ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Ψηφιακών Συστημάτων

Π.Μ.Σ. «Πληροφοριακά Συστήματα και Υπηρεσίες»

Ειδίκευση: «Μεγάλα Δεδομένα και Αναλυτική»

Διπλωματική Εργασία

Ανάπτυξη Τελεστών Ερώτησης NoDA για Δεδομένα Οδικών Δικτύων με χρήση της Neo4J

Μαλούση Χάλγκερ | ΜΕ2114

Επιβλέπων: Χρήστος Δουλκερίδης

Πειραιάς, Φεβρουάριος 2024

Περιεχόμενα

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη τελεστών για το σύστημα NoDA, με έμφαση στην επίλυση πολύπλοκων ερωτημάτων που σχετίζονται με οδικά δίκτυα.

Το NoDA είναι ένα επίπεδο αφαίρεσης, το οποίο χρησιμοποιείται για την πρόσβαση σε NoSQL μηχανές αποθήκευσης με ένα ενιαίο τρόπο. Η κύρια εστίαση του NoDA είναι στην κλιμακωτή διαχείριση δεδομένων κινητικότητας (mobility data), ιδίως στους χώρο-χρονικούς τελεστές. Το NoDA είναι ιδανικό για προγραμματιστές και αναλυτές δεδομένων που θέλουν να εκτελούν ερωτήματα σε βάσεις δεδομένων NoSQL χωρίς να χρειάζεται να χρησιμοποιούν τους αντίστοιχους drivers. Μέσω μιας ευέλικτης διεπαφής, ο χρήστης μπορεί να εκτελεί απλά και πολύπλοκα ερωτήματα, βασιζόμενος στις ανάγκες του. Μέχρι στιγμής, το NoDA μπορεί να λειτουργήσει με MongoDB, HBase, Redis και Neo4j.

Ο κύριος στόχος της εργασίας είναι αρχικά η σχεδίαση του τρόπου αποθήκευσης του οδικού δικτύου και των αντικειμένων που βρίσκονται πάνω στο οδικό δίκτυο σε μια βάση δεδομένων τύπου γράφου (Neo4j) και ύστερα η δημιουργία τελεστών του NoDA, για την εκτέλεση ερωτημάτων πάνω σε αυτό το οδικό δίκτυο. Τα ερωτήματα αυτά θα αφορούν τον υπολογισμό της συντομότερης διαδρομής δύο σημείων στο οδικό δίκτυο και την εύρεση των αντικειμένων πάνω στο δίκτυο αυτό με βάση μια συγκεκριμένη απόσταση και ανάμεσα σε κάποιο συγκεκριμένο χρονικό διάστημα.

Λέξεις-κλειδιά: NoDA, Neo4j, δίκτυα δρόμων, shortest path, Graph Database, OpenStreetMap

Abstract

The present thesis focuses on developing operators for the NoDA system, with an emphasis on solving complex queries related to road networks.

NoDA is an abstraction layer used for accessing NoSQL storage engines in a unified manner. Its primary focus is on scalable management of mobility data, particularly spatio-temporal operators. NoDA is ideal for developers and data analysts who want to execute queries on NoSQL databases without needing to use the respective drivers. Through a flexible interface, users can perform both simple and complex queries, depending on their needs. So far, NoDA can operate with MongoDB, HBase, Redis, and Neo4j.

The main objective this thesis is initially the design of storing the road network and the objects located on the road network in a Graph Database (Neo4j), followed by the creation of NoDA operators for executing queries on this road network. These queries will involve calculating the shortest path between two points on the road network and finding objects on this network based on a specific distance and within a certain time frame.

Key-words: NoDA, Neo4j, road network, shortest path, Graph Database, OpenStreetMap

Ευχαριστίες

Σε αυτό το σημείο, θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας κ. Χρήστο Δουλκερίδη, Αναπληρωτή Καθηγητή στο Τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς, για την πολύτιμη βοήθειά του, τις συμβουλές του και τις παρατηρήσεις του κατά τη διάρκεια της εκπόνησης της διπλωματικής εργασίας.

Θα ήθελα επίσης να ευχαριστήσω και τον κ. Νικόλαο Κουτρουμάνη, διδακτορικό ερευνητή στο Τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς για την βοήθειά του στην υλοποίηση των εργαλείων που χρειάστηκαν για την ολοκλήρωση της διπλωματικής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που με στηρίξανε σε όλη τη διάρκεια των σπουδών μου.

1 Εισαγωγή

1.1 Σκοπός της διπλωματικής εργασίας

Σκοπός της διπλωματικής εργασίας, είναι να επεκτείνουμε τους operators του NoDA, έτσι ώστε να μπορούν να εφαρμοστούν χωρο-χρονικά ερωτήματα για αντικείμενα που βρίσκονται πάνω σε ένα δίκτυο δρόμων μιας πόλης, έτσι ώστε να μπορέσει ένας χρήστης μέσω του NoDA να τα καλέσει χωρίς δυσκολία. Στην συγκεκριμένη εργασία, οι τελεστές που θα δημιουργηθούν, θα είναι για την βάση δεδομένων Neo4j.

Αρχικά θα γίνει μια προεπεξεργασία για το πως θα αποθηκευτεί το οδικό δίκτυο μέσα σε μια μη σχεσιακή βάση δεδομένων και ύστερα θα γίνει μια προεπεξεργασία για το πως θα αποθηκευτούν τα αντικείμενα που βρίσκονται πάνω στο οδικό δίκτυο, έτσι ώστε να μπορούν να εφαρμοστούν τα ερωτήματα αυτά. Για να γίνει αυτό εφικτό, χρησιμοποιήθηκε ένα csv αρχείο με πραγματικά δεδομένα από στίγματα GPS αυτοκινήτων μέσα στην Ελλάδα. Το αρχείο αυτό θα περάσει από μια προεπερξεγασία για να περιοριστούν τα στίγματα μόνο στον χώρο της Αθήνας και αργότερα θα χρησιμοποιηθούν διάφορα scripts για να μπορέσει να γίνει η ανάθεση του δρόμου σε κάθε στίγμα, να βρεθεί δηλαδή σε ποιόν δρόμο βρισκόταν το αυτοκίνητο εκείνη την στιγμή που δημιουργήθηκε το στίγμα. Τα δεδομένα αυτά των στιγμάτων μαζί με το αναγνωριστικό του δρόμου , θα αποθηκευτούν σε μια μη σχεσιακή βάση δεδομένων MongoDB, ως νέο collection. Τα δεδομένα για το δίκτυο δρόμων θα ανακτηθούν από το OpenStreetMap, το οποίο προσφέρει την δυνατότητα δωρεάν πρόσβασης σε τρέχουσες γεωγραφικές πληροφορίες και θα αποθηκευτούν σε διαφορετικό collection της MongoDB, οπού με την βοήθεια scripts που υλοποιήθηκαν και σε συνδυασμό με διάφορα aggregations, θα προβούν σε μια προεπεξεργασία, έτσι ώστε να βοηθήσουν στην εύρεση του δρόμου για το κάθε στίγμα, όπως αναφέρθηκε πιο πάνω. Όταν δημιουργηθεί το collection με τα στίγματα, τότε αυτό γίνεται export σε μορφή json και είναι έτοιμο να χρησιμοποιηθεί αργότερα στην Neo4j.

Αφού λοιπόν έχει φτιαχτεί το αρχείο με τα στίγματα, επόμενο βήμα είναι η δημιουργία του οδικού δικτύου στην Neo4j. Για να γίνει αυτό εφικτό, δημιουργήθηκε ένα script στην python, οπού χρησιμοποιείται μια βιβλιοθήκη που με τις κατάλληλες παραμέτρους μπορεί να ανακτήσει από το OpenStreetMap δεδομένα και να τα μετατρέψει σε GeoDataframes. Ύστερα με διάφορες συναρτήσεις της βιβλιοθήκης, το δίκτυο δρόμων μπόρεσε να αποθηκευτεί στην Neo4j. Στο δίκτυο αυτό, οι κόμβοι αναπαριστούν διασταυρώσεις δρόμων ενώ οι ακμές τους πραγματικούς δρόμους.

Έχοντας αποθηκεύσει και το δίκτυο δρόμων, επόμενο βήμα ήταν να χρησιμοποιηθεί και το αρχείο σε μορφή .json με τα στίγματα, έτσι ώστε να αποθηκευτούν και αυτά στην Neo4j. Με την βοήθεια scripts που δημιουργήθηκαν στην javascript, τα

δεδομένα αυτά αποθηκεύτηκαν πάνω στο δίκτυο δρόμων, ως νέοι κόμβοι με διαφορετική ετικέτα και ενώνονται μέσω νέας ακμής με τον κοντινότερο κόμβο του οδικού δικτύου. Αυτή είναι και η τελική εικόνα της βάσης και πάνω σε αυτή τώρα μπορούν να υλοποιηθούν τα αντίστοιχα ερωτήματα. Τα ερωτήματα αυτά δημιουργήθηκαν με την βοήθεια των procedures, μια λειτουργία που προσφέρει το Neo4j και βοηθά στην επέκταση του, έτσι ώστε να μπορεί ένας χρήστης να εκτελεί πιο περίπλοκα ερωτήματα και τέλος δημιουργήθηκαν οι τελεστές του NoDA, οι οποίοι καλούνε τα procedures αυτά.

Τα ερωτήματα που υλοποιήθηκαν είναι:

- η εύρεση της γρηγορότερης διαδρομής μεταξύ δυο κόμβων του οδικού δικτύου: χρήση του αλγορίθμου Dijkstra
- η εύρεση των κοντινότερων αντικειμένων από έναν κόμβο ενός δικτύου μέσα σε μια συγκεκριμένη απόσταση και ένα συγκεκριμένο χρονικό διάστημα: αρχικά αναζήτηση των κόμβων του οδικού δικτύου που βρίσκονται μέσα στην συγκεκριμένη ακτίνα που έχει δώσει ο χρήστης και ύστερα την αναζήτηση των κόμβων των στιγμάτων που καλύπτουν και τις χρονικές προϋποθέσεις
- η εύρεση του κοντινότερου αντικειμένου από έναν κόμβο του οδικού δικτύου: αρχικά αναζήτηση των κόμβων του οδικού που βρίσκονται μέσα σε μια συγκεκριμένη απόσταση και ύστερα την αναζήτηση του κόμβου εκείνου που καλύπτει και τις χρονικές προϋποθέσεις

1.2 Διάρθρωση της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία θα αναπτυχθεί σε 7 συνολικά κεφάλαια. Και η δομή της εργασίας είναι η εξής:

- Στο Κεφάλαιο 2 παρατίθεται το θεωρητικό υπόβαθρο και κάποια εργαλεία που πρέπει να γνωρίζει κάποιος για να κατανοήσει την εργασία
- Στο Κεφάλαιο 3 αναφέρονται τα εργαλεία που χρησιμοποιήθηκαν για την επίλυση του προβλήματος και σύντομη περιγραφή των εργαλείων αυτών.
- Στο Κεφάλαιο 4 περιγράφεται όλη η διαδικασία της προεπεξεργασίας των δεδομένων για την εισαγωγή τους στην βάση δεδομένων.
- Στο Κεφάλαιο 5 περιγράφεται η δημιουργία των procedures της Neo4j για την δημιουργία των κατάλληλων ερωτημάτων
- Στο Κεφάλαιο 6 περιγράφεται η δημιουργία των τελεστών του συστήματος NoDA
- Στο Κεφάλαιο 7 γίνεται πειραματική μελέτη για την απόδοση των τελεστών

Ακόμη περιέχει και μετά τα 3 κύρια κεφάλαια ειδικές ενότητες οι οποίες είναι για τα:

- Συμπεράσματα όπου γίνεται μια τελική αναφορά στην εργασία και έπειτα χρήσεις της και εξέλιξης της.
- Ένας πίνακας από όλες τις εικόνες που έχουν δημιουργηθεί για να βοηθήσουν στην κατανόηση των διαφορετικών κομματιών που αναλύονται μέσα στην διπλωματική εργασία.
- Βιβλιογραφία όπου είναι επίσης ένας πίνακας με τις βιβλιογραφικές αναφορές και συνδέσμους για την διπλωματική εργασία.

2 Θεωρητικό υπόβαθρο

2.1 NoDA: Unified NoSQL Data Access Operators for Mobility Data

Το NoDA είναι ένα επίπεδο αφαίρεσης που αποτελείται από τελεστές πρόσβασης σε χωροχρονικά δεδομένα, το οποίο χρησιμοποιείται για την πρόσβαση σε NoSQL μηχανές αποθήκευσης με ένα ενιαίο τρόπο.

Το NoDA βρίσκεται ανάμεσα στον κώδικα της εφαρμογής και την αποθήκευση δεδομένων ως γέφυρα για την πρόσβαση στα δεδομένα και στοχεύει να "κρύψει" τη γλώσσα ερωτημάτων της υποκείμενης αποθήκευσης από τον προγραμματιστή. Βασικά, ένας προγραμματιστής μεγάλων δεδομένων μπορεί να εκφράζει τον κώδικά του χρησιμοποιώντας την αφαίρεση NoDA, καταφέρνοντας έτσι να κρύψει τις ιδιαιτερότητες και την πολυπλοκότητα της πρόσβασης στη συγκεκριμένη αποθήκευση NoSQL. Με αυτόν τον τρόπο, ο ακριβώς ίδιος κώδικας μπορεί να χρησιμοποιηθεί για το ερώτημα δεδομένων που αποθηκεύεται σε διάφορες αποθηκεύσεις NoSQL, ανεξάρτητα από το πόσο διαφορετικές είναι οι γλώσσες ερωτημάτων τους.

Ο στόχος του NoDA είναι να προσφέρει μια φιλική προς τον προγραμματιστή αφαίρεση, η οποία μπορεί να εκμεταλλευθεί για να παρέχει απλή και ενιαία πρόσβαση σε κλιμακούμενες αποθηκεύσεις NoSQL [1] [2].

2.2 Μη σχεσιακές βάσεις δεδομένων (NoSQL)

Μια βάση δεδομένων NoSQL, είναι ένας τύπος συστήματος διαχείρισης βάσεων δεδομένων που παρέχει ένα μηχανισμό για την αποθήκευση και ανάκτηση δεδομένων που είναι μοντελοποιημένα με τρόπους διαφορετικούς από τους παραδοσιακούς πίνακες βάσεων δεδομένων σχέσεων. Οι βάσεις δεδομένων NoSQL σχεδιάζονται για να χειρίζονται μεγάλα όγκοι μη δομημένων δεδομένων και χρησιμοποιούνται συχνά για μεγάλα δεδομένα και εφαρμογές πραγματικού χρόνου [3].

Τα βασικά χαρακτηριστικά των βάσεων δεδομένων NoSQL περιλαμβάνουν:

- Ευελιξία σχήματος: Οι βάσεις δεδομένων NoSQL επιτρέπουν ένα πιο ευέλικτο μοντέλο δεδομένων σε σύγκριση με τις παραδοσιακές σχέσεις βάσεις δεδομένων. Μπορούν να φιλοξενήσουν μια ποικιλία μορφών δεδομένων, συμπεριλαμβανομένων των ζευγαριών κλειδιού-τιμής, των εγγράφων, των στηλών-οικογένειας και των γραφημάτων.
- Κλιμάκωση: Οι βάσεις δεδομένων NoSQL σχεδιάζονται γενικά για να κλιμακωθούν οριζόντια, πράγμα που σημαίνει ότι μπορούν να χειριστούν αυξημένο φορτίο προσθέτοντας περισσότερους servers στη βάση δεδομένων.

- Επίδοση: Οι βάσεις δεδομένων NoSQL συχνά βελτιστοποιούνται για συγκεκριμένους τύπους ερωτημάτων και λειτουργιών, οδηγώντας σε βελτιωμένη απόδοση για ορισμένες χρήσεις σε σύγκριση με τις παραδοσιακές σχέσεις βάσεις δεδομένων.
- Κατανεμημένη αρχιτεκτονική: Πολλές βάσεις δεδομένων NoSQL είναι κατασκευασμένες με κατανεμημένη αρχιτεκτονική υπόψη, επιτρέποντας τη διανομή των δεδομένων σε πολλούς servers ή nodes.

Παραδείγματα μη σχεσιακών βάσεων δεδομένων είναι η MongoDB και η Neo4j, οι οποίες θα χρησιμοποιηθούν και στην συγκεκριμένη εργασία.

2.3 MongoDB

Η βάση MongoDB είναι ένα δημοφιλές σύστημα διαχείρισης βάσεων δεδομένων NoSQL που αποθηκεύει δεδομένα σε έγγραφα, παρόμοια με JSON, γνωστά ως BSON (Binary JSON). Είναι σχεδιασμένο για υψηλή διαθεσιμότητα, οριζόντια κλιμάκωση και ευκολία ανάπτυξης και επεκτασιμότητας. Η βάση MongoDB ταξινομείται ως βάση δεδομένων που εστιάζει στα έγγραφα, ανήκει στην οικογένεια των NoSQL και χρησιμοποιείται ευρέως σε σύγχρονες εφαρμογές web, κινητά εφαρμογές, συστήματα διαχείρισης περιεχομένου και άλλους τύπους λογισμικού.

2.4 Neo4j

Η βάση δεδομένων Neo4j είναι ένας τύπος βάσης δεδομένων που βασίζεται στη δομή των γράφων. Σε αντίθεση με τις παραδοσιακές σχεσιακές βάσεις δεδομένων, όπου η πληροφορία αποθηκεύεται σε πίνακες με σειρές και στήλες, σε μια βάση δεδομένων τύπου γράφου, η πληροφορία οργανώνεται σε κόμβους, σχέσεις και ιδιότητες.

Οι βάσεις δεδομένων τύπου γράφου χρησιμοποιούνται για να απεικονίσουν και να αναλύσουν δεδομένα που έχουν πολλαπλές σχέσεις μεταξύ τους, όπως κοινωνικά δίκτυα, συστήματα προτάσεων, δίκτυα οδών, και πολλά άλλα.

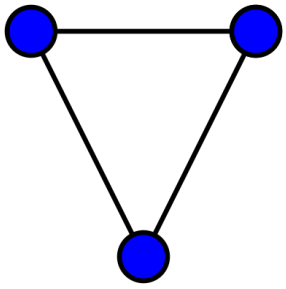
Η Neo4j είναι μια από τις πιο δημοφιλείς βάσεις δεδομένων τύπου γράφου και προσφέρει έναν πλούτο λειτουργιών για τη διαχείριση και την ανάλυση δεδομένων πάνω σε γράφο. Παρέχει έναν πανίσχυρο αλγόριθμο ερωτήσεων, διασυνδέσεις με πολλές γλώσσες προγραμματισμού και ένα ευέλικτο μοντέλο δεδομένων που επιτρέπει την εύκολη αναπαράσταση των σχέσεων μεταξύ διαφόρων τύπων κόμβων και σχέσεων.

Η Neo4j χρησιμοποίει την γλώσσα ερωτημάτων Cypher, που είναι σχεδιασμένη ειδικά για την έκφραση μοτίβων και σχέσεων μέσα στο μοντέλο δεδομένων γράφου. Η γλώσσα ερωτημάτων Cypher, επιτρέπει στους χρήστες να εκτελούν πολύπλοκες λειτουργίες στη βάση δεδομένων γράφου με σύντομη και ευανάγνωστη σύνταξη.

2.5 Γράφος

Στα μαθηματικά, και πιο συγκεκριμένα στη θεωρία γραφημάτων, ένα γράφημα είναι μια δομή που αντιστοιχεί σε ένα σύνολο αντικειμένων στα οποία κάποια ζεύγη αντικειμένων είναι κατά κάποιο τρόπο "συγγενή" [4]. Ένας γράφος, όπως απεικονίζεται στην Εικόνα 1, αποτελείται από δύο κύρια στοιχεία:

- Κορυφές (Κόμβοι): Κάθε κορυφή αντιπροσωπεύει ένα αντικείμενο ή ένα σημείο και μπορεί να επισημανθεί με ένα μοναδικό αναγνωριστικό, όπως έναν αριθμό ή ένα όνομα.
- Ακμές: Οι ακμές αποτελούν συνδέσεις ή συνδέσμους μεταξύ ζευγαριών κορυφών στο γράφημα. Αντιπροσωπεύουν τις σχέσεις ή τις αλληλεπιδράσεις μεταξύ των αντίστοιχων αντικειμένων. Μια ακμή μπορεί να είναι κατευθυνόμενη (με ένα βέλος που υποδηλώνει μια μονόδρομη σύνδεση) ή μη κατευθυνόμενη (χωρίς κατεύθυνση, αντιπροσωπεύοντας μια διπλή σύνδεση). Οι ακμές μπορεί επίσης να έχουν βάρη ή κόστη που σχετίζονται μαζί τους, τα οποία μπορούν να αντιπροσωπεύουν αποστάσεις, κόστη ή οποιαδήποτε άλλη σχετική πληροφορία.



ΕΙΚΟΝΑ 1 : ΕΝΑΣ ΑΠΛΟΣ ΓΡΑΦΟΣ ΜΕ ΤΡΕΙΣ ΚΟΡΥΦΕΣ ΚΑΙ ΤΡΕΙΣ ΑΚΜΕΣ

2.6 Αλγόριθμος Dijkstra

Ο αλγόριθμος Dijkstra, είναι ένας από τους πιο δημοφιλής αλγόριθμους για την εύρεση της συντομότερης διαδρομής(shortest path) [5], μεταξύ 2 κόμβων σε έναν κατευθυνόμενο γράφο με βάρη στις ακμές του.

Ο αλγόριθμος λειτουργεί διατηρώντας ένα σύνολο κόμβων για τους οποίους έχει καθορίσει τη συντομότερη γνωστή απόσταση από τον αρχικό κόμβο και ένα σύνολο κόμβων για τους οποίους η συντομότερη απόσταση δεν είναι ακόμη γνωστή. Εξερευνά επαναληπτικά κόμβους, επιλέγοντας πάντα τον κόμβο με τη μικρότερη γνωστή απόσταση από την αρχή και ενημερώνει τις αποστάσεις προς τους γείτονές του αν βρει ένα μονοπάτι με μικρότερη απόσταση. Ο αλγόριθμος συνεχίζει μέχρι να

καθορίσει το συντομότερο μονοπάτι προς όλους τους κόμβους ή μέχρι να φτάσει στον κόμβο προορισμού.

Ο αλγόριθμος του Dijkstra εξασφαλίζει ότι βρίσκει το συντομότερο μονοπάτι σε έναν γράφο με βάρη, υποθέτοντας ότι όλα τα βάρη των ακμών είναι μη αρνητικά Ωστόσο, ενδέχεται να μη λειτουργεί σωστά εάν ο γράφος περιέχει αρνητικά βάρη ακμών και στην περίπτωση αυτή θα πρέπει να χρησιμοποιηθούν άλλοι αλγόριθμοι για την εύρεση συντομότερης διαδρομής.

2.7 UTM

Το "UTM" (Universal Transverse Mercator) [6]είναι ένα παγκόσμιο σύστημα χαρτογράφησης που χρησιμοποιείται για να αναπαριστά την επιφάνεια της Γης σε χάρτες. Διαιρεί τη Γη σε μια σειρά ζωνών, κάθε μία από τις οποίες προβάλλεται ξεχωριστά για να μειώσει την παραμόρφωση. Το σύστημα UTM χρησιμοποιείται ευρέως για σκοπούς χαρτογράφησης και πλοήγησης.

Στο σύστημα UTM, η επιφάνεια της Γης διαιρείται σε ζώνες πλάτους 6 μοιρών, αριθμημένες από δυτικά προς ανατολικά. Κάθε μια από αυτές τις ζώνες διαιρείται περαιτέρω σε μικρότερα τετράγωνα πλέγματος. Οι προβολές UTM είναι συμβατικές, πράγμα που σημαίνει ότι οι γωνίες και οι σχήματα, αναπαρίστανται με ακρίβεια εντός κάθε ζώνης. Αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές όπως η πλοήγηση και η γεωμετρική αποτύπωση.

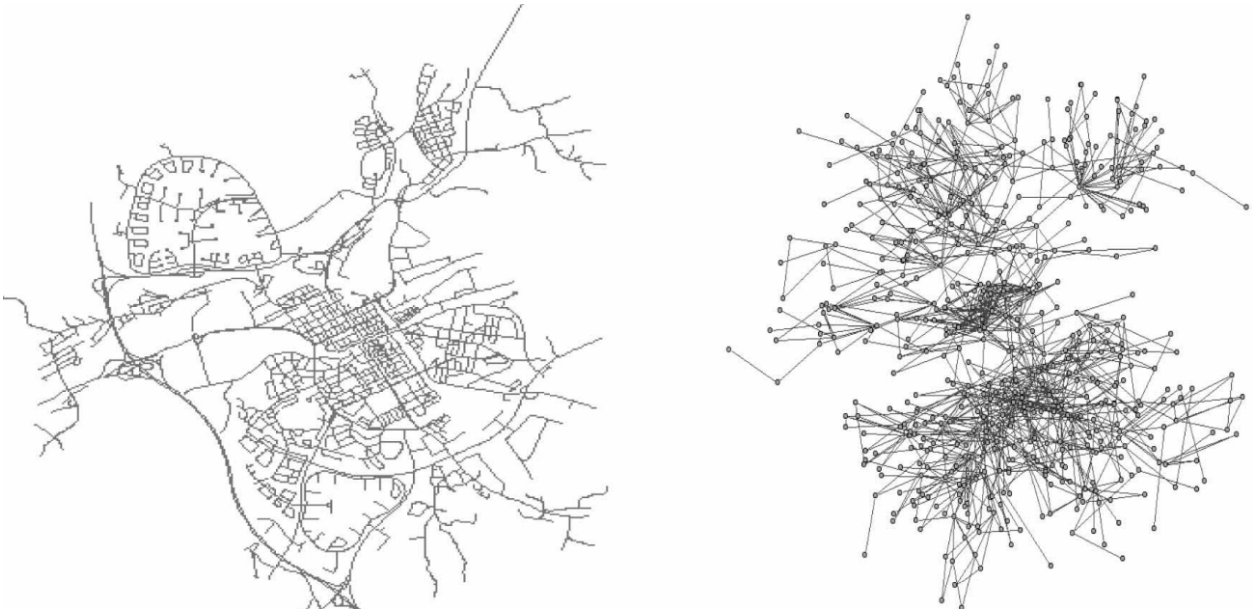
Οι συντεταγμένες UTM εκφράζονται συνήθως σε μέτρα και αποτελούνται από έναν αριθμό ζώνης, ένα γράμμα που υποδηλώνει το ημισφαίριο (N για το βόρειο ημισφαίριο και S για το νότιο ημισφαίριο), μια τιμή ανατολικής συντεταγμένης (μετρημένη σε μέτρα ανατολικά του κεντρικού μεσημβρινού της ζώνης) και μια τιμή βόρειας συντεταγμένης (μετρημένη σε μέτρα από την ισημερινή γραμμή). Αυτή η μορφή καθιστά πιο εύκολο τον χειρισμό αποστάσεων και κατευθύνσεων για διάφορους σκοπούς, όπως η πλοήγηση με GPS και η γεωγραφική χαρτογράφηση.

2.8 Οδικό δίκτυο

Την τελευταία δεκαετία, η τοπολογική ανάλυση έχει ευρέως υιοθετηθεί για τον εντοπισμό προτύπων ή δομών σε διάφορα συστήματα του πραγματικού κόσμου, συμπεριλαμβανομένων πληροφοριακών, κοινωνικών, τεχνολογικών και βιολογικών συστημάτων. Αυτό το νέο κύμα ενδιαφέροντος αναζωογονήθηκε κυρίως από την αυξανόμενη διαθεσιμότητα διαφόρων πραγματικών δεδομένων δικτύων του πραγματικού κόσμου, όπως το Διαδίκτυο και το World Wide Web.

Η εμφάνιση των συστημάτων πληροφοριών γεωγραφικής θέσης (GIS) έχει δημιουργήσει μια αυξανόμενη ποσότητα γεωχωρικών δεδομένων για τα αστικά

δίκτυα οδικών δρόμων (ενδεικτικά Εικόνα 2). Ένα αστικό δίκτυο οδικών δρόμων αντιλαμβάνεται συχνά ως ένα γράφημα όπου οι ακμές του αντιπροσωπεύουν τμήματα δρόμων και τα κόμβοι αντιπροσωπεύουν τις διασταυρώσεις (ή κόμβους) των τμημάτων [7].



ΕΙΚΟΝΑ 2 ΤΟ ΔΙΚΤΥΟ ΜΙΑΣ ΠΟΛΗΣ ΚΑΙ Ο ΑΝΤΙΣΤΟΙΧΟΣ ΓΡΑΦΟΣ ΤΗΣ [8]

Κάποιες από τις πιο βασικές λειτουργίες δικτύου περιλαμβάνουν υπολογιστικές διαδικασίες για την εύρεση της συντομότερης, λιγότερο κοστοβόρας ή πιο αποδοτικής διαδρομής (αναζήτηση διαδρομής) και επιπλέον και για την ανάλυση της συνδεσιμότητας του δικτύου (ανίχνευση) [8].

2.9 OpenStreetMap

Το OpenStreetMap (OSM) είναι μια δωρεάν, ανοικτή γεωγραφική βάση δεδομένων που ενημερώνεται και διατηρείται από μια κοινότητα εθελοντών μέσω ανοικτής συνεργασίας. Το OpenStreetMap προσφέρει την δυνατότητα δωρεάν πρόσβασης σε τρέχουσες γεωγραφικές πληροφορίες, όπου, στις ευρωπαϊκές χώρες, ακριβείς ψηφιακές γεωγραφικές πληροφορίες θεωρούνται ακριβές και εκτός της δυνατότητας ατόμων, μικρών επιχειρήσεων και κοινοτήτων [9]. Το OpenStreetMap χρησιμοποιεί τη δική του τοπολογία για να αποθηκεύει γεωγραφικά χαρακτηριστικά τα οποία μπορούν στη συνέχεια να εξαχθούν σε άλλες μορφές αρχείων GIS [10].

3 Εργαλεία που χρησιμοποιήθηκαν

3.1 MongoDB Compass

Το MongoDB Compass είναι ένα εργαλείο γραφικού περιβάλλοντος χρήστη (GUI) που αναπτύχθηκε από τη MongoDB Inc. Σχεδιάστηκε για να βοηθήσει τους προγραμματιστές και τους διαχειριστές βάσεων δεδομένων να εργάζονται με τις βάσεις δεδομένων MongoDB πιο εύκολα και αποτελεσματικά. Το MongoDB είναι ένα δημοφιλές σύστημα βάσης δεδομένων NoSQL, και το Compass παρέχει ένα φιλικό προς τον χρήστη τρόπο για την αλληλεπίδραση με τις βάσεις δεδομένων MongoDB χωρίς την ανάγκη χρήσης της γραμμής εντολών [11]. Το εργαλείο αυτό, χρησιμοποιήθηκε κατά τη διάρκεια της προεπεξεργασίας των δεδομένων.

3.2 Jupyter Notebook

Το Jupyter Notebook είναι μια open source διαδικτυακή εφαρμογή που επιτρέπει την δημιουργία εγγράφων, που περιλαμβάνουν runtime κώδικα, εξισώσεις, απεικονίσεις και αφηγηματικό κείμενο. Χρησιμοποιείται ευρέως στην επιστήμη των δεδομένων, την επιστημονική έρευνα και την εκπαίδευση για διαδραστικό υπολογισμό και ανάλυση δεδομένων. Το Jupyter Notebook αποτελεί μέρος του μεγαλύτερου Έργου Jupyter, το οποίο περιλαμβάνει επίσης το JupyterLab (ένα πιο σύγχρονο και ευέλικτο περιβάλλον), το JupyterHub (για πολλαπλά περιβάλλοντα χρήστη) και διάφορους πυρήνες (για διάφορες γλώσσες προγραμματισμού). Έχει γίνει ένα απαραίτητο εργαλείο στο εργαλειοθήκη των επιστημόνων δεδομένων, των ερευνητών, των εκπαιδευτών και οποιουδήποτε χρειάζεται ένα διαδραστικό και συνεργατικό περιβάλλον για τον υπολογισμό και την ανάλυση δεδομένων [12]. Το εργαλείο αυτό, χρησιμοποιήθηκε κατά τη διάρκεια της προεπεξεργασίας των δεδομένων.

3.3 Node.Js / Express

Το Node.js και το Express.js είναι δύο δημοφιλείς τεχνολογίες που χρησιμοποιούνται για τη δημιουργία εφαρμογών ιστού και εφαρμογών στην πλευρά του διακομιστή. Το Node.js παρέχει το περιβάλλον εκτέλεσης για την εκτέλεση κώδικα JavaScript στην πλευρά του διακομιστή, ενώ το Express.js χτίζει πάνω στο Node.js για να απλοποιήσει τη διαδικασία δημιουργίας εφαρμογών ιστού. Οι προγραμματιστές χρησιμοποιούν συχνά το Express.js για τον καθορισμό διαδρομών, την χειρισμό αιτημάτων HTTP και

τη διαχείριση μεσαίων, κάνοντας την κατασκευή αξιόπιστων και αποδοτικών εφαρμογών ιστού σε JavaScript πιο εύκολη. Τα δυο εργαλεία αυτά, χρησιμοποιήθηκαν για την δημιουργία μιας server side εφαρμογής, η οποία θα είναι απαραίτητη για την προεπεξεργασία των δεδομένων της διπλωματικής, αλλά και απαραίτητη για την παρουσίαση των αποτελεσμάτων της .

3.4 Flutter

Το Flutter είναι ένα ευέλικτο framework, που δημιουργήθηκε από την Google και μπορεί να χρησιμοποιηθεί για την δημιουργία μιας ποικιλίας εφαρμογών, συμπεριλαμβανομένων κινητών εφαρμογών, web εφαρμογών, εφαρμογών επιφάνειας εργασίας και ακόμη και ενσωματωμένων συστημάτων. Έχει αποκτήσει δημοτικότητα στην κοινότητα των προγραμματιστών λόγω της ευελιξίας του, της απόδοσής του και της ευκολίας χρήσης για την ανάπτυξη διασυνοριακών εφαρμογών [13]. Το Flutter χρησιμοποιήθηκε μαζί με το Node.Js / Express , για να δημιουργηθεί ένα front end application, όπου θα παρουσιάζονται τα αποτελέσματα των ερωτημάτων της διπλωματικής, πάνω σε ένα χάρτη του OpenStreetMap.

3.5 Neo4j Desktop

Το Neo4j Desktop είναι μια εφαρμογή λογισμικού που αναπτύχθηκε από τη Neo4j, Inc. Είναι ένα γραφικό περιβάλλον (GUI) σχεδιασμένο για την απλοποίηση της ανάπτυξης και διαχείρισης των βάσεων δεδομένων Neo4j [14]. Το Neo4j είναι ένα δημοφιλές σύστημα διαχείρισης βάσεων δεδομένων τύπου γράφου ανοικτού κώδικα που χρησιμοποιείται για την εργασία με συνδεδεμένα δεδομένα. Το Neo4j Desktop παρέχει ένα βολικό τρόπο για τους προγραμματιστές και τους διαχειριστές βάσεων δεδομένων να δημιουργούν, ρυθμίζουν και διαχειρίζονται βάσεις δεδομένων Neo4j. Το Neo4j είναι η βάση, η οποία θα χρησιμοποιηθεί σε αυτήν την διπλωματική, για την αποθήκευση του οδικού δικτύου και των αντικειμένων που βρίσκονται πάνω σε αυτό το οδικό δίκτυο.

3.6 OSMnx

Το OSMnx είναι ένα εργαλείο, που δημιουργήθηκε από ερευνητές στο πανεπιστήμιο της California και που καθιστά απλή, αυτοματοποιημένη και αξιόπιστη τη συλλογή δεδομένων και τη δημιουργία και ανάλυση δικτύων δρόμων από την άποψη της θεωρίας των γράφων, των μεταφορών και του αστικού σχεδιασμού.

Το OSMnx προσφέρει πέντε σημαντικές δυνατότητες για ερευνητές και επαγγελματίες:

- το αυτοματοποιημένο κατέβασμα των πολιτικών συνόρων και των αποτυπωμάτων κτιρίων
- το εξατομικευμένο και αυτοματοποιημένο κατέβασμα και κατασκευή δεδομένων δικτύου δρόμων από το OpenStreetMap
- αλγοριθμική διόρθωση της τοπολογίας του δικτύου
- τη δυνατότητα αποθήκευσης δικτύων δρόμων σε δίσκο ως αρχεία shapefiles, GraphML ή SVG
- τη δυνατότητα ανάλυσης των δικτύων δρόμων, συμπεριλαμβανομένου του υπολογισμού διαδρομών, της προβολής και απεικόνισης δικτύων και του υπολογισμού μετρικών και τοπολογικών μετρήσεων.

Αυτές οι μετρήσεις περιλαμβάνουν τις συνήθεις στον αστικό σχεδιασμό και τις μελέτες μεταφορών, καθώς και προηγμένες μετρήσεις της δομής και της τοπολογίας του δικτύου [15].

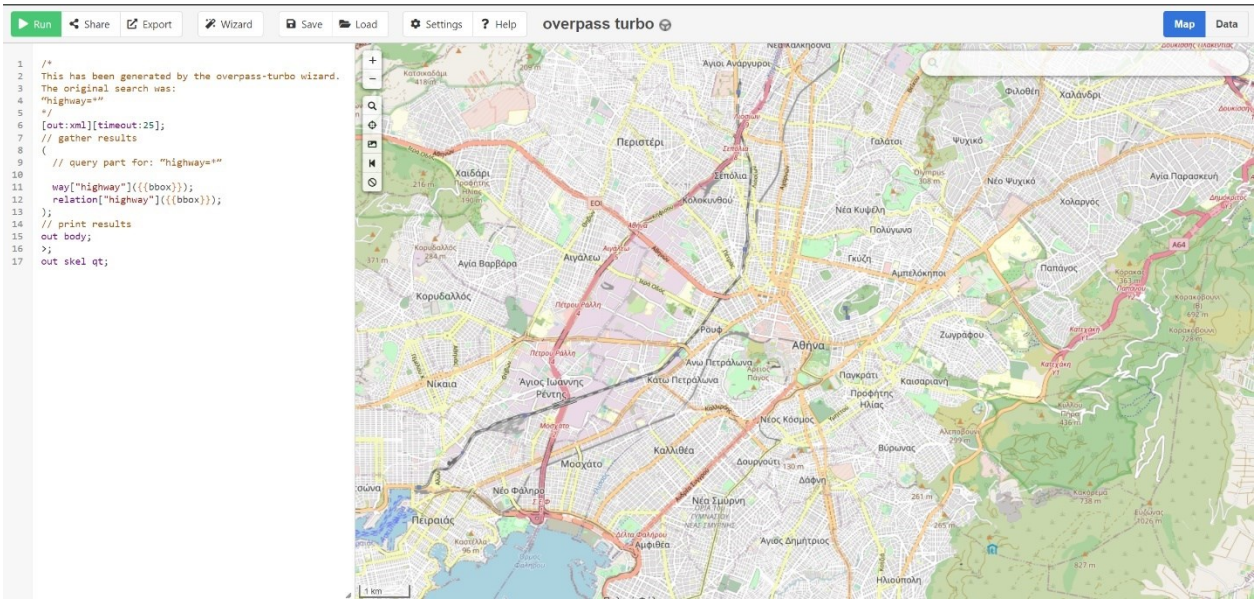
Το εργαλείο αυτό, χρησιμοποιήθηκε σε αυτήν την διπλωματική εργασία, για την ανάκτηση του οδικού δικτύου της Αθήνας και την προεπεξεργασία του οδικού αυτού δικτύου για την εισαγωγή του στην βάση δεδομένων.

3.7 Overpass API

Το Overpass API είναι ένα ισχυρό και ανοικτού κώδικα framework για την δημιουργία ερωτημάτων και την εξαγωγή δεδομένων από το OpenStreetMap (OSM).Το Overpass API επιτρέπει στους χρήστες να ανακτούν συγκεκριμένα γεωγραφικά δεδομένα από το OSM, όπως πληροφορίες σχετικά με δρόμους, κτίρια, πάρκα και άλλα, χρησιμοποιώντας μια γλώσσα ερωτήσεων που ονομάζεται Overpass QL. Συνήθως χρησιμοποιείται από προγραμματιστές και ερευνητές, που χρειάζονται πρόσβαση σε δεδομένα OSM με προγραμματιστικό τρόπο [16].

3.8 Overpass Turbo

Το Overpass Turbo είναι μια διαδικτυακή γραφική διεπαφή (GUI) που διευκολύνει τους χρήστες να δημιουργούν, δοκιμάζουν και εκτελούν ερωτήματα στο Overpass API χωρίς την ανάγκη προγραμματισμού ή γνώσεων για τη γραμμή εντολών. Παρέχει μια διεπαφή του χάρτη όπου οι χρήστες μπορούν να επιλέξουν οπτικά μια περιοχή και να καθορίσουν τον τύπο δεδομένων που θέλουν να ανακτήσουν από το OSM. Στη συνέχεια, το Overpass Turbo δημιουργεί το αντίστοιχο ερώτημα Overpass QL και το στέλνει στο Overpass API για να ανακτήσει τα ζητούμενα δεδομένα. Οι χρήστες μπορούν επίσης να οπτικοποιήσουν τα αποτελέσματα στον χάρτη, να εξάγουν δεδομένα και να μοιραστούν τα ερωτήματά τους [17]. Μία απεικόνιση του παρουσιάζεται στην Εικόνα 3.



ΕΙΚΟΝΑ 3 : OVERPASS TURBO GUI

4 Προεπεξεργασία Δεδομένων

Για να φτάσουμε στο σημείο να μπορούν να δημιουργηθούν οι operators του NoDA, που είναι και ο σκοπός αυτής της διπλωματικής, θα πρέπει πρώτα να γίνει μια προ επεξεργασία κάποιων δεδομένων και κάποιες παραδοχές, για την εικόνα που θα πρέπει να περιέχει η τελική βάση το οδικό δίκτυο και τα δεδομένα που βρίσκονται πάνω σε αυτό. Για αυτήν την διπλωματική, θα πειραματιστούμε με το οδικό δίκτυο της Αθήνας.

4.1 Καθαρισμός αρχείου csv

Για την εργασία αυτή, δόθηκε ένα .csv αρχείο, το οποίο έχει δημιουργηθεί από τα στίγματα οχημάτων τα οποία κινούνται στην Ελλάδα. Το αρχείο αυτό περιέχει τις εξής στήλες :

- Id: Το αναγνωριστικό id του οχήματος
- Timestamp: η χρονική στιγμή που δημιουργήθηκε το στίγμα
- Longitude: το γεωγραφικό μήκος του στίγματος
- Latitude: το γεωγραφικό πλάτος του στίγματος

Έχοντας λοιπόν αυτό το csv, θα έπρεπε να περιορίσουμε τα στίγματα, έτσι ώστε να παραμείνουν μόνο αυτά τα οποία βρίσκονται στο εύρος του οδικού δικτύου της Αθήνας και τα υπόλοιπα στίγματα να διαγραφούν.

Για τον λόγο αυτό, δημιουργήθηκε ένα jupyter notebook, οπού φορτώθηκε το csv αρχείο μέσα σε ένα pandas dataframe και ψάχνοντας τις στήλες με το γεωγραφικό πλάτος και γεωγραφικό μήκος του csv, δημιουργήθηκε ένα άλλο αρχείο csv, το οποίο περιείχε μέσα μόνο τα στίγματα τα οποία χρειάζονται για την συνέχεια της εργασίας.

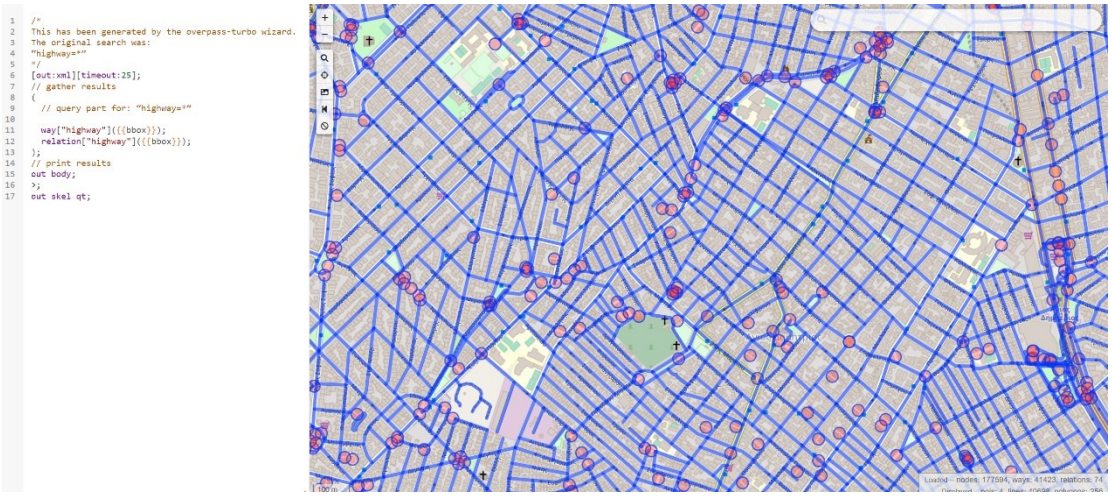
4.2 Ανάκτηση του οδικού δικτύου

Για να ανακτηθεί το οδικό δίκτυο της Αθήνας, χρησιμοποιήθηκε το εργαλείο Overpass turbo και το Overpass api (Εικόνα 4) . Εκτελώντας το παρακάτω Overpass QL query στο Overpass Turbo, μπορούμε να βρούμε όλα τα OSM objects, που έχουν το tag = “highway” και κατά συνέπεια όλο το οδικό δίκτυο του χάρτη που φαίνεται στην δεξιά πλευρά της οθόνης (Εικόνα 5).

```
/*
This has been generated by the overpass-turbo wizard.
The original search was:
“highway=*”
*/
[out:xml][timeout:25];
// gather results
(
  // query part for: “highway=*”

  way["highway"]({{bbox}});
  relation["highway"]({{bbox}});
);
// print results
out body;
>;
out skel qt;
```

ΕΙΚΟΝΑ 4: ΤΟ ΕΡΩΤΗΜΑ ΓΙΑ ΝΑ ΕΠΙΣΤΡΕΨΕΙ ΤΟ ΔΙΚΤΥΟ ΔΡΟΜΟΥ



ΕΙΚΟΝΑ 5 ΤΑ ΑΠΟΤΕΛΕΣΜΑΤΑ ΣΕ ΑΝΑΠΑΡΑΣΤΑΣΗ ΠΑΝΩ ΣΤΟΝ ΧΑΡΤΗ

Έχοντας λοιπόν ανακτήσει από την βάση δεδομένων του OpenStreetMap το οδικό δίκτυο που χρειαζόμαστε, κάνουμε export τα δεδομένα αυτά σε μορφή GeoJSON.

Το GeoJSON είναι ένα format για την κωδικοποίηση γεωγραφικών δομών δεδομένων και χρησιμοποιείται για την αναπαράσταση γεωγραφικών χαρακτηριστικών. Οι γεωμετρικοί τύποι που υποστηρίζει το GeoJSON είναι:

- Point: Αντιπροσωπεύει μια μοναδική γεωγραφική συντεταγμένη.
- LineString: Αντιπροσωπεύει μια σειρά συνδεδεμένων τμημάτων γραμμής.
- Polygon: Αντιπροσωπεύει ένα κλειστό σχήμα με έναν εξωτερικό κύκλο και προαιρετικούς εσωτερικούς κύκλους (τρύπες).
- MultiPoint: Αντιπροσωπεύει πολλά σημεία.

- MultiLineString: Αντιπροσωπεύει πολλές σειρές γραμμής.

Στο συγκεκριμένο GeoJSON, ο γεωγραφικός τύπος που υπάρχει είναι το linestring, αφού με αυτόν τον τρόπο αναπαρίσταται ένας δρόμος μέσα στο OpenStreetMap. Άρα μέσα σε αυτό υπάρχει ένας μεγάλος πίνακας από linestrings.

Αφού λοιπόν γίνει export σε GeoJSON αρχείο, επόμενο βήμα είναι να γίνει download και ύστερα να γίνει import , σαν ένα collection, σε μια βάση δεδομένων στην Mongo, μέσω του εργαλείου MongoDB Compass. Μετά την εισαγωγή αυτή του αρχείου αυτού, φαίνεται ότι κάθε δρόμος είναι ένα document μέσα στο collection (Εικόνα 6).

ΕΙΚΟΝΑ 6: ΤΟ DOCUMENT ΕΝΟΣ ΔΡΟΜΟΥ ΜΕΣΑ ΣΤΟ COLLECTION ΤΗΣ MONGODB

Όπως διακρίνεται στην παραπάνω εικόνα, το κάθε document έχει όλα τα στοιχεία που χρειάζονται για τον προσδιορισμό ενός δρόμου. Το μοναδικό πρόβλημα είναι ότι το πεδίο Linestring, αποτελείται από έναν πίνακα στιγμάτων και ιδανικά για το επόμενο βήμα της προ-επεξεργασίας θα πρέπει το κάθε document να σπάσει σε περισσότερα documents, τα οποία περιέχουν ζευγάρια των στιγμάτων αυτών. Με το τρόπο αυτό, πλέον η απεικόνιση του κάθε δρόμου θα γίνεται από περισσότερα documents, οπού πλέον το linestring πεδίο θα περιέχει μόνο δύο συνεχόμενα στίγματα του δρόμου.

Για να γίνει αυτό εφικτό, υλοποιήθηκε ένα script, το οποίο παίρνει όλους τους δρόμους που έχουν αποθηκευτεί στην βάση δεδομένων και δημιουργεί ένα καινούργιο document, το οποίο κρατάει μόνο τα απαραίτητα πεδία και στο πεδίο

linestring κρατάει 2άδες διαδοχικών στιγμάτων. Τα καινούργια αυτά documents που δημιουργούνται αποθηκεύονται σε ένα νέο collection στην mongodb, για να μην υπάρχει κάποια σύγχυση με το ήδη υπάρχων collection. Μετά το βήμα αυτό, το νέο document που δημιουργείται έχει την παρακάτω μορφή που απεικονίζεται στην Εικόνα 7.

ΕΙΚΟΝΑ 7: ΝΕΑ ΜΟΡΦΗ ΤΟΥ ΑΝΤΙΣΤΟΙΧΟΥ ΔΡΟΜΟΥ

Όπως φαίνεται λοιπόν, κρατάμε όλα τα στοιχεία του δρόμου που μας ενδιαφέρουν και πλέον το Linestring περιέχει μόνο 2 σημεία. Ο κάθε δρόμος πλέον αποτελείται από περισσότερα documents σε σχέση με το προηγούμενο βήμα και έτσι μπορούμε τώρα να πάμε στο επόμενο βήμα, που είναι η εύρεση των δρόμων, πάνω στον οποίον ανήκουν τα οχήματα από το csv αρχείο.

4.3 Ανάθεση δρόμου στα οχήματα και διόρθωση των θέσεων τους

Αφού λοιπόν έχει δημιουργηθεί το collection με τους δρόμους στην κατάλληλη μορφή και το csv με τα στίγματα των οχημάτων, μόνο για το εύρος των Αθηνών, μπορούμε πλέον να προχωρήσουμε στο επόμενο βήμα, το οποίο είναι αρχικά να βρεθεί σε ποιόν δρόμο ανήκουν τα οχήματα και στη συνέχεια να διορθωθεί η θέση τους. Όπως είναι γνωστό, το στίγμα που προέρχεται από το GPS, έχει ένα μικρό σφάλμα [18], με αποτέλεσμα πολλές φορές τα στίγματα να βρίσκονται εκτός δρόμου και για αυτό τον λόγο πρέπει να γίνει το βήμα της διόρθωσης.

Για να υλοποιηθεί αυτό το βήμα, φτιάχτηκε ένα script στην javascript, το οποίο περιέχει τις παρακάτω βοηθητικές συναρτήσεις.

- **getNearestRoadFromCsv**: Είναι η βασική συνάρτηση η οποία παίρνει σαν παράμετρο το csv αρχείο με τα στίγματα των οχημάτων και ορίζει τον δρόμο στον οποίο ανήκει το όχημα. Επιπλέον διορθώνει και την θέση του οχήματος.

- **calcNearestPointOnLine**: Συνάρτηση η οποία παίρνει ως παραμέτρους τα άκρα ενός ευθυγράμμου τμήματος σε έναν δισδιάστατο χώρο και ένα σημείο στον δισδιάστατο χώρο και επιστρέφει τις συντεταγμένες (x,y) της ορθής προβολής του σημείου αυτού πάνω στο ευθύγραμμο τμήμα.

- **calcDistancePointToLine**: Συνάρτηση που παίρνει ως παραμέτρους τα άκρα ενός ευθυγράμμου τμήματος σε έναν δισδιάστατο χώρο και τις συντεταγμένες ενός σημείου στον δισδιάστατο χώρο και επιστρέφει την απόσταση του σημείου αυτού από το ευθύγραμμο τμήμα.

- **calcIsInsideLineSegment**: Συνάρτηση που παίρνει ως παραμέτρους τα άκρα ενός ευθυγράμμου τμήματος σε ένα δισδιάστατο χώρο και τις συντεταγμένες ενός σημείου στον χώρο αυτό και επιστρέφει αν η προβολή του σημείου αυτού βρίσκεται εντός ή εκτός του ευθυγράμμου τμήματος.

- **convertLatLngToUtm:** Συνάρτηση από την βιβλιοθήκη “utm-latlng” του npm, που παίρνει σαν παράμετρο το γεωγραφικό πλάτος και γεωγραφικό μήκος ενός σημείου και επιστρέφει ένα UTM object, με UTM συντεταγμένες.

Αρχικά ο αλγόριθμος παίρνει σαν παράμετρο το csv αρχείο με τα στίγματα των οχημάτων και διαβάζει γραμμή-γραμμή το κάθε στοιχείο του. Για κάθε γραμμή του csv εκτελείται το παρακάτω aggregation στην mongo DB (Εικόνα 8):

ΕΙΚΟΝΑ 8: AGGREGATION ΓΙΑ ΤΗΝ ΑΝΑΚΤΗΣΗ ΚΟΝΤΙΝΟΤΕΡΟΥ ΔΡΟΜΟΥ

Το aggregation αυτό γίνεται στο collection που περιέχει τους δρόμους με μόνο δύο στοιχεία μέσα στο πεδίο του linestring. Το aggregation αυτό επιστρέφει τον δρόμο που βρίσκεται σε απόσταση 50m ή λιγότερο από το στίγμα που διαβάστηκε από την γραμμή του csv, δηλαδή τον κοντινότερο δρόμο από το σημείο του οχήματος. Με αυτόν τον τρόπο, έχει βρεθεί ο δρόμος, πάνω στον οποίο βρίσκεται το στίγμα του οχήματος.

Αφού λοιπόν έχει βρεθεί και ο δρόμος, επόμενο βήμα είναι να μετατραπούν τα σημεία του linestring και το στίγμα του οχήματος, από γεωγραφικές συντεταγμένες(latitude longitude) σε UTM συντεταγμένες. Για αυτό τον λόγο καλείται η συνάρτηση convertLatLngToUtm για καθένα από τα διαφορετικά σημεία. Με αυτόν τον τρόπο, μπορούν να χρησιμοποιηθούν οι συναρτήσεις που αναφέρθηκαν παραπάνω και εφαρμόζονται για σημεία σε δυσδιάστατο χώρο.

Αρχικά, καλείται η calcIsInsideLineSegment με παραμέτρους τα δύο άκρα του δρόμου και το στίγμα του οχήματος, για να βρεθεί αν η προβολή του οχήματος πέφτει πάνω στο κομμάτι του δρόμου ή εκτός. Αν πέφτει πάνω στο κομμάτι του δρόμου, τότε καλούνται οι συναρτήσεις calcDistancePointToLine και calcNearestPointOnLine με τις ίδιες παραμέτρους, για να υπολογιστεί η απόσταση του στίγματος από τον δρόμο και να βρεθεί η προβολή του σημείου αυτού πάνω στον δρόμο. Αν δεν πέφτει πάνω στο κομμάτι του δρόμου, τότε το κοντινότερο σημείο του στίγματος πάνω στον δρόμο, θα είναι ένα από τα κοντινότερα άκρα του δρόμου

Ύστερα, μετατρέπεται το νέο στίγμα σε γεωγραφικές συντεταγμένες (lat,lon) και αποθηκεύεται σε μια μεταβλητή.

Αφού τελειώσει όλος ο υπολογισμός, τότε δημιουργείται ένα αντικείμενο της κλάσης Vehicle, που περιέχει τις εξής μεταβλητές:

- latitude: Το νέο γεωγραφικό πλάτος του οχήματος.
- longitude: Το νέο γεωγραφικό μήκος του οχήματος.
- timestamp: Ο χρόνος που δημιουργήθηκε το στίγμα.
- vehicle_id: Το αναγνωριστικό του οχήματος.
- osmid: Το αναγνωριστικό του δρόμου πάνω στο οποίο βρίσκεται το όχημα.

Κάθε φορά που δημιουργείται ένα τέτοιο αντικείμενο, γίνεται insert μέσα σε ένα collection της mongodb.

Τέλος, αφού τελειώσει όλο το iteration στο csv και έχει γεμίσει το νέο collection της mongo με τα οχήματα, τότε γίνεται export όλο το collection, το οποίο θα χρησιμοποιηθεί πιο μετά για να εισαχθούν τα οχήματα στην νέα βάση δεδομένων της Neo4j και να ενωθούν με το οδικό δίκτυο.

4.4 Εισαγωγή οδικού δικτύου στην Neo4j

Για να ανακτηθεί το οδικό δίκτυο της Αθήνας στο Neo4j, θα χρησιμοποιηθεί το εργαλείο OSMnx.

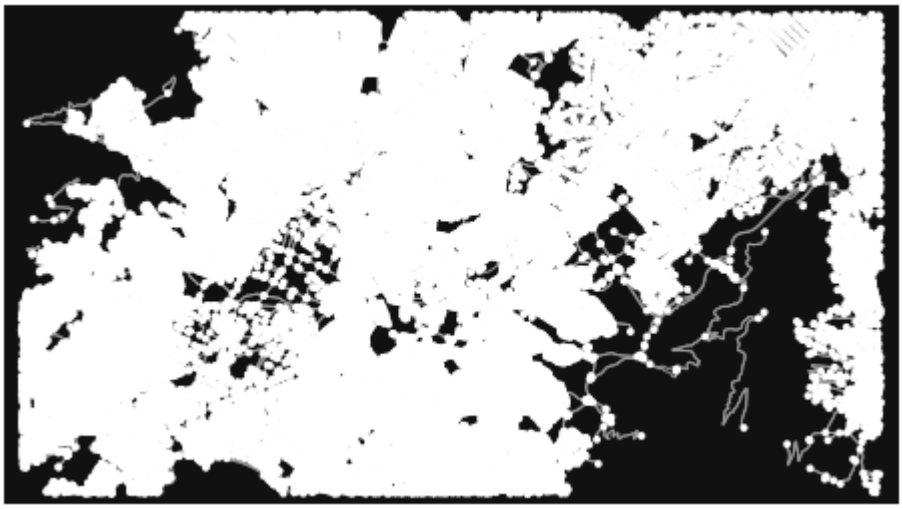
Αρχικά δημιουργείται ένα jupyter αρχείο, οπού γίνονται import οι βιβλιοθήκες Neo4j και osmnx για την γλώσσα python. Ύστερα γίνεται η σύνδεση στην βάση δεδομένων Neo4j που είναι εγκατεστημένη στον υπολογιστή με τα απαραίτητα credentials (Εικόνα 9).

```
NEO4J_URI = "bolt://localhost:7687"
NEO4J_USER = "neo4j"
NEO4J_PASSWORD = "12345678"


driver = neo4j.GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWORD))
```

ΕΙΚΟΝΑ 9 ΔΗΜΙΟΥΡΓΙΑ DRIVER NEO4J ΓΙΑ ΤΗΝ ΕΚΤΕΛΕΣΗ ΕΡΩΤΗΜΑΤΩΝ ΑΠΟ ΤΟ JUPYTER

Αφού δημιουργηθεί ο driver, τότε καλείται η συνάρτηση του osmnx, ox.graph_from_bbox(north, south, east, west, network_type='drive'), η οποία παίρνει σαν παραμέτρους το βορειότερο και νοτιότερο γεωγραφικό πλάτος, το ανατολικότερο και δυτικότερο γεωγραφικό μήκος και τον τύπο του χάρτη και αρχίζει και «κατεβάζει» τον χάρτη και τον αποθηκεύει σε έναν γράφο του osmnx. Με την συνάρτηση plot_graph(G1), γίνεται plot ο γράφος (Εικόνα 10).



ΕΙΚΟΝΑ 10: Ο ΓΡΑΦΟΣ ΤΟΥ ΟΔΙΚΟΥ ΔΙΚΤΥΟΥ ΤΗΣ ΑΘΗΝΑΣ

Ύστερα καλείται η συνάρτηση graph_to_gdfs(G1), η οποία παίρνει σαν παράμετρο τον παραπάνω γράφο και δημιουργεί δύο διαφορετικά GeoDataframes, οπού στο ένα αποθηκεύει τις διασταυρώσεις των δρόμων (Εικόνα 11), οι οποίοι είναι οι κόμβοι του γράφου και στο άλλο αποθηκεύει τους δρόμους σαν ακμές των κόμβων του γράφου (Εικόνα 12).

	osmid	y	x	street_count	ref	highway	geometry
0	242134	37.975138	23.725447	3	NaN	NaN	POINT (23.72545 37.97514)
1	242135	37.975604	23.725719	1	NaN	NaN	POINT (23.72572 37.97560)
2	242136	37.975814	23.724716	1	NaN	NaN	POINT (23.72472 37.97581)
3	242137	37.974902	23.726495	3	NaN	NaN	POINT (23.72650 37.97490)
4	31179466	37.987172	23.726682	4	NaN	NaN	POINT (23.72668 37.98717)

ΕΙΚΟΝΑ 11: GEODATAFRAME ΤΩΝ ΚΟΜΒΩΝ

	u	v	key	osmid	oneway	name	highway	reversed	length	geometry	width	lanes	ref	maxspeed	bridge	junction	tunnel	access	est_width
0	242134	727429546	0	23216541	True	Άρεως	living_street	False	11.989	LINESTRING (23.72545 37.97514, 23.72538 37.97504)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	242134	242135	0	193421328	True	Άρεως	living_street	False	57.011	LINESTRING (23.72545 37.97514, 23.72572 37.97560)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	242136	727429513	0	[58702688, 1175120417]	True	Βρυσακίου	living_street	False	84.740	LINESTRING (23.72472 37.97581, 23.72462 37.975...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	242137	251136130	0	847548593	True	Πανός	living_street	False	21.147	LINESTRING (23.72650 37.97490, 23.72640 37.97473)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	242137	242134	0	8181838	True	Δεξίππου	living_street	False	95.570	LINESTRING (23.72650 37.97490, 23.72597 37.975...	4.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

ΕΙΚΟΝΑ 12: GEODATAFRAME ΤΩΝ ΑΚΜΩΝ ΤΟΥ ΓΡΑΦΟΥ

Μετά την δημιουργία των δυο GeoDataframe, χρησιμοποιείται ο driver που φτιάχτηκε στην αρχή για να εκτελέσει το παρακάτω ερώτημα (Εικόνα 13).

```
UNWIND $rows AS row
WITH row WHERE row.osmid IS NOT NULL
MERGE (i:Intersection {osmid: row.osmid})
    SET i.location =
     point({latitude: row.y, longitude: row.x }),
        i.ref = row.ref,
        i.highway = row.highway,
        i.street_count = toInteger(row.street_count)
RETURN COUNT(*) as total
```

ΕΙΚΟΝΑ 13: ΕΙΣΑΓΩΓΗ ΚΟΜΒΩΝ ΣΤΗΝ ΒΑΣΗ

Με το ερώτημα αυτό, διαβάζεται η κάθε γραμμή του GeoDataframe των κόμβων και δημιουργείται ένας νέος κόμβος στην βάση δεδομένων Neo4j. Πριν την εισαγωγή των κόμβων, δημιουργείται ένα constraint για το πεδίο osmid και ένα index για το πεδίο location, έτσι ώστε να γίνονται πιο αποδοτικά τα ερωτήματα.

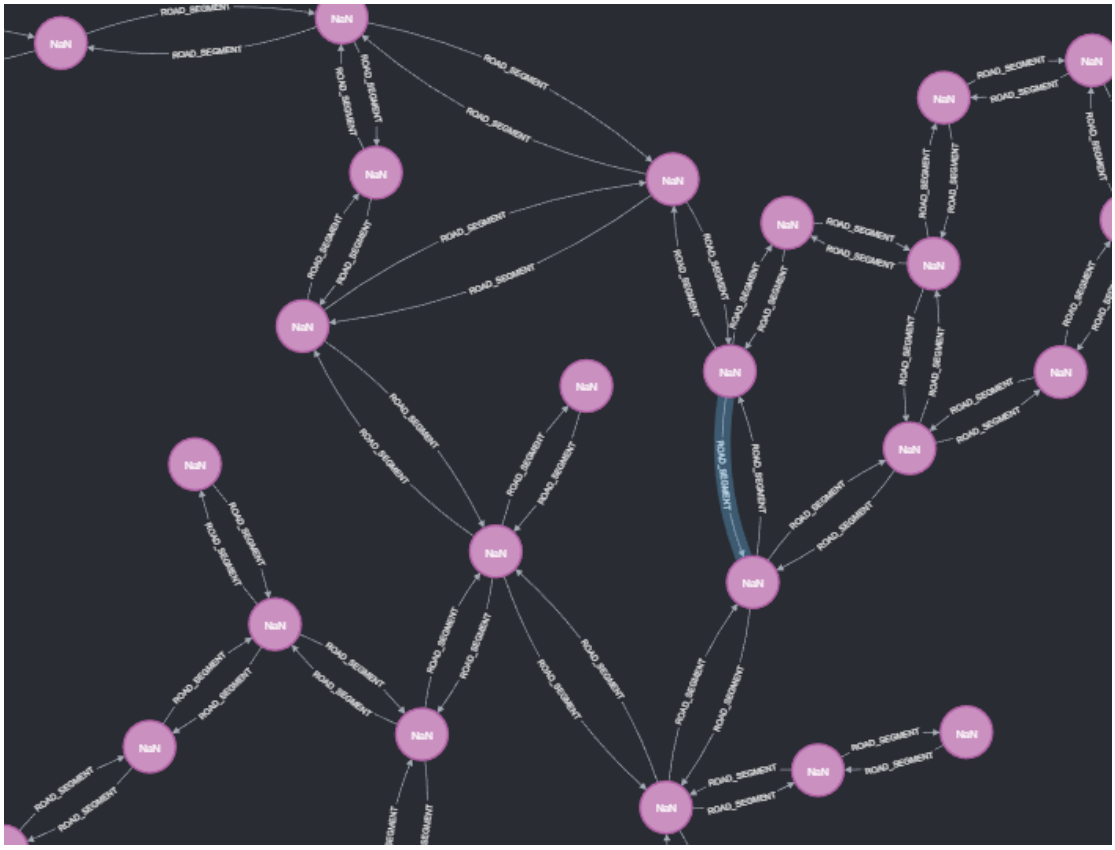
Αφού τελειώσει η εισαγωγή των κόμβων, τότε καλείται το παρακάτω ερώτημα για την δημιουργία των ακμών (Εικόνα 14):

```
UNWIND $rows AS road
MATCH (u:Intersection {osmid: road.u})
MATCH (v:Intersection {osmid: road.v})
MERGE (u)-[r:ROAD_SEGMENT {osmid: road.osmid}]->(v)
    SET r.oneway = road.oneway,
        r.lanes = road.lanes,
        r.ref = road.ref,
        r.name = road.name,
        r.highway = road.highway,
        r.max_speed = road.maxspeed,
        r.length = toFloat(road.length)
RETURN COUNT(*) AS total
```

ΕΙΚΟΝΑ 14: ΕΙΣΑΓΩΓΗ ΑΚΜΩΝ ΣΤΗΝ ΒΑΣΗ

Με τον αντίστοιχο τρόπο όπως και με το Geodataframe των κόμβων, δημιουργείται μια νέα ακμή, βρίσκοντας μέσα στην βάση τους κατάλληλους κόμβους να ενώσει. Δημιουργείται επιπλέον και ένα index πάνω στο πεδίο osmid της ακμής.

Μετά και από την εισαγωγή των ακμών στην βάση, το σχήμα του οδικού δικτύου της Αθήνας είναι ως εξής (Εικόνα 15):



ΕΙΚΟΝΑ 15: ΔΕΙΓΜΑ ΑΠΟ ΤΟ ΟΔΙΚΟ ΔΙΚΤΥΟ ΤΗΣ ΑΘΗΝΑΣ ΜΕΣΑ ΣΤΗΝ NEO4J

Ο κάθε κόμβος περιέχει τα πεδία:

- Id: το id του κόμβου στην βάση.
- Osmid: το id του κόμβου (από το OpenStreetMap).
- Street_count: με πόσες ακμές είναι συνδεδεμένος ο κόμβος.
- Location: το γεωγραφικό μήκος και πλάτος του κόμβου.

Και η κάθε ακμή περιέχει τα πεδία :

- Id: το id της ακμής στην βάση.
- Osmid: το id του δρόμου (από το OpenStreetMap).
- Highway: τύπος δρόμου (από το OpenStreetMap).
- Max_speed: μέγιστη ταχύτητα (από το OpenStreetMap).
- Name: το όνομα της οδούς (από το OpenStreetMap).
- Length: το μήκος της ακμής που δημιουργήθηκε όταν ενώθηκε στην βάση με τους δύο κόμβους.

Ο κάθε κόμβος που αντιστοιχεί στο οδικό δίκτυο έχει label: Intersection ενώ η κάθε ακμή που ενώνει δύο κόμβου του οδικού δικτύου έχει Label : ROAD_SEGMENT.

4.5 Εισαγωγή των οχημάτων στην βάση δεδομένων Neo4j

Για την εισαγωγή των οχημάτων στην βάση δεδομένων Neo4j, χρησιμοποιήθηκε το collection που έγινε export στην ενότητα 4.3, όπου η κάθε γραμμή του collection αντιστοιχεί σε ένα στίγμα ενός οχήματος. Με την βοήθεια του APOC [19] plugin του Neo4j, χρησιμοποιήθηκε το procedure apoc.iterate, αρχικά για να διαβαστεί το json αρχείο με τα στίγματα και ύστερα για την δημιουργία ενός νέου κόμβου , που περιέχει τα μεταδεδομένα του στίγματος ενός οχήματος. Με αυτόν τον τρόπο λοιπόν, δημιουργήθηκαν στην ίδια βάση δεδομένων του Neo4j οπού είναι αποθηκευμένοι οι κόμβοι του οδικού δικτύου, νέοι κόμβοι με τα δεδομένα των οχημάτων αλλά με Label: Vehicle, έτσι ώστε να μπορούν να ξεχωρίζονται από τους κόμβους του οδικού δικτύου, οι οποίοι έχουν Label: Intersection.

Στο επόμενο βήμα, θα γίνει η χρήση μιας βοηθητικής συνάρτησης που φτιάχτηκε για να μπορέσει να γίνει η ανάθεση των στιγμάτων των οχημάτων στο κοντινότερο κόμβο. Αρχικά βρίσκει τα ids των κόμβων των οχημάτων και ύστερα τα αποθηκεύει σε μια λίστα. Ύστερα γίνεται μια προσπέλαση της λίστας και για κάθε ξεχωριστό id, τρέχει το παρακάτω ερώτημα (Εικόνα 16):

```
MATCH (i:Intersection)-[r:ROAD_SEGMENT]-() USING INDEX i:Intersection(location)
WHERE( point.distance(i.location, c.location) < 1000 and( ANY(item IN r.osmid WHERE item = c.osmid)
or r.osmid = c.osmid))

with i,c
ORDER BY point.distance(c.location, i.location) ASC
LIMIT 1
MERGE (c)-[m:NEAREST_INTERSECTION]->(i)
SET m.length = point.distance(c.location, i.location)
return count(m) `
```

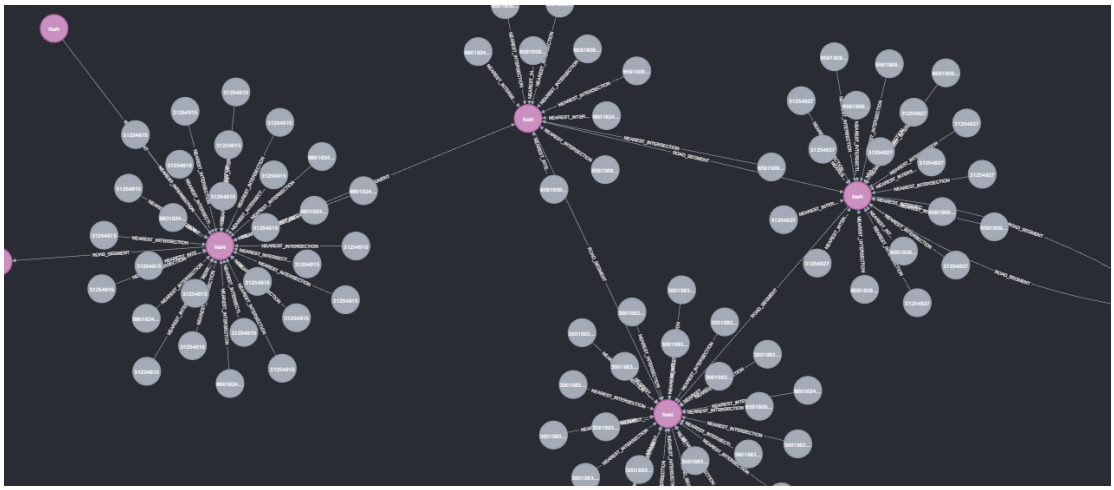
ΕΙΚΟΝΑ 16: ΕΡΩΤΗΜΑ ΑΝΑΘΕΣΗΣ ΠΛΗΣΙΕΣΤΕΡΟΥ ΚΟΜΒΟΥ ΟΔΙΚΟΥ ΔΙΚΤΥΟΥ ΣΤΟΥΣ ΚΟΜΒΟΥΣ ΟΧΗΜΑΤΩΝ

Αυτό που συμβαίνει στο ερώτημα, είναι να βρεθεί ο κόμβος του οδικού δικτύου, πού βρίσκεται σε απόσταση μέχρι και ένα χιλιόμετρο από το στίγμα του οχήματος και η ακμή του κόμβου να έχει το ίδιο osmid με το id του στίγματος του οχήματος και ύστερα να συνδέεται με μια νέα ακμή, η οποία έχει label: NEAREST_INTERSECTION, ο κόμβος του στίγματος με τον πλησιέστερο κόμβο του οδικού δικτύου και να υπολογίζεται και η μεταξύ τους απόσταση. Σε μεταγενέστερο βήμα, το id του κόμβου του οδικού δικτύου, αποθηκεύεται σαν πεδίο και στον κόμβο του στίγματος του αυτοκινήτου, για να διευκολύνει τα χωροχρονικά ερωτήματα. Έτσι λοιπόν, μετά από όλη αυτήν την διαδικασία, ο κόμβος των οχημάτων έχει τα εξής πεδία:

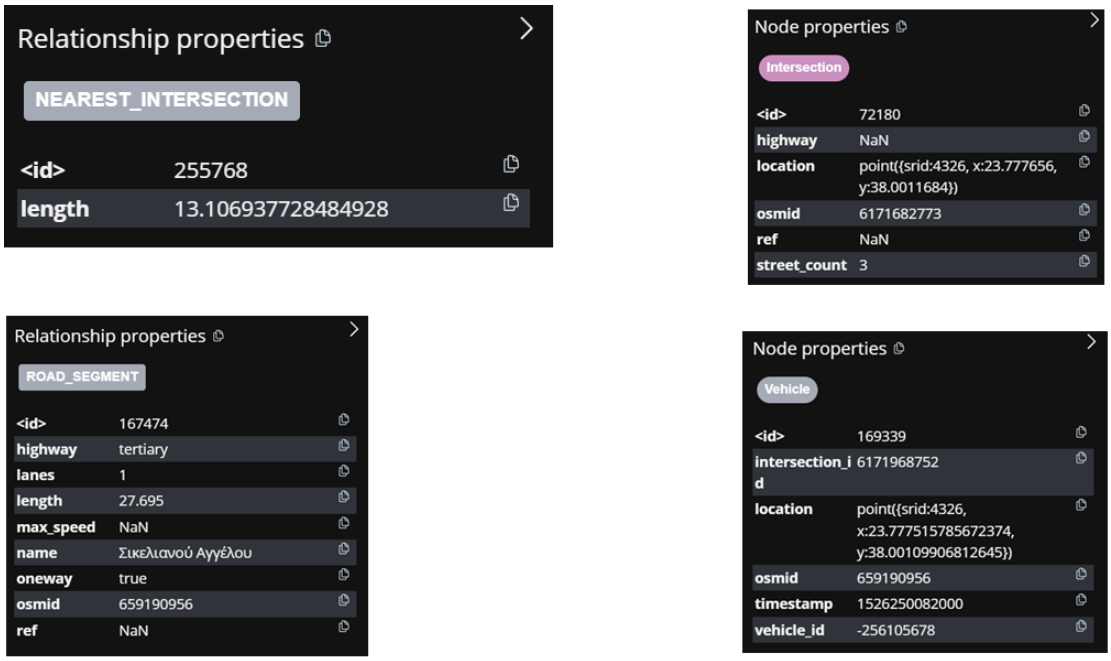
- Id: Το id του κόμβου του οχήματος.
- Intersection_id: Το id του πλησιέστερου κόμβου του οδικού δικτύου.
- Location: Το γεωγραφικό πλάτος και γεωγραφικό μήκος του στίγματος.
- Osmid: Το id του δρόμου (από OpenStreetMap).

- Timestamp: Ο χρόνος που δημιουργήθηκε το στίγμα.
- Vehicle_id: Το id του οχήματος.

Αυτό είναι και το τελικό βήμα της προ επεξεργασίας των δεδομένων και έτσι τώρα μπορούν να δημιουργηθούν τα κατάλληλα χώρο-χρονικά ερωτήματα. Στις παρακάτω εικόνες (Εικόνα 17, Εικόνα 18) φαίνονται το τελικό σχήμα που έχει η βάση δεδομένων στο Neo4j.



ΕΙΚΟΝΑ 17: ΤΕΛΙΚΟ ΣΧΗΜΑ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ



ΕΙΚΟΝΑ 18: ΤΕΛΙΚΗ ΜΟΡΦΗ ΤΩΝ ΚΟΜΒΩΝ ΚΑΙ ΤΩΝ ΑΚΜΩΝ ΣΤΗΝ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

5 Δημιουργία χώρο-χρονικών ερωτημάτων

5.1 Procedures της Neo4j

Για την δημιουργία των ερωτημάτων, έπρεπε να χρησιμοποιηθεί ένας μηχανισμός που προσφέρει το Neo4j, τα procedures [20]. Στο Neo4j, τα procedures αναφέρονται σε ορισμένες λειτουργίες ή εργασίες που έχουν καθοριστεί από τον χρήστη και μπορούν να εκτελεστούν εντός της βάσης δεδομένων Neo4j. Τα procedures γράφονται συνήθως σε Java ή άλλη υποστηριζόμενη γλώσσα, και επιτρέπουν την επέκταση της λειτουργικότητας του Neo4j με την προσθήκη προσαρμοσμένης λογικής ή λειτουργιών που ενδεχομένως να μην είναι διαθέσιμες στην πυρήνα γλώσσα ερωτήματος Cypher. Τα procedures μπορούν να κληθούν από ερωτήματα Cypher και είναι χρήσιμες για την εκτέλεση πολύπλοκων υπολογισμών, την ολοκλήρωση με εξωτερικά συστήματα ή την υλοποίηση εξειδικευμένων αλγορίθμων, όπως και οι αλγόριθμοι που έχουν δημιουργηθεί για τα χώρο-χρονικά ερωτήματα. Για να μπορέσει να χρησιμοποιηθεί το procedure στο Neo4j, θα πρέπει να ακολουθηθεί η εξής διαδικασία:

- Αρχικά θα πρέπει να γίνει η υλοποίηση του procedure σε μια γλώσσα προγραμματισμού που υποστηρίζεται από το Neo4j. Η Java είναι η πιο κοινή γλώσσα για αυτόν το σκοπό, αλλά μπορούν επίσης να χρησιμοποιηθούν και άλλες γλώσσες όπως το Scala ή το Kotlin.
- Ύστερα θα πρέπει να γίνει το compilation του κώδικά σε ένα αρχείο Java Archive (JAR).
- Το επόμενο βήμα είναι να μεταφερθεί το .jar αρχείο αυτό, μέσα στον φάκελο plugins του Neo4j. Αυτό καθιστά το procedure διαθέσιμο για χρήση μέσα στη βάση δεδομένων.
- Αφού λοιπόν το procedure είναι καταχωρημένο στην βάση, μπορεί να γίνει η κλήση του από το Cypher, χρησιμοποιώντας την εντολή **CALL**

Στην συγκεκριμένη διπλωματική, δημιουργήθηκε ένα java project, το οποίο εμπεριέχει τα τρία βασικά procedures για τα τρία διαφορετικά ερωτήματα και επιπλέον βοηθητικές συναρτήσεις που βοηθάνε τα procedures στην αναζήτηση. Για την δημιουργία του jar αρχείου, χρησιμοποιήθηκαν οι παρακάτω εκδόσεις:

- Apache Maven 3.9.4
- Java 17.0.7
- jdk-17

Οι εκδόσεις αυτές είναι συμβατές με την έκδοση 5.3.0 του Neo4j, η οποία χρησιμοποιήθηκε στην διπλωματική. Επιπλέον χρησιμοποιήθηκε το pom.xml [21] από το github project που προτείνεται επίσημα από το Neo4j για τα procedures.

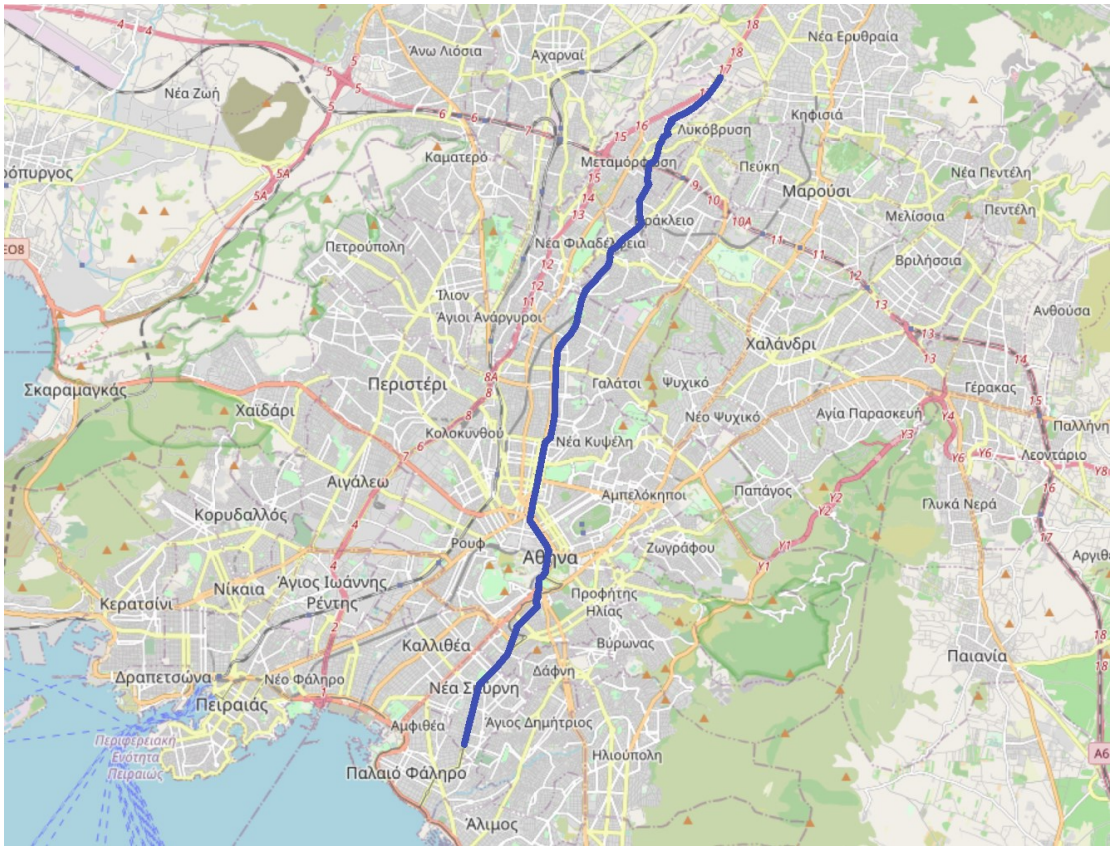
Τα τρία βασικά procedures για το κάθε ερώτημα είναι:

- noda.roadnetwork.closestPath: Παίρνει παραμέτρους δύο σημεία στον χάρτη, το όνομα του κόμβου στο οδικό δίκτυο, το όνομα της μεταβλητής που ορίζει το μήκος του δρόμου και βρίσκεται στις ακμές του οδικού δικτύου και επιστρέφει την συντομότερη διαδρομή μέσα στο οδικό δίκτυο.
- noda.roadnetwork.nearest: Παίρνει παραμέτρους ένα σημείο στον χάρτη, ένα εύρος , δύο χρονικά σημεία, τα ονόματα των κόμβων και των ακμών του οδικού δικτύου και των στιγμάτων των οχημάτων και επιστρέφει όλα τα οχήματα που βρίσκονται σε απόσταση το πολύ όσο το εύρος που δόθηκε στην συγκεκριμένη χρονική περίοδο που δόθηκε.
- noda.roadnetwork.closestObject: Παίρνει τις ίδιες παραμέτρους όπως και το παραπάνω και επιστρέφει το κοντινότερο όχημα στο σημείο που επιλέχτηκε.

5.2 Noda.roadnetwork.closestPath

Για την δημιουργία του συγκεκριμένου procedure, χρησιμοποιήθηκε το ήδη υπάρχων procedure από την APOC βιβλιοθήκη της Neo4j, **apoc.algo.dijkstra** το οποίο παίρνει σαν παραμέτρους δυο κόμβους του οδικού δικτύου και επιστρέφει την συντομότερη διαδρομή μεταξύ των δυο αυτών κόμβων στον γράφο και το βάρος της διαδρομής. Για την εύρεση της συντομότερης διαδρομής, το procedure αυτό χρησιμοποιεί τον αλγόριθμο του Dijkstra [5]. Στην συγκεκριμένη περίπτωση το βάρος είναι το μήκος της ακμής του οδικού δικτύου (Εικόνα 19). Το procedure αυτό έχει την εξής λογική:

- Αρχικά χρησιμοποιείται το πρώτο ζευγάρι γεωγραφικού μήκους και πλάτους, από τις παραμέτρους της συνάρτησης και γίνεται μια αναζήτηση για να βρεθεί ο κοντινότερος κόμβος του οδικού δικτύου που αντιστοιχεί στις συντεταγμένες αυτές. Ο κόμβος αυτός θα είναι ο πρώτος κόμβος του κοντινότερου μονοπατιού.
- Ύστερα γίνεται μια δεύτερη αναζήτηση με το δεύτερο ζευγάρι συντεταγμένων για να βρεθεί ο τελικός κόμβος του μονοπατιού.
- Τέλος οι δυο κόμβοι αυτοί χρησιμοποιούνται ως παράμετροι στο procedure της APOC βιβλιοθήκης, για να υπολογιστεί το κοντινότερο μονοπάτι.



ΕΙΚΟΝΑ 19: ΑΠΟΤΕΛΕΣΜΑ ΑΝΑΖΗΤΗΣΗΣ ΣΥΝΤΟΜΟΤΕΡΗΣ ΔΙΑΔΡΟΜΗΣ ΑΠΟ ΤΗΝ ΒΟΗΘΗΤΙΚΗ FLUTTER ΕΦΑΡΜΟΓΗ

5.3 Noda.roadnetwork.nearest

Για την δημιουργία του συγκεκριμένου procedure, υλοποιήθηκαν διάφορες βοηθητικές συναρτήσεις και κλάσεις.

Οι κλάσεις που δημιουργήθηκαν είναι οι εξής :

Nearest: Είναι η κλάση η οποία κρατάει όλες τις πληροφορίες από τα ερωτήματα που πρόκειται να γίνουν κατά την διάρκεια του αλγορίθμου. Έχει τα παρακάτω πεδία:

- Node_id (Long) : Είναι το id του κόμβου, από όπου ξεκινάει η αναζήτηση
- Array_of_nodes_ids (List<Long>): Ένας πίνακας, όπου αποθηκεύονται τα ids των κόμβων πάνω στους οποίους θα γίνει αναζήτηση για τα κοντινότερα στίγματα οχημάτων.
- Array_of_nodes (List<TempNodeWithDistance>): Λίστα που αποθηκεύει αντικείμενα της κλάσης TempNodeWithDistance, τα οποία αντικείμενα αυτά δημιουργούνται κατά την διάρκεια του αλγορίθμου που περιγράφεται αναλυτικότερα πιο κάτω.
- Vehicles (List<Vehicle>): Λίστα από αντικείμενα της κλάσης Vehicles.
- Vehicles_ids (List<Long>): Λίστα από τα ids των οχημάτων.

- finalNodes (List<Node>): Λίστα από αντικείμενα της κλάσης Node του Neo4j. Είναι η τελική λίστα των στιγμάτων των οχημάτων που επιστρέφει το procedure.

TempNodeWithDistance: Βοηθητική κλάση στην οποία αποθηκεύονται προσωρινά τα δεδομένα των κόμβων του οδικού δικτύου, τα οποία δεδομένα αυτά υπολογίζονται κατά την διάρκεια εκτέλεσης του αλγορίθμου. Η κλάση αυτή έχει τα παρακάτω πεδία:

- node_id (Long): Το id του κόμβου.
- distance_left (double): Η απόσταση που μένει ακόμα για να διανυθεί μέχρι να συμπληρωθεί το εύρος, το οποίο έχει δοθεί ως παράμετρος.
- latitude (double): Το γεωγραφικό πλάτος του κόμβου.
- longitude (double): Το γεωγραφικό μήκος του κόμβου.
- parent_id (long): Το id του κόμβου ο οποίος προηγείται του συγκεκριμένου κόμβου.

Vehicle: Βοηθητική κλάση στην οποία αποθηκεύονται τα δεδομένα του στίγματος των οχημάτων πάνω στο οδικό δίκτυο. Έχει τα παρακάτω πεδία:

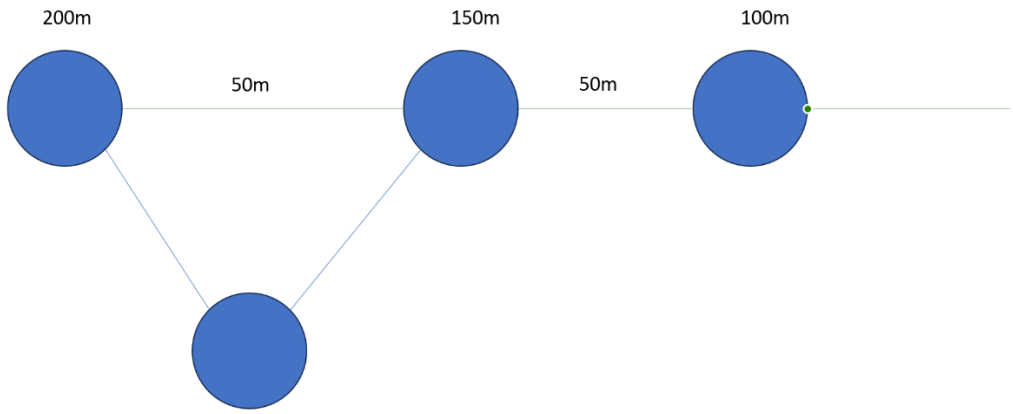
- id(String): Το id του κόμβου του στίγματος
- intersection_id (Long): Το id του κοντινότερου κόμβου του οδικού δικτύου στο στίγμα αυτό
- vehicle_id (Long): Το id του οχήματος, το οποίο δημιούργησε το στίγμα
- Osmid (Long): Το id του δρόμου πάνω στο οποίο βρίσκεται το στίγμα.
- Lat (double): Το γεωγραφικό πλάτος του στίγματος
- Lon (double): Το γεωγραφικό μήκος του στίγματος
- Timestamp (Long): Η χρονική στιγμή που δημιουργήθηκε το στίγμα.
- Distance_from_initial_node: Η απόσταση του στίγματος από τον κόμβο του οδικού δικτύου από τον οποίο ξεκινάει η αναζήτηση για τα κοντινότερα στίγματα μέσα στο εύρος που έχει δοθεί ως παράμετρος.

Οι βοηθητικές συναρτήσεις που δημιουργήθηκαν είναι η εξής:

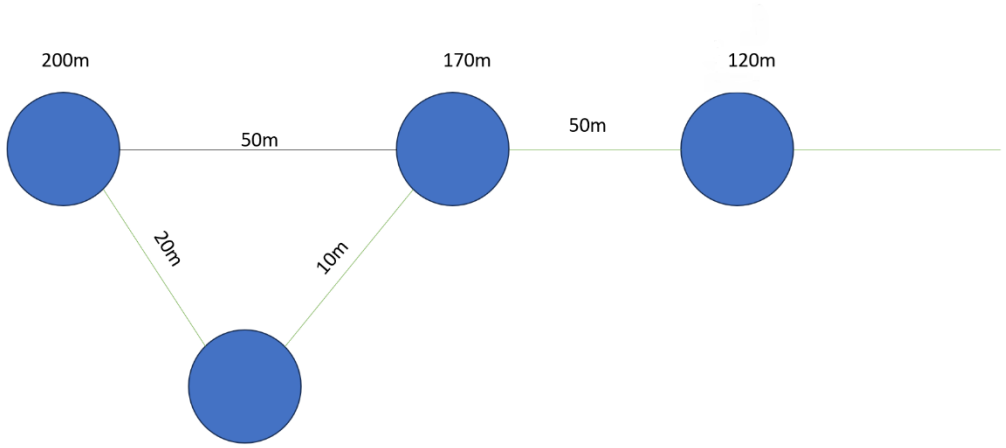
- distFrom: Παίρνει ως παραμέτρους δύο γεωγραφικά μήκη και πλάτη και επιστρέφει την απόσταση μεταξύ τους.
- extractLat: Παίρνει ως παράμετρο το location πεδίο του κόμβου και επιστρέφει το γεωγραφικό πλάτος ως double.
- extractLon: Παίρνει ως παράμετρο το location πεδίο του κόμβου και επιστρέφει το γεωγραφικό μήκος ως double.
- recursiveSearch: Κάνει αναδρομική αναζήτηση των κόμβων, οι οποίοι βρίσκονται μέσα στο εύρος που έχει δοθεί ως παράμετρο. Θα γίνει αναλυτικότερη περιγραφή παρακάτω στην επεξήγηση του αλγορίθμου.
- findNearestObject: Χρησιμοποιεί τους κόμβους που εντόπισε στην αναζήτηση της η παραπάνω συνάρτηση, για την εύρεση των στιγμάτων των οχημάτων που βρίσκονται μέσα στο εύρος που έχει δοθεί ως παράμετρο.

Ο αλγόριθμος για την εύρεση των κοντινότερων οχημάτων σε ένα εύρος μέσα στο οδικό δίκτυο, έχει τα εξής βήματα:

- Αρχικά, το procedure παίρνει από τις παραμέτρους το γεωγραφικό μήκος και πλάτος που έχει δώσει ο χρήστης και αναζητά το κοντινότερο κόμβο του οδικού δικτύου.
- Αφού ανακτηθεί ο κόμβος αυτός, δημιουργείται ένα αντικείμενο της κλάσης Nearest και αρχικοποιούνται τα πεδία node_id με το osmid του κόμβου και το distance_to_cover με το εύρος που θα πραγματοποιηθεί η αναζήτηση και έχει δοθεί από τον χρήστη ως μία από τις παραμέτρους.
- Ύστερα, καλείται η αναδρομική συνάρτηση recursiveSearch, με παραμέτρους το id του αρχικού κόμβου και το αρχικό εύρος. Μέσα στην recursiveSearch, γίνεται αναζήτηση για τους γειτονικούς κόμβους του κόμβου που έχει δοθεί σαν παράμετρος. Αφού ανακτηθούν οι κόμβοι αυτοί, υπολογίζεται η απόσταση που έχει διανυθεί από τον αρχικό κόμβο μέχρι τους γειτονικούς κόμβους ξεχωριστά και ύστερα γίνεται μία αναζήτηση για τον κάθε γειτονικό κόμβο, για το αν υπάρχει το id του μέσα στον πίνακα array_of_nodes_ids της κλάσης Nearset.
- Σε περίπτωση που δεν υπάρχει, τότε δημιουργείται ένα αντικείμενο της κλάσης TempNodeWithDistance, με παραμέτρους το id του κόμβου, distance_left η απόσταση που μένει ακόμα να διανυθεί από το αρχικό εύρος που έχει δώσει ο χρήστης, lat και lon τα γεωγραφικά μήκη και πλάτη του κόμβου και parent_id τον αρχικό κόμβο από τον οποίο προήλθε αυτός ο κόμβος. Αφού δημιουργηθεί και το αντικείμενο της κλάσης TempNodeWithDistance, προστίθεται στον πίνακα array_of_nodes και το id του γειτονικού κόμβου προστίθεται στον πίνακα array_of_nodes_ids. Τέλος, καλείται πάλι η recursiveSearch, με παραμέτρους τώρα το id του γειτονικού κόμβου και την απόσταση που έχει απομείνει ακόμα από το αρχικό εύρος.
- Στην περίπτωση που υπάρχει το id στο array_of_nodes_id, τότε ανακτάται από το array_of_nodes ο κόμβος αυτός και γίνεται έλεγχος στο πεδίο distance_left. Αν το πεδίο distance_left που είναι αποθηκευμένος, είναι μικρότερος από το distance_left που υπολογίζεται στην αρχή, τότε ανανεώνεται ο κόμβος αυτός με τα νέα δεδομένα και ξανακαλείται η recursiveSearch για αυτόν τον κόμβο με τις νέες παραμέτρους. Ο λόγος που συμβαίνει αυτό το βήμα, είναι διότι στην αναδρομική αναζήτηση που συμβαίνει, μπορεί να βρεθεί ένας κόμβος, ο οποίος έχει βρεθεί ξανά πιο πριν από διαφορετικό μονοπάτι, το οποίο μονοπάτι αυτό να είναι μεγαλύτερο από το μονοπάτι που βρέθηκε την δεύτερη φορά. Έτσι λοιπόν αν ο κόμβος αυτός έχει μικρότερο μονοπάτι, θα πρέπει να ανανεωθούν τα δεδομένα για αυτόν τον κόμβο και όλους του υπόλοιπους κόμβους που είχαν βρεθεί (Εικόνα 20 και Εικόνα 21).



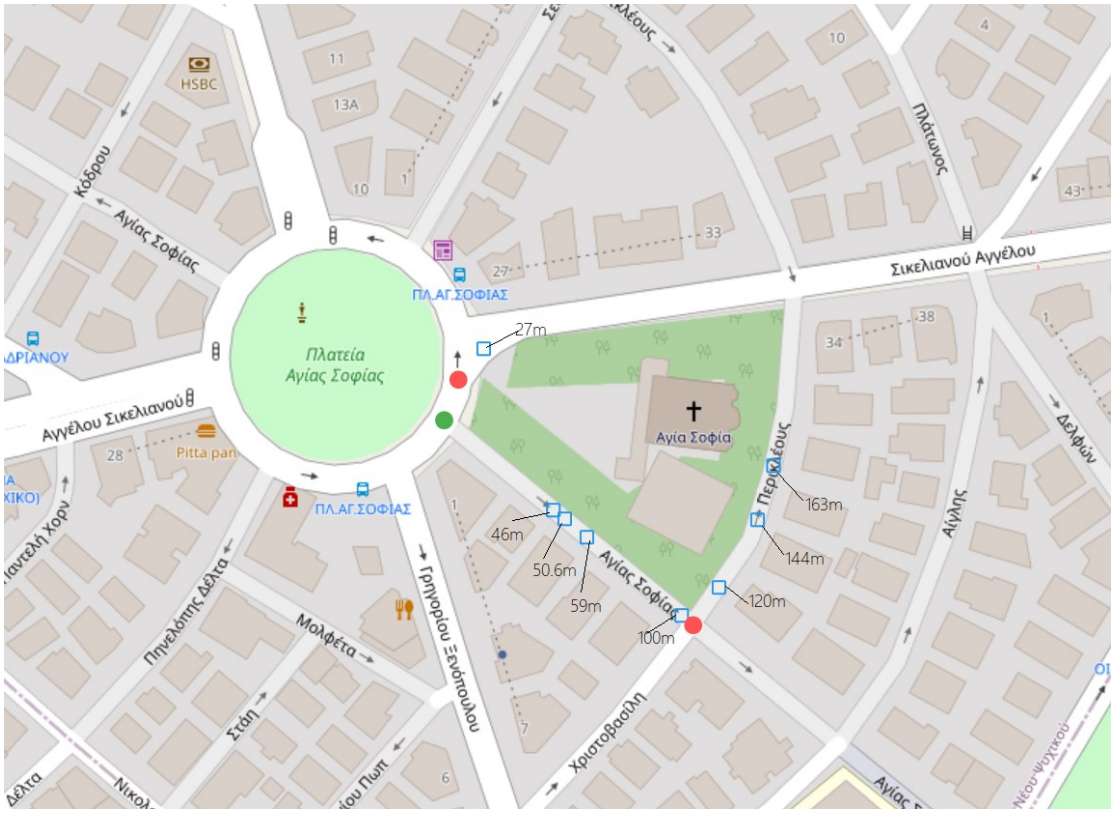
ΕΙΚΟΝΑ 20: ΚΟΜΒΟΙ ΜΕ ΤΑ ΠΟΣΑ ΜΕΤΡΑ ΕΧΟΥΝ ΑΚΟΜΑ ΝΑ ΔΙΑΝΥΣΟΥΝ. ΕΔΩ ΞΕΚΙΝΑΕΙ Ο ΑΛΓΟΡΙΘΜΟΣ ΑΠΟ ΤΟΝ ΠΡΩΤΟ ΚΟΜΒΟ ΚΑΙ ΣΥΝΕΧΙΖΕΙ



ΕΙΚΟΝΑ 21: ΑΝΑΝΕΩΣΗ ΤΩΝ ΑΠΟΣΤΑΣΕΩΝ ΣΕ ΚΑΘΕ ΚΟΜΒΟ ΜΕΤΑ ΤΗΝ ΕΥΡΕΣΗ ΚΟΝΤΙΝΟΤΕΡΟΥ ΜΟΝΟΠΑΤΙΟΥ

- Η recursiveSearch σταματάει, όταν το distance_left είναι μικρότερο ή ίσο με το 0.
- Με την βοήθεια της recursiveSearch, αυτό που γίνεται στην ουσία, είναι να γεμίσει ο πίνακας array_of_nodes_ids με όλους τους κόμβους του οδικού δικτύου που βρίσκονται μέσα στο εύρος. Και αφού ολοκληρωθεί η recursiveSearch , καλείται η συνάρτηση find_nearest_object.

- Η find_nearest_object, αρχικά αναζητεί όλα τα στίγματα των οχημάτων τα οποία είναι συνδεδεμένα με τους κόμβους του οδικού δικτύου, που ανακτήθηκαν από την recursiveSearch και βρίσκονται μέσα στο χρονικό όριο, το οποίο έχει δοθεί από τον χρήστη. Μέσα σε αυτήν την συνάρτηση, υπολογίζονται οι αποστάσεις των στιγμάτων από τον αρχικό κόμβο που ξεκίνησε η αναζήτηση και υπάρχει ένας μικρός αλγόριθμος που υπολογίζει για το αν η απόσταση των στιγμάτων θα πρέπει να προστίθεται ή να αφαιρείται αναλόγως με το που βρίσκεται το στίγμα σε σχέση με τον κοντινότερο του κόμβο. Το κάθε στίγμα που καλύπτει τις προϋποθέσεις, προστίθεται στον πίνακα finalNodes του αντικειμένου της κλάσης Nearsest που δημιουργήθηκε στην αρχή.
- Τέλος , αφού βρεθούν όλα τα στίγματα, το procedure επιστρέφει τον πίνακα finalNodes, ο οποίος περιέχει όλα τα στίγματα από ένα σημείο στον χάρτη, μέσα σε ένα συγκεκριμένο χωροχρονικό όριο, με αποστάσεις υπολογισμένες μέσω του οδικού δικτύου (Εικόνα 22).



ΕΙΚΟΝΑ 22:ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΥΡΕΣΗΣ ΚΟΝΤΙΝΟΤΕΡΩΝ ΣΤΙΓΜΑΤΩΝ ΑΠΟ ΤΗΝ ΒΟΗΘΗΤΙΚΗ ΕΦΑΡΜΟΓΗ FLUTTER

Όπως φαίνεται και στην παραπάνω εικόνα, τα μπλε τετράγωνα είναι τα στίγματα που βρέθηκαν από μια αναζήτηση και οι αποστάσεις τους από τον πράσινο κόμβο, που είναι ο αρχικός κόμβος στην συγκεκριμένη αναζήτηση.

5.4 Noda.roadnetwork.closestObject

Το procedure αυτό, ακολουθεί ακριβώς την ίδια διαδικασία που ακολουθεί το procedure noda.roadnetwork.nearest, με την μόνη διαφορά ότι το εύρος δεν δίνεται από τον χρήστη άλλα είναι ορισμένο μέσα στο procedure με σταθερή τιμή και στο τέλος, δεν επιστρέφεται όλος ο πίνακας finalNodes, αλλά λίγο πριν επιστραφεί γίνεται ταξινόμηση με την απόσταση του στίγματος από τον αρχικό κόμβο και επιστρέφεται μόνο το στίγμα εκείνο που βρίσκεται στην πρώτη θέση του πίνακα, αφού η ταξινόμηση γίνεται με αύξουσα σειρά. Επίσης, μαζί με τον κόμβο του στίγματος, επιστρέφεται και η απόσταση του από τον αρχικό κόμβο του οδικού δικτύου, από όπου ξεκίνησε η αναζήτηση.

6 Δημιουργία των τελεστών NoDA

Για τους τελεστές NoDA του οδικού δικτύου, δημιουργήθηκαν τρεις τελεστές κάτω από τον φάκελο **roadNetworkOperators.**

- OperatorShortestPath: Για την εύρεση της συντομότερης διαδρομής.
- OperatorNearestInNetworkRange: Για την εύρεση των κοντινότερων αντικειμένων μέσα σε ένα οδικό δίκτυο από ένα σημείο και σε συγκεκριμένο χωρο-χρονικό εύρος.
- OperatorNearestObject: Για την εύρεση του κοντινότερου αντικειμένου μέσα στο οδικό δίκτυο από ένα σημείο και σε συγκεκριμένο χωρο-χρονικό εύρος.

Για να μπορέσει ένας χρήστης να χρησιμοποιήσει τους operator αυτούς, θα πρέπει αρχικά να κάνει μια σύνδεση στην βάση Neo4j, όπως φαίνεται παρακάτω (Εικόνα 23):

```
NoSqlDbSystem n = NoSqlDbSystem.Neo4j().Builder( username: "neo4j",  password: "12345678").host("localhost").port(7687).build();
```

ΕΙΚΟΝΑ 23: ΣΥΝΔΕΣΗ ΣΕ NEO4J ΜΕΣΩ ΤΟΥ ΕΡΓΑΛΕΙΟΥ

Τον πρώτο operator, μπορεί να τον καλέσει με τον εξής τρόπο (Εικόνα 24):

```
 //Shortest Path
n.operateOn( s: "Intersection;ROAD_SEGMENT")
        .filter(shortestPathInRoadNetwork( locationField: "length",
                Point.newPoint(Coordinates.newCoordinates( longitude: 23.7773689, latitude: 38.00089)),
                Point.newPoint(Coordinates.newCoordinates( longitude: 23.5287424, latitude: 38.0599682))))
        .printScreen();
```

ΕΙΚΟΝΑ 24: ΕΚΤΕΛΕΣΗ ΠΡΩΤΟΥ ΤΕΛΕΣΤΗ

Γίνεται το operateOn πάνω στον κόμβο Intersection του οδικού δικτύου και στην ακμή ROAD_SEGMENT. Η συνάρτηση shortestPathInRoadNetwork παίρνει σαν παραμέτρους το όνομα του πεδίου που είναι αποθηκευμένο το μήκος της ακμής (length) και το αρχικό και τελικό σημείο, που είναι τύπου Point.

Το αποτέλεσμα της παραπάνω κλήσης είναι (Εικόνα 25):

```
(9)-[ROAD_SEGMENT,24]->(8)<-[ROAD_SEGMENT,128339]-(55126)-[ROAD_SEGMENT,128337]->(55186)-[ROAD_SEGMENT,128495]->(18) weight:26829.64199999999]">]
```

ΕΙΚΟΝΑ 25: ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΠΡΩΤΟΥ ΤΕΛΕΣΤΗ

Στην ουσία εκτυπώνεται όλη η διαδρομή από τον πρώτο κόμβο μέχρι και τον τελευταίο και στο τέλος εκτυπώνεται και η απόσταση που διένυσε. Στην συγκεκριμένη περίπτωση είναι 26,9 km.

Ο δεύτερος operator μπορεί να καλεστεί με τον εξής τρόπο (Εικόνα 26):

ΕΙΚΟΝΑ 26: ΕΚΤΕΛΕΣΗ ΔΕΥΤΕΡΟΥ ΤΕΛΕΣΤΗ

Το operateOn γίνεται πάνω στους κόμβου Intersection και Vehicle και στις ακμές ROAD_SEGMENT και NEAREST_INTERSECTION. Η συνάρτηση nearestInRangeInRoadNetwork παίρνει ως παραμέτρους το όνομα του χωρικού πεδίου (location) και το όνομα του χρονικού πεδίου (timestamp), όπως και το σημείο από όπου θα ξεκινήσει η αναζήτηση, που είναι τύπου Point και το χρονικό εύρος που ορίζεται από δύο παραμέτρους τύπου Date. Επιπλέον έχει και το εύρος μέσα στο οποίο θα γίνει η αναζήτηση. Στο συγκεκριμένο παράδειγμα είναι 200m.

Το αποτέλεσμα της παραπάνω κλήσης είναι (Εικόνα 27):

ΕΙΚΟΝΑ 27: ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΚΤΕΚΕΣΗΣ ΔΕΥΤΕΡΟΥ ΤΕΛΕΣΤΗ

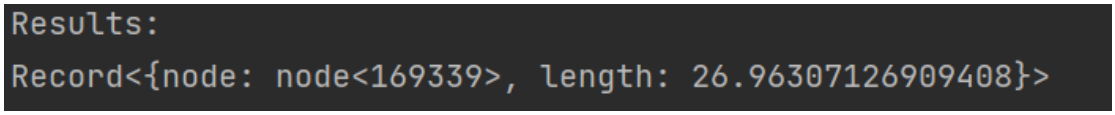
Εδώ επιστρέφονται οι κόμβοι των αντικειμένων που βρίσκονται μέσα στο εύρος των 200m.

Τέλος ο τελευταίος τελεστής καλείται με τον εξής τρόπο (Εικόνα 28):

ΕΙΚΟΝΑ 28:ΕΚΤΕΛΕΣΗ ΤΡΙΤΟΥ ΤΕΛΕΣΤΗ

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, ο τελεστής αυτός είναι παρόμοιος με τον προηγούμενο, με την διαφορά ότι δεν παίρνει εύρος από τον χρήστη. Αυτό φαίνεται και στην συνάρτηση nearestObjectInRoadNetwork, οπού έχει τις ίδιες παραμέτρους με την προηγούμενη συνάρτηση, με την διαφορά ότι εδώ λείπει η παράμετρος του εύρους.

Το αποτέλεσμα της παραπάνω κλήσης είναι (Εικόνα 29):



ΕΙΚΟΝΑ 29: ΑΠΟΤΕΛΕΣΜΑ ΕΚΤΕΛΕΣΗΣ ΤΡΙΤΟΥ ΤΕΛΕΣΤΗ

Εδώ επιστρέφεται ο κόμβος του κοντινότερου αντικειμένου και η απόσταση του από τον κόμβο του οδικού δικτύου που ξεκινά η αναζήτηση.

7 Πειραματική μελέτη

Για την αξιολόγηση της απόδοσης των τελεστών που δημιουργήθηκαν, έγιναν πειράματα σε διαφορετικές βάσεις δεδομένων. Το οδικό δίκτυο σε όλες τις βάσεις δεδομένων αποτελείται από 77515 κόμβους και 178404 ακμές. Πάνω σε αυτό το οδικό δίκτυο, δημιουργήθηκε μια βάση δεδομένων με επιπλέον 100000 κόμβους οχημάτων και μια άλλη βάση με 500000 κόμβους οχημάτων, αυξάνοντας την πολυπλοκότητα των ερωτημάτων. Σε αυτές τις δυο βάσεις δεδομένων, δοκιμάστηκε η απόδοση και των τριών ερωτημάτων που δημιουργήθηκαν. Επιπλέον το ερώτημα για την συντομότερη διαδρομή δοκιμάστηκε και σε μια βάση δεδομένων η οποία έχει αποθηκευμένη μόνο το οδικό δίκτυο και σε διαφορετικά μήκη.

Οι δοκιμές εκτελέστηκαν σε τοπικό μηχάνημα με τα ακόλουθα χαρακτηριστικά:

- Λειτουργικό σύστημα : Windows 10
- CPU: Ryzen 5 3600 (6 cores, 12 threads, 3.6 GHz)
- RAM: 16 GB

7.1 Noda.roadnetwork.closestPath

Εδώ έγιναν δοκιμές για τον operator για την συντομότερη διαδρομή ανάμεσα σε 2άδες κόμβων που έχουν αποστάσεις μεταξύ τους 2, 5, 10, 20 και 30 χιλιόμετρα αντίστοιχα. Οι δοκιμές έγιναν σε όλες τις βάσεις που περιέχουν το οδικό δίκτυο.

	2km	**5km**	**10km**	**20km**	**30km**
Οδ. Δίκτυο	4ms	65ms	280ms	390ms	480ms
ΟΔ+100k	5ms	85ms	372ms	440ms	558ms
ΟΔ+500κ	6ms	90ms	390ms	458ms	560ms
nodes	45	57	105	167	147

Από τον πίνακα φαίνεται ότι στην βάση δεδομένων που έχει μόνο το οδικό δίκτυο, έχουμε και τους καλύτερους χρόνους και προφανώς όσο πιο κοντά είναι οι δύο κόμβοι πάνω στους οποίους θα τρέξει ο τελεστής, τόσο καλύτεροι είναι και οι χρόνοι.

Παρατηρώντας και τους χρόνους στις βάσεις οι οποίες έχουνε μέσα όχι μόνο το οδικό δίκτυο, αλλά και τα στίγματα των αντικειμένων, προφανώς και υπάρχει μια μείωση της απόδοσης, η οποία είναι αμελητέα και για τις δύο περιπτώσεις. Στην τελευταία στήλη, έχουνε καταγραφεί και από πόσους κόμβους αποτελείται η συντομότερη διαδρομή για την κάθε περίπτωση.

7.2 Noda.roadnetwork.nearest

Έγιναν δοκιμές στις 2 βάσεις που περιέχουνε και το οδικό δίκτυο αλλά και τα στίγματα των οχημάτων με εύρος 200, 400,600, 800 μέτρα αντίστοιχα. Στην πρώτη βάση δεδομένων υπάρχουν 100.000 στίγματα ενώ στην δεύτερη 500.000 στίγματα. Μετρήθηκαν 2 διαφορετικοί χρόνοι, ο ένας για το πρώτο κομμάτι του αλγορίθμου που αναζητά τους κοντινότερους κόμβους και ο άλλος για το κομμάτι του αλγορίθμου που αναζητά τα στίγματα. Στους δύο πρώτους πίνακες, επιλέχτηκε μια περιοχή της Αθήνας, η οποία είχε μεγάλη πυκνότητα κόμβων στο οδικό δίκτυο και στους δύο επόμενους πίνακες, επιλέχτηκε μια περιοχή με λιγότερους κόμβους. Στην πρώτη στήλη των πινάκων καταγράφεται ο χρόνος εκτέλεσης του πρώτου μέρους του αλγορίθμου που είναι υπεύθυνος για την εύρεση των κόμβων του οδικού δικτύου. Στην δεύτερη στήλη καταγράφεται ο χρόνος εκτέλεσης του δευτέρου τμήματος του αλγορίθμου ο οποίος είναι υπεύθυνος για την εύρεση και ανάκτηση των στιγμάτων ενώ στην τρίτη στήλη, ο αριθμός των κόμβων του οδικού δικτύου που υπολογίζεται στο πρώτο τμήμα του αλγορίθμου.

Για την βάση με τα 100.000 στίγματα

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
200m	2649ms	67ms	57
400m	14307ms	56ms	141
600m	49256ms	20ms	266
800m	120502ms	23ms	409

Για την βάση με τα 500.000 στίγματα

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
200m	5544ms	52ms	57
400m	34576ms	30ms	141
600m	144218ms	25ms	266
800m	374348ms	26ms	409

Για την βάση με τα 100.000 στίγματα

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
200m	187ms	4ms	11

400m	1943ms	6ms	55
600m	7631ms	7ms	115
800m	21515ms	9ms	216

Για την βάση με τα 500.000 στίγματα

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
200m	452ms	9ms	11
400m	5518ms	8ms	55
600m	21432ms	5ms	115
800m	64287ms	7ms	216

Παρατηρώντας τους πίνακες με τις αποδόσεις για την εκτέλεση του τελεστή εύρεσης τον κοντινότερων αντικειμένων, βλέπουμε ότι υπάρχει πολύ μεγάλη αύξηση ακόμα και για μερικά μέτρα παραπάνω. Αυτό συμβαίνει εξαιτίας του αλγορίθμου που έχει υλοποιηθεί για την εύρεση των κοντινότερων κόμβων σε ένα δίκτυο. Ο τρόπος που λειτουργεί ο αλγόριθμος και επειδή καλείται αναδρομικά, είναι και ο λόγος που βλέπουμε τόσο μεγάλη μείωση στην απόδοση, ακόμα και αν αυξήσουμε πολύ λίγο τα μέτρα. Επιπλέον σημαντικό ρόλο παίζει και η πυκνότητα των κόμβων του οδικού δικτύου στο εύρος που έχει δοθεί.

7.3 Noda.roadnetwork.closestObject

Αντίστοιχα και με τον προηγούμενο τελεστή και εδώ έγιναν δοκιμές στις 2 βάσεις δεδομένων που περιέχουν και το οδικό δίκτυο και τα στίγματα των οχημάτων, με την ίδια λογική όπως και πριν και στα δυο σημεία του οδικού δικτύου όπως και πριν

Για το σημείο με την μεγάλη πυκνότητα κόμβων:

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
100k	2682ms	38ms	57
500k	5312ms	35ms	57

Για την βάση με τα 500.000 στίγματα

	Χρόνος εκτέλεσης πρώτου μέρους	**Χρόνος εκτέλεσης δεύτερου μέρους**	**Κόμβοι οδικού δικτύου**
100k	187ms	4ms	11

500k	452ms	52ms	11

Όπως φαίνεται και από τα αποτελέσματα, ο τελεστής αυτός, επειδή βασίζεται επίσης στον ίδιο αλγόριθμο που καλείται αναδρομικά για την εύρεση των κοντινότερων κόμβων και για αυτό παρατηρούνται παρόμοιοι χρόνοι εκτέλεσης με τον δεύτερο τελεστή για το εύρος των 200m, αφού ο τελεστής αυτός έχει μέσα στον αλγόριθμο σταθερό εύρος τα 200m.

Όπως φαίνεται στις παραπάνω μετρήσεις, όλοι οι χρόνοι σε όλους τους τελεστές, μπορούν να μειωθούν ή να αυξηθούν δραματικά, αναλόγως με την πυκνότητα των κόμβων. Ειδικά για τον δεύτερο και τρίτο τελεστή που αφορούν ο ένας την εύρεση των κοντινότερων αντικειμένων πάνω στο οδικό δίκτυο από ένα σημείο και ο άλλος την εύρεση του κοντινότερου αντικειμένου από ένα σημείο, παίζει πολύ σημαντικό ρόλο για το πόσο κοντά είναι οι κόμβοι μεταξύ τους. Για παράδειγμα για τον δεύτερο τελεστή αν βάλουμε σαν παράμετρο εύρος 500m σε ένα σημείο όπου οι κόμβοι του οδικού δικτύου είναι πιο «αραιοί» , τότε θα εκτελεστεί πολύ πιο γρήγορα από το να το βάζαμε να τρέξει με τις ίδιες παραμέτρους σε ένα σημείο του οδικού δικτύου, το οποίο περιέχει περισσότερους κόμβους. Επιπλέον, η απόδοση εξαρτάται και από το πόσα αντικείμενα είναι αποθηκευμένα στο οδικό δίκτυο, αλλά η αναζήτηση και ανάκτηση αυτών των δεδομένων είναι αμελητέα μπροστά στην πυκνότητα των κόμβων.

8 Συμπεράσματα και μελλοντικές επεκτάσεις

Η εργασία αυτή είχε ως αντικείμενο την δημιουργία τελεστών για το NoDA, οι οποίοι θα εφαρμόζουν χωρο-χρονικά ερωτήματα σε ένα οδικό δίκτυο και σε αντικείμενα που βρίσκονται πάνω στο οδικό δίκτυο, σε μια Neo4j βάση δεδομένων. Για να γίνει αυτό εφικτό, προ-επεξεργάστηκαν δεδομένα από πραγματικά στίγματα αυτοκινήτων που κυκλοφορούσαν πάνω στο οδικό δίκτυο της Αθήνας και δεδομένα από την πλατφόρμα του OpenStreetMap για το οδικό δίκτυο της Αθήνας. Αναπτύχθηκαν procedures για την Neo4j , μέσα στα οποία υλοποιήθηκαν οι αντίστοιχοι αλγόριθμοι για το κάθε χωρο-χρονικό ερώτημα. Τα procedures αυτά είναι εκείνα πάνω στα οποία βασίζεται η δημιουργία των τελεστών του NoDA. Τα ερωτήματα αυτά αφορούν την εύρεση κοντινότερου μονοπατιού ανάμεσα σε 2 σημεία στον χάρτη, την εύρεση κοντινότερων αντικειμένων από ένα σημείο στο οδικό δίκτυο, μέσα σε ένα συγκεκριμένο χρονικό διάστημα και εύρος και την εύρεση του κοντινότερου αντικειμένου από ένα σημείο στο οδικό δίκτυο επίσης μέσα σε ένα συγκεκριμένο χρονικό σημείο και εύρος. Τέλος , έγινε και μια πειραματική μελέτη για την απόδοση των τελεστών αυτών πάνω σε διαφορετικές βάσεις δεδομένων, οι οποίες δημιουργήθηκαν για αυτόν τον σκοπό.

Ως μελλοντική επέκταση, θα μπορούσε να γίνει κάποια βελτιστοποίηση στον αλγόριθμο για την εύρεση κοντινότερων αντικειμένων. Το κομμάτι του αλγορίθμου που αφορά την αναζήτηση των κοντινότερων κόμβων δικτύου , όσο και σημαντικό να θεωρείται, είναι εκείνο το οποίο καθυστερεί αρκετά τον αλγόριθμο και θα μπορούσε σε εκείνο το σημείο να γίνει κάποια βελτιστοποίηση, αναλόγως και με το πως κάθε χρήστης χρησιμοποιεί τον τελεστή που υλοποιεί αυτόν τον αλγόριθμο. Για παράδειγμα θα μπορούσε να γίνεται cached μια τέτοια αναζήτηση και να μην χρειάζεται να υπολογίζεται συνέχεια το cluster με τους κοντινότερους κόμβους. Ή αντίστοιχα θα μπορούσε αντί να γίνεται κατά βάθος αναζήτηση με αναδρομή, να γίνεται κατά πλάτος αναζήτηση.

9 Πίνακας Εικόνων

10 Βιβλιογραφία

1. N. Koutroumanis, P. Nikitopoulos, A. Vlachou και C. Doulkeridis, “NoDA: Unified NoSQL Data Access Operators for Mobility Data”. SSTD '19: Proceedings of the 16th International Symposium on Spatial and Temporal Databases. 2019. pp. 174-177. DOI: 10.1145/3340964.3340981.

2. N. Koutroumanis, C. Doulkeridis και A. Vlachou, "Tearing Down the Tower of Babel: Unified and Efficient Spatio-temporal Queries for NoSQL Stores," 2022 23rd IEEE International Conference on Mobile Data Management (MDM), Paphos, Cyprus, 2022, pp. 19-28, DOI: 10.1109/MDM55031.2022.00024.

3. J. Han, E. Haihong, G. Le and J. Dual. “Survey on NoSQL database”. In 6th IEEE International Conference on Pervasive Computing and Applications (ICPCA’2011), 2011. Port Elizabeth, South Africa, pp. 363-366.

4. S. Kitaev, “A Comprehensive Introduction to the Theory of Word-Representable Graphs”. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2017. pp. 36-67. DOI: 10.1007/978-3-319-62809-7_2.

5. K. Wei, Y. Gao, W.X. Zhang & S. Lin.” A Modified Dijkstra’s Algorithm for Solving the Problem of Finding the Maximum Load Path”. 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), 2019. pp. 10-13. DOI:10.1109/infoct.2019.8711024.

6. R.E. Toohey, W. Brown και J. H. & Stebbings. “Systematic geographic sampling with UTM coordinates”. Environment International, 1988. 14(3), pp. 207–212. DOI:10.1016/0160-4120(88)90140-7.

7. B. Jiang, “A topological pattern of urban street networks: Universality and peculiarity”. Physica A: Statistical Mechanics and Its Applications, 2007. 384(2), pp. 647–655. DOI: 10.1016/j.physa.2007.05.064.

8. B. Jiang & C. Claramunt. “Topological Analysis of Urban Street Networks”. Environment and Planning B: Planning and Design. 2004. 31. Pp. 151-162. DOI: 10.1068/b306.

9. M. Haklay & P. Weber. “OpenStreetMap: User-Generated Street Maps”. IEEE Pervasive Computing, 2008. 7(4). pp. 12-18. ISSN 15361268. 7. DOI: 10.1109/MPRV.2008.80.

10. Wikipedia. “OpenStreetMap”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://en.wikipedia.org/wiki/OpenStreetMap. [Πρόσβαση 01/10/2023].

11. MongoDB. “Compass. The GUI for MongoDB”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://www.mongodb.com/products/tools/compass [Πρόσβαση 01/10/2023].

12. Jupyter. “Jupyter Notebooks”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://jupyter.org/. [Πρόσβαση 01/10/2023].

13. Google. “Flutter Framework”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://flutter.dev/. [Πρόσβαση 01/10/2023].

14. Neo4j.” Neo4j Desktop”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://neo4j.com/download/. [Πρόσβαση 01/10/2023].

15. G. Boeing. (2017). “OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks”. Computers, Environment and Urban Systems. 2017. 65. pp. 126-139, DOI: 10.31235/osf.io/q86sd.

16. Wikipedia. “Overpass API”, 2023. [Ηλεκτρονική Πηγή]. Available at: https://wiki.openstreetmap.org/wiki/Overpass_API. [Πρόσβαση 01/10/2023].