ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δρομολόγηση Οχημάτων Περιορισμένης Χωρητικότητας με Χρονικά Όρια

Ιωσήφ Ε. Αγγελίδης

Επιβλέποντες: **Δελής Αλέξιος,** Καθηγητής ΕΚΠΑ
Λιάκος Παναγιώτης, Υποψήφιος Διδάκτωρ ΕΚΠΑ

ΑΘΗΝΑ

ΑΥΓΟΥΣΤΟΣ 2015

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δρομολόγηση Οχημάτων Περιορισμένης Χωρητικότητας με Χρονικά Όρια

Ιωσήφ Ε. Αγγελίδης
Α.Μ.: 1115201100127

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δελής Αλέξιος,** Καθηγητής ΕΚΠΑ
Λιάκος Παναγιώτης, Υποψήφιος Διδάκτωρ ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Το Πρόβλημα Δρομολόγησης Οχημάτων Περιορισμένης Χωρητικότητας με Χρονικά Όρια (Capacitated Vehicle Routing Problem with Time Windows, CVRPTW) έχει απασχολήσει την ακαδημαϊκή κοινότητα για πολλά χρόνια λόγω της δυσκολίας του (ανήκει στην κλάση NP) και έχουν δοθεί πολλές προσεγγιστικές λύσεις που είναι επαρκώς ικανοποιητικές. Φυσικά, υπάρχει μια πληθώρα εφαρμογών των λύσεων αυτών σε πρακτικές εφαρμογές.

Στόχος του προγράμματος που τεκμηριώνεται στην παρούσα πτυχιακή είναι να λύνει αρχικά το πρόβλημα ακολουθώντας κλασσικές μεθόδους της βιβλιογραφίας χρησιμοποιώντας πραγματικά γεωγραφικά δεδομένα σε συντεταγμένες Γης. Στην συνέχεια, επικοινωνεί με μια σχεσιακή βάση (relational database) που περιέχει όλους του δρόμους της Αθήνας και πληροφορίες για την κατεύθυνσή τους (μονής ή διπλής).

Επιπλέον, επικοινωνεί με μια αποθήκη κλειδού-τιμής στην μνήμη (in-memory key-value store) η οποία περιέχει επιπλέον δυναμικές πληροφορίες που αφορούν δρόμους. Πιο συγκεκριμένα, περιέχει θετικά και αρνητικά ποσοστιαία βάρη που δηλώνουν δυναμικά δεδομένα δρόμων. Θετικά βάρη σημαίνουν κίνηση και παίρνουν τιμές στο διάστημα $[0.0, 1.0]$. Αρνητικά βάρη σημαίνουν εμπειρία οδηγών που τους επηρεάζει να ακολουθήσουν συγκεκριμένους δρόμους γιατί ξέρουν ότι θα φτάσουν πιο γρήγορα και παίρνουν τιμές στο διάστημα $[-0.5, 0.0]$. Τα αρνητικά δεν έχουν το ίδιο εύρος τιμών σε σχέση με τα θετικά για διαισθητικούς λόγους αφού η κίνηση δρόμων επηρεάζει την λήψη απόφασης πολύ περισσότερο από την εμπειρία του οδηγού αφού μπορεί ένας δρόμος να έχει μποτιλιάρισμα που δεν είναι αναμενόμενο.

Κατά την εκτέλεση του προγράμματος εκτυπώνονται μηνύματα που εξηγούν τις αλγοριθμικές αποφάσεις που λαμβάνονται σε κρίσιμα βήματα και αποθηκεύονται ενδιάμεσα αρχεία για μελέτη των αποτελεσμάτων αφού τελειώσει η εκτέλεση. Τέλος, παράγονται ένα αρχείο kml και ένα αρχείο html για να είναι δυνατόν το τελικό αποτέλεσμα να μπορεί να προβληθεί σε οποιαδήποτε εφαρμογή χειρίζεται kml αρχεία (ή html αν θέλουμε να δούμε την οπτικοποίηση στο Διαδίκτυο). Συνήθεις εφαρμογές είναι το Google Earth, το Google Maps και το Marble.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κατανομή Πόρων, Μεταφορά

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: δρομολόγηση οχημάτων, χρονικά όρια, δρομολόγηση, μεταφορά, κατευθυνόμενη τοπική αναζήτηση

ABSTRACT

Capacitated Vehicle Routing Problem with Time Windows is a problem that has troubled academics for many years due to its difficulty (it belongs to the class NP) and many approximate solutions that are fairly sufficient have been proposed. Of course, a plethora of application of those solutions exists in practical applications.

The goal of the program documented in the present thesis is to initially solve the problem based on classic methods from bibliography using real geographical data in Earth coords. After that, it communicates with a relational database which contains roads of Athens and info about their direction (unidirectional or bidirectional).

Furthermore, it communicates with an in-memory key-value store which contains additional dynamic info on roads. To be more specific, it contains positive and negative percentage weights that express dynamic roads' info. Positive weights represent traffic and have a value in the range of $[0.0, 1.0]$. Negative weights represent drivers' experience that influences them to follow specific roads because they are aware that they will reach their destination faster that way and have a value in the range of $[-0.5, 0.0]$. Negative weights do not have the same range as positive ones due to intuition as road traffic affects the decisions made a lot more than the driver's experience since a road could have a traffic jam in an unexpected scenario.

During the program's execution messages are being printed explaining algorithmic decision-making in critical parts and saved in intermediate files in order to study the results after execution. Finally, a kml and a html file are produced so that it's possible for the final result to be shown in any application capable of handling kml files (or html if the user wishes to view the visualization on the web). Most common applications capable of handling kml files are Google Earth, Google Maps and Marble.

SUBJECT AREA: Resources Distribution, Transportation

KEYWORDS: vehicle routing, time windows, scheduling, transportation, guided local search

Η παρούσα πτυχιακή είναι αφιερωμένη στους γονείς μου. Θα ήθελα να τους ευχαριστήσω για την αγάπη που μου δείχνουν καθημερινά και για το ήθος που μου έχουν μεταδώσει. Χωρίς αυτούς απλά δεν θα ήμουν αυτός που είμαι.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα κ. Αλέξη Δελή που μου έδωσε την ευκαιρία να ασχοληθώ με αυτό το ενδιαφέρον πρόβλημα και για την καθοδήγηση που μου πρόσφερε καθ'όλη τη διάρκεια εκπόνησης αυτής της πτυχιακής.

Επίσης, Θα ήθελα να ευχαριστήσω τον επιβλέποντα κ. Παναγιώτη Λιάκο για τη συνεργασία και τη βοήθεια που μου προσέφερε. Η συνεισφορά του ήταν κρίσιμη για την περαίωση του έργου.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για τις πολύτιμες παρατηρήσεις και προτάσεις τους στις αρχικές εκδόσεις της εφαρμογής και του παρόντος κειμένου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΚΩΔΙΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή αποτελεί μέρος των υποχρεώσεων για την λήψη πτυχίου στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Αντικείμενό της είναι η επέκταση ενός κλασσικού τρόπου επίλυσης του CVRPTW έτσι ώστε να μπορούν να χρησιμοποιηθούν σε μια πραγματική εφαρμογή.

Η εκπόνησή της ήταν μια ενδιαφέρουσα εμπειρία, καθώς μέσα από αυτή τη διαδικασία έμαθα πολλά νέα πράγματα που δεν γνώριζα και επέκτεινα τις γνώσεις μου σε διάφορα πεδία.

Η πρόκληση αντιμετώπισης ενός προβλήματος αυτής της δυσκολίας αποτέλεσε κίνητρο για να προσπαθήσω να φτιάξω μια εφαρμογή που ελπίζω να είναι απλή και κατανοητή. Πολλοί άνθρωποι ενδεχομένως να αποθαρρύνονταν να ασχοληθούν με το πρόβλημα λόγω της δυσκολίας του. Ελπίζω πως η τεκμηρίωση της εφαρμογής είναι αρκετά λεπτομερής και επεξηγηματική για να πείσει και τον πιο δύσπιστο άνθρωπο ότι αξίζει να το μελετήσει.

Αναμφίβολα, μελετάται επί δεκαετίες από την ακαδημαϊκή κοινότητα, αλλά είναι αναγκαίο τα αποτελέσματα τόσης έρευνας να έχουν και κάποιο αντίκρυσμα στην κοινωνία.

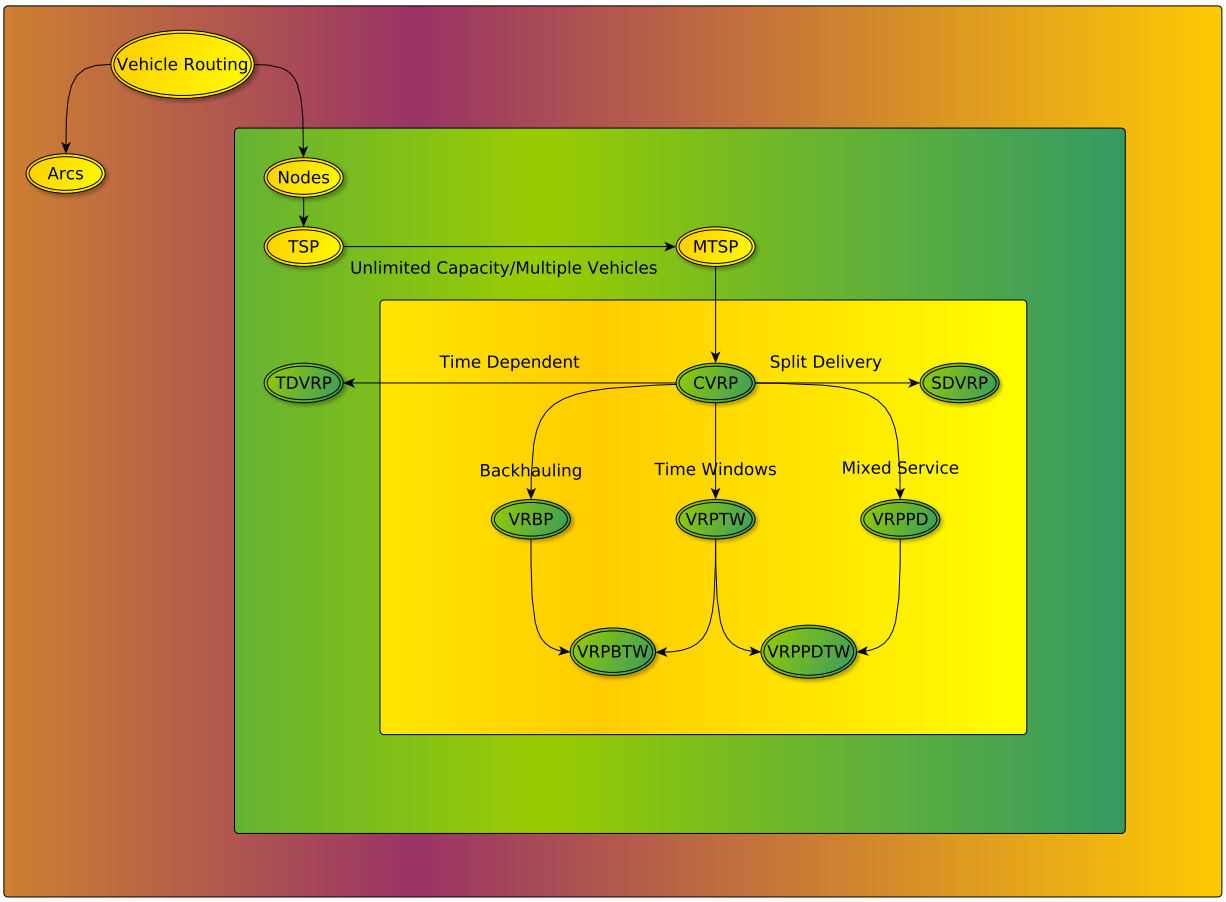
1. ΕΙΣΑΓΩΓΗ

Το Πρόβλημα του Περιοδεύοντος Πωλητή (Travelling Salesman Problem, TSP) είναι ένα από τα πιο γνωστά προβλήματα της Αλγοριθμικής Επιχειρησιακής Έρευνας. Σύμφωνα με αυτό, ο πωλητής πρέπει να επισκεφτεί κάποιους πελάτες ακριβώς μια φορά και να επιστρέψει στην αρχική του θέση, διανύοντας την ελάχιστη δυνατή απόσταση. Παρότι το πρόβλημα διατυπώνεται πολύ εύκολα, αποτελεί μια μεγάλη πρόκληση για τους ακαδημαϊκούς καθώς είναι ένα NP-Δύσκολο πρόβλημα (NP-Hard).

Στο Πρόβλημα Δρομολόγησης Οχημάτων (Vehicle Routing Problem, VRP) [1, 2], έχουμε ένα αρχηγείο (depot) και μια αρμάδα οχημάτων (vehicle fleet). Σκοπός των οχημάτων είναι να επισκεφτούν ένα σύνολο πελατών ακριβώς μια φορά με τέτοιο τρόπο ώστε να ελαχιστοποιείται το πλήθος των οχημάτων αλλά και η συνολική απόσταση που πρέπει να διανυθεί. Είναι προφανές ότι πρόκειται για μια γενίκευση του TSP, αφού προσφέρει περισσότερες ελευθερίες και μεγαλύτερη πολυπλοκότητα, αποσπώντας έτσι και την προσοχή των ακαδημαϊκών.

Πρόκειται για ένα ενδιαφέρον πρόβλημα που έχει πραγματικές εφαρμογές και μπορεί να το συναντήσει κανείς σε πολλές παραλλαγές [3, 4] (Σχήμα 1), όπως:

- Vehicle Routing Problem with Pickup and Delivery (VRPPD): οι πόροι που μεταφέρουν τα φορτηγά πρέπει να μετακινούνται μεταξύ σημείων παραδόσεων.
- Vehicle Routing Problem with LIFO: ίδιο με το προηγούμενο με τον επιπλέον περιορισμό ο πόρος που μεταφέρεται να είναι ο πιο πρόσφατος που φορτώθηκε. Η ιδέα αυτού του περιορισμού είναι η μείωση του χρόνου φόρτωσης-εκφόρτωσης πόρων στα φορτηγά.
- Vehicle Routing Problem with Time Windows: οι πελάτες που εξυπηρετούνται από τα οχήματα δίνουν ένα χρονικό περιθώριο μέσα στο οποίο πρέπει να εξυπηρετηθούν, όχι νωρίτερα και όχι αργότερα.
- Capacitated Vehicle Routing Problem: CVRP or CVRPTW: Τα οχήματα έχουν επιπλέον περιορισμένη χωρητικότητα πόρων που μπορούν να μεταφέρουν.
- Vehicle Routing Problem with Multiple Trips (VRPMT): Τα οχήματα επιτρέπεται να κάνουν πάνω από μια διαδρομές.
- Open Vehicle Routing Problem (OVRP): Τα οχήματα δεν απαιτείται να επιστρέψουν στο αρχηγείο.



Σχήμα 1: Η σχέση μεταξύ συνηθισμένων παραλλαγών του VRP

Στην παρούσα πτυχιακή θα ασχοληθούμε με το CVRPTW [5]. Επειδή το πρόβλημα έχει πολλαπλούς στόχους και περιορισμούς, πρέπει να ικανοποιούνται τα παρακάτω:

- Τα φορτηγά έχουν περιορισμένη χωρητικότητα.
- Οι πελάτες έχουν συγκεκριμένα χρονικά όρια μέσα στα οποία πρέπει να εξυπηρετηθούν.
- Τα φορτηγά πρέπει να έχουν επιστρέψει όλα στο αρχηγείο μέχρι το τέλος της λειτουργίας του.
- Κάθε πελάτης εξυπηρετείται ακριβώς μια φορά και κάθε φορτηγό κάνει το πολύ μια διαδρομή.
- Η συνολική απόσταση που θα διανυθεί από τα φορτηγά πρέπει να γίνει όσο το δυνατόν μικρότερη.
- Το πλήθος των φορτηγών που θα εξυπηρετήσουν τους πελάτες πρέπει να είναι όσο το δυνατόν μικρότερο.

Επειδή το πρόβλημα είναι τόσο δύσκολο, αν αυξήσουμε πολύ τον χώρο αναζήτησης για μια λύση, το πρόβλημα μπορεί να μην λύνεται σε αποδεκτό χρόνο, έτσι συνήθως καταφεύγουμε σε αλγορίθμους που λύνουν προσεγγιστικά το πρόβλημα. Αυτοί πάλι έχουν κατηγορίες ανάλογα με το πώς ιεραρχούν τους στόχους που πρέπει να ικανοποιηθούν. Αυτό

σημαίνει ότι κάποιος αλγόριθμος μπορεί να λύσει το πρόβλημα βρίσκοντας την ελάχιστη διαδρομή βάζοντας όμως πολλά φορτηγά, ενώ ένας άλλος να ελαχιστοποιεί το πλήθος των φορτηγών θυσιάζοντας την βέλτιστα ελάχιστη διαδρομή.

Είναι γνωστό ότι υπάρχουν πολλοί αλγόριθμοι και υλοποιήσεις (δωρεάν ή επαγγελματικές) που αντιμετωπίζουν αποτελεσματικά το πρόβλημα. Στην παρούσα πτυχιακή γίνεται μια προσπάθεια η λύση που βρίσκεται να ανταποκρίνεται στον πραγματικό κόσμο και συνάμα να μπορεί να λάβει υπόψην πραγματικά δεδομένα κίνησης δρόμων και εμπειρία οδηγών που κάνουν δρομολόγια. Στόχος είναι να δίνεται μια πλήρης λύση στο πρόβλημα δρομολόγησης οχημάτων που μπορεί να διαφέρει ακόμα και για τα ίδια δεδομένα αν έχουμε νέες πληροφορίες κίνησης στους δρόμους.

Στο Κεφάλαιο 2 γίνεται αναφορά σε βασική θεωρία που είναι απαραίτητη για να καταλάβει κανείς κάποια βασικά σημεία των αλγορίθμων που χρησιμοποιούνται.

Στο Κεφάλαιο 3 γίνεται πλήρης ανάλυση των αλγορίθμων με επεξήγηση τεχνικών λεπτομερειών. Γίνεται εκτενής αναφορά στην συνολική δομή του προγράμματος και την αλληλεξάρτηση των κλάσεών (classes) του, καθώς επίσης και στα εργαλεία που είναι απαραίτητα για να μπορεί να εκτελεστεί σωστά. Στο κεφάλαιο περιλαμβάνονται οδηγίες και δείγματα αρχείων εισόδου-εξόδου και εικόνες που δείχνουν την οπτικοποίηση του τελικού αποτελέσματος.

Στο Κεφάλαιο 4 παρατίθενται πειραματικά αποτελέσματα που συγκρίνεται η επίδοση του προγράμματος με γνωστές μετρήσεις της βιβλιογραφίας.

Στο Κεφάλαιο 5 περιγράφονται κάποιες σχεδιαστικές επιλογές που πάρθηκαν κατά την εκπόνηση της πτυχιακής, δίνονται παραδείγματα που αναδεικνύουν την αξία του προγράμματος και περιγράφεται η προσέγγιση που ακολουθήθηκε.

Στο Παράρτημα υπάρχει παράθεση της συνάρτησης που υπολογίζει την απόσταση δύο σημείων στη Γη (Κώδικας 6) και κάποιων ενδιαφερόντων PostgreSQL ερωτημάτων (Κώδικας 7-11) που χρησιμοποιούνται στο πρόγραμμα μαζί με λίγα σχόλια για το καθένα.

2. ΒΑΣΙΚΗ ΘΕΩΡΙΑ

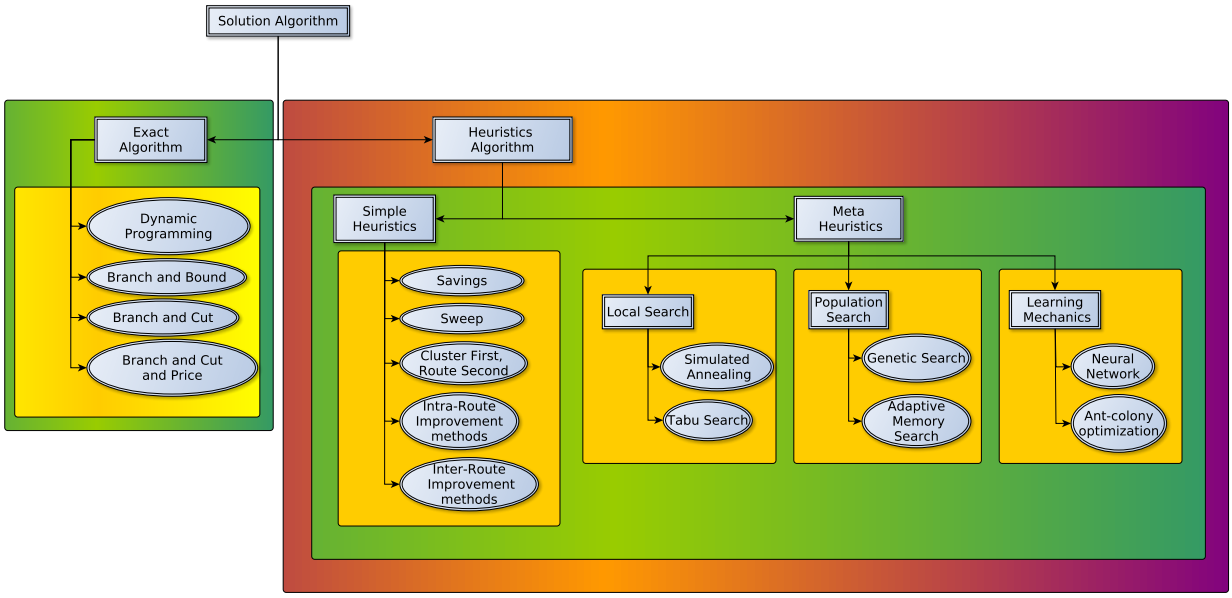
Σε αυτό το κεφάλαιο γίνεται αναφορά σε βασική θεωρία που είναι απαραίτητη για να είναι κατανοητό το υπόλοιπο κείμενο. Πιο συγκεκριμένα, εξηγούνται οι ευρετικές (heuristics), οι μετα-ευρετικές (meta-heuristics) και ο αλγόριθμος A*.

2.1 Ευρετικές και Μετα-ευρετικές

Πολλά προβλήματα στον χώρο της Πληροφορικής μπορούν να λυθούν ακριβώς σε αποδεκτό χρόνο μόνο για μικρό πλήθος δεδομένων. Αν θέλουμε να έχουμε κάποια λύση για περισσότερα δεδομένα καταφεύγουμε σε προσεγγιστικές λύσεις μέσα από την χρήση ευρετικών (Σχήμα 2). Μια ευρετική προσέγγιση είναι κάποιος αλγόριθμος ή μια απλή ιδέα που διαισθητικά και πειραματικά περιμένουμε ότι θα δώσει μια “καλή” λύση που θα προσεγγίζει αρκετά την βέλτιστη (που δεν μπορούμε να περιμένουμε για να αποκτήσουμε). Όμως τι σημαίνει “καλή”;

Ανάλογα το πρόβλημα, μπορεί να μην μας ενδιαφέρει καν η βέλτιστη λύση, αλλά οποιαδήποτε λύση να αρκεί. Σε άλλες περιπτώσεις μπορεί να παίρνουμε μια λύση που να είναι διαισθητικά καλή αλλά να απέχει πάρα πολύ από τη βέλτιστη.

Αρκετά συχνά όμως, η χρήση ευρετικών δεν επαρκεί για μια αποδεκτή λύση. Τότε καταφεύγουμε στην χρήση μετα-ευρετικών. Οι μετα-ευρετικές λειτουργούν περισσότερο σαν μαύρα κουτιά και ξεκινώντας από κάποια αρχική εφικτή (feasible) λύση, κάνουν γρήγορη και έξυπνη αναζήτηση των λύσεων προσπαθώντας να βελτιώσουν την αρχική λύση για να πλησιάσει όσο περισσότερο γίνεται τη βέλτιστη.



Σχήμα 2: Κατηγορίες αλγορίθμων επίλυσης του VRP και των παραλλαγών του

2.1.1 Ευρετική Εισαγωγής με Απευθείας Προώθηση

Η Ευρετική Εισαγωγής με Απευθείας Προώθηση (Push-Forward Insertion Heuristic, PFIH) είναι μια από τις πιο παλιές ευρετικές που υπάρχουν για το πρόβλημα και προτάθηκε από τον Solomon [6] το 1987. Πρόκειται για μια ευρετική που ακολουθεί την ανθρώπινη διαίσθηση και μπορεί να δώσει γρήγορα μια εφικτή λύση. Ωστόσο, ανάλογα με τη φύση των δεδομένων, η λύση αυτή μπορεί να απέχει από ελάχιστα έως και πάρα πολύ από την βέλτιστη λύση. Κατά μέσο όρο, έχει καλή απόδοση, αλλά σπάνια προτιμάται η χρήση της χωρίς κάποια άλλη ευρετική ή μετα-ευρετική.

Διαισθητικά, ο αλγόριθμος (Κώδικας 1) κρατά μια λίστα με όλους τους πελάτες που δεν έχει αναλάβει κανένα όχημα. Από αυτούς επιλέγει τον πιο κοντινό και τον βάζει σε μια νέα διαδρομή. Ελέγχοντας ότι ικανοποιούνται όλοι οι περιορισμοί του προβλήματος, τον τοποθετεί μέσα στη διαδρομή στη θέση με το μικρότερο κόστος (στην περίπτωσή μας το κόστος είναι η συνολική απόσταση που πρέπει να διανυθεί). Αν η διαδρομή δεν χωράει άλλους πελάτες, ο επόμενος προστίθεται σε νέα διαδρομή και η διαδικασία επαναλαμβάνεται μέχρι όλοι οι πελάτες να έχουν πάει σε κάποιο όχημα.

Είναι φανερό ότι αυτή η προσέγγιση θα δώσει αναγκαστικά εφικτή λύση, αλλά είναι εύκολο να απέχει πολύ από τη βέλτιστη. Βέβαια, ένα επιπλέον καλό που κερδίζουμε από αυτή την ευρετική είναι και ένα άνω φράγμα του πλήθους των φορτηγών. Ξέρουμε ότι αν με αυτή την ευρετική θέλουμε N φορτηγά για να λύσουμε το πρόβλημα και η βέλτιστη λύση θα έχει το πολύ τόσα. Αυτό δίνει μια ελευθερία για ενσωμάτωση μετα-ευρετικών και βελτίωση της αρχικής λύσης περιορίζοντας πολύ τον χώρο αναζήτησης.

Κώδικας 1: Ο αλγόριθμος PFIH

```
unserviced_customers:=customers;
serviced_customers:={};
fleet feasible_solution:={};
feasible_solution:=feasible_solution U new_route;
current_route:=new_route;

while unserviced!={}{
   if last candidate added to vehicle satisfies constraints{
      candidates:={};
      for unserviced_customer in unserviced{
         for every valid_position in current_route{
            candidates:=candidates U candidate;
         }
      }
      if candidates=={}{
         //the latest addition remains in the old route
         feasible_solution:=feasible_solution U new_route;
      }
```

```
      else{
         best_candidate:=candidates.getFirst();
         for candidate in candidates{
            if candidate.isBetterFrom(best_candidate){
               best_candidate=candidate;
            }
         }
         current_route:=current_route U best_candidate;
      }
   }
   else{
      generate a new route and add the best candidate there;
   }
   current_route := last route in fleet;
}
return fleet;
```

2.1.2 Μετα-ευρετική Τοπικής Κατευθυνόμενης Αναζήτησης

Η μετα-ευρετική Τοπικής Κατευθυνόμενης Αναζήτησης (Guided Local Search, GLS) είναι μια μέθοδος που εφαρμόζεται πάνω σε κάποιο αλγόριθμο αναζήτησης για να κατευθύνει μια λύση έτσι ώστε να πλησιάσουμε σε τοπικό ελάχιστο σε σχέση με την αρχική λύση. Είναι φανερό ότι ανάλογα με την δομή του χώρου λύσεων του προβλήματος και την θέση της αρχικής λύσης, το ελάχιστο μπορεί να απέχει λιγότερο ή περισσότερο από την βέλτιστη λύση.

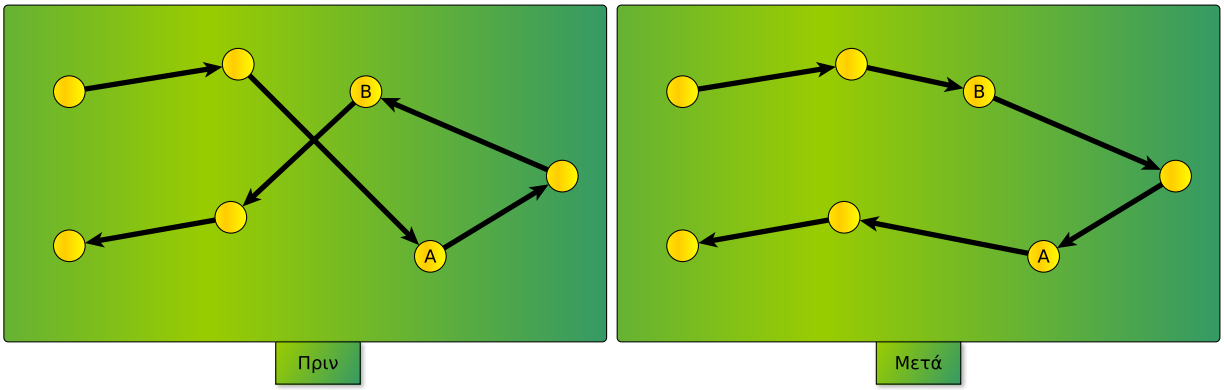
Η βελτίωση της υπάρχουσας λύσης γίνεται με την χρήση κάποιων τελεστών που αλλάζουν τις διαδρομές και εφαρμόζονται μόνο εφόσον ικανοποιούνται οι περιορισμοί του προβλήματος. Οι τελεστές που θα αναφερθούμε στην παρούσα πτυχιακή είναι οι ακόλουθοι:

- 2-βελτιστοποίηση με αλλαγή φοράς. Η 2-βελτιστοποίηση με αλλαγή φοράς (2-opt swap [7]) είναι ένας τελεστής που εφαρμόζεται εσωτερικά σε μια διαδρομή (intra-route operator). Επιλέγει δύο πελάτες της διαδρομής και το κομμάτι της διαδρομής που βρίσκεται ανάμεσά τους αλλάζει φορά (Σχήμα 3, Κώδικας 2). Έτσι αλλάζει η σειρά εξυπηρέτησης με την ελπίδα ότι το κόστος λόγω αποστάσεων θα μειωθεί.

Κώδικας 2: Ο αλγόριθμος 2-opt

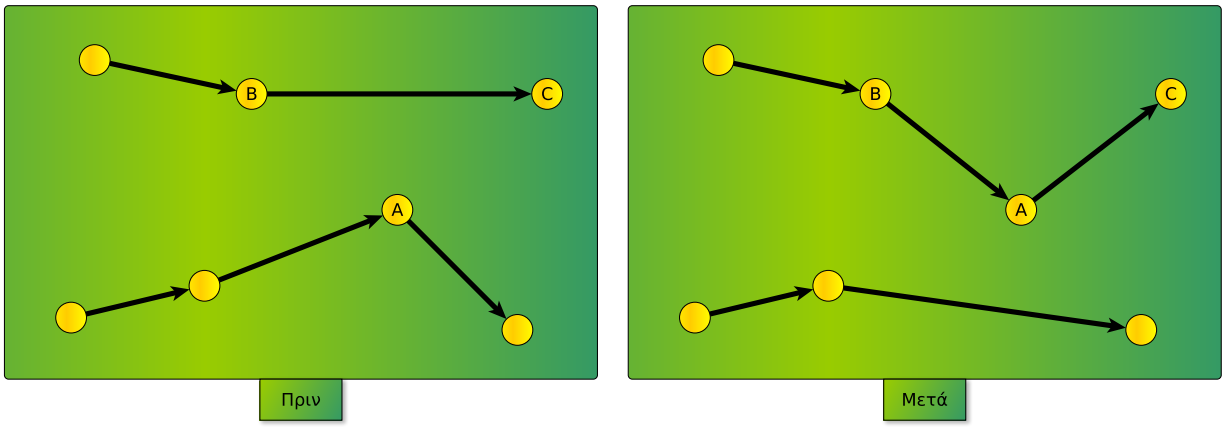
```
customers:=fleet.getVehicle(route);
i=customer1.getPos(customers);
j=customer2.getPos(customers);
vehicle changed_route=new_route;

for(int c=0;c<i;c++){
      changed_route.addCustomer(vehicle.getCustomers().get(c));
}
for(int c=j;c>=i;c--){
      changed_route.addCustomer(vehicle.getCustomers().get(c));
}
for(int c=j+1;c<customers.size();c++){
      changed_route.addCustomer(vehicle.getCustomers().get(c));
}
return changed_route;
```



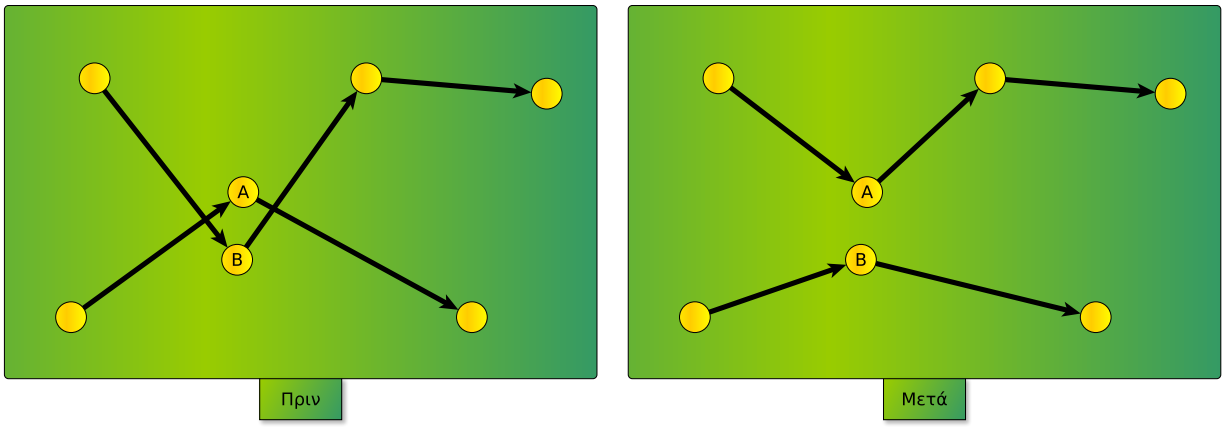
Σχήμα 3: Ο τελεστής 2-opt

- 2-βελτιστοποίηση. Η 2-βελτιστοποίηση είναι μια επέκταση του προηγούμενου τελεστή όταν περιμένουμε πιο αισιόδοξη βελτίωση. Ουσιαστικά είναι η εφαρμογή της 2-βελτιστοποίησης με αλλαγή φοράς συνεχόμενα όσο μπορεί να υπάρχουν επανειλημμένες βελτιώσεις και κρατάμε την καλύτερη βελτίωση που μπορέσαμε να πετύχουμε. Η εφαρμογή του τελεστή θεωρείται επιτυχής όταν υπάρχει τουλάχιστον μια βελτίωση.
- επανατοποθέτηση. Η επανατοποθέτηση (relocate) είναι ένας τελεστής που εφαρμόζεται μεταξύ διαδρομών (inter-route operator). Δοκιμάζει για κάθε πελάτη μιας διαδρομής να τον μετακινήσει σε μια άλλη διαδρομή και από όλες τις δυνατές θέσεις επιλέγεται αυτή που έχει το λιγότερο κόστος (Σχήμα 4).



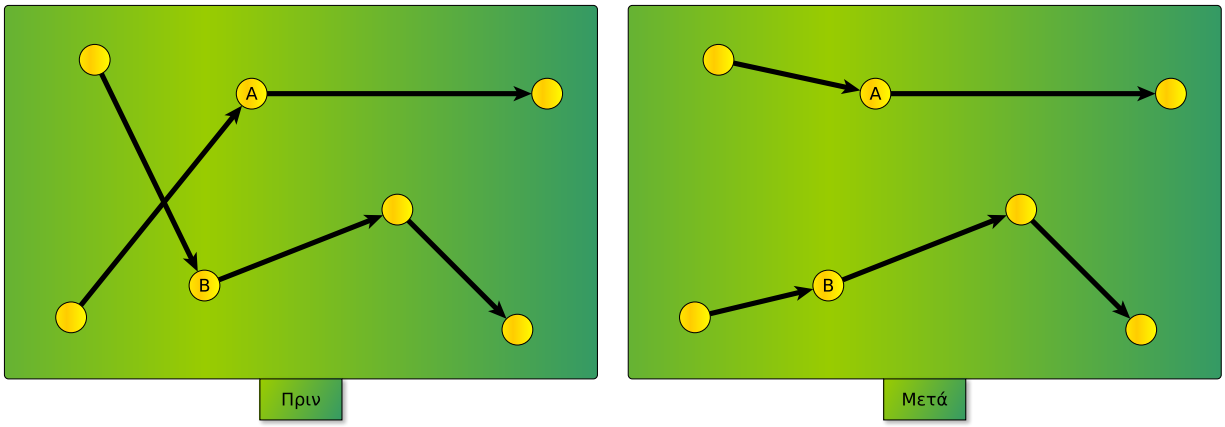
Σχήμα 4: Ο τελεστής relocate

- επανατοποθέτηση σε νέα διαδρομή. Η επανατοποθέτηση μπορεί να αποτύχει αν και οι δυο διαδρομές που εξετάζονται δεν μπορούν να έχουν και άλλον πελάτη λόγω περιορισμών. Σε αυτή την περίπτωση, εφαρμόζεται η επανατοποθέτηση σε νέα διαδρομή. Ουσιαστικά, από τους πελάτες της υπάρχουσας διαδρομής, δοκιμάζονται ένας-ένας να εισάγονται σε μια εξολοκλήρου νέα διαδρομή και κρατάμε τη λύση με το λιγότερο κόστος. Έτσι αυξάνουμε το πλήθος των οχημάτων κατά ένα, ενώ ήδη είχαμε λύση με λιγότερα οχήματα. Παρά ταύτα, η εφαρμογή άλλων τελεστών που αναλύονται αμέσως μετά σε συνδυασμό με αυτόν διασφαλίζουν ότι καθώς συγκλίνουμε σε τοπικό ελάχιστο, ο αριθμός των οχημάτων θα επανέλθει (εκτός κι αν η διαφορά κόστους είναι πολύ μεγάλη).
- ανταλλαγή. Η ανταλλαγή (exchange) είναι τελεστής που εφαρμόζεται μεταξύ δυο διαδρομών. Από τις δυο διαδρομές επιλέγεται ένας πελάτης από την πρώτη και ένας από την δεύτερη έτσι ώστε η μεταξύ τους απόσταση να είναι η μικρότερη από όλα τα υποψήφια ζευγάρια και οι διαδρομές ανταλλάσσουν μεταξύ τους αυτούς τους πελάτες (Σχήμα 5).



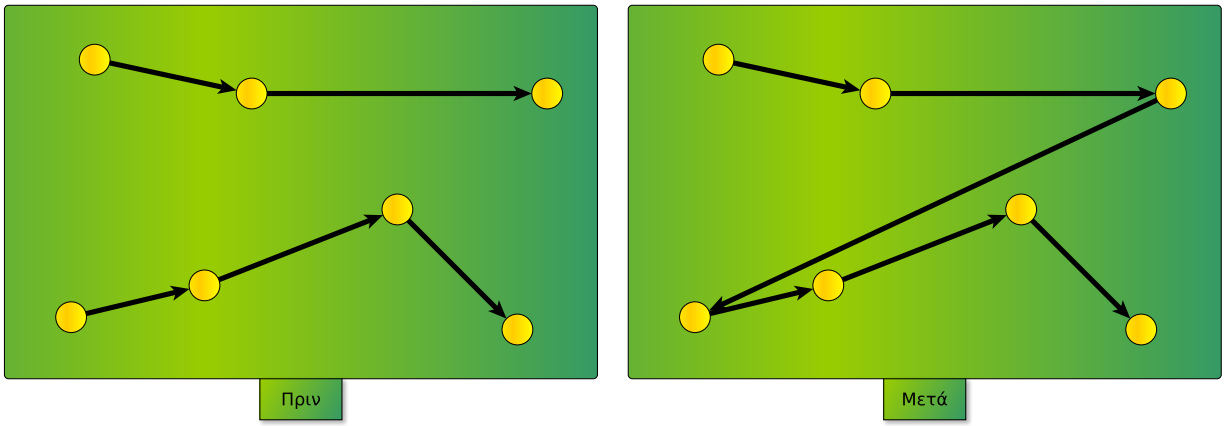
Σχήμα 5: Ο τελεστής exchange

- διασταύρωση. Ο τελεστής διασταύρωσης (cross operator) εφαρμόζεται μεταξύ δυο διαδρομών. Ουσιαστικά, αν έχουμε δυο διαδρομές A και B, δοκιμάζει να δίνει το τέλος της A στην B και της B στην A. Σαν τέλος ορίζεται οποιοδήποτε υποσύνολο της διαδρομής με τους πελάτες στην σειρά που βρίσκονται (Σχήμα 6). Από όλες τις υποψήφιες αλλαγές διατηρούμε αυτή με το μικρότερο κόστος.



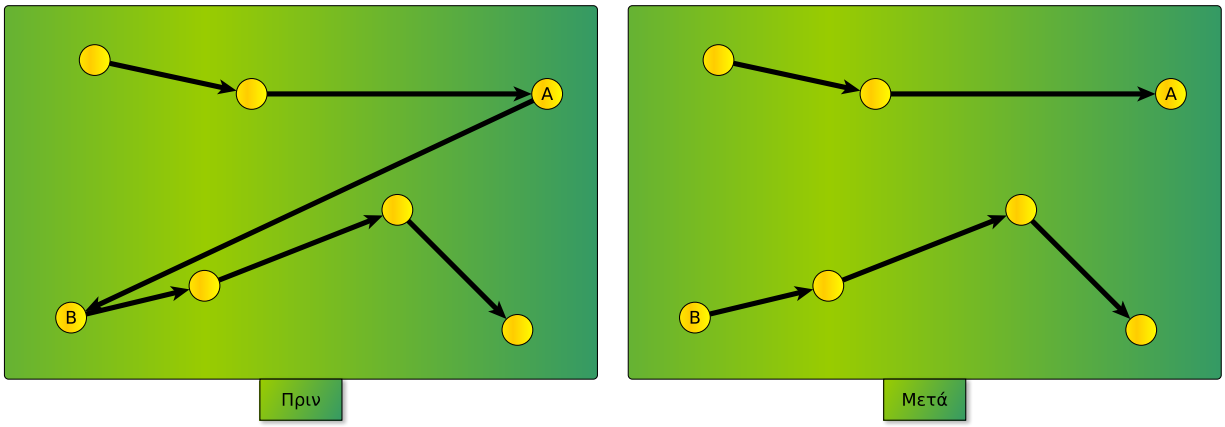
Σχήμα 6: Ο τελεστής cross

- διασταύρωση-προσάρτηση στο τέλος. Ο τελεστής διασταύρωσης-προσάρτησης στο τέλος (cross-append operator) εφαρμόζεται μεταξύ δυο διαδρομών. Ουσιαστικά δοκιμάζει να ενσωματώσει εξολοκλήρου μια διαδρομή στο τέλος μιας άλλης, μειώνοντας τον αριθμό των οχημάτων (Σχήμα 7).



Σχήμα 7: Ο τελεστής cross-append

- διασταύρωση-διαχωρισμός. Ο τελεστής διασταύρωσης-διαχωρισμού (cross-split operator) εφαρμόζεται μεταξύ δυο διαδρομών. Ουσιαστικά κάνει το αντίστροφο από την διασταύρωση-ένωση στο τέλος, δηλαδή έχοντας μια δεδομένη διαδρομή, την σπάει στα δύο παράγοντας έτσι ένα επιπλέον όχημα με κάποιους πελάτες (Σχήμα 8).



Σχήμα 8: Ο τελεστής cross-split

Έχοντας εξηγήσει την λειτουργία των τελεστών, μπορούμε να εξηγήσουμε περισσότερο τον αλγόριθμο GLS (Κώδικας 3). Προκειμένου να γίνεται αξιολόγηση των λύσεων, ο GLS κρατάει ένα διάνυσμα ποινών για κάθε κομμάτι της λύσης (στην περίπτωσή μας ακμές που ενώνουν πελάτες). Κάθε φορά που εφαρμόζεται κάποιος τελεστής, υπολογίζεται το κόστος της νέας λύσης και επιβάλλονται ποινές στα στοιχεία που συνθέτουν την λύση.

Η ποινή κάθε ακμής υπολογίζεται από τον τύπο $c_i/(p_i+1)$, ενώ σε κάθε τοπική αναζήτηση που πραγματοποιείται, επιπλέον πόντο ποινής έχουν μόνο οι ακμές με τη μεγαλύτερη τέτοια τιμή. Ο λόγος είναι ότι έτσι στοιχεία που κοστίζουν (μεγάλο μήκος ακμής) δέχονται γρήγορα ποινές, αλλά αν εμφανίζονται πολλές φορές στη λύση (δηλαδή έχουν δεχτεί πολλαπλές ποινές), ο παρανομαστής του κλάσματος αυξάνεται και έτσι το κόστος μειώνεται, δείχνοντας διαισθητικά ότι αυτή η ακμή μάλλον είναι αναπόφευκτο μέρος της λύσης. Έτσι, ο αλγόριθμος γίνεται πιο ελαστικός για κάποια “απαραίτητα” στοιχεία της λύσης όσο προχωράει.

Οι ποινές όλων των ακμών συνεισφέρουν στην αντικειμενική συνάρτηση του GLS, η οποία υπολογίζει το συνολικό κόστος μιας λύσης. Η συνάρτηση κόστους του GLS είναι μια επαυξημένη αντικειμενική συνάρτηση (augmented objective function) που ορίζεται σαν $O'(S) = O(S) + \lambda \sum_{i \in F} f_i(S) p_i c_i$. $O(S)$ είναι το συνολικό κόστος των διαδρομών χωρίς ποινές. Ο παράγοντας με το άθροισμα συνολικά δηλώνει το επιπλέον κόστος από στοιχεία της λύσης που θέλουμε να αποφύγουμε και τους επιβάλλουμε ποινές (για εμάς αυτά είναι μεγάλες ακμές) και το λ είναι μια παράμετρος που επιλέγουμε εμείς και έχει συνήθως μικρή τιμή (η βιβλιογραφία προτείνει την τιμή 0.2) και δηλώνει πόσο σημαντικές είναι οι ποινές για την λύση μας.

Κώδικας 3: Ο αλγόριθμος GLS

```
penalty_vector:=[0,..,0]^T;
S:=Initial_Solution();
S*:=LocalSearch(S);
while !stoppingCondition(){
   f:=choosePenaltyFeatures(S,penalty_vector);
   for x in f{
      p[x]:=p[x]+1;
      S:=LocalSearch(S);
      if O(S)<O(S*){
         S*:=S;
      }
   }
}
return S*;
```

2.2 Αναζήτηση A*

Ο αλγόριθμος A* [8](Κώδικας 4) είναι ένας από τους πιο παλιούς αλγόριθμους αναζήτησης στον χώρο της Πληροφορικής και έχει γίνει ιδιαίτερα δημοφιλής σε εφαρμογές Τεχνητής Νοημοσύνης. Προτάθηκε για πρώτη φορά το 1968 από τους Peter Hart, Nils Nilsson και Bertram Raphael του Ερευνητικού Ινστιτούτου του Στάνφορντ (Stanford Research Institute, SRI). Πρόκειται για μια γενική περίπτωση του αλγορίθμου του Dijkstra για αναζήτηση γράφων. Αυτό που του δίνει το επιπλέον πλεονέκτημα είναι η χρήση μιας ευρετικής για να εκτιμά πόσο κοντά βρίσκεται από αυτό που ψάχνει, οπότε καταφέρνει να μειώνει τον χώρο αναζήτησης. Η απουσία ευρετικής στον A* τον κάνει ακριβώς ισοδύναμο με τον Dijkstra. Λόγω της γενικότητας του αλγορίθμου, μπορεί να χρησιμοποιηθεί για μια σειρά από σκοπούς.

Η επιλογή ευρετικής μπορεί να είναι ένα πολύ δύσκολο έργο ανάλογα με την εφαρμογή. Στην περίπτωσή μας, η απόσταση της τρέχουσας θέσης από τον στόχο σε συντεταγμένες Γης επαρκεί. Ο A* ορίζει σαν συνάρτηση κόστους την $f(n) = g(n) + h(n)$, όπου $g(n)$ το βασικό κόστος της λύσης και $h(n)$ το κόστος που επιστρέφει η ευρετική συνάρτηση. Είναι κρίσιμο το κόστος της ευρετικής να είναι πάντοτε μικρότερο ή ισο του πραγματικού κόστους από τον στόχο για να εκτελεστεί σωστά ο αλγόριθμος.

Κώδικας 4: Ο αλγόριθμος A*

```
  function A*(start,goal)
   closedset := the empty set // The set of nodes already evaluated.
   openset := {start} // The set of tentative nodes to be evaluated, initially
       containing the start node
   came_from := the empty map // The map of navigated nodes.

   g_score := map with default value of Infinity
   g_score[start] := 0 // Cost from start along best known path.
   // Estimated total cost from start to goal through y.
   f_score = map with default value of Infinity
   f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

   while openset is not empty
       current := the node in openset having the lowest f_score[] value
       if current = goal
           return reconstruct_path(came_from, goal)

       remove current from openset
       add current to closedset
       for each neighbor in neighbor_nodes(current)
           if neighbor in closedset
               continue

           tentative_g_score := g_score[current] + dist_between(current,neighbor)

           if neighbor not in openset or tentative_g_score < g_score[neighbor]
               came_from[neighbor] := current
               g_score[neighbor] := tentative_g_score
               f_score[neighbor] := g_score[neighbor] +
                   heuristic_cost_estimate(neighbor, goal)
               if neighbor not in openset
                   add neighbor to openset

   return failure

function reconstruct_path(came_from,current)
   total_path := [current]
   while current in came_from:
       current := came_from[current]
       total_path.append(current)
   return total_path
```

3. ΔΟΜΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Σε αυτό το κεφάλαιο θα αναλυθεί σε βάθος η δομή του προγράμματος (Σχήμα 9). Πιο συγκεκριμένα, θα εξηγήσουμε την προσέγγιση που ακολουθείται και θα περιγράψουμε συνοπτικά τη διαδικασία με την οποία γίνεται η εκτέλεση από την είσοδο του χρήστη μέχρι την εγγραφή των αποτελεσμάτων. Έπειτα, θα αναφερθούμε στα εργαλεία που είναι απαραίτητα για την εκτέλεση του προγράμματος και θα μιλήσουμε για τα δύο βασικά κομμάτια του προγράμματος, GLS και A*, την σύνδεσή τους και την επικοινωνία με PostgreSQL (μέσω JDBC Driver) και Redis.

3.1 Προσέγγιση

Όπως έχει ήδη αναλυθεί, θέλουμε να δρομολογήσουμε πελάτες με χρονικά όρια σε οχήματα προκειμένου να ικανοποιούνται οι περιορισμοί του προβλήματος. Αυτό είδαμε ότι μπορεί να γίνει λαμβάνοντας μια αρχική λύση εφαρμόζοντας τον αλγόριθμο PFIH. Η λύση αυτή μπορεί να βελτιωθεί με την χρήση τελεστών που αναφέρθηκαν παραπάνω και οι οποίοι βασίζονται στην ανθρώπινη διαίσθηση.

Ωστόσο, δεδομένου ότι η περιοχή αναζήτησης λύσεων μπορεί να είναι μεγάλη, δεν μπορούμε να εφαρμόζουμε με τυχαίο τρόπο τελεστές. Γι’ αυτόν τον λόγο, χρειαζόμαστε κάποιον αλγόριθμο που να αναζητά βελτίωση μιας λύσης σε σχέση με την τρέχουσα κατάσταση προκειμένου να φτάσουμε σε μια τοπικά βέλτιστη λύση (που όμως δεν είναι απαραίτητα και ολικά βέλτιστη). Αυτό επιτυγχάνεται με την εφαρμογή του αλγορίθμου GLS, ο οποίος κάνοντας χρήση ποινών προσπαθεί να βελτιώνει τη λύση που πήραμε από τον αλγόριθμο PFIH όσο περισσότερο γίνεται.

Μέχρι αυτό το σημείο, έχουμε μια προσέγγιση της πραγματικής λύσης, αφού στην πραγματικότητα το μόνο που έχουμε πει ως τώρα είναι πόσα φορτηγά θα χρειαστούμε, ποιους πελάτες και με ποια σειρά θα αναλάβει το καθένα και τους αναμενόμενους χρόνους άφιξης στους πελάτες έτσι ώστε η άφιξη να είναι εντός των χρονικών ορίων. Για να ολοκληρωθεί η εύρεση της λύσης, πρέπει να βρούμε το γρηγορότερο μονοπάτι ανάμεσα σε κάθε ζευγάρι πελατών και όλα μαζί να συνδυάσουν την τελική διαδρομή που πρέπει να ακολουθηθεί. Αυτό γίνεται με την βοήθεια του A*, ο οποίος θα βρίσκει γρήγορα και αποδοτικά την συντομότερη διαδρομή για κάθε ζευγάρι.

Φυσικά, έτσι θα βρούμε μια λύση που δεν λαμβάνει καθόλου υπόψην της δεδομένα κίνησης στους δρόμους. Αυτό είναι προβληματικό για την δομή μιας πραγματικής λύσης, αφού όπως γνωρίζουμε κάποιος δρόμος μπορεί να είναι σύντομος, αλλά να έχει πολλή κίνηση, καθιστώντας τον τελικά μη προτιμητέο. Προκειμένου να μεριμνήσουμε και για αυτό, ο A* χρησιμοποιεί βάρη στους δρόμους που εξετάζει επικοινωνώντας με μια βάση από την οποία λαμβάνει πληροφορίες κίνησης σε κάποιους δρόμους.

Βέβαια, επειδή στην πράξη είναι δυνατό να υπάρχουν ραγδαίες ή μη αναμενόμενες αλλαγές κίνησης στους δρόμους, λαμβάνουμε υπόψην και την εμπειρία των οδηγών ως ένα

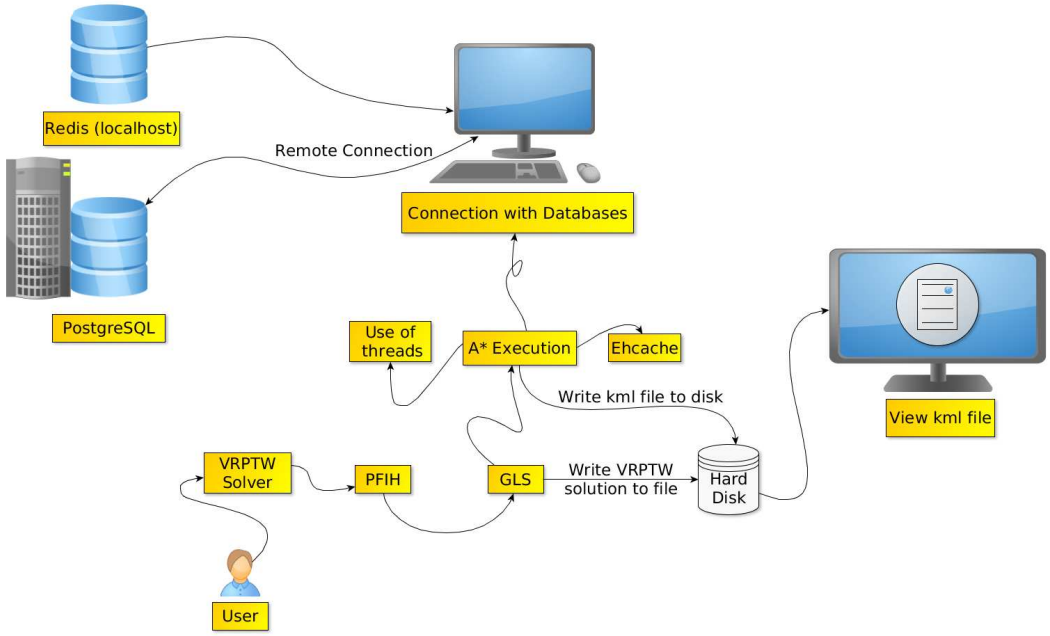
βαθμό, με πρόσθετα βάρη που ενσωματώνονται στον A*.

Δεδομένου ότι ο A* πρέπει να τρέξει για κάθε ζευγάρι, υπάρχουν πολλά δεδομένα που πρέπει να εξεταστούν και η επικοινωνία με μια βάση κοστίζει πολύ χρονικά. Αυτό το πρόβλημα επιλύεται με την χρήση νημάτων για παραλληλία εκτέλεσης και με την χρήση μιας κρυφής μνήμης που κρατάει τοπικά κάθε πληροφορία που ζητήθηκε μια φορά, δίνοντας δραματική επιτάχυνση στην εκτέλεση. Το τελικό αποτέλεσμα γράφεται σε αρχείο (σαν σύνολο σημείων από τα οποία ανακατασκευάζεται η διαδρομή οπτικά) που μπορεί να προβληθεί η τελική λύση από τον χρήστη.

3.2 Συνοπτική περιγραφή

Συνοπτικά, η διαδικασία που ακολουθείται είναι η εξής:

- Ο χρήστης τοποθετεί ένα αρχείο εισόδου με τα δεδομένα του CVRPTW στον φάκελο resources/testCases.
- Ο CVRPTW solver θα λύσει το κλασσικό πρόβλημα περνώντας από δύο στάδια: πρώτα από τον αλγόριθμο PFIH για να πάρουμε μια αρχική εφικτή λύση και μετά από τον αλγόριθμο GLS για να την βελτιώσουμε όσο περισσότερο γίνεται. Τα αποτελέσματα του GLS γράφονται στον φάκελο target/classes/ στο αρχείο Output_[input_file] αν το αρχείο εισόδου του χρήστη ήταν το [input_file].
- Τα δεδομένα του CVRPTW solver περνάνε στο κομμάτι του A*. Συνολικά, το πρόγραμμα χρησιμοποιεί την ehcache και επικοινωνεί με μια PostgreSQL βάση μέσω JDBC Driver και τοπικά με μια Redis. Κάθε ζευγάρι πελατών αντιστοιχίζεται σε ένα νήμα (ενός συνόλου νημάτων) που εκτελεί τον A* για να βρεθεί η συντομότερη διαδρομή μεταξύ τους. Κάθε νήμα που τελειώνει γράφει σε ένα kml αρχείο (που επιλέγει ο χρήστης πώς θα ονομάζεται και που θα αποθηκευτεί) τα σημεία που συνθέτουν την διαδρομή. Όταν τελειώσουν όλα τα νήματα την εκτέλεσή τους και το αρχείο γραφτεί πλήρως, γράφεται αυτόματα ένα συνοδευτικό html αρχείο στο ίδιο μονοπάτι με το kml για περισσότερες επιλογές οπτικοποίησης. Το πρόγραμμα τερματίζει επιτυχώς και ο χρήστης μπορεί να δει το αποτέλεσμα στον αγαπημένο του φυλλομετρητή (web browser) ή πρόγραμμα που προβάλλει αρχεία kml (π.χ. Google Earth, Marble).



Σχήμα 9: Η δομή του προγράμματος

3.3 Απαραίτητα εργαλεία

Σε αυτή την ενότητα θα αναφέρουμε μερικά βασικά εργαλεία που είναι απαραίτητα για την ομαλή εκτέλεση του προγράμματος:

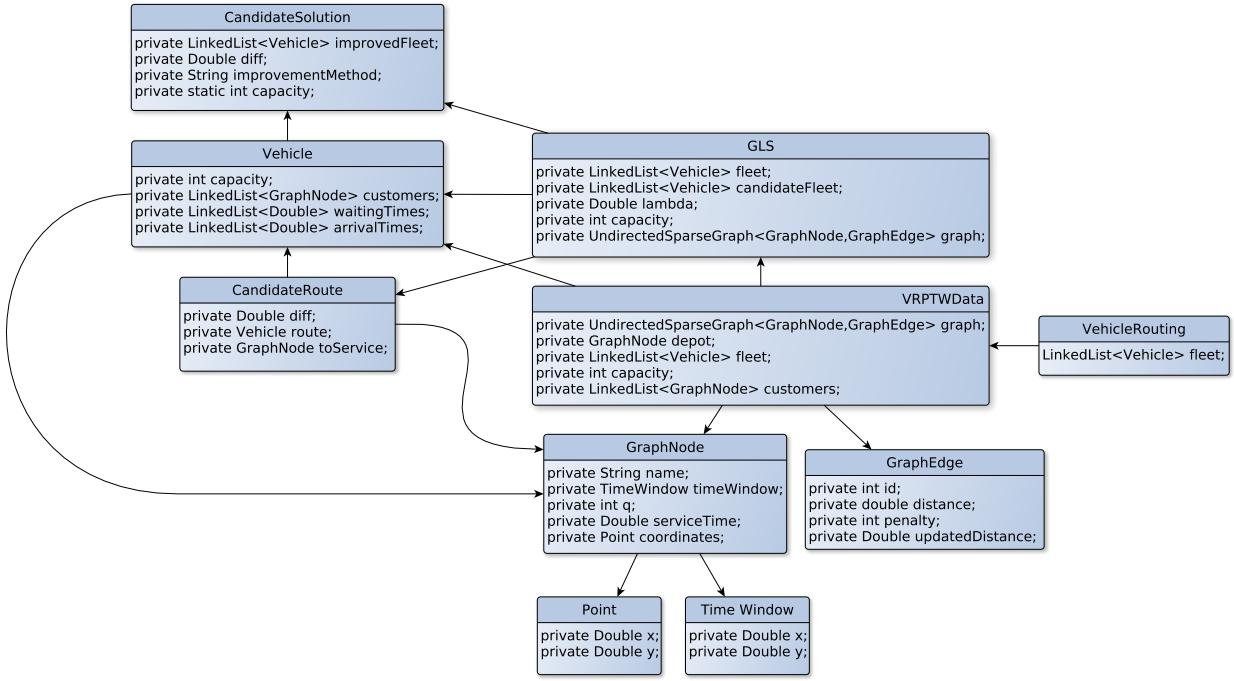
- Το πρόγραμμα γράφτηκε σε Java 7, οπότε για να τρέξει επιτυχώς χρειάζεται τουλάχιστον αυτή την έκδοση. Υπάρχει διαθέσιμο στο bitbucket [9], απ'όπου και μπορεί να μεταφορτωθεί.
- Μερικά μηνύματα κατά την εκτέλεση εκτυπώνονται σε τερματικό (terminal) οπότε είναι απαραίτητο το περιβάλλον που θα εκτελεστεί να έχει κάποιο τερματικό.
- Είναι απαραίτητη η επικοινωνία του προγράμματος με μια PostGIS βάση.
- Το Διαδίκτυο είναι απαραίτητο για να μπορεί να προβληθεί το html αρχείο με την οπτικοποίηση.
- Πριν εκτελεστεί το πρόγραμμα, πρέπει να έχει φορτωθεί η Redis βάση (ακόμα και αν είναι άδεια).
- Για να έχουμε επιτάχυνση του προγράμματος γίνεται χρήση της ehcache (κρυφή μνήμη) η οποία έχει ρυθμίσεις για το πώς χειρίζεται τη μνήμη στο αρχείο ehcache.xml.
- Η οπτικοποίηση του html αρχείου γίνεται με χρήση html+javascript και με τη βοήθεια του Google Maps API v3.0 [10].

3.4 Ο κώδικας του CVRPTW

Σε αυτή την ενότητα θα συζητήσουμε την εκτέλεση του CVRPTW και την πρώτη φάση του προγράμματος (Σχήμα 10).

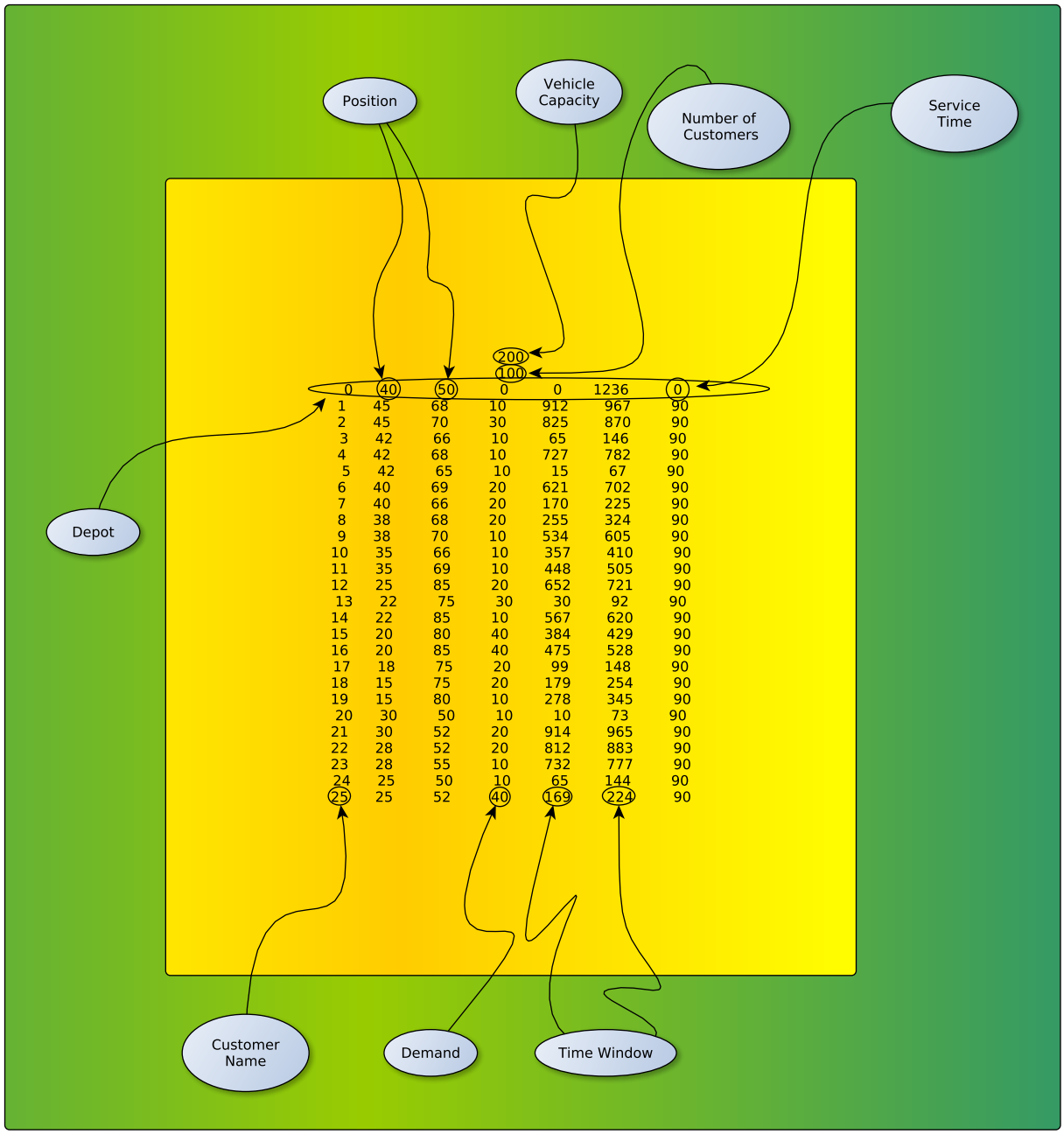
Πρώτα απ'όλα, ας εξηγήσουμε λίγο τι μοντελοποιεί η κάθε κλάση:

- Η κλάση Vehicle μοντελοποιεί το κάθε όχημα κρατώντας την χωρητικότητα που έχει διαθέσιμη και 3 λίστες που έχουν τους πελάτες του στη σειρά που θα εξυπηρετηθούν μαζί με τους χρόνους άφιξης και μετακίνησης στον επόμενο πελάτη.
- Η κλάση CandidateRoute μοντελοποιεί μια υποψήφια αλλαγή που γίνεται σε μια διαδρομή κρατώντας τη διαδρομή, την διαφορά του κόστους από την παλιά και τον πελάτη που προσπαθεί να αλλάξει.
- Η κλάση CandidateSolution μοντελοποιεί μια υποψήφια λύση που (ενδεχομένως) προκύπτει από κάποια αλλαγή στην αρχική κρατώντας την αρμάδα των οχημάτων, την αρχική τους χωρητικότητα, τη διαφορά κόστους από την αρχική λύση και το όνομα του τελεστή που εφαρμόστηκε για να πάρουμε την νέα υποψήφια λύση.
- Η κλάση GraphNode μοντελοποιεί τους πελάτες του προβλήματος κρατώντας το όνομα του πελάτη, το χρονικό όριο που πρέπει να εξυπηρετηθεί, την ποσότητα του προϊόντος που χρειάζεται, τον χρόνο που χρειάζεται να εξυπηρετηθεί και τις συντεταγμένες του.
- Η κλάση GraphEdge μοντελοποιεί τις ακμές του γράφου που συνδέει τους πελάτες ανά δύο κρατώντας έναν μοναδικό αναγνωριστικό κωδικό για κάθε ακμή, την αρχική απόσταση των δύο πελατών, τον αριθμό ποινών που έχει δεχτεί (κατά την εκτέλεση του GLS) και την ενημερωμένη απόσταση κατά την εκτέλεση του GLS που επηρεάζεται από το πλήθος των ποινών.
- Οι κλάσεις Point και TimeWindow μοντελοποιούν συντεταγμένες σημείου και τα όρια ενός χρονικού παραθύρου αντίστοιχα. Θα μπορούσαν να αποτελούν μια κλάση αλλά υπάρχουν χωριστά για λόγους αντικειμενοστράφειας.
- Η κλάση VRPTWData μοντελοποιεί τα δεδομένα του προβλήματος (και βρίσκει την αρχική λύση) κρατώντας τον γράφο που συνδέει τους πελάτες, το αρχηγείο, την αρμάδα των οχημάτων, την αρχική τους χωρητικότητα και τη λίστα με τους πελάτες που πρέπει να εξυπηρετηθούν.
- Η κλάση GLS μοντελοποιεί την εκτέλεση του αλγορίθμου GLS κρατώντας την τρέχουσα λύση, μια υποψήφια, την τιμή του λ, την αρχική χωρητικότητα των οχημάτων και τον γράφο που συνδέει τους πελάτες ανά δύο.
- Η κλάση Vehicle Routing μοντελοποιεί το ίδιο το πρόβλημα αφού επικοινωνεί με τις υπόλοιπες για να πάρει την τελική λύση. Επίσης έχει την ευθύνη να ανοίξει το αρχείο του χρήστη και να γράψει τα αποτελέσματα του GLS στο αρχείο εξόδου. Κρατάει την τελική λύση η οποία θα χρησιμοποιηθεί στον A*.



Σχήμα 10: Εξάρτηση των κλάσεων του κώδικα για το CVRPTW

Έχοντας κατεβάσει το έργο (project), ο χρήστης πρέπει να βάλει στον φάκελο resources/testCases ένα αρχείο με τα δεδομένα του προβλήματος (Σχήμα 11). Οι πρώτες δυο γραμμές πρέπει να περιέχουν δυο αριθμούς που δηλώνουν την χωρητικότητα των φορτηγών και το πλήθος των πελατών (μαζί με το αρχηγείο) και με αυτή τη σειρά. Στη συνέχεια πρέπει να υπάρχουν γραμμές με 7 πεδία, χωρισμένα με κενά. Αυτά θα είναι το όνομα του πελάτη, οι συντεταγμένες του, η ποσότητα προϊόντος που ζητάει, η αρχή και το τέλος του χρονικού ορίου που πρέπει να εξυπηρετηθεί και ο χρόνος που χρειάζεται για να εξυπηρετηθεί. Το χρονικό όριο του αρχηγείου δηλώνει και το χρονικό όριο μέσα στο οποίο πρέπει να έχουν γίνει όλες οι παραδόσεις με τα φορτηγά να βρίσκονται πίσω στο αρχηγείο. Τέλος, είναι σημαντικό το αρχηγείο να είναι ο πρώτος πελάτης στο αρχείο.



Σχήμα 11: Δείγμα αρχείου εισόδου

Μόλις το αρχείο διαβαστεί από το πρόγραμμα, θα κατασκευαστεί γράφος που συνδέει τους πελάτες ανά δύο και θα εφαρμοστεί ο αλγόριθμος PFIH για την κατασκευή αρχικής λύσης.

Η λύση που θα έχουμε από το PFIH θα αποτελέσει την αρχική λύση για τον αλγόριθμο GLS, ο οποίος για 2000 βήματα θα προσπαθήσει να την βελτιώσει. Αν για δύο διαδοχικά βήματα δεν μπορέσει να την βελτιώσει και έχει ακριβώς την ίδια λύση σταματά πριν τα 2000 για να μην σπαταλάται χρόνος. Η τιμή των 2000 βημάτων θεωρείται καλή στην βιβλιογραφία και μπορεί να αλλαχτεί από τον χρήστη στο σημείο ορισμού της.

Μόλις τελειώσει και το κομμάτι του GLS, το αποτέλεσμα της τελικής λύσης θα γραφτεί στον φάκελο target/classes στο αρχείο Output_[input_file] αν το αρχείο εισόδου του χρήστη

ήταν το [input_file].

3.5 Ο κώδικας του A*

Μέχρι το σημείο που τελειώνει ο GLS, δεν έχουμε καμία διαφοροποίηση σε σχέση με τις κλασσικές προσεγγίσεις. Στην περίπτωσή μας, ο A* χρησιμοποιείται για να ξεφύγουμε από το θεωρητικό μοντέλο που προσεγγίζουν οι κλασσικοί αλγόριθμοι του CVRPTW. Προκειμένου να έχουμε μια πραγματική εφαρμογή στα χέρια μας, μπορούμε να δώσουμε πραγματικές συντεταγμένες και τοποθεσίες από κάποιον χάρτη. Έχοντας καθορίσει την σειρά που θα εξυπηρετηθούν οι πελάτες, πρέπει να βρούμε την συντομότερη διαδρομή που τους συνδέει ανά δύο έτσι ώστε τα φορτηγά να μειώσουν όσο περισσότερο γίνεται την διαδρομή που θα διανύσουν. Βέβαια υπάρχουν άλλοι δυο παράγοντες που πρέπει να λάβουμε υπόψην. Ο πρώτος παράγοντας είναι η ύπαρξη κίνησης σε κάποιους δρόμους που κάνει μια “σύντομη” διαδρομή μη προτιμητέα. Ο δεύτερος παράγοντας είναι η ίδια η εμπειρία των οδηγών που μπορεί να τους βοηθά να επιλέγουν σύντομες διαδρομές και να γλιτώνουν την κίνηση.

Προκειμένου να έχουμε γρήγορη εκτέλεση του προγράμματος, υπάρχει ένα σύνολο νημάτων (thread pool) που, για κάθε φορτηγό και για κάθε διαδρομή μεταξύ δύο πελατών, κάθε νήμα αναλαμβάνει να τρέξει τον αλγόριθμο A* για να βρει την συντομότερη διαδρομή μεταξύ τους λαμβάνοντας υπόψην τα δεδομένα κίνησης δρόμων και την εμπειρία των οδηγών με την μορφή θετικών και αρνητικών βαρών, όπως αυτά λαμβάνονται από την Redis που τρέχει τοπικά (localhost).

Για να πετύχουμε ακόμα μεγαλύτερη επιτάχυνση, γίνεται χρήση της ehcache, μιας κρυφή μνήμη (cache) για την Java, η οποία κάθε φορά που αναζητούμε κάτι στην PostgreSQL και στην Redis, αποθηκεύεται τοπικά για γρηγορότερη πρόσβαση. Οι ρυθμίσεις της υπάρχουν στο αρχείο ehcache.xml.

3.5.1 Επικοινωνία με PostgreSQL μέσω Driver

Ο A* για να μπορέσει να εκτελεστεί χρειάζεται πληροφορίες για τους δρόμους της Αθήνας για να μπορέσει να βρει μια διαδρομή μεταξύ δυο σημείων. Για να γίνει αυτό, το πρόγραμμα ανοίγει μια σύνδεση μέσω JDBC Driver και επικοινωνεί με την PostgreSQL. Έτσι, έχουμε πρόσβαση σε μια βάση open street map που φιλοξενείται (hosted) στο ΕΚΠΑ με δρόμους της Αθήνας. Κάθε πλειάδα της βάσης έχει σαν πρωτεύον κλειδί (primary key) τον γεωγραφικό κωδικό κάθε δρόμου και σαν επιπλέον πεδία αυτά που είναι χρήσιμα για την εφαρμογή μας είναι το όνομα του δρόμου, η κατευθυντικότητά του (μονή ή διπλή) και το geom του που είναι ένας κωδικός που παριστάνει τη γεωμετρία του δρόμου σαν σύνολο σημείων. Ο χειρισμός των geom εσωτερικά στο πρόγραμμα γίνεται μέσω PostGIS ερωτημάτων (queries). Στο Παράρτημα ο αναγνώστης μπορεί να δει τα queries που χρησιμοποιούνται στο πρόγραμμα μαζί με επεξήγηση του τρόπου λειτουργίας και ειδικών περιπτώσεων που καλύπτουν.

3.5.2 Επικοινωνία με Redis

Η Redis αποτελεί ανεξάρτηση βάση από την PostgreSQL, διότι έχει δεδομένα που για κάθε δρόμο είναι δυναμικά και άρα πρέπει να υπάρχει ανεξάρτητα από την PostgreSQL που έχει σταθερά δεδομένα. Οι εγγραφές που περιέχει πρέπει να έχουν τη μορφή gid, weight. Το gid πρέπει να είναι κάποιο από τα gid της PostgeSQL. Το weight είναι το βάρος του δρόμου που σχετίζεται με την κίνηση που έχει ή την εμπειρία των οδηγών. Αν πρόκειται για δεδομένα κίνησης πρέπει να παίρνουν τιμές στο διάστημα $[0.0, 1.0]$, ενώ αν πρόκειται για δεδομένα εμπειρίας οδηγών στο $[-0.5, 0.0]$. Αν δεν έχουμε πληροφορία για κάποιο δρόμο, το βάρος του έχει την προεπιλεγμένη τιμή 0.0.

3.5.3 Χρήση της ehcache

Όπως έχει ήδη αναφερθεί, η ehcache χρησιμοποιείται για να επιταχύνει την εκτέλεση του προγράμματος σε συνεργασία με τα υπάρχοντα νήματα. Προκειμένου να μην γίνονται συνέχεια χρονοβόρα ερωτήματα στην PostgreSQL, κάθε φορά που ανακτάται κάποια πληροφορία, αποθηκεύεται στην κρυφή μνήμη και πλέον έχουμε πρόσβαση από αυτήν στην αντίστοιχη πληροφορία. Για τον σκοπό αυτό, υπάρχουν 5 ξεχωριστές κρυφές μνήμες (καθεμιά έχει ένα κλειδί αναζήτησης για να ανακτούμε γρήγορα την πληροφορία που το συνοδεύει):

- Η cache geoms που αποθηκεύει τις γεωμετρίες σημείων των δρόμων με κλειδί αναζήτησης το gid τους.
- Η cache roadData που αποθηκεύει τα δεδομένα των δρόμων (όνομα και κατευθυντικότητα) με κλειδί αναζήτησης το gid τους.
- Η cache successors που αποθηκεύει μα λίστα από υποψήφια σημεία από την τρέχουσα θέση και δρόμο με κλειδί αναζήτησης το gid των δρόμων. Είναι σημαντικό να τονιστεί ότι σαν υποψήφιοι αποθηκεύονται όλοι όσοι θα μπορούσαν να είναι ανεξάρτητα από την εκτέλεση του αλγορίθμου. Αυτό γίνεται για να μπορούν πληροφορίες της ehcache να χρησιμοποιηθούν και σε μελλοντικές εκτελέσεις.
- Η cache intersections που αποθηκεύει το σημείο που τέμνονται δύο δρόμοι με κλειδί την συνένωση (concatenation) των gid τους.
- Η cache roadWeights που αποθηκεύει τα βάρη των δρόμων με κλειδί το gid τους. Αν δεν υπάρχει πληροφορία βάρους για το δεδομένο gid στην Redis, αποθηκεύεται η τιμή 0.0. Σε περίπτωση μελλοντικής ενημέρωσης της Redis, το βάρος προσαρμόζεται κατάλληλα.

3.5.4 Επιλογή ευρετικής

Σαν κόστος της ευρετικής χρησιμοποιούμε το $(0.5 + weight)EarthDist(p1, p2)$. Αυτό είναι πάρα πολύ σημαντικό και συνδυάζεται με τον υπολογισμό του κόστους μεταξύ δύο σημείων στην PathFindingStepCostFunction με κόστος $(1.0 + weight)EarthDist(p1, p2)$. Τα

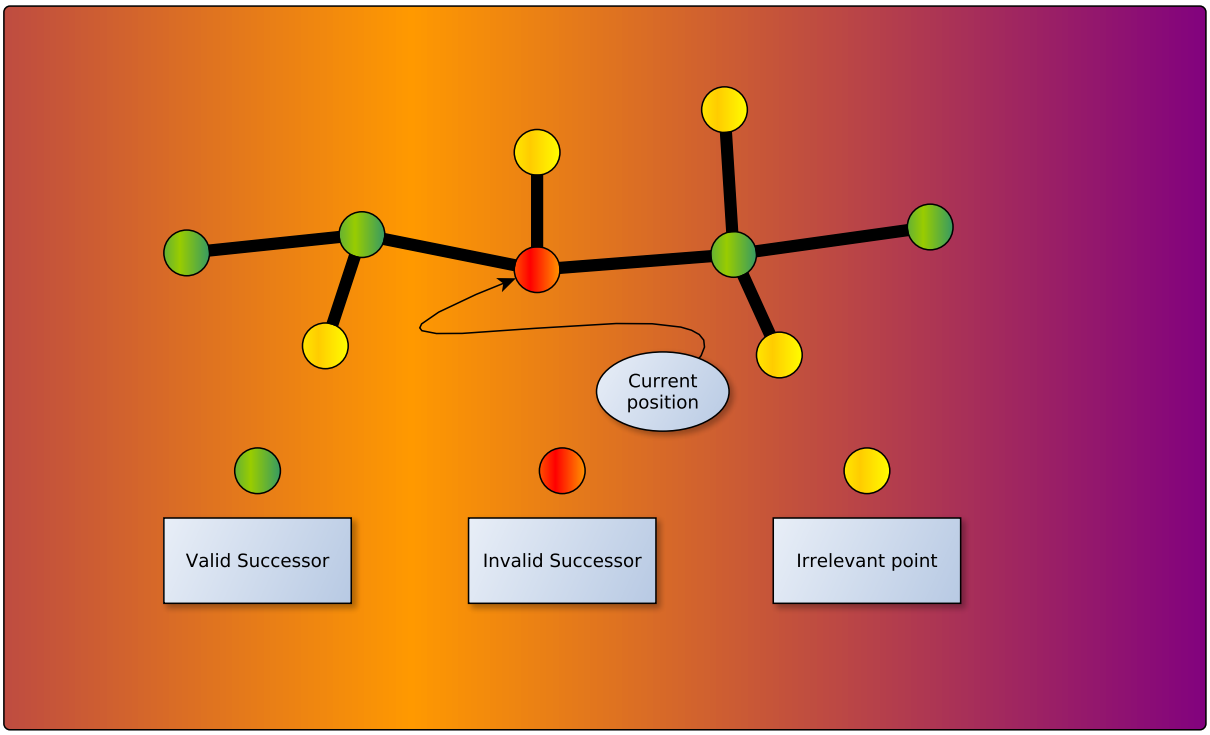
κόστη ορίζονται με αυτόν τον τρόπο για δύο λόγους. Ο πρώτος λόγος είναι για να μπορούμε να λαμβάνουμε υπόψην και θετικά και αρνητικά βάρη της Redis στον A*. Ο όρος $1.0 + weight$ εξασφαλίζει ότι θα έχουμε τιμές στο διάστημα $[0.5, 2.0]$. Αυτό σχετίζεται με τον δεύτερο λόγο, το γεγονός ότι πρέπει πάντα το κόστος της ευρετικής να είναι μικρότερο ή ίσο του πραγματικού κόστους. Σε αυτή την περίπτωση διασφαλίζουμε ότι γίνεται ακριβώς αυτό, αφού ο όρος στην ευρετική είναι πάντα 0.5.

3.5.5 Διάδοχοι τρέχουσας κατάστασης

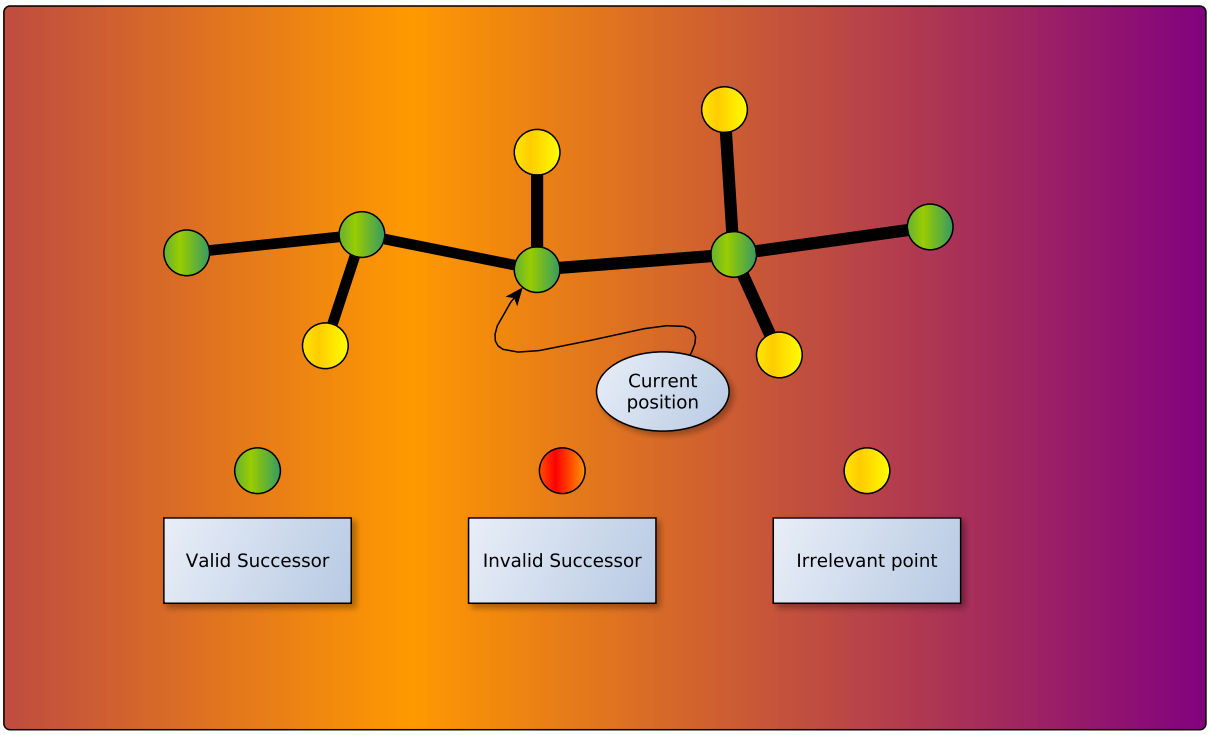
Δεδομένου ότι πρέπει να λάβουμε υπόψην και την κατευθυντικότητα των δρόμων αλλά και λόγω του τρόπου αποθήκευσής του στην PostgreSQL η επιλογή επόμενων καταστάσεων πρέπει να γίνεται *πολύ προσεκτικά*. Διακρίνουμε τις εξής περιπτώσεις:

- Ο δρόμος είναι διπλής κατεύθυνσης και δεν είναι ο αρχικός. Σε αυτή την περίπτωση, όλα τα σημεία εκτός από το τρέχον είναι υποψήφια για διασταυρώσεις με άλλους δρόμους (Σχήμα 12).
- Ο δρόμος είναι διπλής κατεύθυνσης και είναι ο αρχικός. Σε αυτή την περίπτωση, επειδή είναι πιθανό να υπάρχει κάποια διασταύρωση ακριβώς στην τρέχουσα θέση, όλα τα σημεία (και το τρέχον) θεωρούνται υποψήφιες διασταυρώσεις (Σχήμα 13).
- Ο δρόμος είναι μονής κατεύθυνσης και δεν είναι ο αρχικός. Σε αυτή την περίπτωση, όλα τα σημεία που προηγούνται με βάση τον προσανατολισμό του τρέχοντος (και το τρέχον) πρέπει να απορριφθούν από υποψήφια. Όλα τα επόμενα μπορούν να είναι υποψήφια για διασταυρώσεις με άλλους δρόμους (Σχήμα 14).
- Ο δρόμος είναι μονής κατεύθυνσης και είναι ο αρχικός. Σε αυτή την περίπτωση, αν το αρχικό σημείο υπάρχει στη γεωμετρία του δρόμου απορρίπτονται όλα τα σημεία πριν από το τρέχον αλλά αυτό και τα επόμενά του διατηρούνται (Σχήμα 15). Αυτό γίνεται γιατί όντας ο πρώτος δρόμος, υπάρχει πιθανότητα να υπάρχει διασταύρωση ακριβώς στη θέση που βρίσκεται. Σε περίπτωση που το αρχικό σημείο είναι ένα τυχαίο σημείο, πρέπει να βρούμε το κοντινότερο σημείο που υπάρχει πάνω στη γεωμετρία του δρόμου και να απορρίψουμε όλα τα σημεία που προηγούνται και να κρατήσουμε το τρέχον αφού αποτελεί υποψήφιο (Σχήμα 16).

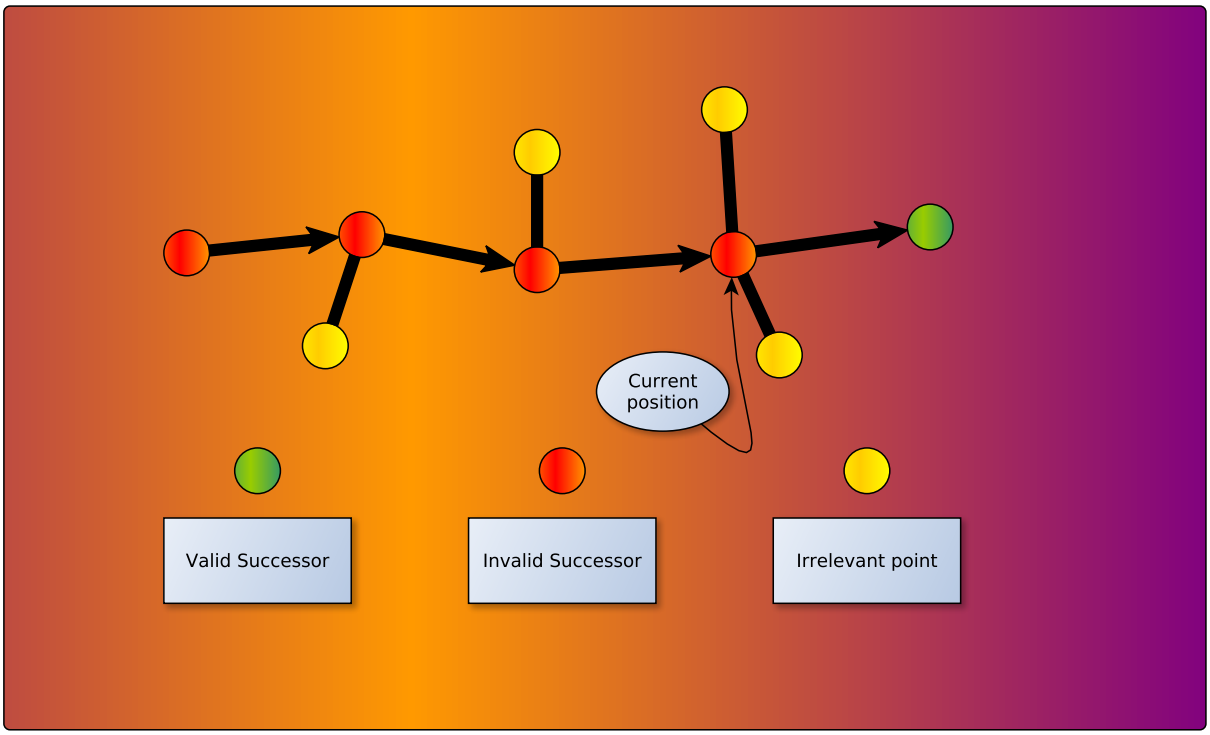
Αφού απορρίψουμε με βάση την παραπάνω περιπτωσιολογία υποψήφιους, μπορούμε να απορρίψουμε μερικούς επιπλέον στην περίπτωση των δρόμων μονής κατεύθυνσης. Αρχικά εξετάζουμε αν το σημείο διασταύρωσης του υποψήφιου δρόμου με τον τρέχοντα είναι και το τελευταίο του (Σχήμα 17). Σε αυτή την περίπτωση πρέπει να εξαιρεθεί από υποψήφιος αφού δεν έχουμε που να πάμε μετά. Έπειτα, σε περίπτωση που ο υποψήφιος δρόμος είναι και ο τελικός, εξετάζουμε αν γίνεται να τον φτάσουμε από την τρέχουσα θέση (Σχήμα 19), αλλιώς απορρίπτεται (Σχήμα 18). Αυτό γίνεται για να διασφαλιστεί ότι μπορούμε να φτάσουμε στον τελικό δρόμο από την σωστή κατεύθυνση.



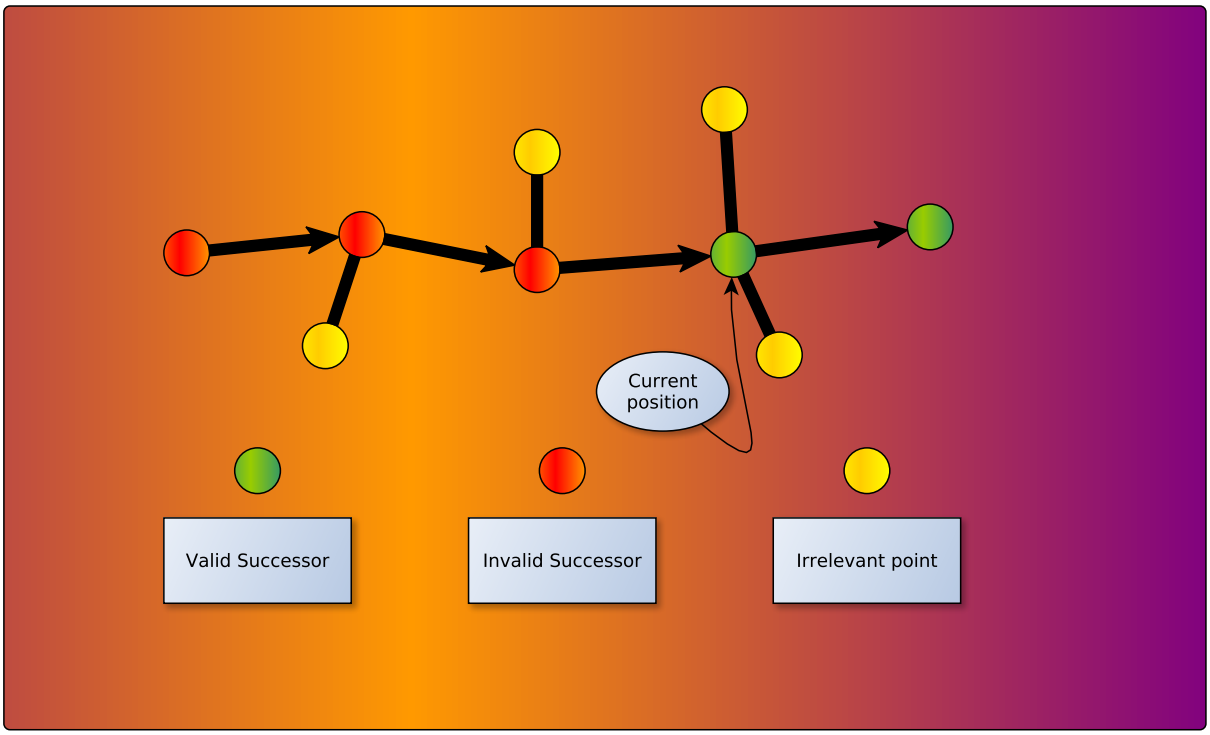
Σχήμα 12: Διάδοχοι δρόμου διπλής κατεύθυνσης που δεν είναι ο αρχικός



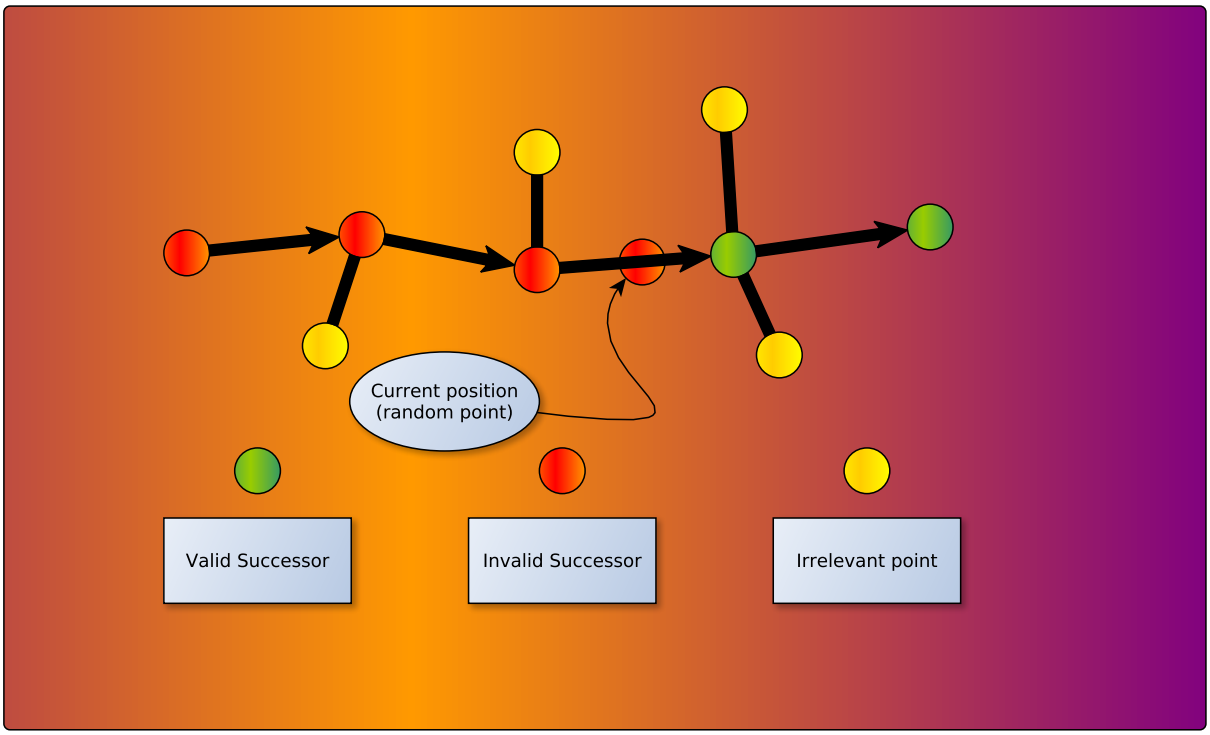
Σχήμα 13: Διάδοχοι δρόμου διπλής κατεύθυνσης που είναι ο αρχικός



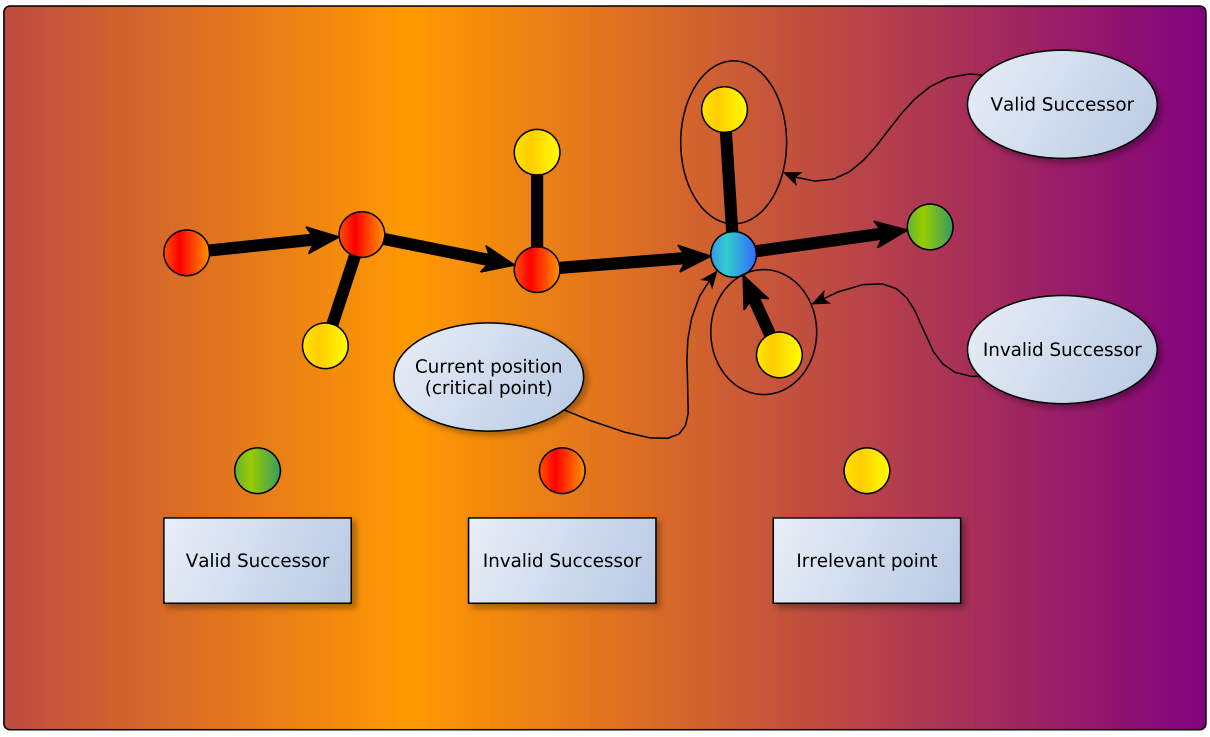
Σχήμα 14: Διάδοχοι δρόμου μονής κατεύθυνσης που δεν είναι ο αρχικός



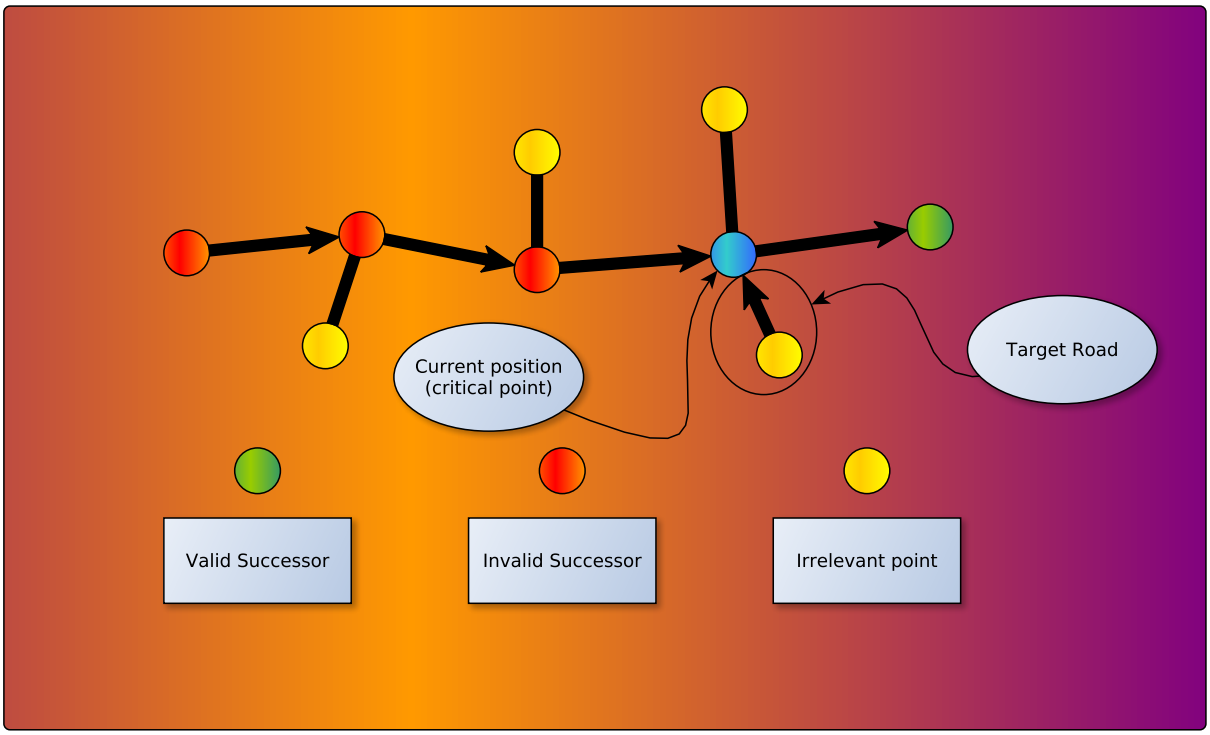
Σχήμα 15: Διάδοχοι δρόμου μονής κατεύθυνσης που είναι ο αρχικός - υπάρχον σημείο



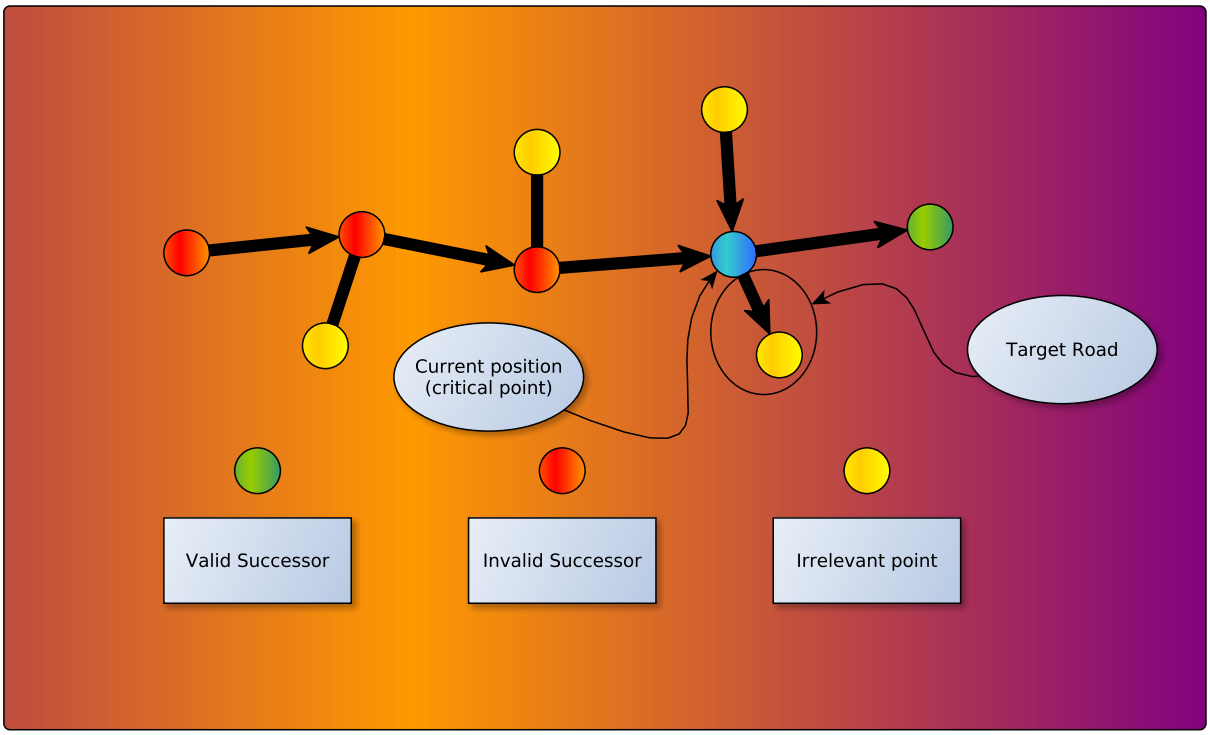
Σχήμα 16: Διάδοχοι δρόμου μονής κατεύθυνσης που είναι ο αρχικός - τυχαίο σημείο



Σχήμα 17: Διάδοχοι δρόμου μονής κατεύθυνσης - υποψήφιος που το σημείο τομής είναι το τελευταίο του



Σχήμα 18: Τελικός δρόμος ως διάδοχος δρόμου μονής κατεύθυνσης - μη αποδεκτή περίπτωση

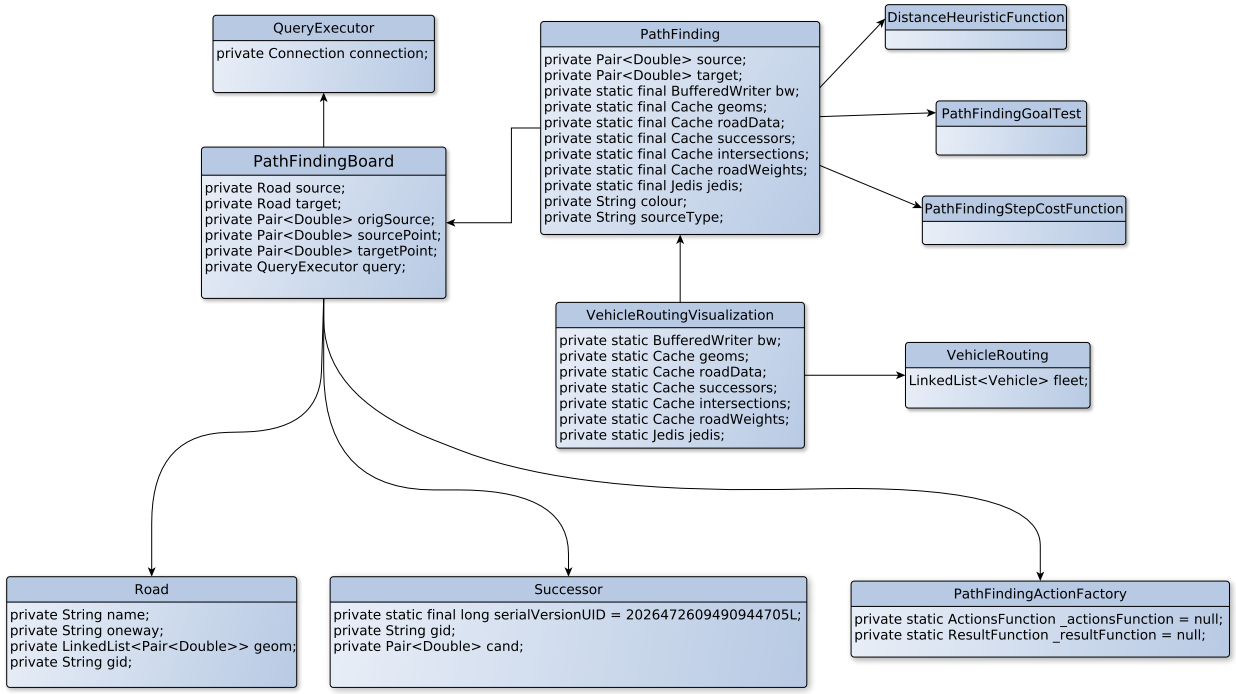


Σχήμα 19: Τελικός δρόμος ως διάδοχος δρόμου μονής κατεύθυνσης - αποδεκτή περίπτωση

3.5.6 Συνολική δομή

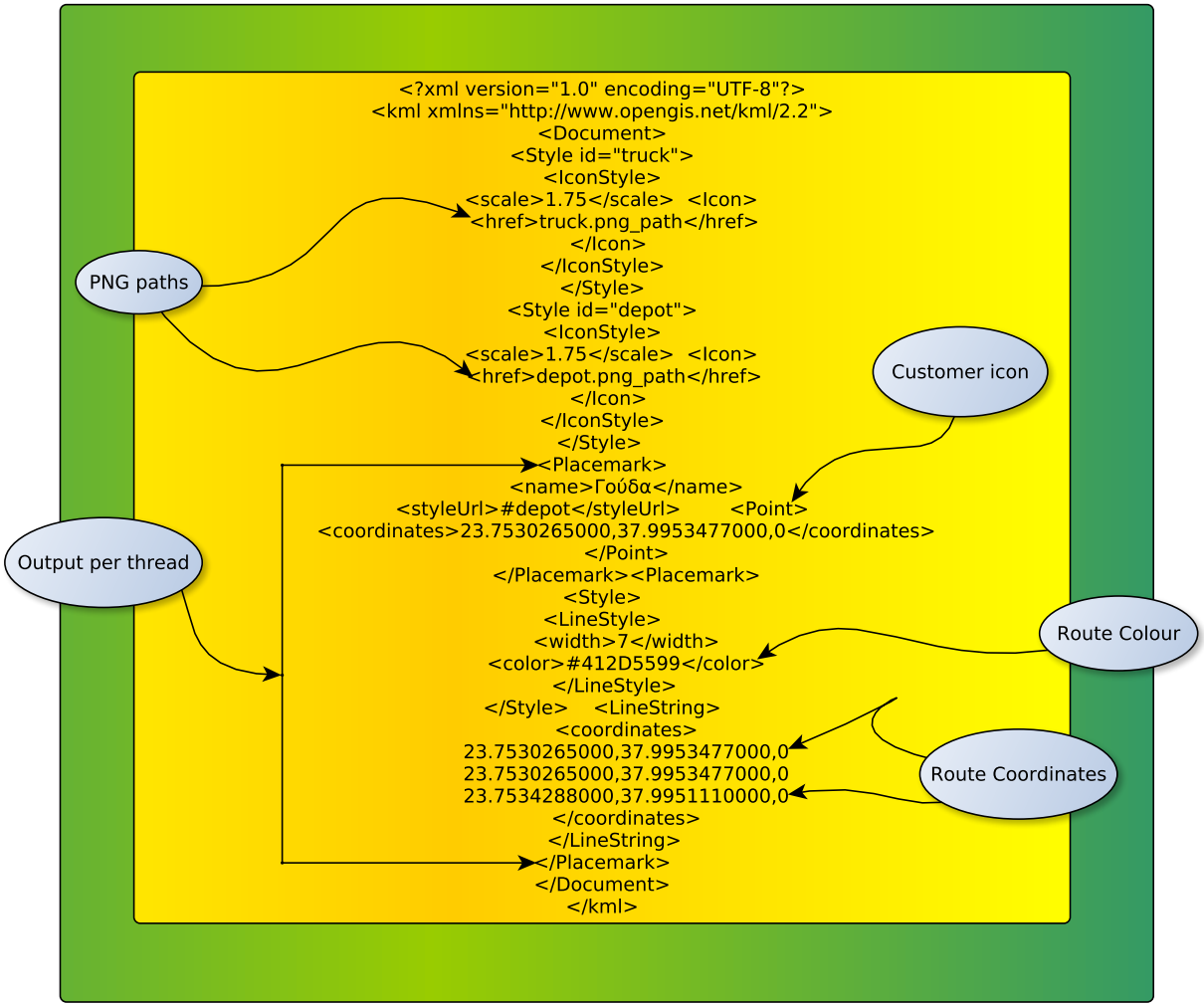
Έχοντας δει μερικά βασικά εργαλεία που χρειάζονται για την εκτέλεση του A*, ας εξηγήσουμε λίγο τι μοντελοποιεί η κάθε κλάση (Σχήμα 20):

- Η κλάση Road μοντελοποιεί τους δρόμους της PostgreSQL κρατώντας το όνομα, το gid, την κατευθυντικότητα και την γεωμετρία του δρόμου.
- Η κλάση Successor μοντελοποιεί πιθανό διάδοχο της τρέχουσας κατάστασης του A* κρατώντας το gid του δρόμου και τις συντεταγμένες της θέσης της νέας κατάστασης. Το επιπλέον πεδίο serialVersionUID υπάρχει γιατί η κλάση πρέπει να είναι σειριοποιήσιμη (serializable) προκειμένου να αποθηκεύεται και στην ehcache.
- Η κλάση QueryExecutor μοντελοποιεί την σύνδεση στην PostgreSQL μέσω του JDBC Driver κρατώντας σαν πεδίο τη σύνδεση και εκτελεί SQL ερωτήματα που της περνιούνται σαν παράμετροι.
- Η κλάση PathFindingGoalTest μοντελοποιεί τον έλεγχο του A* για το αν η τρέχουσα κατάσταση είναι και η τελική.
- Η κλάση PathFindingSteopCostFunction μοντελοποιεί τον υπολογισμό κόστους μετάβασης μεταξύ δύο καταστάσεων.
- Η κλάση DistanceHeuristicFunction μοντελοποιεί την ευρετική του A*.
- η κλάση PathFindingBoard μοντελοποιεί την συλλογή επόμενων από την τρέχουσα καταστάσεων. Αυτό γίνεται κρατώντας μια λίστα με υποψήφιες επόμενες καταστάσεις και μια λίστα με τα αποτελέσματα της (πιθανής) εφαρμογής τους.
- Η κλάση PathFindingBoard μοντελοποιεί μια κατάσταση του A*. Κρατάει τον αρχικό και τελικό δρόμο, μια σύνδεση από την κλάση QueryExecutor, το τρέχον σημείο που βρισκόμαστε, το αρχικό σημείο που ξεκίνησε ο αλγόριθμος και το τελικό σημείο που θέλουμε να φτάσουμε.
- Η κλάση PathFinding μοντελοποιεί την εκτέλεση του A* και ένα μέρος της εκτύπωσης της τελικής λύσης στο kml αρχείο. Κρατάει το αρχικό και το τελικό σημείο, ένα αρχείο στο οποίο γράφει δεδομένα, τις κρυφές μνήμες της ehcache, την σύνδεση με την Redis, το χρώμα της γραμμής της διαδρομής (κάθε φορτηγό έχει το δικό του χρώμα διαδρομής) και τον τύπο του πελάτη αφετηρίας (αρχηγείο ή απλός πελάτης).
- Η κλάση VehicleRoutingVisualization αποτελεί την κύρια κλάση ολόκληρου του προγράμματος. Επικοινωνεί με την κλάση VehicleRouting για να πάρει την λύση του CVRPTW και χρησιμοποιεί τα δεδομένα αυτά για βρει τις κοντινότερες διαδρομές μεταξύ των πελατών. Τέλος, γράφει τα αποτελέσματα σε ένα αρχείο kml και ένα αρχείο html (Κώδικας 5) για οπτικοποίηση του αποτελέσματος. Οι πληροφορίες που κρατάει είναι το αρχείο kml που θα γράψει την τελική λύση, οι κρυφές μνήμες και η σύνδεση με την Redis.



Σχήμα 20: Εξάρτηση των κλάσεων του κώδικα για τον A*

Μόλις τελειώσει η εκτέλεση του GLS, το πρόγραμμα κρατάει τη λύση που βρέθηκε και για κάθε δυο πελάτες δημιουργεί από ένα νήμα που θα τρέξει τον αλγόριθμο A* για να βρει την συντομότερη διαδρομή μεταξύ δυο σημείων. Σε κάθε φορτηγό αποδίδεται τυχαία και από ένα χρώμα για την διαδρομή του, ενώ κάθε πελάτης έχει ένα εικονίδιο φορτηγού που αντιπροσωπεύει τη θέση του. Το αρχηγείο έχει ειδικό εικονίδιο με ένα σπιτάκι για να ξεχωρίζει. Μόλις εκτελεστεί επιτυχώς ο A* έχουμε μια συλλογή από σημεία που φτιάχνουν τα κομμάτια των διαδρομών για κάθε φορτηγό και για κάθε ζευγάρι πελατών. Όλα τα νήματα συγχρονίζονται και γράφουν όταν τελειώνει το καθένα ατομικά στο kml (Σχήμα 21) αρχείο. Ο χρήστης επιλέγει μέσω ενός παραθύρου πού θα αποθηκευτεί και με τι όνομα. Μόλις γραφτούν τα σημεία από όλες τις διαδρομές, το πρόγραμμα τερματίζει την εκτέλεσή του.



Σχήμα 21: Δείγμα αρχείου kml

Κώδικας 5: Το html αρχείο που προβάλλει τα δεδομένα του kml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>VRPTW Visualization</title>
<script
type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false">
</script>
<style type="text/css">
  html {
    height: 100%
  }
  body {
    height: 100%;
    margin: 0;
```

```
      padding: 0
   }
   #map_canvas{
      width: 100%;
      height: 100%;
   }
</style>
<script
src="http://code.jquery.com/jquery-1.10.1.min.js">
</script>
<script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&
sensor=false&libraries=geometry">
</script>
<script
src="http://geoxml3.googlecode.com/svn/branches/polys/geoxml3.js">
</script>
<script
src="http://geoxml3.googlecode.com/svn/trunk/ProjectedOverlay.js">
</script>
<script
type="text/javascript">

function initialize() {
   var options = {
      center: new google.maps.LatLng(-34.397, 150.644),
      mapTypeId: google.maps.MapTypeId.ROADMAP,
      draggable: true,
      panControl:true,
      zoomControl:true,
      zoomControlOptions:{
         style:google.maps.ZoomControlStyle.LARGE,
         posi tion:google.maps.ControlPosition.LEFT_TOP
      },
      mapTypeControl:true,
      mapTypeControlOptions: {
         style: google.maps.MapTypeControlStyle.DEFAULT
      },
      scaleControl:true,
      streetViewControl:true,
      overviewMapControl:true,
      rotateControl:true,
      scrollwheel:true
```

```
  };

  var map = new google.maps.Map(document.getElementById("map_canvas"), options);
  var parser = new geoXML3.parser({map: map, processStyles: true});
  parser.parse("/kml/file/path/kml_file.kml");

  var worldBounds = new google.maps.LatLngBounds(new
      google.maps.LatLng(-85,-180),
  new google.maps.LatLng(85,180));
  map.fitBounds(worldBounds);

  if (window.attachEvent) {
     window.attachEvent("onresize", function( ) {this.map.onResize( )} );
     window.attachEvent("onload", function( ) {this.map.onResize( )} );
  }
  else if (window.addEventListener) {
     window.addEventListener("resize", function( ) {this.map.onResize( )},false);
     window.addEventListener("load", function( ) {this.map.onResize( )} , false);
  }
}

google.maps.event.addDomListener(window, 'load', initialize);
</script>
</head>
<body onload="initialize()">
<div id="map_canvas"></div>
</body>
</html>
```

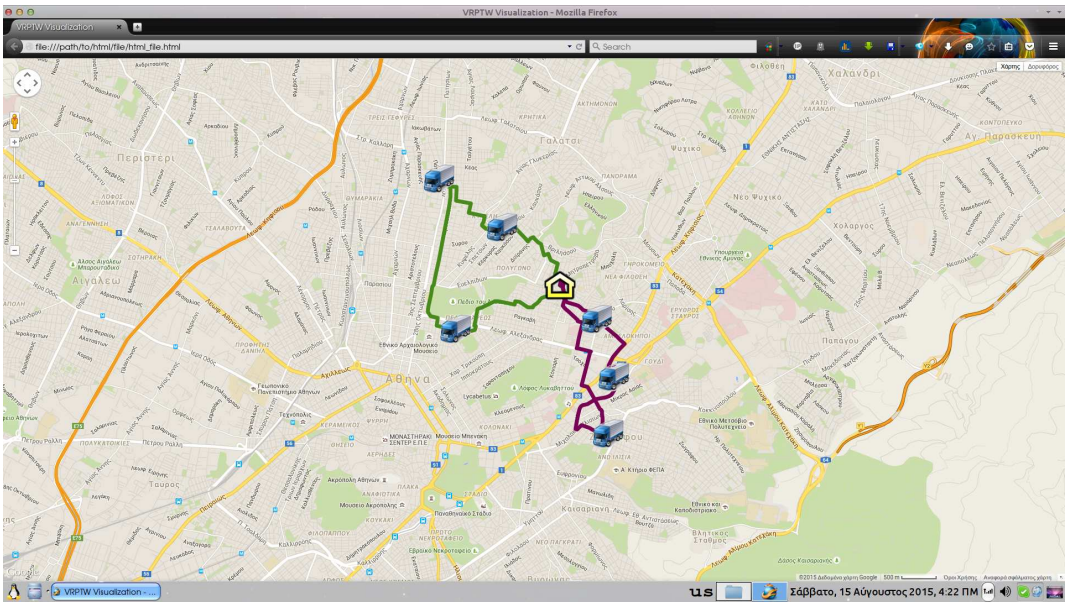
3.6 Οπτικοποίηση του τελικού αποτελέσματος

Έχοντας αναλύσει όλη την εκτέλεση του προγράμματος και την δομή του, μπορούμε να δείξουμε πώς φαίνεται το τελικό αποτέλεσμα.

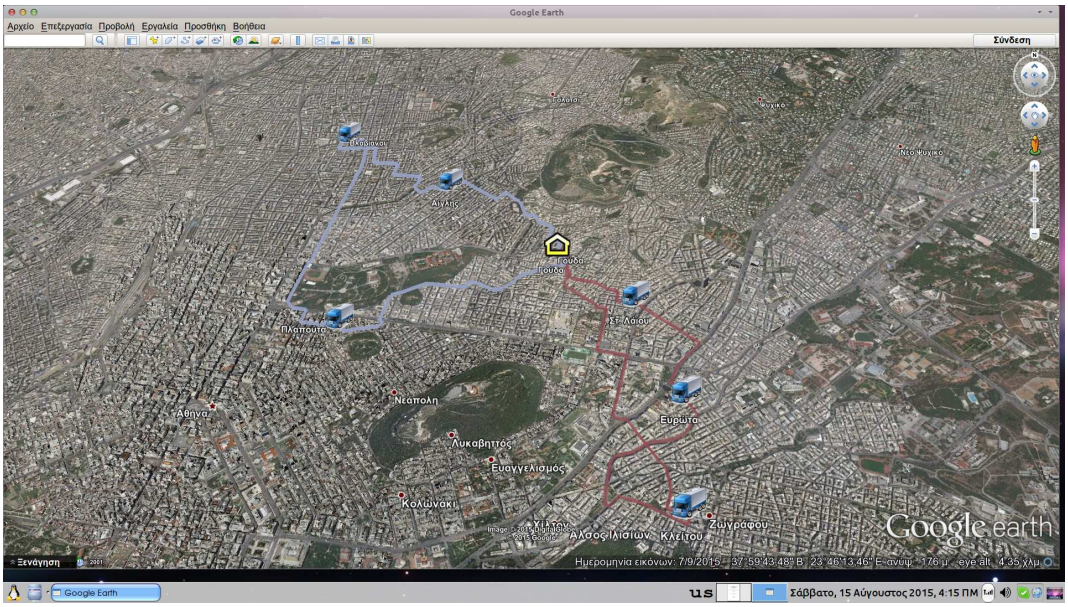
Χρήσιμα μηνύματα που αφορούν το CVRPTW εκτυπώνονται σε τερματικό. Ο λόγος είναι ότι η εφαρμογή στην παρούσα κατάσταση απευθύνεται κυρίως σε ακαδημαϊκούς. Σε κάθε περίπτωση πάντως, η λύση που βρίσκεται αποθηκεύεται στο αντίστοιχο αρχείο για μελλοντική προβολή ή μελέτη των αποτελεσμάτων.

Όπως έχει ήδη αναφερθεί, είναι δυνατόν να προβάλλουμε τα δεδομένα σε κάποια εφαρμογή ή ακόμα και στον αγαπημένο μας web browser. Έχει διαπιστωθεί ότι σίγουρα δεν γίνεται να προβληθεί το τοπικό αρχείο στον Google Chrome, διότι για λόγους ασφαλείας δεν επιτρέπει την προβολή τοπικών αρχείων που περιέχουν μονοπάτια.

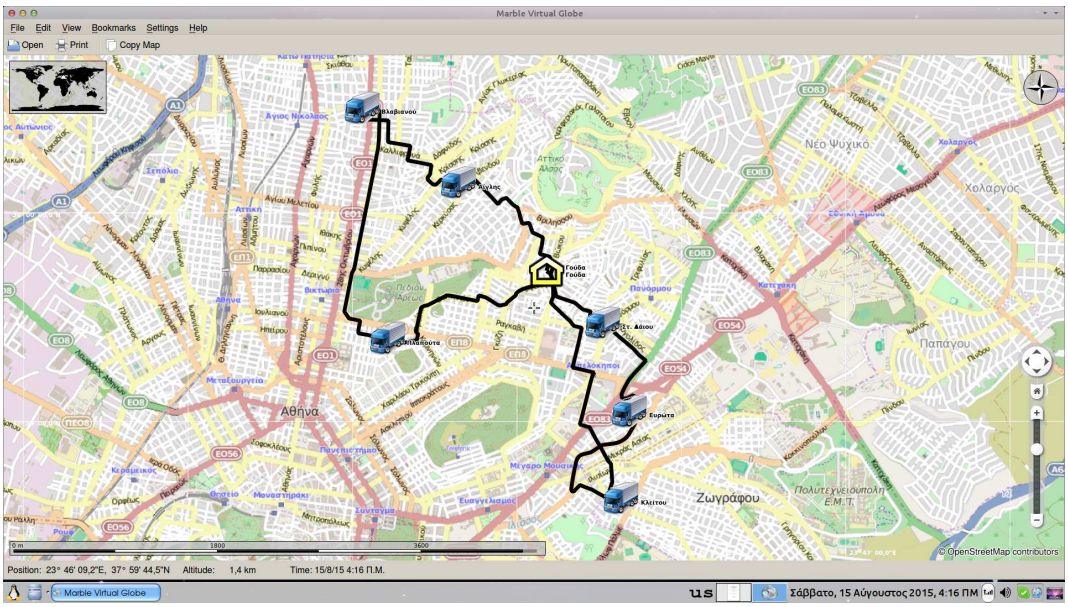
Στα επόμενα σχήματα (Σχήμα 22-24) φαίνεται η οπτικοποίηση ενός βασικού δοκιμαστικού στον Mozilla Firefox (προβολή του html), στο Google Earth (προβολή του kml) και στο Marble (ομοίως). Στο Marble όλες οι διαδρομές έχουν μαύρο χρώμα γιατί δεν είναι φτιαγμένο να χειρίζεται τα χρώματα του kml.



Σχήμα 22: Οπτικοποίηση της λύσης στον Mozilla Firefox



Σχήμα 23: Οπτικοποίηση της λύσης στο Google Earth



Σχήμα 24: Οπτικοποίηση της λύσης στο Marble

4. ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

Σε αυτό το κεφάλαιο θα γίνει παράθεση των πειραματικών αποτελεσμάτων και ενός παραδείγματος εκτέλεσης.

4.1 Μετρήσεις

Σε αυτή την ενότητα θα συγκρίνουμε μόνο για το CVRPTW κομμάτι την επίδοση του προγράμματος (Πίνακας 1) με κάποια από τα τεστ του Solomon [11]. Τα τεστ αυτά χωρίζονται γενικά σε 3 κατηγορίες:

- Τα τεστ CX0Y. Το X είναι το πλήθος των πελατών του τεστ $/100$ (στα υπάρχοντα τεστ του Solomon έχουμε 100 ή 200 πελάτες). Το Y είναι ο αύξοντας αριθμός του τεστ. Τα τεστ της ομάδας CX0Y αναφέρονται σε δοκιμαστικά με συσταδοποιημένα (clustered) δεδομένα. Αυτό σημαίνει ότι η γεωμετρία του προβλήματος θέλει πολλούς πελάτες να αντιστοιχίζονται “φυσικά” σε μια διαδρομή και άρα εδώ τις πιο πολλές φορές θα πρέπει να έχουμε πολύ καλές επιδόσεις.
- Τα τεστ RX0Y. Το X είναι το πλήθος των πελατών του τεστ $/100$ (στα υπάρχοντα τεστ του Solomon έχουμε 100 ή 200 πελάτες). Το Y είναι ο αύξοντας αριθμός του τεστ. Τα τεστ της ομάδας RX0Y αναφέρονται σε δοκιμαστικά με τελείως τυχαία (random) δεδομένα. Αυτό σημαίνει ότι κάθε τεστ έχει τυχαία τοποθετημένους πελάτες και επομένως οποιαδήποτε υλοποίηση μπορεί να έχει από άριστη έως εξαιρετικά άσχημη επίδοση.
- Τα τεστ RCX0Y. Το X είναι το πλήθος των πελατών του τεστ $/100$ (στα υπάρχοντα τεστ του Solomon έχουμε 100 ή 200 πελάτες). Το Y είναι ο αύξοντας αριθμός του τεστ. Τα τεστ της ομάδας RCX0Y αναφέρονται σε δοκιμαστικά με συσταδοποιημένα (clustered) και τυχαία (random) δεδομένα. Σε αυτή την περίπτωση περιμένουμε μια ενδιάμεση επίδοση σε σχέση με τις προηγούμενες κατηγορίες.

Πίνακας 1: Αποτελέσματα του GLS για τα τεστ του Solomon σε σχέση με τις καλύτερες γνωστές λύσεις

Test case	Best known cost	GLS cost	Best known number of vehicles	GLS number of vehicles	Iterations
C101	828.93664	1052.270973199374	10	11	8
C102	828.93664	1425.3398534387452	10	13	10
C107	828.93664	953.737506596058	10	11	6
C201	591.55626	603.8792840880658	3	3	1
C202	591.55626	1906.1764325360728	3	4	4
R101	1650.79864	1753.1747833182615	19	20	16
R102	1486.85859	1890.5706507401735	17	24	25
R201	1252.37074	2353.0812119439597	4	6	6
RC101	1696.94871	1803.467856259144	14	17	14
RC201	1406.93961	2380.040157421201	4	5	4

Στον παραπάνω πίνακα ο αναγνώστης μπορεί να δει την επίδοση του προγράμματος ως προς το CVRPTW κομμάτι σε σχέση με τις καλύτερες γνωστές λύσεις για τα τεστ του Solomon. Με πράσινο χρώμα εμφανίζονται τα στοιχεία μιας λύσης που είναι πολύ κοντινά στη βέλτιστη λύση. Με μπλε εμφανίζονται τα στοιχεία μιας λύσης που θα μπορούσαν να

είναι κάπως καλύτερα αλλά η απόκλιση από την αντίστοιχη καλύτερη γνωστή δεν είναι τόσο μεγάλη. Με κόκκινο εμφανίζονται τα στοιχεία μιας λύσης που απέχουν πολύ από την αντίστοιχη καλύτερη γνωστή.

Όσον αφορά τον χρόνο εκτέλεσης, το πρόγραμμα έχει καλή απόδοση. Τα αποτελέσματα του GLS για κάποιο από τα τεστ του Solomon με 100 πελάτες βγαίνουν σε 6-7 δευτερόλεπτα. Αντίστοιχα, ένα δοκιμαστικό με 7 πελάτες που τρέχουν 7 νήματα παράλληλα τον A* θέλουν περίπου 1-2 λεπτά στην πρώτη εκτέλεση και 3-4 δευτερόλεπτα σε κάθε επόμενη λόγω της ehcache. Συνολικά, οι χρόνοι είναι ικανοποιητικοί για ένα τέτοιο πρόβλημα. Οι μετρήσεις έγιναν σε υπολογιστή με τις ακόλουθες προδιαγραφές:

- Λειτουργικό Σύστημα (Operating System, OS): Xubuntu 15.04 με πυρήνα x86_64 Linux 3.19.0-25-generic και κέλυφος (shell) bash 4.3.30
- Επεξεργαστής (Central Processing Unit, CPU): Intel Core i5-4460 CPU @ 3.4GHz
- Κάρτα Γραφικών (Graphics Card): NVIDIA GeForce GTX 970 4GB
- Μνήμη RAM: 16GB DDR3
- Ολοκληρωμένο Περιβάλλον Ανάπτυξης (Integrated Desktop Environment, IDE): JetBrains IntelliJ 14.1.4

4.2 Παράδειγμα εκτέλεσης

Σε αυτήν την ενότητα θα δούμε ένα παράδειγμα εκτέλεσης. Εξηγήσαμε πώς το πρόγραμμα μπορεί να εφαρμόσει την επίλυση του κλασσικού προβλήματος και να χρησιμοποιήσει τη λύση αυτή στην πράξη.

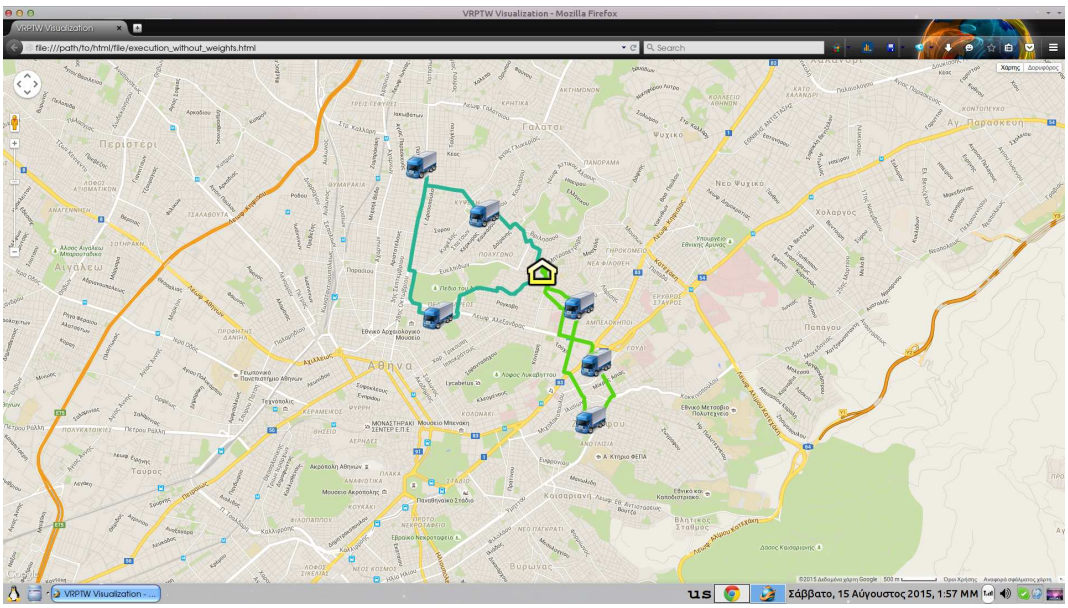
Επιπλέον η χρήση βαρών με δεδομένα κίνησης προσδίδει μια δυναμικότητα στο πρόβλημα. Ακόμα και για τα ίδια ακριβώς δεδομένα η διαδρομή που θα λαμβάνουμε μπορεί να διαφέρει.

Έστω ότι έχουμε το ακόλουθο πρόβλημα:

- Το αρχηγείο βρίσκεται στη διασταύρωση Γούδα και Κουγκίου.
- Έχουμε 6 πελάτες προς εξυπηρέτηση (χωρίς το αρχηγείο). Αυτοί βρίσκονται στη διασταύρωση:
    - Φθιώτιδος και Στ.Λάιου.
    - Σινώπης και Ευρώτα.
    - Λιβύης και Κλείτου.
    - Υπάτρου και Πλαπούτα.
    - Πατησίων και Βλαβιανού.
    - Αίγλης και Ελάτειας.

- Για να είναι πιο εύκολη η διατύπωση του παραδείγματος, δίνουμε για το αρχηγείο και τους 6 πελάτες τα κωδικά ονόματα Γούδα, Στ.Λάιου, Ευρώτα, Κλείτου, Πλαπούτα, Βλαβιανού και Αίγλης αντίστοιχα. Υποθέτουμε ότι ο GLS έδωσε σαν λύση δυο φορτηγά που το ένα πρέπει να κάνει τη διαδρομή Γούδα-Πλαπούτα-Βλαβιανού-Αίγλης-Γούδα, ενώ το άλλο τη διαδρομή Γούδα-Στ.Λάιου-Ευρώτα-Κλείτου-Γούδα.

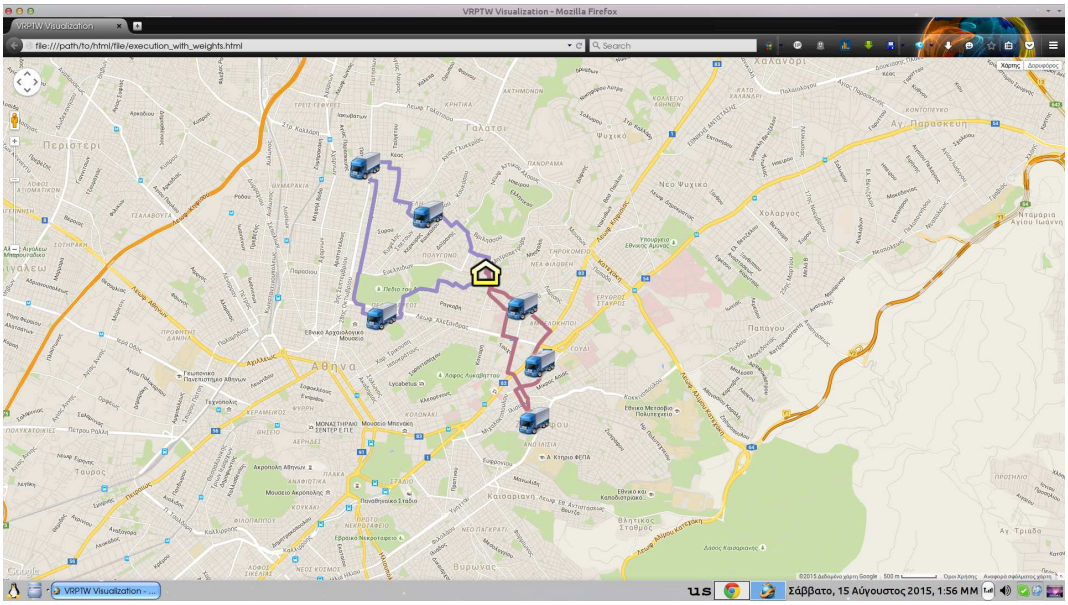
Αρχικά, ας υποθέσουμε ότι κάνουμε το τεστ με μια άδεια Redis βάση. Αυτό σημαίνει ότι θα πρέπει να βρούμε τη συντομότερη διαδρομή χωρίς επιρροές βαρών. Πράγματι, το αποτέλεσμα που λαμβάνουμε είναι το αναμενόμενο (Σχήμα 25).



Σχήμα 25: Εκτέλεση του προγράμματος με απουσία δεδομένων κίνησης

Τώρα ας δοκιμάσουμε να λύσουμε ακριβώς το ίδιο πρόβλημα προσθέτοντας βάρη στη Redis.

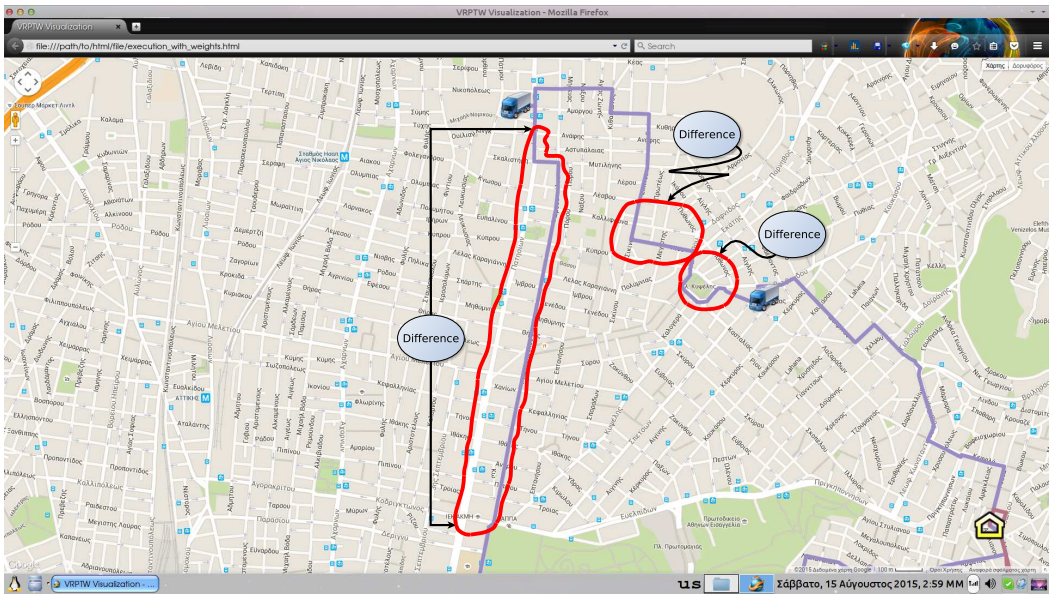
Για χάρη παραδείγματος και ακολουθώντας κοινή λογική, βάζουμε δεδομένα κίνησης 70-80% (βάρος 0.7-0.8 αντίστοιχα) στην Πανόρμου, 59% στην Πύθωνος (βάρος 0.59) και 100% στην Πατησίων (βάρος 1.0). Επιπλέον, βάζουμε δεδομένα εμπειρίας οδηγών 35% (βάρος -0.35) στην Κίου και 45% (βάρος -0.45) στην Μιχαλακοπούλου. Η λύση πλέον θα διαφέρει (Σχήμα 26) σε μερικά ενδιαφέροντα σημεία (Σχήμα 27-28).



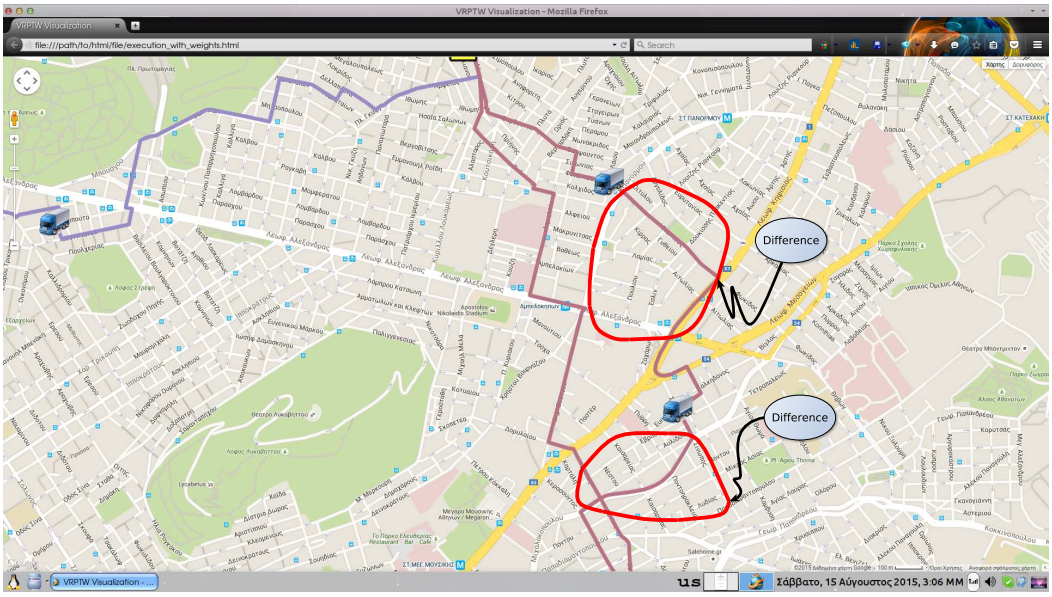
Σχήμα 26: Εκτέλεση του προγράμματος με ύπαρξη δεδομένων κίνησης

Ας απαριθμήσουμε τις σημαντικές διαφορές:

- Το πρώτο φορτηγό:
  - Κανονικά θα οδηγούσε κατά μήκος σχεδόν όλης της Πατησίων. Ωστόσο, θα την αποφύγει γιατί έχει πολλή κίνηση και θα οδηγήσει κατά μήκος της Ι.Δροσοπούλου. Κάτι που αξίζει να παρατηρήσουμε εδώ είναι ότι όταν πλησιάζει τον πελάτη στην Βλαβιανού, θα οδηγήσει και για λίγο στην Πατησίων. Αυτό οφείλεται στο γεγονός ότι παρά την ύπαρξη κίνησης, είναι η συντομότερη διαδρομή από εκείνο το σημείο και μετά· δεν πρέπει να ξεχνάμε ότι οι δρόμοι έχουν και κατευθύνσεις οπότε κάποιος δρόμος που οπτικά θα φαινόταν καλύτερη επιλογή μπορεί να μην έχει την σωστή κατεύθυνση για να είναι υποψήφιος!
  - Θα επηρεαστεί από την εμπειρία των οδηγών και θα ακολουθήσει την Κίου.
  - Κανονικά θα κατέβαινε όλη την Πύθωνος και θα πλησίαζε τον πελάτη στην Αίγλης. Ωστόσο, θα την αποφύγει λόγω υψηλής κίνησης και θα επιλέξει να φτάσει περνώντας από την Πλατεία Κυψέλης.
- Το δεύτερο φορτηγό:
  - Κανονικά θα περνούσε από την Πανόρμου. Ωστόσο, θα την αποφύγει γιατί έχει πολλή κίνηση και θα ακολουθήσει άλλη διαδρομή μεταξύ των πελατών Στ.Λάιου και Ευρώτα.
  - Θα επηρεαστεί από την εμπειρία των οδηγών και θα ακολουθήσει την Μιχαλακοπούλου αντί να πάει από την Σινώπης και την Ούλωφ Πάλμε.



Σχήμα 27: Οι αλλαγές στη διαδρομή του πρώτου φορτηγού



Σχήμα 28: Οι αλλαγές στη διαδρομή του δεύτερου φορτηγού

Είναι εύκολο να παρατηρήσουμε ότι τρέχοντας το ίδιο τεστ με αλλαγή στα βάρη, το αποτέλεσμα μπορεί να αλλάξει σημαντικά. Επίσης, αυτές οι αλλαγές επηρεάζουν και άλλα κομμάτια της διαδρομής. Ο λόγος είναι ότι έχοντας βρεθεί πλέον σε άλλες ενδιάμεσες καταστάσεις, ο A* θα βρει *νέα* συντομότερη προς τον εκάστοτε στόχο.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτή την ενότητα θα σχολιάσουμε τα συμπεράσματα που εξάγονται από την εκτέλεση του προγράμματος. Αρχικά, γίνεται μια αναφορά σε κάποιες σχεδιαστικές επιλογές που επηρεάζουν την ποιότητα των αποτελεσμάτων και στη συνέχεια διατυπώνονται μερικές τελικές παρατηρήσεις και ιδέες για μελλοντικές επεκτάσεις/εφαρμογές.

5.1 Σχεδιαστικές επιλογές

Όπως εξηγήθηκε, στόχος της παρούσας πτυχιακής είναι να προτείνει μια πρακτική εφαρμογή των κλασσικών μεθόδων επίλυσης του CVRPTW. Προκειμένου όμως να μπορούμε να έχουμε τουλάχιστον ως ένα βαθμό σύγκριση με ακαδημαϊκά τεστ, το CVRPTW κομμάτι δέχεται δεδομένα εισόδου με τη λογική των τεστ του Solomon.

Αν ο αναγνώστης αναζητήσει μεθόδους επίλυσης του προβλήματος στη βιβλιογραφία [12] θα δει ότι η εφαρμογή του PFIH για εύρεση αρχικής λύσης και του GLS για βελτίωσή της είναι η λιγότερο αποδοτική προσέγγιση. Ειδικά όταν βλέπουμε μετρήσεις των διαφόρων μεθόδων, ο GLS συνήθως δουλεύει με τον αλγόριθμο Φανταστικού Οχήματος (Virtual Vehicle) για την εύρεση αρχικής λύσης και παρουσιάζει αρκετά καλύτερα αποτελέσματα έτσι. Ο αλγόριθμος αυτός όμως προϋποθέτει ότι έχουμε εκ των προτέρων γνώση για τα δεδομένα που θα δοκιμάσουμε και αυτό θα αναιρούσε την γενική χρήση της εφαρμογής.

Αντί για τον GLS θα μπορούσαν να χρησιμοποιηθούν πιο εκλεπτυσμένοι αλγόριθμοι. Ωστόσο, ο στόχος δεν ήταν να βρούμε μια νέα ανταγωνιστική μέθοδο σε σχέση με τις υπάρχουσες, αλλά να προτείνουμε μια πρακτική εφαρμογή ενός κλασσικού προβλήματος. Επιπλέον, ο GLS λόγω της απλότητάς του είναι πολύ εύκολο να γίνει κατανοητός και σε κάποιον που δεν είναι ειδικός.

Τέλος, η χρήση της PostgreSQL βάσης μπορεί να επηρεάσει καθοριστικά τη λύση. Αν υπάρχουν τυχόν αποκλίσεις σε σχέση με κάποια βέλτιστη διαδρομή μεταξύ δύο σημείων στο Google Maps, αυτές θα οφείλονται σε λανθασμένες πληροφορίες κατευθυντικότητας κάποιων από τους δρόμους στη βάση. Το πρόγραμμα εξακολουθεί να είναι ορθό παρά ταύτα και αν ο αναγνώστης διαπιστώσει τέτοια φαινόμενα μπορεί απλώς να δοκιμάσει κάποια βάση δρόμων που έχει την ίδια δομή και θεωρεί περισσότερο αξιόπιστη.

5.2 Τελικές παρατηρήσεις

Η ανάπτυξη του προγράμματος έγινε με την πρόθεση ο κώδικας να είναι απλός και να αποτελείται από κατανοητά βήματα. Η επιλογή αλγορίθμων που συνδυάζονται για το τελικό αποτέλεσμα έγινε με τρόπο που να είναι επαρκώς αποτελεσματικοί και κατανοητοί στον χρήστη· είναι σημαντικό να μπορούμε να καταλαβαίνουμε τι γίνεται σε όλη την εκτέλεση και πώς λειτουργούν εσωτερικά τα συστατικά στοιχεία του προγράμματος ώστε τα ερευνητικά αποτελέσματα που παράγονται να έχουν κάποια αξία.

Η ενσωμάτωση δυναμικών δεδομένων κίνησης στους δρόμους και της εμπειρίας των οδη-

γών προσφέρουν μια πληθώρα από δυνατότητες, μελλοντικές επεκτάσεις και εφαρμογές. Για παράδειγμα, μια επέκταση της εφαρμογής θα μπορούσε να είναι η χρήση αισθητήρων πραγματικής κίνησης στους δρόμους έτσι ώστε σε πραγματικό χρόνο να ενημερώνεται το πρόγραμμα για αλλαγές στην κίνηση και να λύνει το πρόβλημα ικανοποιητικά.

Επίσης, το γεγονός ότι η εφαρμογή μπορεί να λειτουργήσει και σαν προσομείωση προσφέρει την ευκαιρία για μερικά ενδιαφέροντα πειράματα. Θα μπορούσαν για παράδειγμα να μελετηθούν για διάφορες ώρες της μέρας οι διαφορετικές λύσεις που λαμβάνονται και να εξαχθούν συμπεράσματα σχετικά με την κατά μέσο όρο συνολική απόσταση που πρέπει να διανύσουν τα φορτηγά για μια δεδομένη λύση. Ακόμη, θα μπορούσε να γίνεται αξιολόγηση της εμπειρίας των οδηγών συγκρίνοντας μια λύση που προέρχεται από δεδομένα κίνησης δρόμων και μιας λύσης (για το ίδιο πρόβλημα) που προέρχεται μόνο από εμπειρία οδηγών ή να γίνεται έλεγχος πότε η εμπειρία των οδηγών μπορεί να είναι το ίδιο αποτελεσματική με δεδομένα κίνησης για συγκεκριμένα δοκιμαστικά.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός όρος**
Capacitaded Vehicle Routing Problem with Time Windows	Πρόβλημα Δρομολόγησης Οχημάτων Περιορισμένης Χωρητικότητας με Χρονικά Όρια
Relational database	Σχεσιακή βάση
In-memory key-value storage	αποθήκη ζευγών κλειδού-τιμής στην μνήμη
Travelling Salesman Problem	Πρόβλημα του Περιοδεύοντος Πωλητή
Vehicle Routing Problem	Πρόβλημα Δρομολόγησης Οχημάτων
Depot	Αρχηγείο
Vehicle fleet	Αρμάδα οχημάτων
Class	Κλάση
Heuristics	Ευρετικές
Meta-heuristics	Μετα-ευρετικές
Feasible solution	Εφικτή λύση
Push-Forward Insertion Heuristic	Ευρετική Εισαγωγής με Απευθείας Προώθηση
Guided Local Search	Τοπική Κατευθυνόμενη Αναζήτηση
2-opt swap	2-βελτιστοποίηση με αλλαγή φοράς
Intra-route operator	Τελεστής που εφαρμόζεται εσωτερικά σε μια διαδρομή
Relocate	Επανατοποθέτηση
Inter-route operator	Τελεστής που εφαρμόζεται μεταξύ διαδρομών
Exchange	Ανταλλαγή
Cross operator	Τελεστής διασταύρωσης
Cross-append operator	Τελεστής διασταύρωσης-προσάρτησης στο τέλος
Cross-split operator	Τελεστής διασταύρωσης-διαχωρισμού
Augmented objective function	Επαυξημένη αντικειμενική συνάρτηση
Stanford Research Institute	Ερευνητικό Ινστιτούτο του Στάνφορντ
Web browser	Φυλλομετρητής
Terminal	Τερματικό
Project	Έργο
Thread pool	Σύνολο νημάτων
Localhost	Εκτέλεση βάσης τοπικά
Cache	Κρυφή μνήμη

Hosted	Φιλοξενείται
Primary key	Πρωτεύον κλειδί
Queries	Ερωτήματα
Concatenation	Συνένωση
Serializable	Σειριοποιήσιμος
Clustered	Συσταδοποιημένος
Random	Τυχαίος
Operating System	Λειτουργικό Σύστημα
Shell	Κέλυφος
Central Processing Unit	Κεντρική Μονάδα Επεξεργασίας/Επεξεργαστής
Graphics Card	Κάρτα Γραφικών
Integrated Desktop Environment	Ολοκληρωμένο Περιβάλλον Ανάπτυξης
Virtual Vehicle	Φανταστικό Όχημα

ΣΥΝΤΜΗΣΕΙΣ - ΑΡΤΙΚΟΛΕΞΑ - ΑΚΡΩΝΥΜΙΑ

ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
CVRPTW	Capacitated Vehicle Routing Problem with Time Windows
NP	Non-deterministic polynomial time
SQL	Structured Query Language
JDBC	Java Database Connectivity
KML	Keyhole Markup Language
HTML	HyperText Markup Language
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickup and Delivery
LIFO	Last In First Out
VRPTW	Vehicle Routing Problem with Time Windows
CVRP	Capacitated Vehicle Routing Problem
CVRPTW	Capacitated Vehicle Routing Problem with Time Windows
VRPMT	Vehicle Routing Problem with Multiple Trips
OVRP	Open Vehicle Routing Problem
PFIH	Push-Forward Insertion Heuristic
GLS	Guided Local Search
SRI	Stanford Research Institute
OS	Operating System
CPU	Central Processing Unit
RAM	Random Access Memory
IDE	Integrated Desktop Environment

ΠΑΡΑΡΤΗΜΑ

Σε αυτή την ενότητα θα γίνει παράθεση της συνάρτηση που υπολογίζει την απόσταση δύο σημείων στη Γη και των PostGIS ερωτημάτων που χρησιμοποιούνται στο πρόγραμμα με μια μικρή επεξήγηση για το καθένα.

Κώδικας 6: Απόσταση δύο σημείων στη Γη

```
public double getDist(Pair<Double> source,Pair<Double> target){

    Double lat1,lon1,lat2,lon2;

    lat1=source.getFirst();
    lon1=source.getSecond();
    lat2=target.getFirst();
    lon2=target.getSecond();

    Double R = 6371.0; // km
    Double dLat = (lat2-lat1)*(Math.PI/180.0);
    Double dLon = (lon2-lon1)*(Math.PI/180.0);
    Double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
           Math.cos(lat1*(Math.PI/180.0)) *
              Math.cos(lat2*(Math.PI/180.0)) *
                 Math.sin(dLon/2) * Math.sin(dLon/2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return R * c;
 }
```

- Σημεία ενός δρόμου με gid some_gid:

Κώδικας 7: Query 1

```
SELECT ST_AsText(geom) AS result FROM (SELECT (ST_DumpPoints(g.geom)).*
    FROM roads AS g WHERE gid=some_gid ) j;
```

- Σημεία δρόμου με gid some_gid που τέμνεται από άλλους δρόμους και gid αυτών ακολουθώντας τον προσανατολισμό του δρόμου με gid some_gid:

Κώδικας 8: Query 2

```
SELECT ST_AsText((ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom)
   AS result,
 b.gid AS resultGid,
 b.name AS resName,
 ST_LineLocatePoint(ST_LineMerge(a.geom),
 (ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom) AS sorting
 FROM roads AS a,roads AS b
 WHERE a.gid=some\_gid
 AND a.gid!=b.gid
 AND ST_intersects(geomfromewkt(a.geom),b.geom)
 ORDER BY ST_LineLocatePoint(ST_LineMerge(a.geom),
 (ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom);
```

- Όνομα και κατευθυντικότητα του δρόμου με gid some_gid:

Κώδικας 9: Query 3

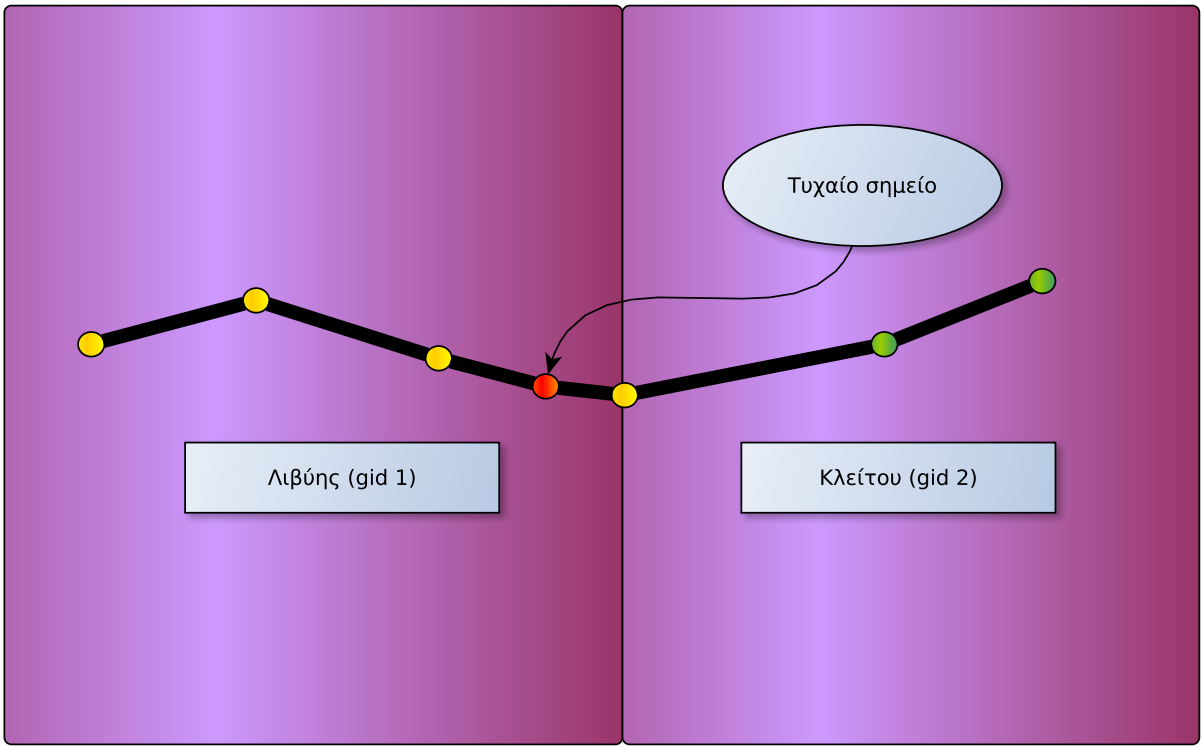
```
SELECT name,oneway FROM roads WHERE gid=some_gid LIMIT 1 ;
```

- Gid, όνομα και κατευθυντικότητα του δρόμου που βρίσκεται το σημείο και το πιο κοντινό γεωγραφικά σημείο πάνω στο δρόμο που βρίσκεται το σημείο με συντεταγμένες (X,Y):

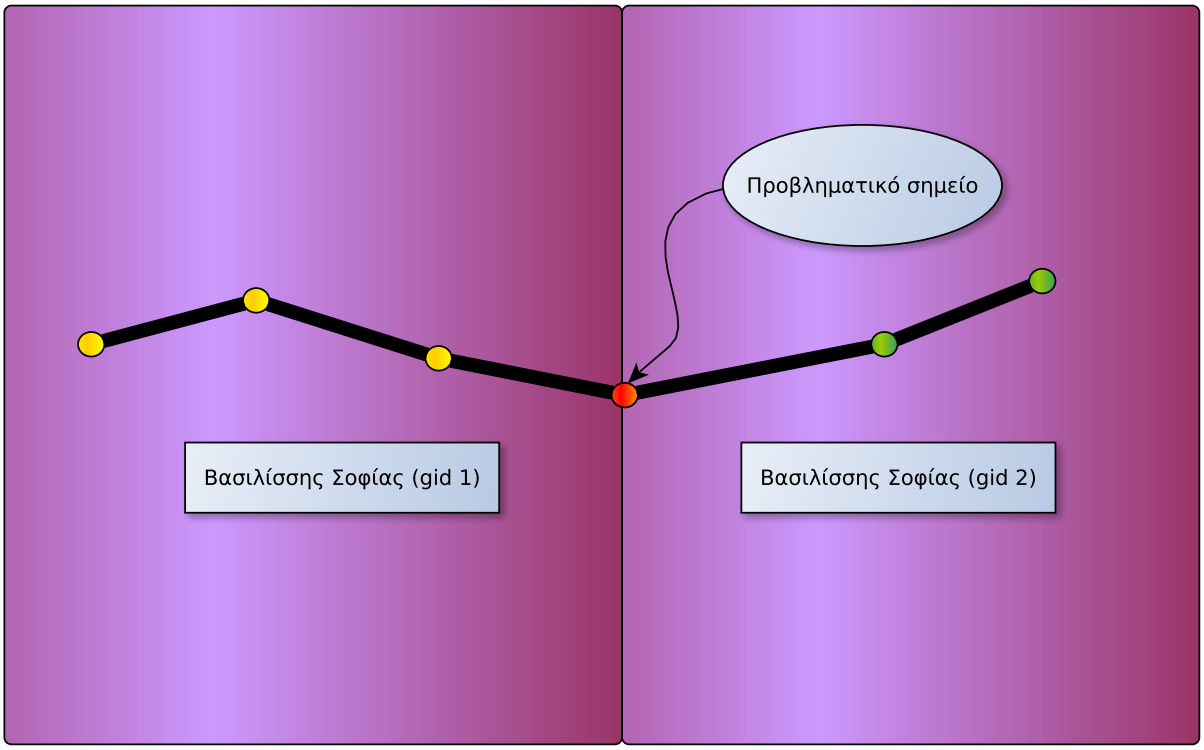
Κώδικας 10: Query 4

```
SELECT gid, name,oneway, st_distance(st_pointfromtext('POINT('X,Y')',
   4326), geom) AS dist,
           ST_AsText(ST_Snap(st_pointfromtext('POINT('X,Y')',
              4326),geom,0.0001)) AS snappedPoint
           FROM roads
           WHERE st_distance(st_pointfromtext('POINT('X,Y')', 4326),
              geom) < 0.002 ORDER BY dist,name,gid LIMIT 1;
```

Επειδή μπορεί να πρόκειται για σημείο διασταύρωσης, το query επιλέγει τον δρόμο που έχει κάποιο σημείο πιο κοντά στο σημείο που εξετάζουμε (Σχήμα 29), αλλά επειδή ούτε κι αυτό επαρκεί (Σχήμα 30), ταξινομεί επιπλέον με βάση αλφαβητική σειρά και με gid (σε περίπτωση που κάποιος μεγάλος δρόμος υπάρχει σε τουλάχιστον δύο κομμάτια στη βάση και το σημείο που εξετάζουμε είναι σημείο διασταύρωσης).



Σχήμα 29: Query 4 (γενική περίπτωση)



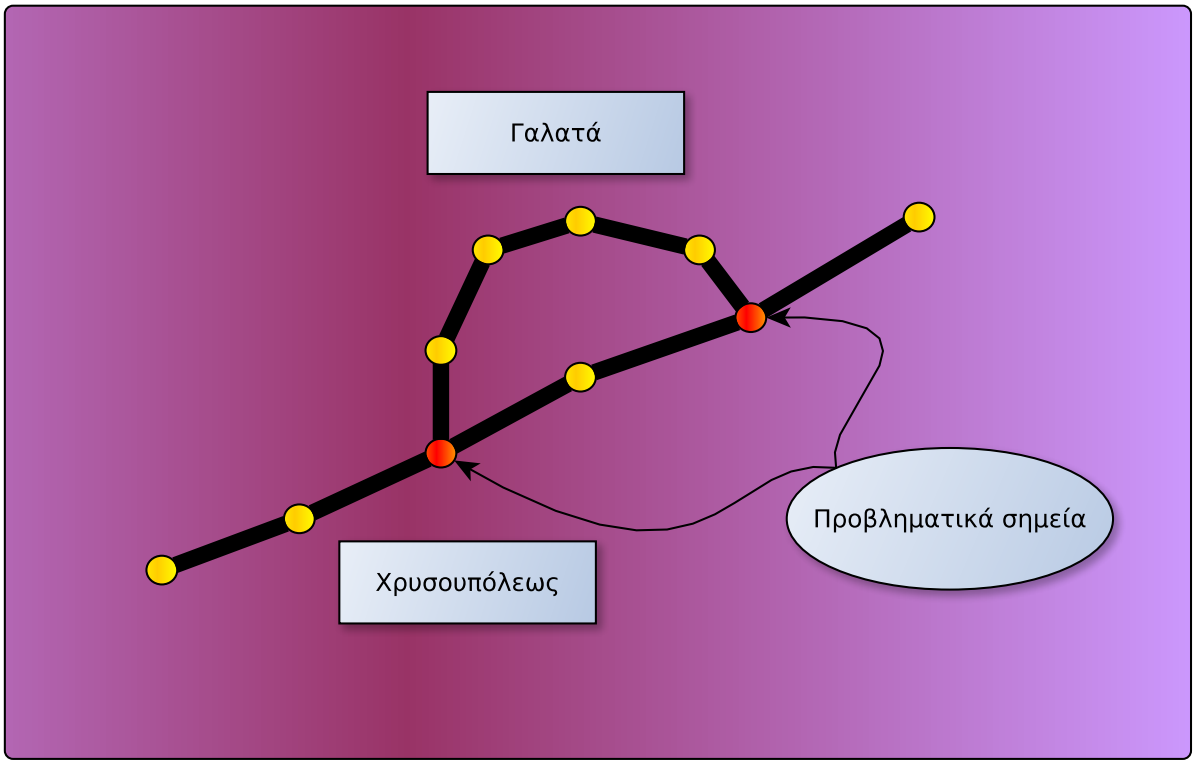
Σχήμα 30: Query 4 (Κρίσιμη περίπτωση)

- Σημείο, gid και όνομα δρόμου με gid gid2 που τέμνεται με τον δρόμο gid1:

Κώδικας 11: Query 5

```
SELECT ST_AsText((ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom)
   AS result,
 b.gid AS resultGid,
 b.name AS resName,
 ST_LineLocatePoint(ST_LineMerge(a.geom),
 (ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom) AS sorting
 FROM roads AS a,roads AS b
 WHERE a.gid=gid1
 AND b.gid =gid2
 AND ST_intersects(geomfromewkt(a.geom),b.geom)
 ORDER BY ST_LineLocatePoint(ST_LineMerge(a.geom),
 (ST_DumpPoints(ST_Intersection(a.geom,b.geom))).geom) LIMIT 1;
```

Το ερώτημα πρέπει να επιστρέφει πάντα ένα σημείο αφού υπάρχει περίπτωση δύο δρόμοι να τέμνονται σε δύο σημεία (Σχήμα 31). Σε αυτή την περίπτωση γίνεται ταξινόμηση με βάση τη σχετική θέση.



Σχήμα 31: Query 5 (Κρίσιμη περίπτωση)

ΑΝΑΦΟΡΕΣ

[1] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem”, *Management Science*, Vol. 6, 1959, pp. 80–91, doi:10.1287/mnsc.6.1.80.

[2] S. Kumar and R. Panneerselvam, A Survey on the Vehicle Routing Problem and Its Variants, *Intelligent Information Management*, Vol. 4, No. 3, 2012, pp. 66-74, doi:10.4236/iim.2012.43010.

[3] Neo.lcc.uma.es (2013, January 7), “Vehicle Routing Problem | NEO Research Group”; http://neo.lcc.uma.es/vrp/ [Προσπελάστηκε 17/8/15].

[4] Bernabe.dorronsoro.es (2006, November), “VRP Web”; http://www.bernabe.dorronsoro.es/vrp/index.html?/Problem_Descriptions/VRPTWDesc.html [Προσπελάστηκε 17/8/15].

[5] Sandhya and Vijay Kumar, “Issues in Solving Vehicle Routing Problem with Time Window and its Variants using Meta heuristics - A Survey”, *International Journal of Engineering and Technology (IJET)*, Vol. 3, No. 6, June, 2013, pp. 668-672.

[6] Marius M. Solomon, Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints, *JSTOR*, Vol. 35, No. 2, Mar. - Apr., 1987, pp. 254-265, doi:10.1287/opre.35.2.254.

[7] Wikipedia contributors (2015, June 26), “2-opt”, Wikipedia, The Free Encyclopedia; https://en.wikipedia.org/w/index.php?title=2-opt&oldid=668709223 [Προσπελάστηκε 17/8/15].

[8] Russell, Stuart J. and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Upper Saddle River, N.J, Prentice Hall, 2010, ch. 4, pp. 94-110.

[9] Maven repository: https://bitbucket.org/Metimdjai/vehicle-routing.

[10] Google Maps API: https://developers.google.com/maps/.

[11] Web.cba.neu.edu (2005, March 24), “Benchmarking Problems”, 2015; http://web.cba.neu.edu/~msolomon/problems.htm [Προσπελάστηκε 17/8/15].

[12] Caric Tonci and Hrvoje Gold, Vehicle Routing Problem, Rijek, Crotia, 2008, doi:10.5772/64.