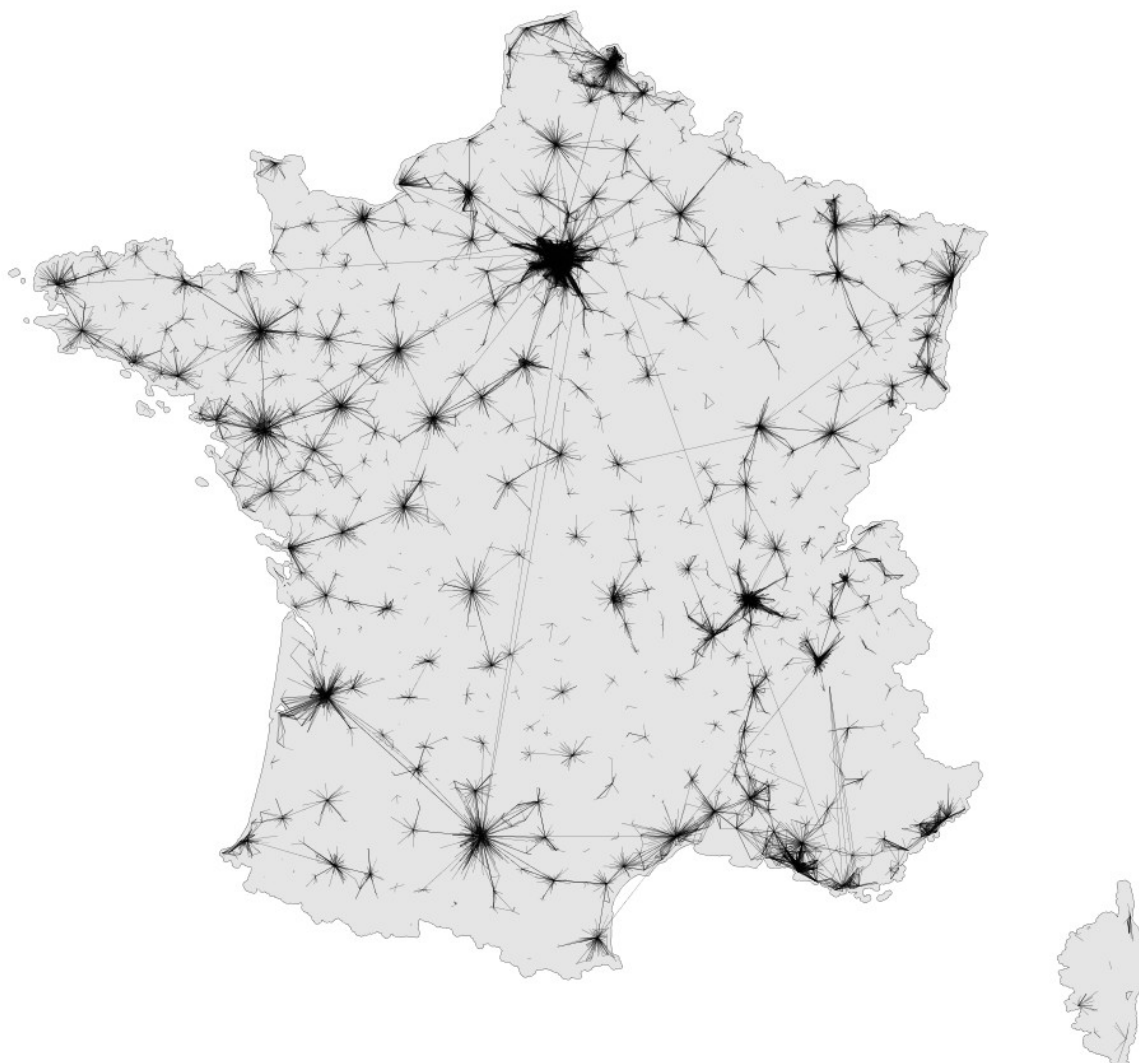


Figure 45: Représentation cartographique des couples "domicile-travail"

⁵⁷ Ce chiffre peut être comparé aux 26 202 entrées initiales (dans la table de l'INSEE). Cette différence peut s'expliquer par des erreurs au niveau des codes Insee dans la table d'entrée, qui génèrent des échecs lors de la jointure avec les données communales

La construction des liens géographiques domicile-travail est réalisée à l'aide du script ci-dessous.

```
-- Nous créons une ligne reliant les points de départ et d'arrivée (source et
destination) grâce à une jointure entre la table des points de navigation et le
distancier. Pour chacune des lignes, nous associons les codes INSEE et le temps de
trajet.

DROP TABLE IF EXISTS LIENS_DOM_TRAV_GEOM;
CREATE TABLE LIENS_DOM_TRAV_GEOM (GID SERIAL, THE_GEOM GEOMETRY, INSEE_SOURCE
CHAR(7), INSEE_DEST CHAR(7), TEMPS_TRAJET DOUBLE(17))
AS SELECT null, ST_MAKELINE(a.THE_GEOM, b.THE_GEOM) AS THE_GEOM,
c.INSEE_SOURCE, c.INSEE_DEST, c.TEMPS_TRAJET
FROM POINTS_NAV a, POINTS_NAV b, DISTANCIER c
WHERE a.INSEE_COM=c.INSEE_SOURCE AND b.INSEE_COM=c.INSEE_DEST;
```

L'analyse des distances montre que :

- l'ensemble des actifs travaillent en dehors de leur commune de résidence (99,8 %) pour les flux supérieurs à 100 actifs,
- le trajet moyen des salariés est de 28 minutes,
- seul 11 % des salariés sont à moins de 10 minutes de leur lieu de travail,
- 35 % des salariés sont à plus de 30 minutes de leur lieu de travail.

Le tableau 7 ci-dessous donne une analyse plus fine sur la répartition des différentes classes de temps de trajet.

Classe de temps (minutes)	Effectif	Pourcentage
[0 ; 5[541	2,12
[5 ; 10[2425	9,51
[10 ; 15[3524	13,83
[15 ; 20[3645	14,30
[20 ; 25[3377	13,25
[25 ; 30[2827	11,09
[30 ; 50[6451	25,31
[50 ; +∞[2666	10,46

Tableau 7 : Classification des temps de trajet "domicile-travail" en pourcentage d'effectifs de salariés

Notons également que le trajet le plus long dure 833 minutes, soit près de 14h. Il s'agit ici d'un trajet effectué entre Toulon et Compiègne.

Remarque :

Les résultats obtenus diffèrent de ceux proposés par des solutions comme GoogleMaps⁵⁸ (8h31 pour cet exemple, en prenant en compte les conditions de trafic lors du test (29/09/2014 à 15h)) (voir figure 46 ci-dessous), Bing Maps⁵⁹ (8h16) ou bien OSRM⁶⁰ (10h - basé sur OpenStreetMap).

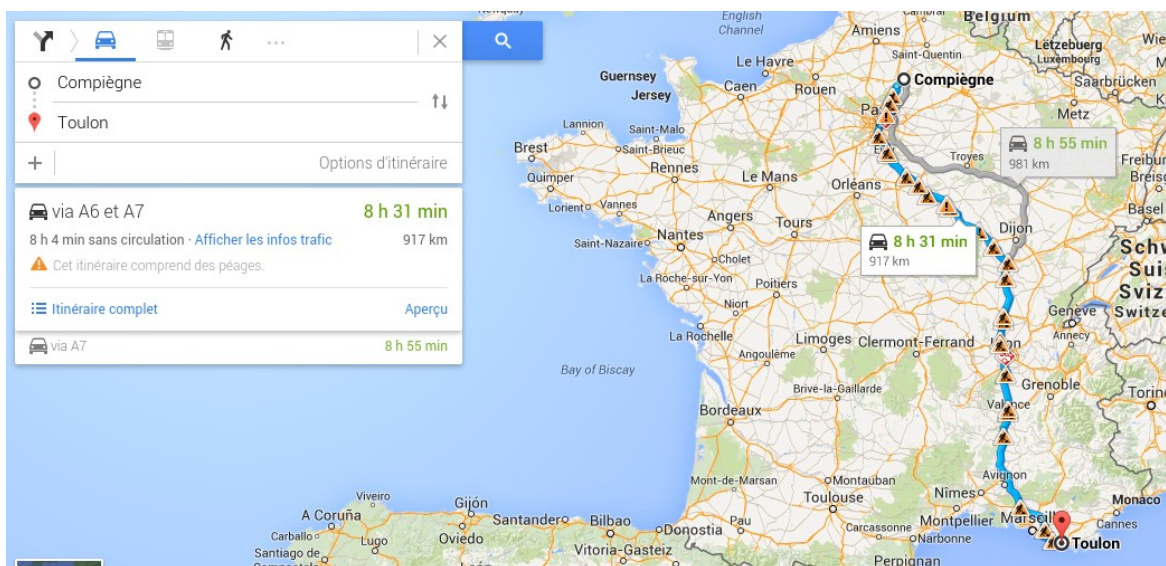


Figure 46: Illustration du trajet entre Toulon et Compiègne calculé avec GoogleMaps

Cette différence est due aux méthodes de pondérations des vitesses et à la richesse de description géométrique et sémantique du réseau routier. A titre d'exemple, dans notre cas, un tronçon autoroutier présent dans une aire urbaine sera pondéré avec la valeur de 70km/h (voir section « 5.1.3 Pondération des tronçons du réseau routiers ») alors que la vitesse légale est de 130km/h et qu'une vitesse moyenne probable est plus proche des 100-110km/h.

58 <http://tinyurl.com/kbhtvug> consulté en septembre 2014.
59 <http://tinyurl.com/l3applh> consulté en septembre 2014.
60 <http://map.project-osrm.org/> consulté en septembre 2014.

Conclusion et perspectives

Conclusion

Le rapport “*H2Network : un outil pour la modélisation et l'analyse de graphes dans le Système d'Information Géographique OrbisGIS*” propose un outillage et une méthodologie pour structurer et questionner des réseaux routiers.

Les fonctions de H2Network combinées avec le langage SQL spatial d'OrbisGIS offrent une plate-forme complète pour construire à façon des chaînes de traitements avancées : de la préparation des données à leur exploitation puis cartographie. Dans la partie 5, nous en avons présenté une application succincte. Néanmoins, ce travail a fait l'objet de plusieurs valorisations scientifiques et pédagogiques. Nous citerons en autre une participation au symposium OGRS'2014⁶¹ (*Open Source Geospatial Research & Education Symposium*) en Finlande. Cette participation s'est traduite par la présentation d'un papier scientifique (Gouge *et al*, 2014) et l'organisation d'un atelier d'une matinée où ingénieurs et chercheurs de la communauté géomatique ont manipulé les outils pour évaluer l'accessibilité aux écoles primaires sur l'agglomération nantaise (Petit *et al*, 2014).

Perspectives

La perspective la plus immédiate est l'amélioration des performances des temps de calcul. Parmi les pistes à étudier, il y a l'implémentation :

- d'une version bidirectionnelle de l'algorithme de Dijkstra qui permettrait de réduire le temps de parcours par deux.
- d'une version parallèle de l'algorithme de Dijkstra. Ceci nécessiterait par contre une copie du graphe par *thread* et donc beaucoup de mémoire vive.
- d'une approche hiérarchique. Cette technique consiste à introduire dans le graphe des arcs dits “raccourcis” et peut réduire de façon drastique le temps de calcul de plus courts chemins. On passe de quelques millisecondes à quelques microsecondes.

Une seconde perspective d'amélioration concerne le modèle de données. En effet, les fonctions H2Network chargent systématiquement en mémoire les indices des arcs et des nœuds dans une structure interne à JGraphT. Cette solution offre l'avantage de fournir des temps d'accès rapides aux objets du graphe, néanmoins elle s'avère pénalisante (1) pour de grands graphes (nécessite de la mémoire vive) (2) pour prendre en compte les changements sur le graphe (perturbation ponctuelles). En effet, avec cette approche, nous considérons que les données nécessaires au graphe sont connues à l'avance et qu'elles sont fiables dans le temps. Une piste pour s'affranchir de cette limitation serait d'exploiter les indexes disponibles dans la base de données H2 Database et son cartouche H2GIS afin de récupérer à la demande les informations stockées dans les tables qui représentent le graphe (EDGES et NODES dans notre cas). Si ce type de technique privilégie les accès disques au détriment de la mémoire, elle offre également la possibilité de traiter des données plus volumineuses et surtout de maintenir un lien constant avec un jeu de données qui peu évoluer.

61 <http://www.ogrs-community.org/> consulté en septembre 2014.

Ressources en ligne

L'ensemble de la documentation en ligne des fonctions de H2Network comprenant de nombreux exemples est disponible à l'adresse suivante :

- <http://www.h2gis.org/docs/dev/h2network/>

De la même manière, les fonctions SQL de H2GIS mentionnées au cours de ce rapport, sont documentées à l'adresse suivante :

- <http://www.h2gis.org/docs/dev/fonctions/>



Le code source de H2Network et de JNA sont disponibles aux adresses suivantes :

- <https://github.com/irstv/H2GIS/>
- <https://github.com/irstv/Java-Network-Analyzer>

Le logiciel OrbisGIS est disponible sur le site officiel <http://www.orbisgis.org/>. La version utilisée dans ce rapport est la n°5, nommée « Molène ».



Références

- Atmakuri-Davidsen and Padmavathamma, (2014) « *A fuzzy closeness centrality using andness-direction to control degree of closeness* », International Conference on Networks & Soft Computing (ICNSC), Vignan Foundation For Science, Technology & Research (Vignan University), India.
- Bocher E. & Petit G. (2012) « *OrbisGIS: Geographical Information System designed by and for research* », in Innovative Software Development in GIS, ISTE & Wiley, "Geographical Information System" series, Bucher B. & Le Ber F. (Editors), (p25-66), 331p.
- Brandes, Ulrik (2001) « *A Faster Algorithm for Betweenness Centrality.* » *Journal of Mathematical Sociology* 25 (2): 163–177.
- Di Salvo M. (2006) « [Calculs d'accessibilité – Impact des spécifications du réseau routier sur les calculs d'accessibilité](#) », Certu
- Dijkstra, E. W. (1959) « [A note on two problems in connexion with graphs](#) ». *Numerische Mathematik* 1: 269–271.
- Éditions CERTU (2010) « [La consommation d'espaces par l'urbanisation. Panorama des méthodes d'évaluation](#) ».
- Études et documents n°57 (2011) « [Indicateurs de développement durable pour les territoires](#) », Commissariat général au développement durable.
- Euler L. - « [Solutio problematis ad geometriam situs pertinentis](#) », *Commentarii academiae scientiarum Petropolitanae* 8, 1741, pages 128-140.
- Fisher D. (2004) « *The java universal network/graph framework (JUNG) : A brief tour* ». UC Irvine KDD Project.
- Fredman M. and Tarjan R. (1987) « *Fibonacci heaps and their uses in improved network optimization algorithms* ». *Journal of the ACM (JACM)*, 34 (3), 596–615.
- Freeman, Linton (1977) « *A set of measures of centrality based on betweenness* ». *Sociometry* 40: 35–41.
- Geisberger, Robert, et al. (2008) « *Contraction hierarchies: Faster and simpler hierarchical routing in road networks.* » *Experimental Algorithms*. Springer Berlin Heidelberg, 319-333.
- Gondran M. et Minoux M. (1979) « *Graphes et algorithmes* », Éditions Eyrolles, coll. « Études et recherches d'Électricité de France », (réimpr. 1986), 548 p., « 2 - Le problème du plus court chemin ».
- Gouge A., Bocher E., Fortin N. and Petit G., (2014) « [H2Network: A tool for understanding the influence of urban mobility plans \(UMP\) on spatial accessibility](#) », Proceedings of the 3rd Open Source Geospatial Research & Education Symposium.
- Hilal M. (2008) [Documentation Odomatrix](#),
- Magalon N. et Graziani P. (2000) « [Démarches de projets d'aménagements urbains, Le Grand Lyon : mise en oeuvre d'une politique d'aménagement d'espace public](#) ».
- Mermet E. and Ruas A. (2010) « [GeoGraphLab: a tool for exploring structural characteristics of transportation networks](#) ».
- Mermet E. et Gleyze J-F. (2010) « [Opérations et cohérence pour l'exploration des propriétés structurelles d'un réseau de transport](#) ».

- O'Madadhain J., Fisher D., Smyth P., White S. and Boey Y.-B. (2005) « *Analysis and Visualization of Network Data using JUNG* », Journal of Statistical Software, VV.
- Peter (2014) « [Fast and Memory-Efficient Road Routing with Graphhopper](#) »,
- Petit G., Gouge A., Fortin N., Bocher E. and Lecouvre M. (2014) « [Road analysis with H2Network](#) », Proceedings of the 3rd Open Source Geospatial Research & Education Symposium.
- Protsenko, N, (2010) « [Distance Problems in Networks - Theory and Praxis](#) »
- Sanders P. and Schultes D. (2005) « *Highway hierarchies hasten exact shortest path queries* ». Lecture Notes in Computer Science, 3669 , 568–579.

Glossaire

Bibliothèque : En informatique, une bibliothèque est un ensemble de fonctions utilitaires, regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire.

Clé primaire : Dans une base de données, contrainte que l'on applique à un champ afin de spécifier qu'aucune valeur dupliquée ne peut s'y trouver. En général, la clé primaire permet de définir des identifiants.

Multi(linestring) : Géométrie de type (multi)ligne.

Opensource : Se dit d'une application pour laquelle le code source est ouvert, la redistribution est libre et la création de dérivés autorisée.

List : Classe java, permettant de lister un élément avec conservation de l'ordre d'insertion.

Set : Classe java, permettant de lister un élément unique sans conservation de l'ordre d'insertion.

String : donnée de type chaîne de caractères.

Vecteur : type de donnée géographique, composé des points, lignes et polygones. On oppose souvent les vecteurs aux rasters (des images constituées de pixels).

Acronyme

SIG : Système d'Information Géographique

SQL : Structured Query Language

Index des figures

Figure 1: Positionnement de la bibliothèque H2Network dans l'architecture d'OrbisGIS.....	21
Figure 2: Environnement graphique d'OrbisGIS et console SQL.....	21
Figure 3: Implémentation des fonctions d'analyses de graphe.....	22
Figure 4: Principe de l'analyse de connectivité d'un graphe.....	26
Figure 5: Centralité de proximité des nœuds du réseau Route 120® de l'IGN en rouge le nœud avec la centralité égale à 1.....	40
Figure 6: Centralité d'intermédiarité des nœuds du réseau routier Route 120® de l'IGN en bleu le nœud avec la centralité égale à 1.....	41
Figure 7: "Moyenne" de la centralité d'intermédiarité pour les arcs du réseau routier Route 120®.....	42
Figure 8: Représentation des sous-graphes du réseau routier Route 120®.....	46
Figure 9: Exemples de croisements.....	47
Figure 10: Exemples de tronçons non-intersectants.....	48
Figure 11: Étapes pour le découpage des arcs se croisant.....	50
Figure 12: Correction des erreurs topologiques aux croisements.....	50
Figure 13: Cas possible d'erreur sur les sommets pendants.....	51
Figure 14: Distinction des cas possibles d'erreur sur les sommets pendants en fonction de la sémantique.....	51
Figure 15: Cartographie des nœuds pendants (points rouge) - Extrait sur le réseau routier ROUTE 120®.....	53
Figure 16: Mise en évidence des sommets pendants à corriger.....	54
Figure 17: Schématisation des différents cas de tronçons pendants corrigés.....	54
Figure 18: Découpage des tronçons routiers à partir des unités communales.....	59
Figure 19: Illustration du problème des tronçons littoraux.....	59
Figure 20: Méthode de découpage des tronçons en limite de zone d'étude.....	60
Figure 21: Raccordement d'un point de navigation à un point existant dans un réseau routier.....	62
Figure 22: Méthode pour l'identification du tronçon le plus proche d'un nœud de navigation.....	63
Figure 23: Calcul de la distance la plus courte entre un point et des lignes.....	64
Figure 24: Raccordement des points de navigation et découpage des tronçons routiers.....	65
Figure 25: Capture d'écran du dossier dans lequel sont enregistrées les données.....	67
Figure 26: Présentation de l'interface d'OrbisGIS et de la Console SQL.....	68
Figure 27: Représentation des couches "ZONE_ETUDE" (en bleu) et "ROUTES" (en noir).....	70
Figure 28: Extrait de la table "COUPLE_DT".....	71
Figure 29: Représentation cartographique (points orange) et attributaire de la table "POINTS_NAV".....	72
Figure 30: Extrait de la table "COMMUNE_AU".....	73
Figure 31: Extrait de la table "ROUTE_AU".....	73
Figure 32: Représentation d'une partie des couches "ZONE_ETUDE_NEG" (en gris) et "ROUTES_NEG" (en rouge).....	75
Figure 33: Extrait de la couche "ROUTES_FINAL".....	76
Figure 34: Illustration de la table "ROUTES_FINAL".....	76
Figure 35: Mise à jour de la colonne "VITESSE" dans la table "ROUTES_FINAL".....	77
Figure 36: Représentation cartographique et attributaire de la table "ROUTES_CORRECT", ainsi qu'un extrait de la table "ROUTES_DECOUP".....	78
Figure 37: Extrait de la table "ROUTES_PROCHES".....	80
Figure 38: Représentation des couches "ROUTES_CORRECT" (en noir), "POINTS_NAV" (points	

orange).....	80
Figure 39: Représentation des liens de navigation “ROUTES_PROCHES” (segments rouge).....	80
Figure 40: Extrait de la couche “ROUTES_GRAPHE”.....	80
Figure 41: Visualisation de la couche “ROUTES_GRAPHE_EDGES_PONDEREES” (à gauche) et extraits des tables issues de la production du graphe (à droite).....	81
Figure 42: Visualisation de la table “DESTINATIONS” (points rouge).....	82
Figure 43: Extrait de la table “SOURCE_DEST”.....	83
Figure 44: Extrait du distancier “Domicile-Travail” réalisé sur la France à partir de la base de données Route 120®.....	84
Figure 45: Représentation cartographique des couples "domicile-travail".....	85
Figure 46: Illustration du trajet entre Toulon et Compiègne calculé avec GoogleMaps.....	87