Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων Και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Υπολογισμός ισοχρονικών καμπύλων χρονοαπόστασης σε οδικά δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Γρίβα

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υπολογισμός ισοχρονικών καμπύλων χρονοαπόστασης σε οδικά δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Γρίβα

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Οκτωβρίου 2012.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Γιάννης Σταύρακας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Ερευνητής Β' ΙΠΣΥ/Ε.Κ. «Αθηνά»

Αθήνα, Οκτώβριος 2012

.....

Νικόλαος Δ. Γρίβας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Καταρχήν, θα ήθελα να ευχαριστήσω τον Αλέξανδρο Εφεντάκη, διδακτορικό υπότροφο του ΙΠΣΥ/Ε.Κ. «Αθηνά», για την πολύτιμη βοήθειά του στην εκπόνηση της παρούσας διπλωματικής εργασίας. Επιπλέον, ευχαριστώ πολύ τον κ. Dieter Pfoser και τον κ. Τιμολέοντα Σελλή για την επίβλεψη της διπλωματικής μου εργασίας και την ευκαιρία που μου έδωσαν να συνεργασθώ με το ΙΠΣΥ/Ε.Κ. «Αθηνά».

Θέλω να ευχαριστήσω, επίσης, τους συμφοιτητές μου Νίκο, Βασίλη, Αδριανό, Γιώργο, Γιάννη και άλλους χάρη στους οποίους η φοίτηση μου στο Πολυτεχνείο έγινε πολύ πιο ευχάριστη και άξια πολλών αναμνήσεων.

Μα κυρίως, θέλω να ευχαριστήσω τους γονείς μου για τη στήριξή τους και τη δυνατότητα που μου έδωσαν να πραγματοποιήσω τις σπουδές μου με αφοσίωση και ευχαρίστηση.

Νίκος Γρίβας, Οκτώβριος 2012

Περίληψη

Ισοχρονικές καμπύλες χρονοαπόστασης είναι οι καμπύλες που ορίζουν ένα πολύγωνο πάνω στον χάρτη το οποίο περιλαμβάνει όλες τις περιοχές οι οποίες είναι προσβάσιμες μέσα σε ένα ορισμένο χρονικό διάστημα από ένα αρχικό σημείο μέσω του οδικού δικτύου.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η δημιουργία μίας εφαρμογής που, δεχόμενη από τον χρήστη το αρχικό σημείο, υπολογίζει με όσο το δυνατόν μεγαλύτερη ταχύτητα και ακρίβεια τις αντίστοιχες ισοχρονικές καμπύλες χρονοαπόστασης και τις παρουσιάζει πάνω στον χάρτη.

Θέλουμε ο υπολογισμός να είναι όσο το δυνατόν ταχύτερος έτσι ώστε η εφαρμογή να είναι άμεσα αποκρίσιμη στην επιλογή του αρχικού σημείου από τον χρήστη. Επιπλέον, θέλουμε όσο το δυνατόν μεγαλύτερη ακρίβεια στον υπολογισμό που σημαίνει ότι οι καμπύλες θα πρέπει να περικλείουν όλες τις προσβάσιμες περιοχές εντός του χρονικού διαστήματος αλλά και να αποκλείει τις μη προσβάσιμες.

Λέξεις Κλειδιά: ισοχρονικές καμπύλες, χρονοαπόσταση, οδικά δίκτυα, εφαρμογή

Abstract

Isochrones define polygons on the map which include all areas that are accessible within a certain time interval from a given starting point through the road network.

The scope of this thesis is the development of an application which, getting as input the starting point, computes as fast and accurately as possible the corresponding isochrones and presents them on the map.

We intend the computation to be fast enough in order for the application to be sharply responsive to the user's selection of the starting point. Moreover, we need the best possible accuracy in the computation meaning that the polygon should include all accessible areas but also it should exclude inaccessible ones.

Keywords: isochrones, travel times, road networks, application

Πίνακας περιεχομένων

1

Εισαγωγή

Η Γεωπληροφορική είναι η επιστήμη που χρησιμοποιεί την πληροφορική για να επιλύσει προβλήματα που αφορούν τον κλάδο της γεωγραφίας και άλλων γεωεπιστημών. Ασχολείται με την αποθήκευση, διαχείριση, επεξεργασία και οπτικοποίηση γεωχωρικών πληροφοριών με στόχο τη δημιουργία χρήσιμων γεωεφαρμογών αλλά και την εξαγωγή χρήσιμων συμπερασμάτων.

1.1 Αντικείμενο διπλωματικής

Στην παρούσα εργασία ασχολούμαστε με τον υπολογισμό των σημείων μίας αστικής περιοχής που μπορεί να φτάσει κάποιος μέσα σε ένα ορισμένο χρονικό διάστημα ξεκινώντας από ένα συγκεκριμένο σημείο και χρησιμοποιώντας το οδικό δίκτυο. Προσπαθούμε δηλαδή να απαντήσουμε σε ερωτήματα όπως “Πού μπορώ να φτάσω σε 20 λεπτά με το αυτοκίνητο ξεκινώντας από το σπίτι μου;” ή “Πόσο διαφέρει η απάντηση σε αυτό αν ξεκινήσω σε ώρα αιχμής;”. Επίσης, ερωτήματα εμπορικής σημασίας όπως “Πόσα σημεία τουριστικής σημασίας είναι σε εύρος 10 λεπτών από ένα ξενοδοχείο;”. Ακόμη, μπορεί να δοθεί απάντηση και στο αντίστροφο ερώτημα δηλαδή “Στους κατοίκους ποιών περιοχών είναι προσπελάσιμο μέσα σε 20 λεπτά ένα πολυκατάστημα;”. Τα παραπάνω

ερωτήματα δίνουν μια αίσθηση του προβλήματος που καλείται να επιλύσει η παρούσα εργασία.

1.1.1 Συνεισφορά

Πιο συγκεκριμένα, σκοπός της εργασίας είναι ο όσο το δυνατόν ταχύτερος και ακριβέστερος υπολογισμός των ισοχρονικών καμπύλων χρονοαπόστασης. Ισοχρονικές καμπύλες χρονοαπόστασης είναι οι καμπύλες που ορίζουν τα όρια των περιοχών που μπορούν να προσπελαστούν σε συγκεκριμένο χρονικό διάστημα μέσω του οδικού δικτύου ξεκινώντας από κάποιο συγκεκριμένο σημείο.

Η συνεισφορά της εργασίας είναι η ανάπτυξη μίας εφαρμογής η οποία δέχεται σαν δεδομένα το αρχικό σημείο και το επιθυμητό χρονικό διάστημα και σαν αποτέλεσμα παρουσιάζει πάνω στον χάρτη τις αντίστοιχες ισοχρονικές καμπύλες χρονοαπόστασης.

1.2 Οργάνωση κειμένου

Στο Κεφάλαιο 2 δίνεται μία σύντομή αναφορά σε παλαιότερες εργασίες που αφορούν τις ισοχρονικές καμπύλες χρονοαπόστασης. Στο Κεφάλαιο 3 περιγράφεται καλύτερα το πρόβλημα καθώς επίσης και όλα τα βήματα που πραγματοποιήσαμε για την αποδοτική επίλυσή του. Στο Κεφάλαιο 4 δίνεται συνοπτικά η σχεδίαση του συστήματος που υλοποιεί την εφαρμογή ενώ στο Κεφάλαιο 5 περιγράφεται αναλυτικά η αρχιτεκτονική του συστήματος που υλοποιήσαμε. Στο Κεφάλαιο 6 παρουσιάζονται τα πειραματικά αποτελέσματα και οι μετρήσεις από την εκτέλεση της εφαρμογής ενώ, τέλος, στο Κεφάλαιο 7 δίνονται τα συμπεράσματα από την όλη εργασία καθώς και πιθανές μελλοντικές επεκτάσεις.

2

Σχετικές εργασίες

Οι ισοχρονικές καμπύλες χρονοαπόστασης, που ορίζονται ως isochrones στα αγγλικά, είναι μία καινούργια έννοια που εμφανίστηκε μόλις το 2008. Από τότε έχει γίνει αρκετή έρευνα πάνω στον υπολογισμό τους και επίσης έχουν αναπτυχθεί κάποιες εφαρμογές που υπολογίζουν και παρουσιάζουν τις ισοχρονικές καμπύλες. Στο κεφάλαιο αυτό αναφέρουμε κάποιες θεωρητικές εργασίες σχετικές με τις ισοχρονικές καμπύλες χρονοαπόστασης καθώς και κάποιες από τις εφαρμογές υπολογισμού τους που έχουν υλοποιηθεί.

2.1 Θεωρητικές εργασίες

Το πρώτο paper που ορίζει την έννοια isochrones είναι το [BGL+08]. Σύμφωνα με αυτό, isochrones είναι τα σύνολα των σημείων από τα οποία ένα συγκεκριμένο σημείο ενδιαφέροντος είναι προσβάσιμο μέσα σε ένα δεδομένο χρονικό διάστημα. Το συγκεκριμένο paper ασχολείται με την υλοποίηση ενός αλγορίθμου που υπολογίζει το σύνολο των ακμών του οδικού δικτύου από τα οποία είναι προσβάσιμο ένα σημείο ενδιαφέροντος μέσα σε ένα συγκεκριμένο χρονικό διάστημα χρησιμοποιώντας τα δημόσια μέσα μεταφοράς και το περπάτημα δεδομένων των στάσεων των λεωφορείων και των δρομολογίων τους. Διαφέρει αρκετά με τη δική μας δουλειά καθώς εμείς

ασχολούμαστε με τη τον υπολογισμό των ισοχρονικών καμπύλων χρησιμοποιώντας αυτοκίνητο και κινούμενοι στο οδικό δίκτυο.

Ένα άλλο paper το οποίο ήταν αρκετά σημαντικό για τη δική μας δουλειά είναι το [MG10]. Αυτό ασχολείται με τον υπολογισμό ολόκληρων περιοχών που είναι προσβάσιμες μέσα σε ένα δεδομένο χρονικό διάστημα από ένα σημείο αρχής. Διαφοροποιείται, δηλαδή, από το προηγούμενο καθώς προσπαθεί να υπολογίσει όχι μόνο τις ακμές του οδικού δικτύου που είναι προσβάσιμες αλλά όλες τις προσβάσιμες περιοχές στην επιφάνεια του χάρτη. Αυτό είναι και το ζητούμενο στην δική μας εργασία αφού οι ισοχρονικές καμπύλες θέλουμε να είναι το περίγραμμα ενός πολυγώνου το οποίο περιλαμβάνει όλες τις προσβάσιμες περιοχές και όχι απλά ένας γράφος με τις προσβάσιμες ακμές.

Το paper αυτό χρησιμοποιεί δύο προσεγγίσεις. Και οι δύο απαιτούν πρώτα τον υπολογισμό των προσβάσιμων ακμών. Η πρώτη προσέγγιση λέει ότι τα isochrones είναι οι επιφάνειες που σχηματίζονται αν παχύνουμε τις προσβάσιμες ακμές. Δηλαδή, ισχυρίζεται ότι εφόσον μπορούμε εγκάιρως να φτάσουμε σε μία ακμή, μπορούμε εγκαίρως να φτάσουμε και σε μία παρακείμενη περιοχή που απέχει λίγα μέτρα από την ακμή. Η άλλη προσέγγιση λέει ότι τα isochrones είναι οι περιοχές που περικυκλώνονται από προσβάσιμες ακμές, Ισχυρίζεται δηλαδή ότι μία περιοχή μπορεί να θεωρηθεί προσβάσιμη εντός του χρόνου όταν οι γύρω δρόμοι της είναι και αυτοί προσβάσιμοι εντός του χρόνου. Αυτή η προσέγγιση είναι αρκετά σημαντική για τον υπολογισμό των ισοχρονικών καμπύλων στην δική μας εργασία όπως θα αναλυθεί στο Κεφάλαιο 3.

Ένα τελευταίο paper που αξίζει να αναφέρουμε είναι το [GBCI11]. Το paper αυτό ασχολείται με τον υπολογισμό των isochrones σε χωρικά δίκτυα πολλαπλών μέσων διακριτών ή συνεχών σε χώρο και χρόνο. Για παράδειγμα, συνεχές σε χρόνο και χώρο μπορεί να θεωρηθεί το περπάτημα. Όμως, η μετακίνηση με το λεωφορείο θεωρείται διακριτή σε χώρο αφού υπάρχουν διακριτές στάσεις αλλά και σε χρόνο αν λάβουμε υπόψη το χρονοπρόγραμμα των λεωφορείων δηλαδή συγκεκριμένες ώρες που περνάει κάποιο λεωφορείο από κάθε στάση.

2.2 Εφαρμογές υπολογισμού

Όπως είπαμε, έχουν αναπτυχθεί και κάποιες εφαρμογές που υπολογίζουν τα isochrones. Κάποιες από αυτές που είναι άξιες προσοχής είναι το Mapnificent [Mapn] και το Isokron [Isok].

Το Mapnificent είναι μία εφαρμογή που υπολογίζει τις προσβάσιμες περιοχές σε δεδομένο χρόνο από κάποιο αρχικό σημείο χρησιμοποιώντας τα δημόσια μέσα μεταφοράς. Λειτουργεί για πολλές πόλεις του κόσμου κυρίως Αμερικάνικες και

Ευρωπαϊκές. Στο σχήμα 2.1 φαίνεται ένα παράδειγμα υπολογισμού των ισοχρονικών καμπύλων για χρονικό διάστημα 15’ από ένα αρχικό σημείο στο Λονδίνο.

Σχήμα 2.1: Ένα παράδειγμα υπολογισμού ισοχρονικών καμπύλων από την εφαρμογή Mapnificent

Παρατηρούμε στο σχήμα 2.1 κάποιες ανεξάρτητες περιοχές που συμπεριλαμβάνονται στις ισοχρονικές καμπύλες. Αυτό συμβαίνει γιατί σε αυτές τις περιοχές υπάρχει σταθμός του υπόγειου σιδηρόδρομου ή στάση λεωφορείου.

Η εφαρμογή Isokron είναι και αυτή παρόμοιας λειτουργίας με το Mapnificent αλλά είναι διαθέσιμη μόνο για δύο Γαλλικές πόλεις, Παρίσι και Ρεν.

Αφού είδαμε σύντομα κάποιες εργασίες σχετικές με τις ισοχρονικές καμπύλες χρονοαπόστασης είμαστε έτοιμοι στα επόμενα κεφάλαια να αναλύσουμε καλύτερα το πρόβλημα που έχουμε να επιλύσουμε και να προχωρήσουμε στη σχεδίαση και υλοποίηση της δικής μας εφαρμογής.

3

Το πρόβλημα και η διαδικασία επίλυσης

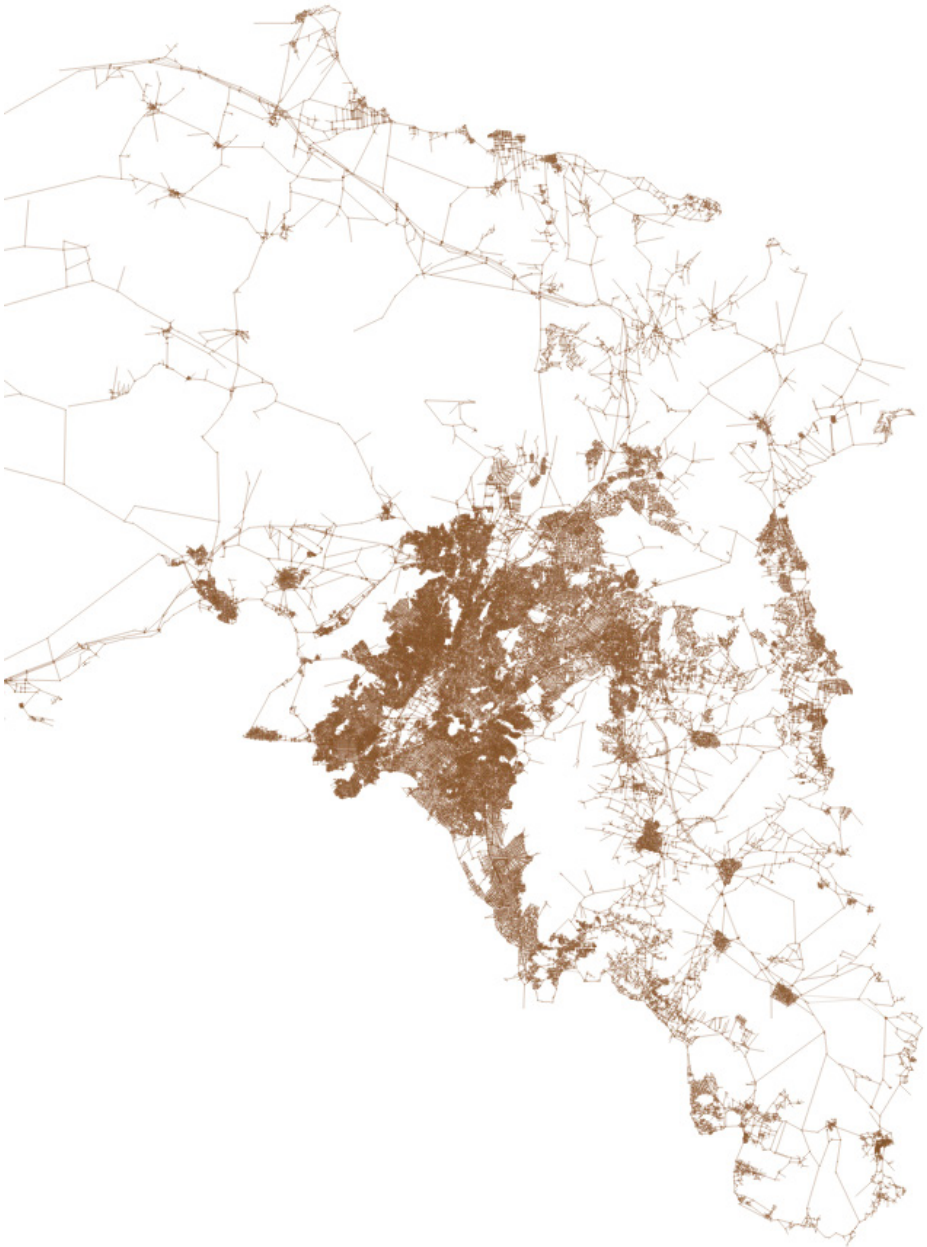
Στο κεφάλαιο αυτό περιγράφεται με ακρίβεια το πρόβλημα και όλα τα βήματα που πραγματοποιούνται για την αποδοτική επίλυσή του. Τα κύρια τμήματα στα οποία εστιάζεται η ανάλυση είναι η περιγραφή του οδικού δικτύου που χρειαζόμαστε για τον υπολογισμό, ο αλγόριθμος συντομότερων μονοπατιών Dijkstra που χρησιμοποιείται καθώς και κάποιες τροποποιήσεις που απαιτούνται για την περίπτωσή μας και, το σημαντικότερο, η περιγραφή των αλγορίθμων που δοκιμάστηκαν για τον υπολογισμό των ισοχρονικών καμπύλων χρονοαπόστασης. Εκτός από αυτά, περιγράφονται και κάποια άλλα μικρότερης σημασίας βήματα που συντελούν όμως στην αποδοτικότερη επίλυση του όλου προβλήματος.

3.1 Ο γράφος του οδικού δικτύου

Το πρώτο πράγμα που χρειαζόμαστε για να κάνουμε τον οποιοδήποτε υπολογισμό είναι δεδομένα για το οδικό δίκτυο. Βασικά, αυτό που μας είναι απαραίτητο είναι ένας κατευθυνόμενος γράφος με βάρη στις ακμές του ο οποίος να περιγράφει το οδικό δίκτυο τουλάχιστον μίας πόλης ή και μεγαλύτερο. Ο γράφος πρέπει να αποτελείται από τα εξής μέρη:

- Κόμβοι οι οποίοι αντιπροσωπεύουν τα σημεία του οδικού δικτύου όπου υπάρχουν διασταυρώσεις. Οι κόμβοι πρέπει να περιγράφονται από τις γεωγραφικές συντεταγμένες τους.
- Ακμές οι οποίες αντιπροσωπεύουν τους δρόμους του οδικού δικτύου. Κάθε ακμή ξεκινάει από έναν κόμβο και καταλήγει σε έναν άλλο.
- Τα βάρη των ακμών τα οποία αντιπροσωπεύουν το χρόνο που χρειάζεται ένα αυτοκίνητο για να διασχίσει το δρόμο που αντιπροσωπεύει η κάθε ακμή.

Στο σχήμα 3.1 φαίνεται για παράδειγμα ένα δίκτυο πάνω στο οποίο δουλέψαμε. Πρόκειται για το οδικό δίκτυο της Αθήνας.



Σχήμα 3.1: Ο γράφος του οδικού δικτύου της Αθήνας που είχαμε στη διάθεσή μας

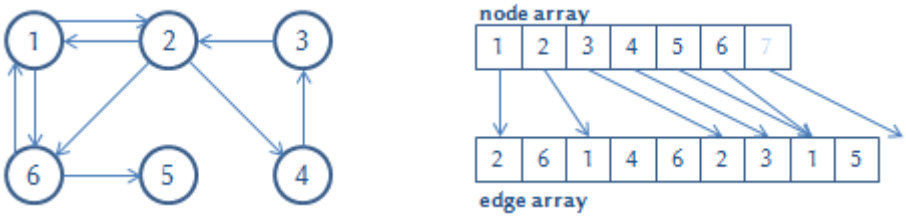
3.1.1 Αναπαράσταση γράφου

Ένα πρώτο δίλημμα που αντιμετωπίσαμε είναι ο τρόπος αναπαράστασης του γράφου του οδικού δικτύου στη μνήμη. Ουσιαστικά είχαμε τρεις επιλογές όπως περιγράφονται αναλυτικά στο [MS08] και συνοπτικά παρακάτω. Οι περιγραφές αφορούν έναν κατευθυνόμενο γράφο G = (V, E) όπου V το σύνολο των κόμβων και E το σύνολο των ακμών με |V| = n.

Adjacency Matrix (πίνακας γειτνίασης). Πρόκειται για έναν n×n πίνακα A. Το A_{ij} είναι 1 όταν η ακμή (i,j) ∈ E, αλλιώς 0. Ουσιαστικά αυτός ο τρόπος αναπαράστασης μπορεί να είναι συμφέρων μόνο σε μικρούς ή μεγάλους αλλά αρκετά πυκνούς γράφους. Για οδικά δίκτυα (των οποίων ο γράφος είναι πολύ αραιός) σίγουρα δεν είναι ο ενδεδειγμένος τρόπος αναπαράστασης καθώς χρησιμοποιεί μεγάλο μέγεθος μνήμης άσκοπα.

Adjacency List (λίστα γειτνίασης). Πρόκειται για ένα διάνυσμα μεγέθους n που στη θέση i έχει αποθηκευμένη τη λίστα με τους κόμβους j για τους οποίους ισχύει (i,j) ∈ E. Η λίστα γειτνίασης είναι ένας πιο συμπαγής τρόπος αναπαράστασης γράφου. Έχει το μειονέκτημα ότι η αναζήτηση ύπαρξης μιας ακμής δεν είναι άμεση όπως στον πίνακα γειτνίασης αλλά χρειάζεται O(deg(i)) όπου deg(i) ο βαθμός του κόμβου i δηλαδή ο αριθμός των ακμών που ξεκινούν από αυτόν.

Adjacency Array (διάνυσμα γειτνίασης). Μπορεί να χρησιμοποιηθεί μόνο σε στατικούς γράφους. Περιλαμβάνει δύο διανύσματα. Το διάνυσμα των ακμών, μεγέθους |E|, περιλαμβάνει όλους τους τελικούς κόμβους των ακμών ταξινομημένους ως προς τους αρχικούς κόμβους. Το διάνυσμα των κόμβων, μεγέθους n+1, περιλαμβάνει για τον κόμβο i τον δείκτη στην πρώτη ακμή του διανύσματος ακμών που έχει σαν αρχικό κόμβο τον i. Το τελευταίο στοιχείο του διανύσματος κόμβων δείχνει στο τέλος του διανύσματος ακμών. Στο σχήμα 3.2 φαίνεται ένα παράδειγμα διανύσματος γειτνίασης.



Σχήμα 3.2: Ένα παράδειγμα διανύσματος γειτνίασης. Αριστερά φαίνεται ο γράφος και δεξιά τα αντίστοιχα διανύσματα κόμβων και ακμών

Το διάνυσμα γειτνίασης έχει το πλεονέκτημα ότι η αναπαράσταση του γράφου γίνεται όσο το δυνατόν πιο συμπαγής καθώς αρκούν δύο διανύσματα ενώ η αναζήτηση

ύπαρξης μιας ακμής είναι της ίδιας πολυπλοκότητας με τη λίστα γειτνίασης. Αυτός είναι και ο λόγος που μας οδήγησε στην επιλογή αυτού του τρόπου για την αναπαράσταση του οδικού δικτύου στη μνήμη.

3.2 Επιλογή του κόμβου αρχής

Στην τελική εφαρμογή θέλουμε ο χρήστης να έχει τη δυνατότητα να επιλέξει ένα σημείο πάνω στο χάρτη από το οποίο θα γίνεται ο υπολογισμός των ισοχρονικών καμπύλων χρονοαπόστασης. Αυτό το σημείο μπορεί να μεταφραστεί σε συντεταγμένες γεωγραφικού πλάτους και μήκους (latitude και longitude). Για να ξεκινήσει, όμως, ο υπολογισμός πρέπει να έχουμε ένα σημείο που αντιστοιχεί σε κόμβο του οδικού δικτύου που έχουμε στη διάθεσή μας. Για αυτό το λόγο χρειαζόμαστε ένα τρόπο να βρίσκουμε αποδοτικά τον κοντινότερο κόμβο του οδικού δικτύου σε ένα συγκεκριμένο γεωγραφικό σημείο του οποίου έχουμε τις συντεταγμένες.

3.2.1 Indexing των κόμβων με R* tree

Μία απλή λύση είναι να υπολογίζουμε με τις κατάλληλες μαθηματικές πράξεις την απόσταση του δεδομένου σημείου από όλους τους κόμβους του οδικού δικτύου κι έτσι να βρίσκουμε αυτόν με τη μικρότερη απόσταση. Όμως, επειδή θέλουμε η εφαρμογή μας να τρέχει όσο το δυνατόν ταχύτερα μετά την επιλογή του σημείου από τον χρήστη πρέπει να βρούμε έναν αποδοτικότερο τρόπο υπολογισμού ο οποίος προϋποθέτει το indexing των κόμβων του οδικού δικτύου με βάση τις συντεταγμένες τους.

Για αυτό το σκοπό μια έτοιμη και αποδοτική λύση είναι το R* tree [BKSS90]. Το R* tree είναι μία δενδρική δομή που χρησιμοποιείται για το indexing χωρικών δεδομένων. Η κεντρική ιδέα πίσω από τη δομή αυτή είναι η ομαδοποίηση κοντινών κόμβων και η αναπαράστασή τους με το ελάχιστο πολύγωνο που τα οριοθετεί στο υψηλότερο επίπεδο του δένδρου.

Για να γίνει το indexing απαιτούνται μερικά δευτερόλεπτα, όμως αυτό συμβαίνει μία φορά κατά την έναρξη της εφαρμογής. Από εκεί και πέρα, ο υπολογισμός του πλησιέστερου κόμβου σε ένα δεδομένο σημείο του χάρτη γίνεται πολύ γρήγορα κι έτσι η εφαρμογή μας δε χάνει αισθητό χρόνο σε αυτό το τμήμα της.

Για την υλοποίηση του χρησιμοποιήσαμε το [ELKI] το οποίο είναι ένα framework που χρησιμοποιείται για την ανάπτυξη εφαρμογών που υποστηρίζονται από δομές indexing.

3.3 Ο αλγόριθμος Dijkstra

Ο κορμός της εφαρμογής δεν είναι άλλος από τον υπολογισμό των σημείων του οδικού δικτύου στα οποία μπορεί να φτάσει ένα όχημα μέσα στο ζητούμενο χρονικό διάστημα ξεκινώντας από το αρχικό σημείο. Για τον υπολογισμό αυτό καταλληλότερος είναι ο αλγόριθμος συντομότερων μονοπατιών Dijkstra [CLRS09]. Βέβαια, για τις ανάγκες της δικής μας εφαρμογής χρειάστηκαν κάποιες τροποποιήσεις στον αλγόριθμο. Έτσι, σε αυτήν την παράγραφο, μετά από μια συνοπτική περιγραφή του αλγορίθμου Dijkstra, περιγράφουμε και αυτές τις απαιτούμενες τροποποιήσεις.

3.3.1 Συνοπτική περιγραφή αλγορίθμου Dijkstra

Δεδομένου ενός κατευθυνόμενου γράφου G = (V, E) με μη αρνητικά βάρη ακμών και ενός αρχικού κόμβου s, ο αλγόριθμος Dijkstra υπολογίζει τις συντομότερες διαδρομές από το σημείο αυτό προς όλους τους κόμβους του γράφου.

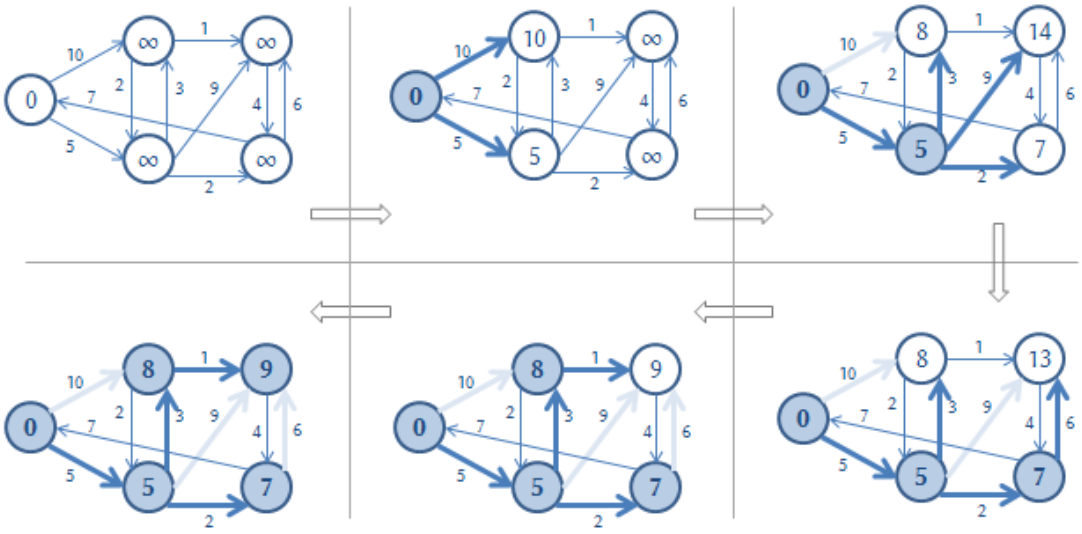
Ο αλγόριθμος διατηρεί έναν πίνακα D με την τρέχουσα υπολογισμένη ελάχιστη απόσταση κάθε κόμβου από τον s. Αρχικά, $D[s] = 0$ ενώ $D[u] = \infty$ για κάθε $u \in V - \{s\}$. Επίσης, διατηρεί ένα σύνολο S που αποτελείται από τους κόμβους για τους οποίους έχει ήδη προσδιοριστεί η ελάχιστη διαδρομή και το οποίο αρχικά είναι κενό.

Η επαναληπτική διαδικασία είναι η εξής:

- Επιλέγεται ο κόμβος $u \in V - S$ με την ελάχιστη εκτιμώμενη απόσταση στον πίνακα D.
- Ο u προστίθεται στο σύνολο S και "χαλαρώνουν" όλες οι ακμές που ξεκινούν από αυτόν. Με τον όρο "χαλάρωμα" μιας ακμής εννοούμε την ανανέωση της εκτιμώμενης απόστασης για τον τελικό κόμβο v της ακμής δηλαδή γίνεται $D[v] = D[u] + w(u,v)$.
- Ο αλγόριθμος τερματίζει όταν στο S προστεθεί και ο τελευταίος κόμβος του γράφου.

Στο σχήμα 3.3 φαίνεται η διαδικασία εκτέλεσης του αλγορίθμου για ένα μικρό γράφο.

Η πιο κρίσιμη για την απόδοση του αλγορίθμου διαδικασία είναι η επιλογή του κόμβου u με την ελάχιστη εκτιμώμενη απόσταση. Για την επιλογή αυτή χρησιμοποιείται ένα priority queue (ουρά προτεραιότητας). Μια καλή επιλογή είναι η χρήση ενός binary heap που έχει πολυπλοκότητα O(logn) στην εισαγωγή ενός στοιχείου. Η ανάκτηση του ελάχιστου στοιχείου έχει πολυπλοκότητα O(1), όμως η διαγραφή του που απαιτεί ανακατάταξη του heap έχει πολυπλοκότητα O(logn). Η πολυπλοκότητα του αλγορίθμου Dijkstra με binary heap σαν priority queue είναι $O(|E| + |V| \log|V|)$.



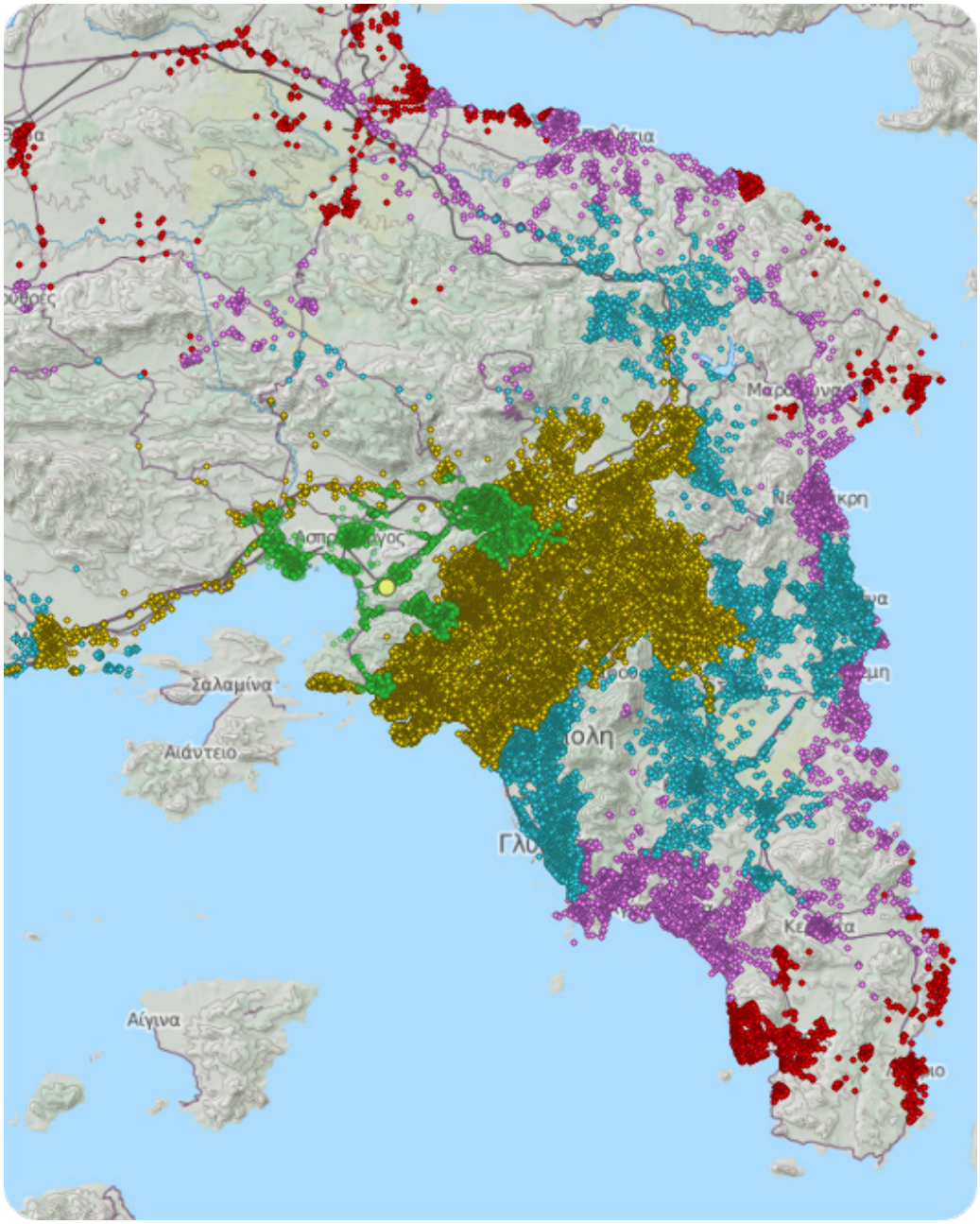
Σχήμα 3.3: Ένα παράδειγμα εκτέλεσης του αλγορίθμου Dijkstra σε ένα μικρό γράφο

3.3.2 Προσαρμογή στην εφαρμογή μας

Για τη δική μας εφαρμογή, ο αλγόριθμος του Dijkstra χρειάζεται προσαρμογή. Πρώτα από όλα, δεν μας ενδιαφέρουν τα συντομότερα μονοπάτια αυτά καθ' εαυτά αλλά μόνο οι χρόνοι των συντομότερων μονοπατιών. Δηλαδή το ζητούμενο για εμάς είναι ο πίνακας D μετά την ολοκλήρωση της εκτέλεσης του αλγορίθμου.

Επιπλέον, πολύ σημαντικό είναι ότι εμείς νοιαζόμαστε μόνο για τους κόμβους που είναι προσπελάσιμοι μέσα στο καθορισμένο χρονικό διάστημα. Αυτό σημαίνει ότι όταν επιλεγεί από το priority queue ένας κόμβος με κόστος μεγαλύτερο από αυτό το χρονικό διάστημα τότε ο αλγόριθμος μπορεί να τερματιστεί καθώς έχουν βρεθεί όλοι οι κόμβοι που είναι εντός χρόνου. Αυτό μας δίνει το πλεονέκτημα ότι μπορούμε να δουλεύουμε σε μεγάλα οδικά δίκτυα (π.χ. σε επίπεδο ηπείρου) με χρόνο εκτέλεσης του Dijkstra παρόμοιο αν δεν αυξήσουμε το χρονικό διάστημα. Κι αυτό γιατί δεν εξαντλεί όλους τους κόμβους του δικτύου παρά μόνο αυτούς που είναι εντός χρόνου.

Στο σχήμα 3.4 παρουσιάζεται το αποτέλεσμα του Dijkstra με σημείο αρχής ένα σημείο στο οδικό δίκτυο της Αθήνας. Ουσιαστικά πρόκειται για τους ισοχρονικούς κόμβους για 5 διαφορετικά χρονικά διαστήματα (10', 20', ... , 50').



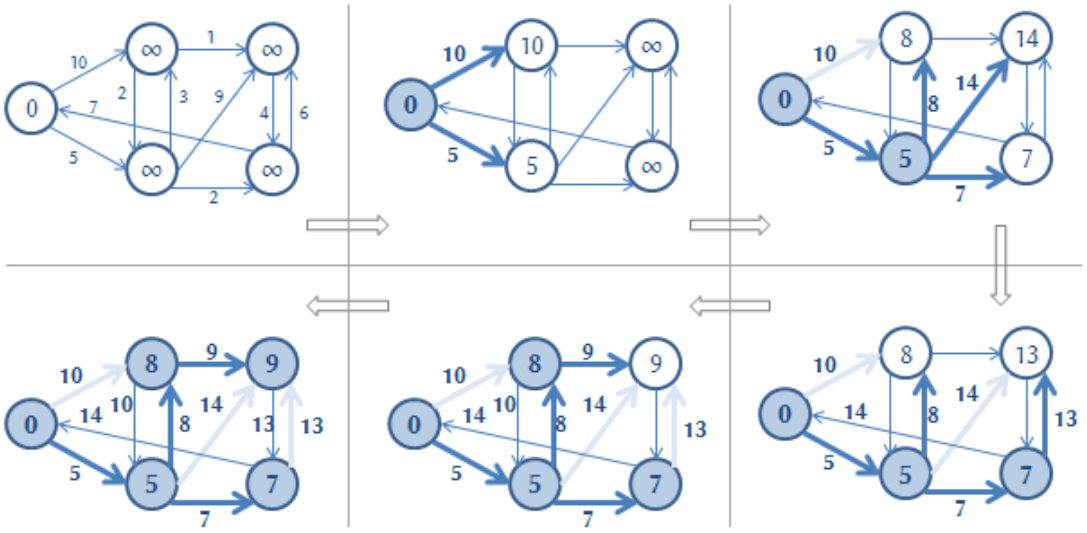
Σχήμα 3.4: Οι ισοχρονικοί κόμβοι για 10', 20', ... , 50' με διαφορετικά χρώματα (από πράσινο για 10' έως κόκκινο για 50') ως αποτέλεσμα της εκτέλεσης του αλγορίθμου Dijkstra

3.3.3 Υπολογισμός ισοχρονικών ακμών

Όπως θα δούμε στην επόμενη παράγραφο που αφορά στις προσεγγίσεις που κάναμε στον υπολογισμό των ισοχρονικών καμπύλων, σε μία περίπτωση είναι απαραίτητη η πληροφορία χρόνου για τις ακμές του δικτύου. Δηλαδή εκτός από τους ισοχρονικούς κόμβους (κόμβους εντός χρόνου) που υπολογίζουμε μέσω Dijkstra θέλουμε και τις ισοχρονικές ακμές. Ισοχρονικές ακμές ονομάζουμε τις ακμές που, ξεκινώντας από το

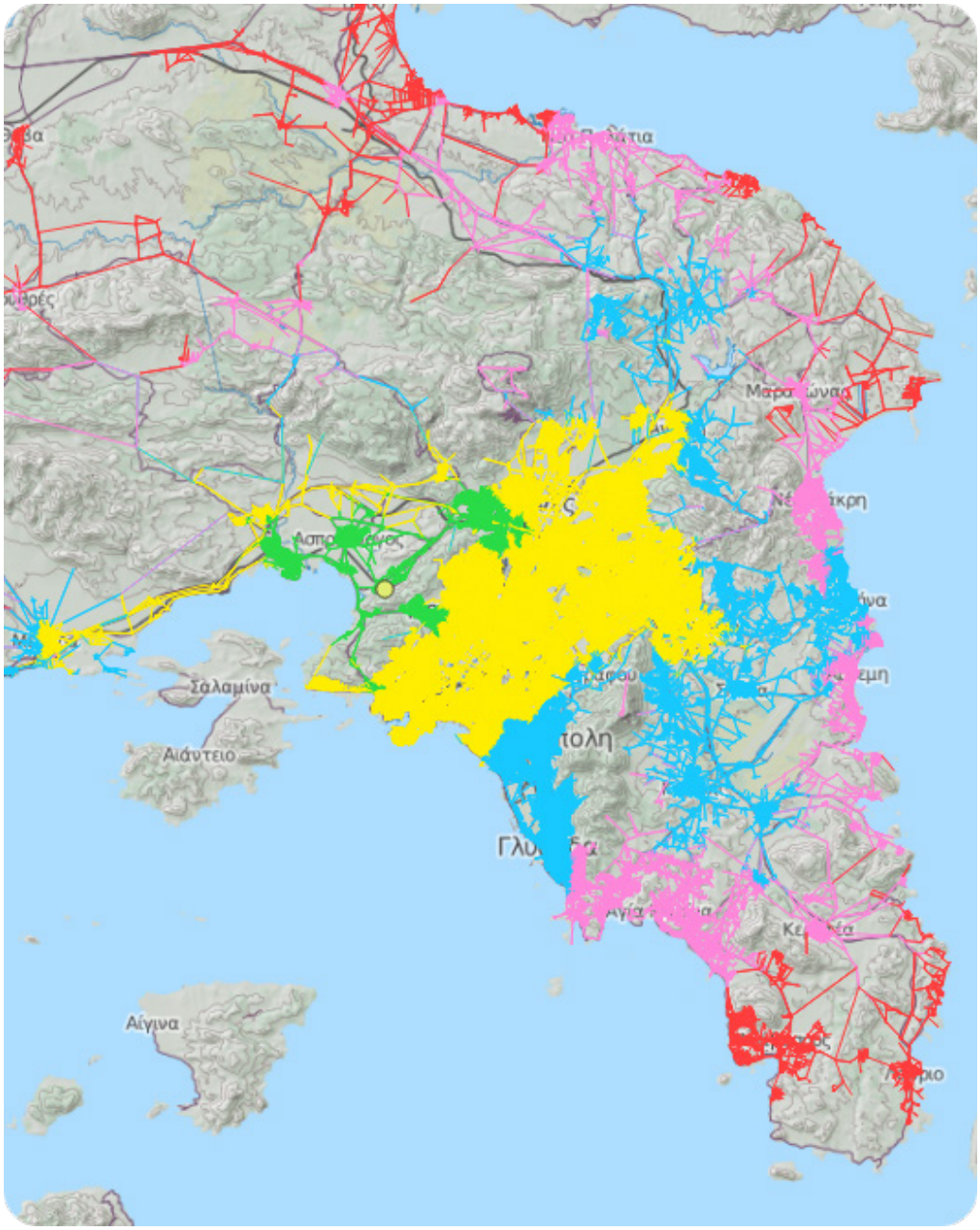
αρχικό σημείο, μπορούμε εντός χρόνου να φτάσουμε στον αρχικό τους κόμβο και να τις διασχίσουμε φτάνοντας στον τελικό τους κόμβο.

Ο ελάχιστος χρόνος διάσχισης μίας ακμής είναι ο ελάχιστος χρόνος για να φτάσουμε στον αρχικό της κόμβο συν το βάρος της ακμής. Αν αυτός ο χρόνος είναι μικρότερος ή ίσος με το καθορισμένο χρονικό διάστημα τότε η ακμή είναι ισοχρονική. Όμως, αυτός ο χρόνος μπορεί πολύ εύκολα να υπολογιστεί κατά την εκτέλεση του Dijkstra. Κι αυτό γιατί ο ελάχιστος χρόνος για να φτάσουμε στον αρχικό της κόμβο καθορίζεται τη στιγμή που αυτός ο κόμβος εξάγεται από το priority queue. Έτσι, κάθε φορά που εξάγουμε κάποιον κόμβο από το priority queue μπορούμε να υπολογίσουμε τον ελάχιστο χρόνο διάσχισης κάθε ακμής που ξεκινάει από αυτόν προσθέτοντας το κόστος του με το βάρος της ακμής. Οπότε, τελικά έχουμε και τις ισοχρονικές ακμές. Στο σχήμα 3.5 φαίνεται το ίδιο παράδειγμα εκτέλεσης του Dijkstra με το σχήμα 3.3 μόνο που τώρα δίπλα σε κάθε ακμή δεν έχουμε το βάρος της αλλά τον ελάχιστο χρόνο διάσχισής της όταν αυτός υπολογίζεται.



Σχήμα 3.5: Το ίδιο παράδειγμα εκτέλεσης του αλγορίθμου Dijkstra με υπολογισμό των χρόνων διάσχισης των ακμών

Στο σχήμα 3.6 παρουσιάζεται η αντίστοιχη περίπτωση με το σχήμα 3.4 για τις ισοχρονικές ακμές αυτή τη φορά.



Σχήμα 3.6: Οι ισοχρονικές ακμές για 10', 20', ... , 50' με διαφορετικά χρώματα (από πράσινο για 10' έως κόκκινο για 50') ως αποτέλεσμα της εκτέλεσης του προσαρμοσμένου αλγορίθμου Dijkstra

3.3.4 Priority queue Dial

Προκειμένου να επιταχύνουμε λίγο τον αλγόριθμο του Dijkstra δοκιμάσαμε τη χρήση μιας ακόμα ουράς προτεραιότητας. Πρόκειται για την ουρά προτεραιότητας του Dial [Dia69] [CGR93].

Η συγκεκριμένη υλοποίηση διατηρεί ένα διάνυσμα από buckets με το i-οστό bucket να συμπεριλαμβάνει όλους τους κόμβους για τους οποίους η τρέχουσα εκτίμηση απόστασης είναι ίση με i. Όταν η εκτίμηση απόστασης ενός κόμβου αλλάζει τότε αυτός ο

κόμβος αφαιρείται από το bucket στο οποίο βρισκόταν (αν είχε μη άπειρη εκτίμηση απόστασης) και προστίθεται στο νέο. Η υλοποίηση διατηρεί και έναν δείκτη L. Αρχικά, L=0. Ισχύει πάντα η ιδιότητα ότι όλα τα buckets με i < L είναι άδεια. Η εξαγωγή ενός στοιχείου από την ουρά προτεραιότητας γίνεται επιλέγοντας έναν κόμβο από το bucket L ή αν αυτό είναι άδειο το L αυξάνεται κατά 1 μέχρι το bucket L να μην είναι άδειο.

Για την υλοποίηση αυτή χρειάζεται ένα διάνυσμα μήκους όσο και η μέγιστη απόσταση που βρίσκει ο Dijkstra. Στην περίπτωσή μας, επειδή ο αλγόριθμος τερματίζεται μόλις ξεπεραστεί το δεδομένο χρονικό διάστημα μπορούμε να έχουμε ένα διάνυσμα με μήκος όσο και αυτό το χρονικό διάστημα. Έχει όμως αποδειχθεί ότι δεν χρειάζεται καν τέτοιο μέγεθος διανύσματος. Συγκεκριμένα, αποδεικνύεται ότι μόνο C+1 διαδοχικά buckets μπορεί να είναι κατειλημμένα σε κάθε δεδομένη χρονική στιγμή όπου C το βάρος της ακμής του δικτύου με το μεγαλύτερο βάρος. Οπότε, η χρήση ενός διανύσματος μεγέθους C+1 είναι αρκετός για την υλοποίηση της ουράς προτεραιότητας.

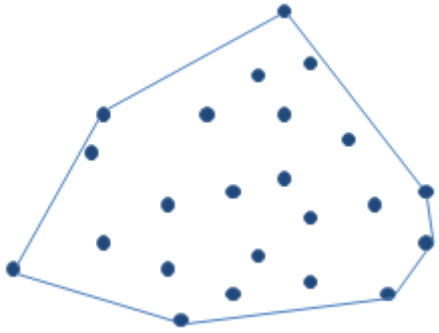
Με τη χρήση της συγκεκριμένης ουράς προτεραιότητας ο αλγόριθμος του Dijkstra τρέχει με πολυπλοκότητα O(|E|+|V|·C). Το γεγονός αυτό δίνει μία σημαντική επιτάχυνση στην εκτέλεση του αλγορίθμου. Στο Κεφάλαιο 6 δίνεται πλήρης ανάλυση με μετρήσεις χρόνων εκτέλεσης για την επιτάχυνση που επιτυγχάνεται.

3.4 Ισοχρονικές καμπύλες

Το πιο σημαντικό βήμα στην εκτέλεση της εφαρμογής είναι, με δεδομένα ισοχρονικούς κόμβους και ακμές από την εκτέλεση του Dijkstra, ο υπολογισμός των ισοχρονικών καμπυλών που είναι και η ουσία της εφαρμογής. Για το σκοπό αυτό κάναμε αρκετά πειράματα και δοκιμάσαμε διάφορους αλγορίθμους ώστε να βρούμε ποιά λύση μπορεί να δώσει το καλύτερο αποτέλεσμα από πλευράς ταχύτητας και ακρίβειας. Σε αυτή την παράγραφο, περιγράφουμε συνοπτικά την διαδρομή αυτή μέχρι την τελική επιλογή και τους λόγους που μας οδήγησαν σε αυτήν.

3.4.1 Convex Hull

Η πρώτη προσέγγιση που κάναμε είναι το Convex Hull. Το Convex Hull ενός συνόλου σημείων σε ένα Ευκλίδειο επίπεδο είναι το ελάχιστο κυρτό πολύγωνο που περιέχει όλα τα σημεία. Πιο περιγραφικά, θα λέγαμε ότι είναι το πολύγωνο που θα σχηματιζόταν αν τεντώναμε μία ελαστική κορδέλα γύρω από τα σημεία. Στο σχήμα 3.7 φαίνεται ένα παράδειγμα του Convex Hull ενός μικρού συνόλου σημείων.

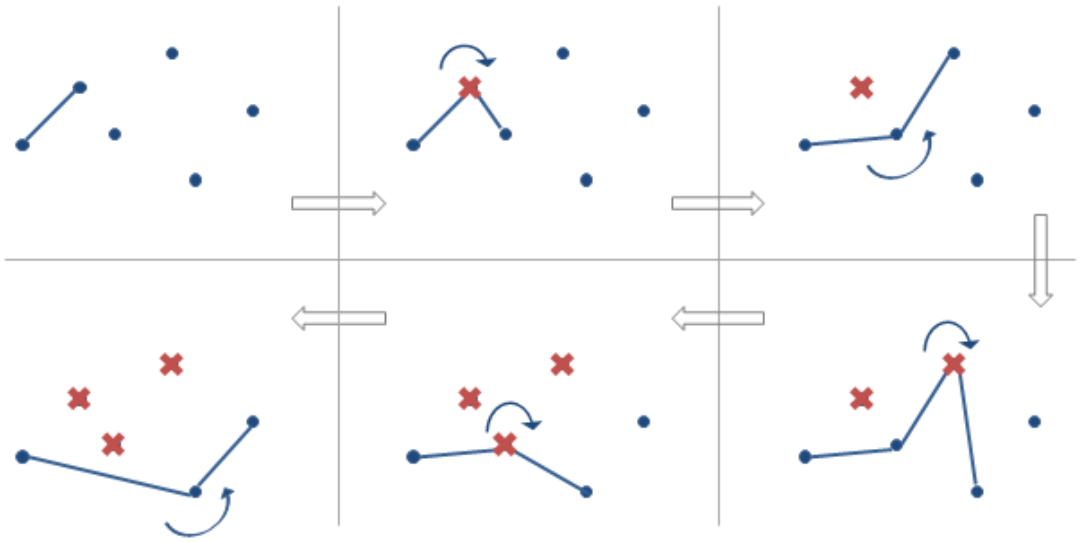


Σχήμα 3.7: Το Convex Hull ενός μικρού συνόλου σημείων

Για την υλοποίηση του Convex Hull χρησιμοποιήσαμε τον αλγόριθμο Monotone Chain Convex Hull [MCCH]. Ακολουθεί μία σύντομη περιγραφή του:

- Ο αλγόριθμος ξεκινάει με την ταξινόμηση των διδιάστατων δεδομένων σημείων ως προς τη συντεταγμένη x. Σε περίπτωση ισότητας ταξινομεί ως προς τη συντεταγμένη y.
- Στη συνέχεια, αρχίζει τον υπολογισμό του lower hull (λίστα L) δηλαδή του τμήματος του Convex Hull που είναι ορατό από την κάτω πλευρά του. Αφού προσθέσει στην L τα δύο πρώτα (αριστερότερα) σημεία αρχίζει την επαναληπτική διαδικασία από το 3ο έως το τελευταίο σημείο (δηλαδή για i από 3 έως n):
    - Όσο το i-οστό σημείο σε συνδυασμό με τα δύο τελευταία της L δε σχηματίζει αριστερόστροφη στροφή, αφαιρείται το τελευταίο σημείο από την L.
    - Προστίθεται το i-οστό σημείο στην L.
- Με αντίστοιχο τρόπο υπολογίζεται και το upper hull. Για το upper hull προστίθενται στη λίστα U πρώτα τα δύο δεξιότερα σημεία και η επαναληπτική διαδικασία γίνεται για i από n-2 έως 1.
- Η ένωση του lower hull με το upper hull αφού αφαιρέσουμε τα τελευταία σημεία από το καθένα μας δίνει το Convex Hull.

Στο σχήμα 3.8 δίνεται ένα πολύ απλό παράδειγμα υπολογισμού του lower hull για ένα πολύ μικρό σύνολο σημείων.



Σχήμα 3.8: Ένα παράδειγμα εκτέλεσης του αλγορίθμου Monotone Chain Convex Hull για το lower hull ενός μικρού συνόλου σημείων

Τα πλεονεκτήματα του Convex Hull είναι η γρήγορη εκτέλεσή του, το μικρό σε μέγεθος αποτέλεσμά του και η εύκολη υλοποίησή του. Η πολυπλοκότητά του είναι O(n·logn) καθώς κάνει αρχικά την ταξινόμηση σε O(n·logn) και στη συνέχεια κατασκευάζει τα lower και upper hulls σε O(n).

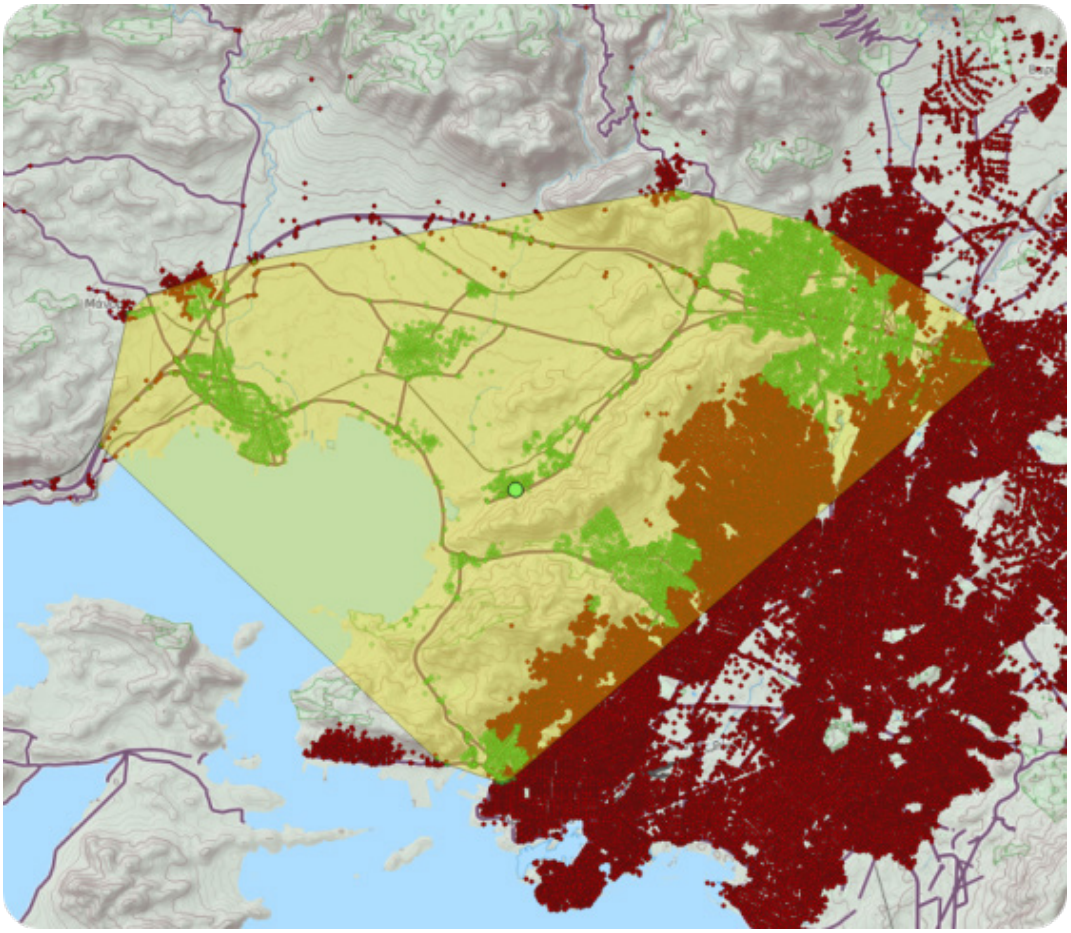
Όμως, έχει ένα σημαντικό μειονέκτημα για την εφαρμογή μας. Ενώ περιλαμβάνει όλους τους ισοχρονικούς κόμβους χωρίς εξαίρεση, περιλαμβάνει και πολλούς κόμβους που δεν είναι ισοχρονικοί δηλαδή για την προσπέλασή τους χρειάζεται χρόνος μεγαλύτερος από το καθορισμένο χρονικό διάστημα. Αυτό φαίνεται καλύτερα μέσα από το σχήμα 3.9 που είναι ένα παράδειγμα Convex Hull στο οδικό δίκτυο της Αθήνας.

Για να γίνουμε πιο συγκεκριμένοι πήραμε κάποιες μετρήσεις για 20.000 αρχικούς κόμβους και 5 διαφορετικά χρονικά διαστήματα για να δούμε το μέσο ποσοστό των ανεπιθύμητων σημείων που περιλαμβάνει το Convex Hull. Αυτές παρουσιάζονται στον Πίνακα 3.1.

χρονικό όριο	**10'**	**20'**	**30'**	**40'**	**50'**
μέσο % μη ισοχρονικών κόμβων μέσα στο Convex Hull	15,4 %	7,7 %	4,2 %	2,1 %	1,0 %

Πίνακας 3.1

Όπως παρατηρούμε, το ποσοστό μειώνεται όσο αυξάνεται το χρονικό όριο αλλά είναι αρκετά μεγάλο ώστε να χρησιμοποιήσουμε το Convex Hull για την εφαρμογή μας.

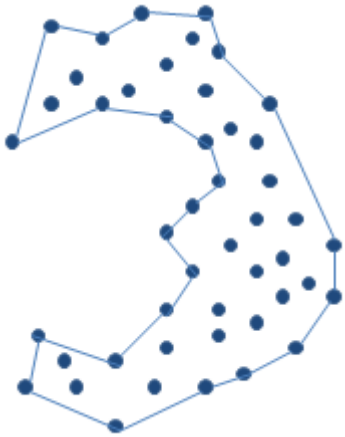


Σχήμα 3.9: Οι πράσινοι είναι οι ισοχρονικοί κόμβοι (για 10') ενώ οι κόκκινοι οι μη ισοχρονικοί. Παρατηρούμε ότι πολλοί κόκκινοι βρίσκονται εντός του Convex Hull.

3.4.2 Concave Hull

Η επόμενη προσέγγιση που κάναμε είναι το Concave Hull [PO12]. Για το Concave Hull δεν υπάρχει κάποιος καθολικός ορισμός αλλά διαισθητικά είναι το Convex Hull επεξεργασμένο έτσι ώστε οι μεγάλες περιοχές χωρίς σημεία που περικλείει να αφαιρούνται. Στο σχήμα 3.10 φαίνεται ένα Concave Hull ενός συνόλου σημείων.

Το κίνητρο για τη χρήση του Concave Hull είναι ότι θα μπορούσε να εξαφανίσει το κύριο μειονέκτημα του Convex Hull που είναι η συμπερίληψη μη ισοχρονικών κόμβων στο παραχθέν πολύγωνο. Δηλαδή, κόβοντας μεγάλες περιοχές του Convex Hull που δεν περιέχουν ισοχρονικούς κόμβους, πιθανότατα θα αποκλείει και πολλούς μη ισοχρονικούς κόμβους που πριν ήταν μέσα στο πολύγωνο.



Σχήμα 3.10: Το Concave Hull ενός συνόλου σημείων

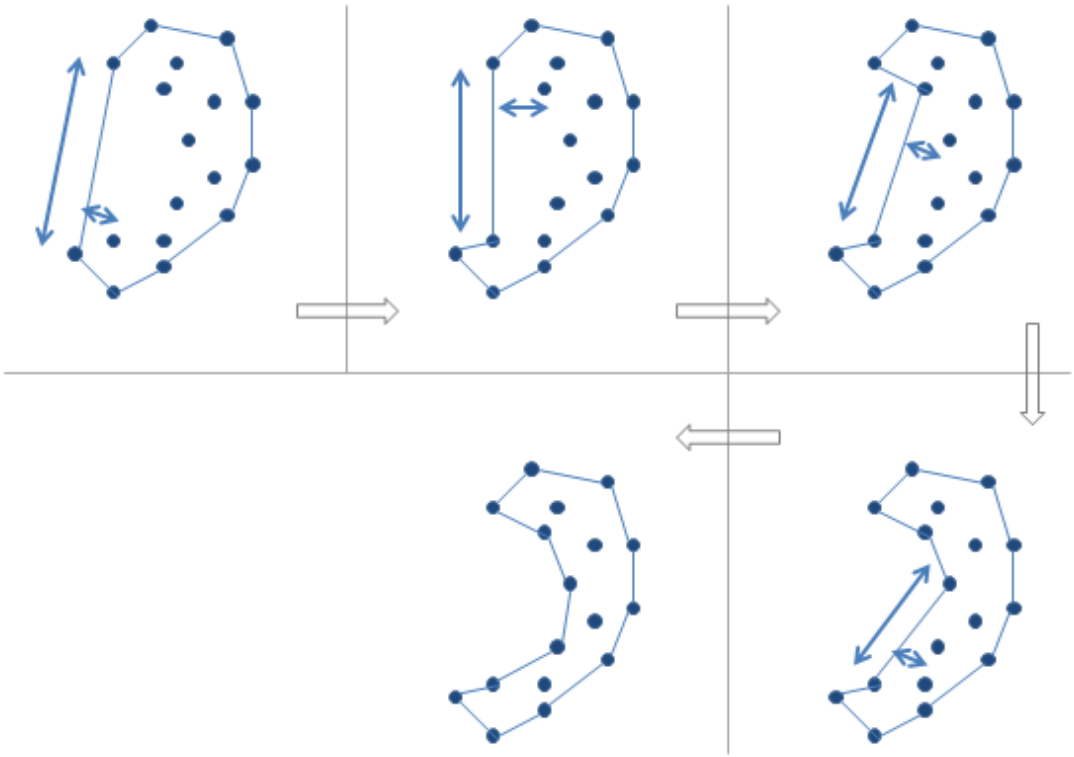
Εφόσον δεν υπάρχει καθολικός ορισμός, δεν υπάρχει και καθολικός αλγόριθμος που να υπολογίζει το Concave Hull. Υπάρχουν διάφορες προσεγγίσεις και αλγόριθμοι που προσπαθούν να δώσουν ένα ικανοποιητικό αποτέλεσμα. Οι αλγόριθμοι αυτοί είναι παραμετρικοί. Το Concave Hull στο σχήμα 3.10 θα μπορούσε να είναι αρκετά διαφορετικό αν η παραμετροποίηση του αλγορίθμου που έδωσε το αποτέλεσμα ήταν διαφορετική. Αυτό είναι σίγουρα ένα μειονέκτημα καθώς ακόμα κι αν υλοποιήσουμε έναν αλγόριθμο, σε κάποιες εκτελέσεις το αποτέλεσμα μπορεί να είναι ως κάποιο βαθμό απρόβλεπτο.

Για να δοκιμάσουμε κατά πόσο το Concave Hull θα ήταν μια καλή λύση για την εφαρμογή μας χρησιμοποιήσαμε μία έτοιμη υλοποίηση ενός αλγορίθμου υπολογισμού του, το [dkuCH]. Ακολουθεί μια συνοπτική περιγραφή της λειτουργίας του αλγορίθμου.

- Αρχικά, υπολογίζεται το Convex Hull όπως περιγράφηκε στην προηγούμενη παράγραφο.
- Για κάθε ακμή του Convex Hull γίνονται τα εξής:
    - Υπολογίζεται το μήκος της ακμής.
    - Αναζητείται το κοντινότερο στην ακμή σημείο εντός του πολυγώνου.
    - Υπολογίζεται η κάθετη απόσταση του κοντινότερου σημείου και της ακμής.
    - Αν ο λόγος του μήκους της ακμής προς την απόσταση του κοντινότερου μέσα σημείου είναι μεγαλύτερος από μία δεδομένη παράμετρο N τότε η αρχική ακμή παύει να υπάρχει και δημιουργούνται δύο ακμές που η κάθε μια έχει άκρα ένα σημείο της αρχικής ακμής και το κοντινότερο μέσα σημείο.

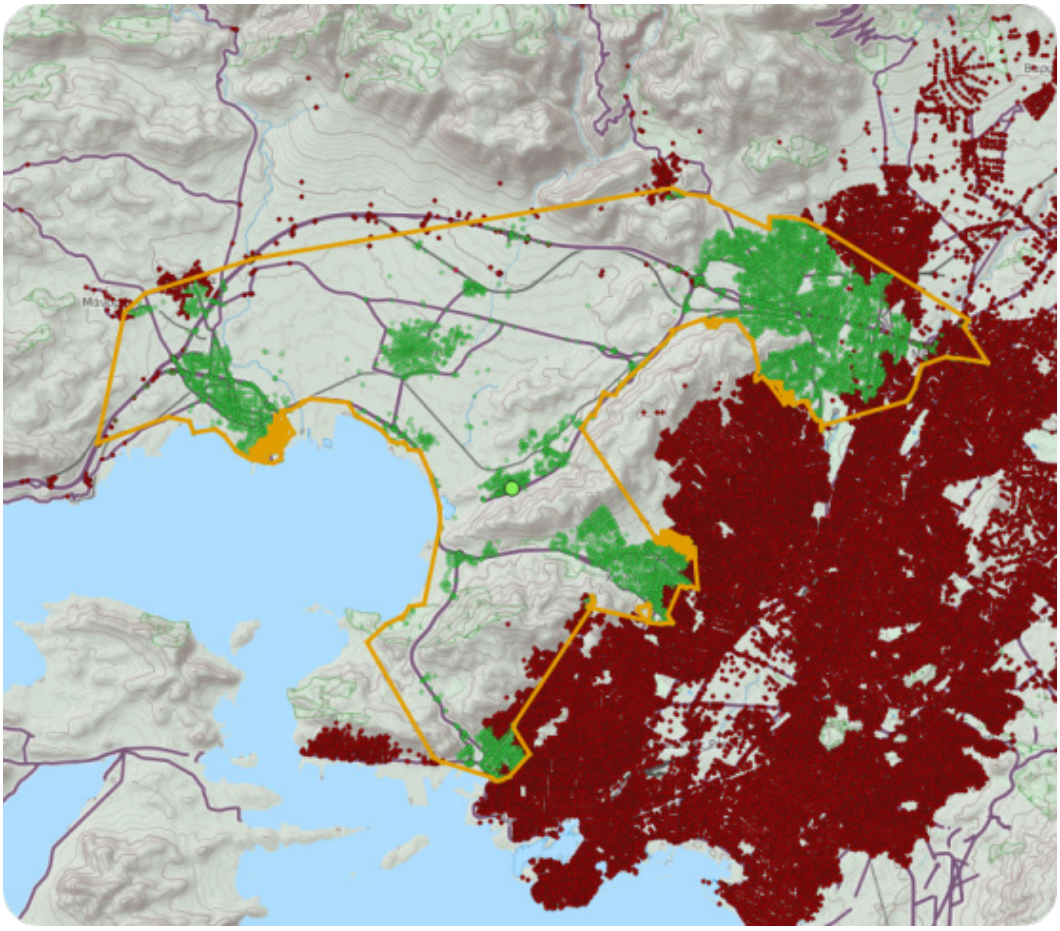
- Η διαδικασία συνεχίζεται για όλες τις ακμές του Convex Hull αλλά και για τις νέες ακμές που δημιουργούνται κατά την εκτέλεση του αλγορίθμου.

Στο σχήμα 3.11 δίνεται ένα παράδειγμα εκτέλεσης μιας επανάληψης του αλγορίθμου για ένα μικρό σύνολο σημείων.



Σχήμα 3.11: Ένα παράδειγμα εκτέλεσης του αλγορίθμου για το Concave Hull για ένα μικρό σύνολο σημείων

Στο σχήμα 3.12 δίνεται το αποτέλεσμα που έδωσε η εκτέλεση του αλγορίθμου στο οδικό δίκτυο της Αθήνας για το ίδιο αρχικό σημείο και χρονικό διάστημα με το αντίστοιχο σχήμα του Convex Hull (σχήμα 3.9).



Σχήμα 3.12: Οι πράσινοι είναι οι ισοχρονικοί κόμβοι (για 10') ενώ οι κόκκινοι οι μη ισοχρονικοί. Τα όρια του Concave Hull διακρίνονται με την ανοιχτή πράσινη γραμμή.

Όπως παρατηρούμε, πράγματι το Concave Hull αποκλείει πολλούς μη ισοχρονικούς κόμβους οι οποίοι περιέχονταν μέσα στο Convex Hull. Επιπλέον, σε αρκετές περιπτώσεις μπορούμε ποιοτικά να πούμε ότι η τελική καμπύλη πλησιάζει αρκετά αυτή που θα λέγαμε ιδανική “με το μάτι”.

Από την άλλη όμως, το Concave Hull έχει και κάποια δυνατά μειονεκτήματα. Ένα από αυτά, όπως είπαμε, είναι ότι το αποτέλεσμα μπορεί να είναι απρόβλεπτο λόγω της εξάρτησης από παραμέτρους. Μία ανωμαλία, για παράδειγμα, που προέρχεται από αυτό το λόγο φαίνεται με λεπτομέρεια στο σχήμα 3.13.

Σχήμα 3.13: Μία από τις ανωμαλίες που προέρχεται από τη μη προβλεψιμότητα του αποτελέσματος της εκτέλεσης του αλγορίθμου

Ένα ακόμα μειονέκτημα και ίσως το σημαντικότερο είναι ο χρόνος εκτέλεσης του αλγορίθμου. Για το οδικό δίκτυο της Αθήνας, ο υπολογισμός χρειάζεται αρκετά δευτερόλεπτα κάτι που είναι απαγορευτικό για τη δημιουργία μιας εφαρμογής που θέλουμε να έχει άμεση απόκριση. Έστω και αν με κάποιες βελτιστοποιήσεις καταφέραμε να μειώσουμε λίγο το χρόνο εκτέλεσης όσο αυξάνονται τα χρονικά διαστήματα υπολογισμού, οι χρόνοι παρέμειναν μεγάλοι. Μία ένδειξη αυτών των χρόνων δίνεται στον Πίνακα 3.2.

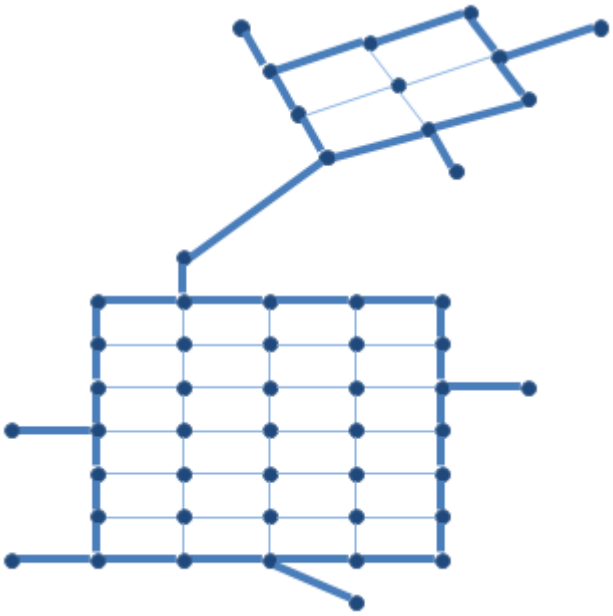
		πριν τις βελτιστοποιήσεις		μετά τις βελτιστοποιήσεις	
χρονικό όριο	**# ακμών Convex Hull**	**# ακμών Concave Hull**	**χρόνος εκτέλεσης**	**# ακμών Concave Hull**	**χρόνος εκτέλεσης**
10'	16	525	4,32 sec	525	4,37 sec
20'	9	529	53,42 sec	515	52,53 sec
30'	11	834	73,39 sec	810	33,62 sec
40'	21	662	57,10 sec	617	16,26 sec
50'	29	995	95,38 sec	841	14,27 sec

Πίνακας 3.2

3.4.3 Edges' Hull

Λόγω των σημαντικών μειονεκτημάτων των δύο προηγούμενων προσεγγίσεων, έπρεπε να βρεθεί μία άλλη προσέγγιση η οποία να δίνει καλύτερα αποτελέσματα. Τόσο από άποψη χρόνου αφού ο τελικός στόχος είναι μία εφαρμογή που θα απαντά άμεσα στα ερωτήματα όσο και από άποψη ακρίβειας όπου οι δύο προηγούμενες προσεγγίσεις υστερούσαν. Έτσι, υπήρξε η ιδέα να χρησιμοποιηθούν για τον υπολογισμό των ισοχρονικών καμπύλων οι ισοχρονικές ακμές και όχι οι κόμβοι.

Μία παρόμοια προσέγγιση υπάρχει στο [MG10] όπως αναφέραμε στο Κεφάλαιο 2. Η βασική ιδέα είναι ότι εφόσον οι ισοχρονικές ακμές αποτελούν ένα συνεκτικό γράφο θα μπορούσαμε να βρούμε όλες τις περιοχές οι οποίες περικυκλώνονται από ισοχρονικές ακμές και το σύνολο αυτών των περιοχών να αποτελεί το τελικό πολύγωνο. Στο σχήμα 3.14 φαίνεται ένα τέτοιο πολύγωνο που προκύπτει από ένα μικρό σύνολο ακμών.



Σχήμα 3.14: Το Edges' Hull ενός συνόλου ακμών

Στην επόμενη παράγραφο αναλύεται με λεπτομέρειες ο αλγόριθμος που υλοποιήσαμε για την προσέγγιση αυτή καθώς και τα προβλήματα που αντιμετωπίστηκαν.

3.5 Edges' Hull

Το Edges' Hull είναι διαισθητικά το πολύγωνο που σχηματίζεται από τις πιο εξωτερικές ισοχρονικές ακμές δηλαδή αυτές που δεν κλείνονται από οποιοδήποτε κύκλο

σχηματίζεται από άλλες. Παρακάτω παρουσιάζεται ο αλγόριθμος που χρησιμοποιούμε για τον υπολογισμό του καθώς και ένα βασικό πρόβλημα που προκύπτει με τα υπάρχοντα δεδομένα και ο τρόπος που το αντιμετωπίσαμε.

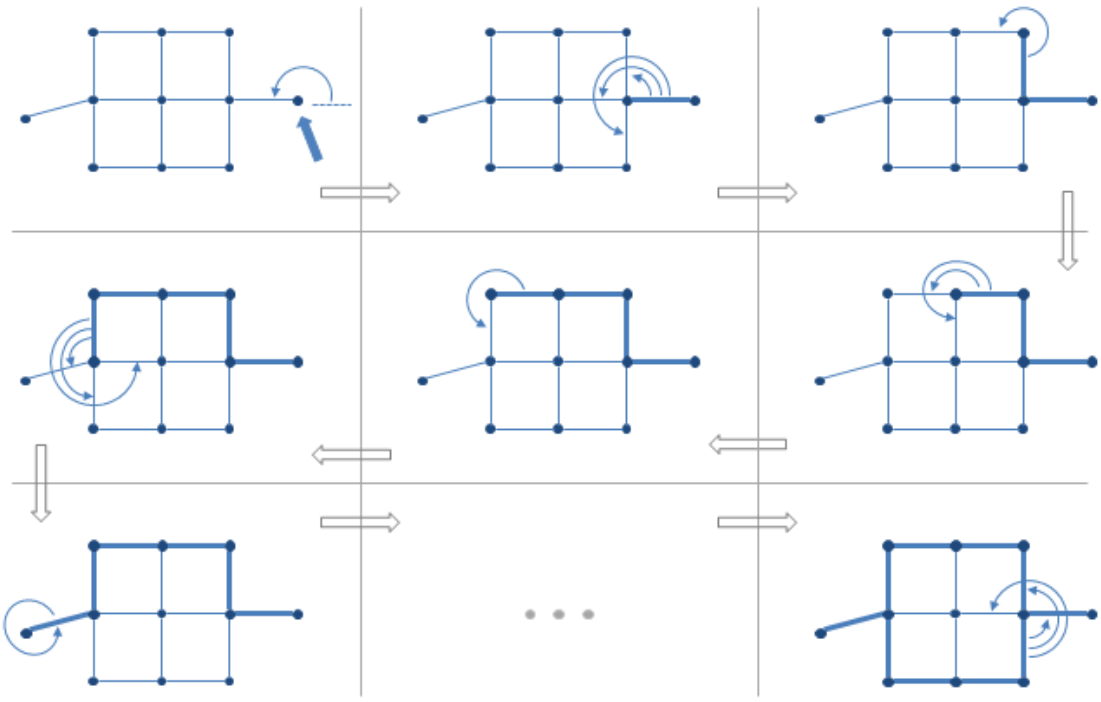
3.5.1 Περιγραφή του αλγορίθμου

Το αποτέλεσμα του αλγορίθμου που θα περιγραφεί είναι μία λίστα σημείων L που αποτελούν τα άκρα των διαδοχικών ακμών οι οποίες σχηματίζουν το τελικό πολύγωνο. Η κατευθυντικότητα των ακμών δεν έχει σημασία στη διαδικασία αυτή. Ο αλγόριθμος λειτουργεί ως εξής:

- Αρχικά, βρίσκεται η ισοχρονική ακμή που έχει σαν άκρο το σημείο με τη μεγαλύτερη συντεταγμένη x. Το σημείο αυτό είναι το πρώτο που εισάγεται στη λίστα. Ουσιαστικά αποτελεί το δεξιότερο άκρο του τελικού πολυγώνου. Το βήμα αυτό γίνεται μόνο μία φορά και μπορεί να γίνει γρηγορότερα κατά την εκτέλεση του Dijkstra για να μην σπαταλήσει χρόνο προσπελαύνοντας όλες τις ισοχρονικές ακμές.
- Βρίσκονται όλες οι ακμές που έχουν σαν ένα άκρο το τελευταίο σημείο που μπήκε στη λίστα L. Για κάθε ακμή e:
    - υπολογίζεται η γωνία που σχηματίζεται μεταξύ της τελευταίας ακμής (αυτή που σχηματίζεται μεταξύ των δύο τελευταίων σημείων της L) και της e.
    - Στην πρώτη επανάληψη όπου υπάρχει μόνο ένα σημείο στη λίστα θεωρούμε οριζόντια ακμή.
- Επιλέγεται η ακμή με την μικρότερη σχηματιζόμενη γωνία. Το επόμενο σημείο που εισάγεται στη λίστα είναι το άλλο άκρο αυτής της ακμής.
- Επαναλαμβάνεται η διαδικασία από το 2ο βήμα μέχρι να εισαχθεί ξανά το πρώτο (δεξιότερο) σημείο στη λίστα οπότε τερματίζεται η διαδικασία.

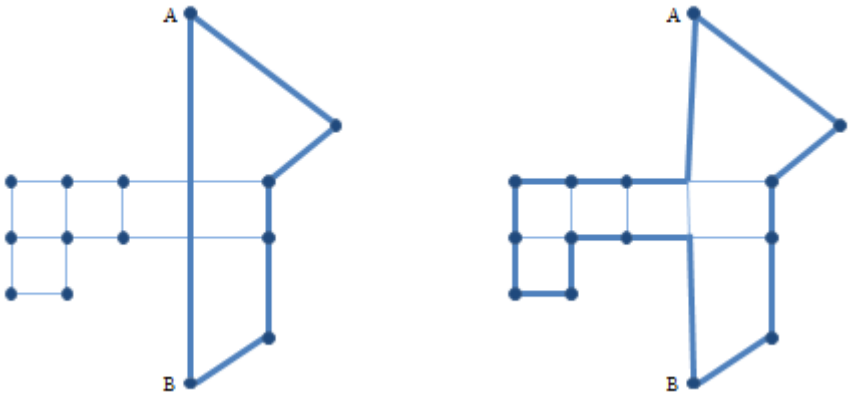
Τονίζουμε ότι λόγω συνεκτικότητας των ισοχρονικών ακμών η παραπάνω διαδικασία θα δώσει το εξωτερικό περίβλημα του συνόλου των ακμών ξεκινώντας από το δεξιότερο σημείο και κατευθυνόμενη αριστερόστροφα. Τελικά, θα καταλήξει στο αρχικό σημείο οπότε κλείνει η καμπύλη και έχουμε το τελικό πολύγωνο.

Στο σχήμα 3.15 δίνεται ένα παράδειγμα εκτέλεσης του αλγορίθμου για έναν μικρό γράφο.



Σχήμα 3.15: Ένα παράδειγμα εκτέλεσης του αλγορίθμου υπολογισμού του Edges' Hull σε έναν μικρό γράφο. Ξεκινάει από το δεξιότερο σημείο που υποδεικνύεται στην πρώτη εικόνα. Κάθε φορά επιλέγει την ακμή με τη μικρότερη σχηματιζόμενη γωνία.

Το μόνο πρόβλημα που δημιουργείται είναι στα σημεία που υπάρχουν τομές μεταξύ των ακμών αλλά χωρίς να επικοινωνούν μεταξύ τους δηλαδή χωρίς κόμβο στο σημείο της τομής. Αυτό συμβαίνει για παράδειγμα όπου υπάρχει γέφυρα στο οδικό δίκτυο. Για να λειτουργήσει ο παραπάνω αλγόριθμος χρειάζεται ένας μονοεπίπεδος γράφος. Η λύση στο πρόβλημα δίνεται παρακάτω. Στο σχήμα 3.16 φαίνεται μία περίπτωση που το συγκεκριμένο πρόβλημα επηρεάζει αρνητικά το αποτέλεσμα του αλγορίθμου.



Σχήμα 3.16: Η ακμή AB είναι μία γέφυρα στο οδικό δίκτυο. Αυτό οδηγεί τον αλγόριθμο στην εξαγωγή του πολυγώνου που φαίνεται στα αριστερά. Το επιθυμητό όμως πολύγωνο είναι αυτό που φαίνεται στα δεξιά

3.5.2 Απαλοιφή μη επικοινωνούντων ακμών στο οδικό δίκτυο

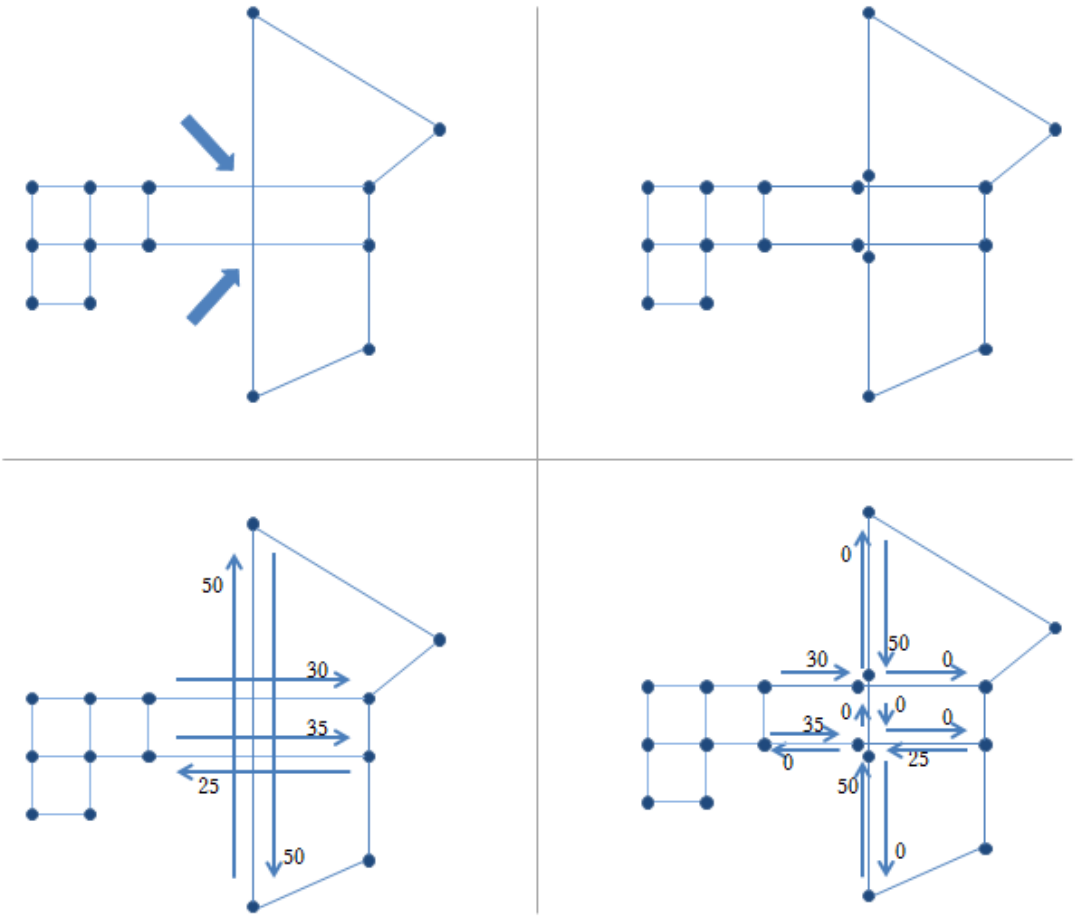
Για να έχουμε ένα μονοεπίπεδο γράφο ώστε να λειτουργήσει σωστά ο παραπάνω αλγόριθμος πραγματοποιούμε μία προ-επεξεργασία στο οδικό δίκτυο. Ο στόχος είναι αυτή η προ-επεξεργασία να μην επηρεάζει καθόλου το αποτέλεσμα που δίνει ο αλγόριθμος Dijkstra, όμως να προσθέτει πληροφορία στο δίκτυο ώστε ο αλγόριθμος του Edges' Hull να εκτελείται σωστά και να δίνει το αναμενόμενο αποτέλεσμα.

Η προ-επεξεργασία διαρκεί μερικά δευτερόλεπτα αλλά αρκεί να γίνει μία φορά κατά την έναρξη της εφαρμογής οπότε δεν επηρεάζει το χρόνο απόκρισης στα ερωτήματα του χρήστη. Γίνεται ως εξής:

- Αρχικά, εντοπίζονται όλες οι ακμές που τέμνονται μεταξύ τους χωρίς να υπάρχει κόμβος στο σημείο τομής που να τις κάνει να επικοινωνούν. Αυτή η διαδικασία διαρκεί αρκετά αλλά αρκεί να γίνει μία φορά για κάποιο οδικό δίκτυο που έχουμε στη διάθεσή μας. Από αυτή τη διαδικασία, για κάθε τομή κρατάμε τα ids των ακμών που τέμνονται καθώς και τις συντεταγμένες του σημείου τομής.
- Στη συνέχεια, για κάθε σημείο τομής δημιουργείται ένας νέος κόμβος με συντεταγμένες αυτές του σημείου τομής για κάθε μία από τις δύο ακμές που τέμνονται. Στόχος είναι να σπάσει κάθε ακμή που τέμνεται σε κομμάτια με ενδιάμεσα άκρα τους κόμβους που δημιουργούνται σε αυτό το βήμα. Οι δύο κόμβοι που δημιουργούνται για το σημείο τομής κρατούν ο καθένας το id του άλλου τον οποίο ονομάζουμε “αδερφό” κόμβο του. Έτσι, ο αλγόριθμος του Edges' Hull όταν τελευταίος κόμβος στη λίστα L είναι ένας από αυτούς που δημιουργήθηκαν εδώ, εκτός από τις ακμές που έχουν σαν άκρο τον ίδιο, εξετάζει και τις ακμές που έχουν σαν άκρο τον “αδερφό” κόμβο του.
- Αφού ολοκληρωθεί η δημιουργία των νέων κόμβων, για κάθε τεμνόμενη ακμή αναδιαμορφώνουμε τον γράφο του οδικού δικτύου ώστε η ακμή αυτή να σπάσει σε κομμάτια ανάλογα με τον αριθμό των τομών. Με λίγα λόγια η ακμή αποκτά ενδιάμεσους κόμβους. Αυτό το βήμα δεν είναι τόσο απλό όσο δείχνει καθώς υπάρχουν ακμές με τα ίδια άκρα και αντίθετες κατευθύνσεις. Θα πρέπει να φροντίσουμε ώστε αυτές οι ακμές να σπάσουν στους ίδιους ενδιάμεσους κόμβους και όχι σε διαφορετικούς η καθεμιά. Τα βάρη των ακμών διαμορφώνονται ως εξής:
    - Η πρώτη ακμή από τα κομμάτια που δημιουργήθηκαν έχει σαν βάρος αυτό που είχε προηγουμένως ολόκληρη η ακμή.
    - Οι επόμενες ακμές έχουν βάρος 0.

  Αυτή η διαμόρφωση συμβαίνει έτσι ώστε το αποτέλεσμα της εκτέλεσης του αλγορίθμου Dijkstra να μην έχει καμία απολύτως διαφορά με αυτό που θα είχαμε αν δε γινόταν η προ-επεξεργασία.

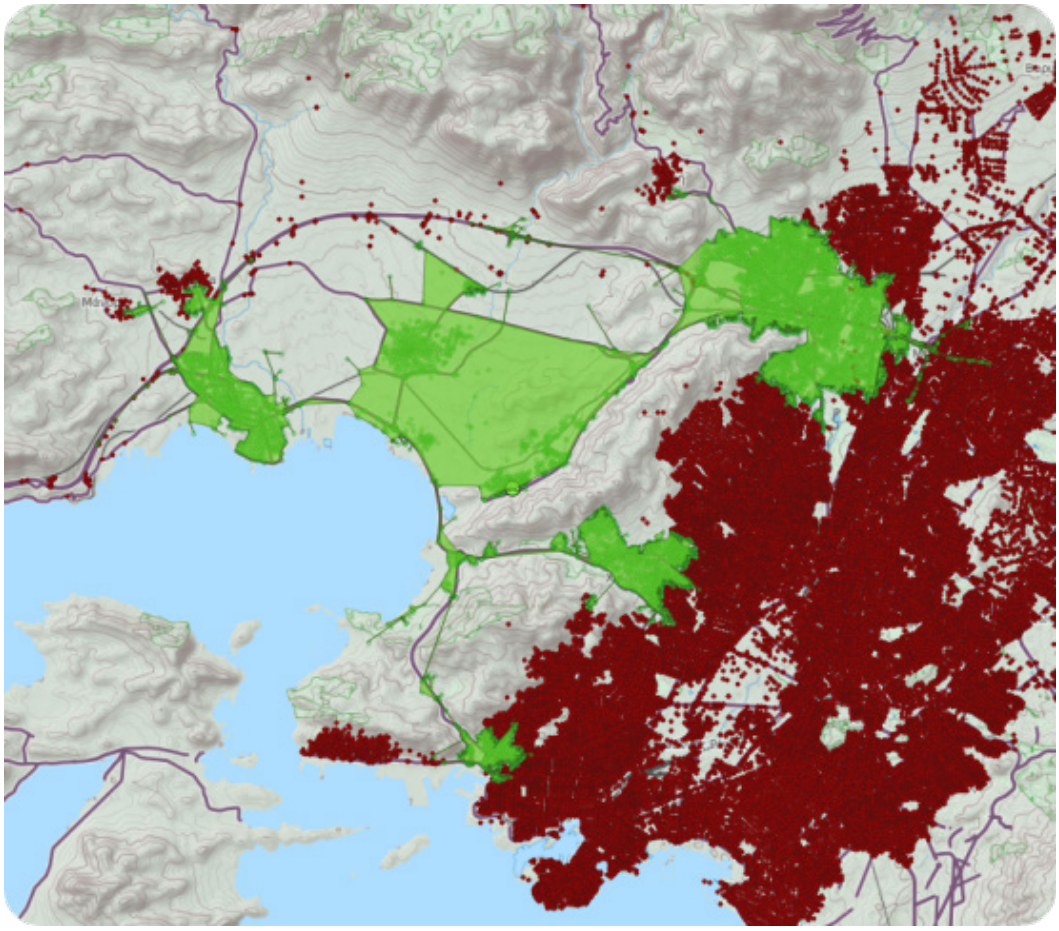
Στο σχήμα 3.17 φαίνεται ένα μικρό παράδειγμα εκτέλεσης της προ-επεξεργασίας για ένα μικρό τμήμα ενός δικτύου.



Σχήμα 3.17: Πάνω-αριστερά: ο αρχικός γράφος με την υπόδειξη των σημείων τομής. Κάτω-αριστερά: Τα βάρη των ακμών όπως είναι αρχικά. Πάνω-δεξιά: Σε κάθε σημείο τομής δημιουργούνται δύο νέοι κόμβοι, ένας για κάθε ακμή. Οι δύο αυτοί κόμβοι ονομάζονται “αδερφοί” κόμβοι. Κάτω-δεξιά: Οι νέες ακμές και τα βάρη τους όπως διαμορφώνονται στο τέλος της προ-επεξεργασίας.

3.5.3 Αξιολόγηση της μεθόδου

Η μέθοδος του Edges' Hull για τον υπολογισμό των ισοχρονικών καμπύλων χρονοαπόστασης έχει αρκετά πλεονεκτήματα εν συγκρίσει με τις δύο άλλες μεθόδους. Στο σχήμα 3.18 δίνεται το αποτέλεσμα που έδωσε η εκτέλεση του αλγορίθμου στο οδικό δίκτυο της Αθήνας για το ίδιο αρχικό σημείο και χρονικό διάστημα με τα αντίστοιχα σχήματα του Convex Hull (σχήμα 3.9) και Concave Hull (σχήμα 3.12).



Σχήμα 3.18: Οι πράσινοι είναι οι ισοχρονικοί κόμβοι (για 10') ενώ οι κόκκινοι οι μη ισοχρονικοί. Το Edges' Hull είναι το πολύγωνο με το πράσινο γέμισμα.

Πολύ σημαντικό προσόν της μεθόδου είναι η ταχύτητα εκτέλεσης του αλγορίθμου. Ο αλγόριθμος είναι πολύ γρήγορος αφού δε χρειάζεται να προσπελαύνει όλες τις ισοχρονικές ακμές του δικτύου. Συγκεκριμένα, εφόσον έχει βρεθεί το δεξιότερο άκρο το οποίο βρίσκεται εύκολα και χωρίς μεγάλο επιπλέον κόστος κατά την εκτέλεση του Dijkstra, ο αλγόριθμος ουσιαστικά πραγματοποιεί τόσες επαναλήψεις όσες και οι ακμές του τελικού παραχθέντος πολυγώνου και σε κάθε επανάληψη προσπελαύνει όλες τις ακμές του τρέχοντος κόμβου υπολογίζοντας μία γωνία. Όμως, αυτές οι ακμές είναι ελάχιστες (λιγότερες από 10 και συνήθως γύρω στις 4) αφού μιλάμε για οδικό δίκτυο. Η ταχύτητα του αλγορίθμου αναλύεται με πειραματικά αποτελέσματα στο Κεφάλαιο 6.

Επίσης, σημαντικό πλεονέκτημα της μεθόδου είναι η ακρίβεια του αποτελέσματος. Συγκεκριμένα, το πολύγωνο που παράγεται περιέχει στο εσωτερικό του ή πάνω στο όριό του όλους τους ισοχρονικούς κόμβους ενώ οι μη ισοχρονικοί κόμβοι αποκλείονται όλοι εκτός από ελάχιστους σε ορισμένες περιπτώσεις που το οδικό δίκτυο έχει κάποιες ιδιαιτερότητες.

Τελικά, η απόφαση για τη μέθοδο που θα χρησιμοποιηθεί στην εφαρμογή για τον υπολογισμό των ισοχρονικών καμπύλων χρονοαπόστασης ήταν εύκολη αφού το Edges' Hull υπερτερεί και μάλιστα είναι άκρως ικανοποιητικό και στα δύο κριτήριά μας, ταχύτητα και ακρίβεια.

3.6 Συμπίεση παραχθέντων καμπύλων

Το επόμενο βήμα που πρέπει να γίνει ώστε να έχουμε μία γρήγορη εφαρμογή είναι η συμπίεση του αποτελέσματος του υπολογισμού. Αυτό είναι ιδιαίτερα σημαντικό αν έχουμε μια web-εφαρμογή και πρέπει τα αποτελέσματα αυτά να αποσταλούν μέσω του δικτύου στον χρήστη. Όσο μικρότερος είναι ο όγκος των δεδομένων που θα αποσταλούν τόσο ταχύτερα θα γίνει η μεταφορά και το δίκτυο θα φορτωθεί λιγότερο.

3.6.1 Ο αλγόριθμος Encoded Polyline

Ένας τρόπος να συμπιέσουμε τις καμπύλες που παράγονται είναι η χρήση του αλγορίθμου Encoded Polyline [EncPo]. Ο αλγόριθμος αυτός παίρνει σαν είσοδο μία σειρά από σημεία με συντεταγμένες latitude και longitude και κωδικοποιεί τις συντεταγμένες με ASCII χαρακτήρες έτσι ώστε το τελικό αποτέλεσμα να είναι αρκετά μικρότερο. Για ακόμα καλύτερη οικονομία χώρου, δεν κωδικοποιούνται οι απόλυτες συντεταγμένες των σημείων αλλά το offset τους από το προηγούμενο σημείο (εκτός φυσικά από το πρώτο σημείο).

Για παράδειγμα, μία σειρά σημείων όπως τα εξής:

37.94395, 23.70918

37.94439, 23.70956

37.94415, 23.71051

37.94524, 23.71077

37.94571, 23.70265

37.95014, 23.70849

37.95590, 23.70201

37.95982, 23.70802

37.96270, 23.70506

37.96324, 23.70712

37.96371, 23.70750

37.96713, 23.70205

37.96794, 23.70231

37.96804, 23.70188

τα οποία υποδεικνύουν μία διαδρομή στο οδικό δίκτυο και αναπαρίστανται με 280 bytes μετά την κωδικοποίηση μετατρέπονται στο εξής string:

“u|qfFkuuoCwAkAn@}DyEs@}Avq@uZoc@_c@ng
@oWqd@_QnQkB {K}AkAkT`a@aDs@StA”

το οποίο αναπαρίσταται με 67 bytes δηλαδή έχουμε γι' αυτό το παράδειγμα συμπίεση της τάξης του 76%.

3.6.2 GZIP compression

Αν ο υπολογισμός γίνεται στον server και το αποτέλεσμα πρέπει να αποσταλεί στον client τότε μπορούμε να προχωρήσουμε και σε περαιτέρω συμπίεση εφόσον ο server και ο client υποστηρίζουν gzip compression [RFC1952]. Το gzip compression, πολύ απλοϊκά, λειτουργεί βρίσκοντας παρόμοια strings σε ένα αρχείο κειμένου και αντικαθιστώντας τα προσωρινά ώστε να μικρύνει το συνολικό μέγεθος του κειμένου. Όταν φτάσει το συμπιεσμένο κείμενο στον client χρειάζεται decompression.

3.6.3 Συνολική συμπίεση

Πραγματοποιώντας τις δύο παραπάνω συμπιέσεις έχουμε μία αρκετά ικανοποιητική τελική συμπίεση. Στον Πίνακα 3.3 φαίνονται τα μεγέθη των αποτελεσμάτων της εκτέλεσης του αλγορίθμου με σημείο αρχής κάποιο σημείο στο οδικό δίκτυο της Αθήνας πριν και μετά τις συμπιέσεις.

Αρχικά	**Encoded Polyline**		**GZIP compression**		**Συνολικά**
μέγεθος	**μέγεθος**	**ποσοστό συμπίεσης**	**μέγεθος**	**ποσοστό συμπίεσης**	**ποσοστό συμπίεσης**
564 KB	63 KB	89 %	32 KB	49 %	94 %

Πίνακας 3.3

3.7 Εξομάλυνση τελικής καμπύλης

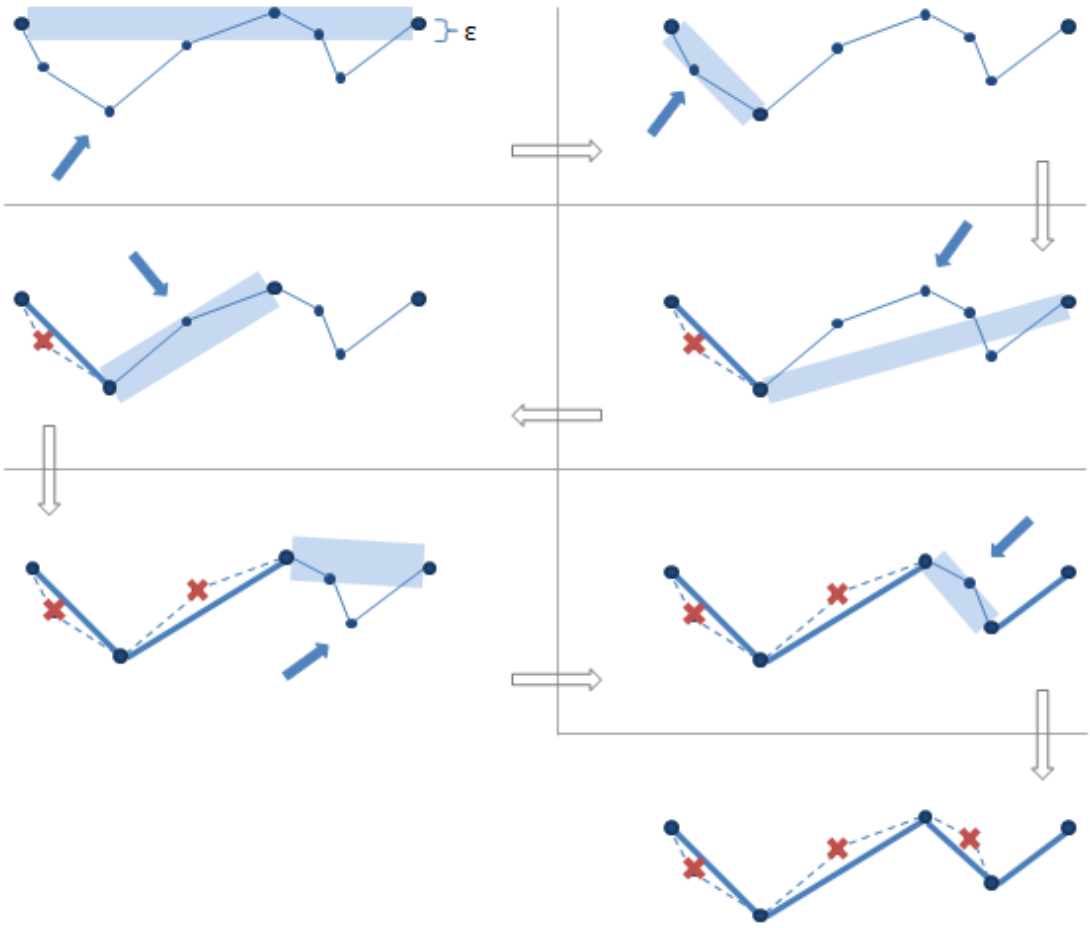
Επειδή η καμπύλη που παράγεται από την εφαρμογή έχει αρκετές γωνίες και δεν είναι ομαλή, θα ήταν σκόπιμο να παρείχαμε τη δυνατότητα της εξομάλυνσης της τελικής καμπύλης σε περιπτώσεις που δεν ενδιαφέρει η απόλυτη ακρίβεια. Αυτό θα βοηθούσε ώστε το αισθητικό αποτέλεσμα της οπτικοποίησης πάνω στο χάρτη να είναι καλύτερο.

3.7.1 Ο αλγόριθμος Ramer-Douglas-Peucker

Ο αλγόριθμος Ramer-Douglas-Peucker [DP06] είναι ένας αλγόριθμος που μειώνει τον αριθμό των σημείων μιας καμπύλης ώστε το τελικό αποτέλεσμα να είναι μία προσέγγιση της αρχικής καμπύλης. Το τελικό σύνολο σημείων της καμπύλης είναι ένα υποσύνολο του αρχικού. Με τον όρο προσέγγιση της αρχικής καμπύλης εννοούμε ότι κανένα σημείο της τελικής καμπύλης δεν απέχει περισσότερο από κάποιο συγκεκριμένο όριο από κάποιο σημείο της αρχικής. Ακολουθεί μία σύντομη περιγραφή του αλγορίθμου:

- Η αρχική καμπύλη είναι ένα διατεταγμένο σύνολο σημείων που αποτελούν τα άκρα των διαδοχικών ακμών της καμπύλης. Επιπλέον, υπάρχει δεδομένο το όριο απόστασης αρχικής και τελικής καμπύλης $\varepsilon > 0$.
- Ο αλγόριθμος λειτουργεί διαιρώντας αναδρομικά ευθύγραμμα τμήματα σε δύο. Αρχικά, ξεκινάει με το ευθύγραμμο τμήμα που έχει σαν άκρα το πρώτο και το τελευταίο σημείο του συνόλου.
- Για κάθε ευθύγραμμο τμήμα κάνει τα εξής:
    - Σημειώνει ότι τα άκρα του θα κρατηθούν οπότε θα ανήκουν και στην τελική καμπύλη.
    - Εξετάζει την κάθετη απόσταση όλων των ενδιάμεσων σημείων από το ευθύγραμμο τμήμα ώστε να βρει αυτό που έχει τη μέγιστη απόσταση.
        - Αν αυτό το σημείο απέχει λιγότερο από ε από το ευθύγραμμο τμήμα τότε αυτό και όλα τα ενδιάμεσα σημεία μπορούν να αγνοηθούν χωρίς η τελική καμπύλη να έχει απόσταση από την αρχική μεγαλύτερη από ε.
        - Αν όμως αυτό το σημείο δεν απέχει λιγότερο από ε τότε πρέπει να κρατηθεί και να είναι μέρος της τελικής καμπύλης. Οπότε ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τα δύο ευθύγραμμα τμήματα που σχηματίζονται από τα άκρα του αρχικού ευθυγράμμου τμήματος και το μακρύτερο σημείο.
- Όταν τερματιστεί η αναδρομή, η τελική καμπύλη μπορεί να ανακτηθεί αποτελούμενη από τα σημεία που είναι σημειωμένα να κρατηθούν και μόνο αυτά.

Στο σχήμα 3.19 φαίνεται ένα παράδειγμα εκτέλεσης του αλγορίθμου σε μια καμπύλη που αποτελείται από λίγα σημεία.



Σχήμα 3.19: Ένα παράδειγμα εκτέλεσης του αλγορίθμου Ramer-Douglas-Peucker σε μία καμπύλη που αποτελείται από λίγα σημεία. Τα βέλη υποδεικνύουν το πιο μακρινό ενδιάμεσο σημείο του τρέχοντος ευθυγράμμου τμήματος. Αν αυτό είναι μακρύτερα από ε κρατείται, αλλιώς απορρίπτεται.

Η πολυπλοκότητα του αλγορίθμου είναι $\Theta(n \cdot \log n)$ ωστόσο στη χειρότερη περίπτωση ο αλγόριθμος μπορεί να χρειαστεί $O(n^2)$.

Στο σχήμα 3.20 φαίνεται ένα παράδειγμα μίας καμπύλης που έχει παραχθεί χωρίς τη χρήση του αλγορίθμου Ramer-Douglas-Peucker και δίπλα η καμπύλη για την ίδια περίπτωση με τη χρήση του αλγορίθμου.

Σχήμα 3.20: Αριστερά η καμπύλη χωρίς την επεξεργασία από τον αλγόριθμο Ramer–Douglas–Peucker και δεξιά η ίδια καμπύλη μετά την επεξεργασία από τον αλγόριθμο

Αφού περιγράψαμε αναλυτικά ένα προς ένα τα βήματα που πρέπει να πραγματοποιηθούν για τον επιτυχή και αποδοτικό υπολογισμό των ισοχρονικών καμπύλων χρονοαπόστασης είμαστε έτοιμοι να προχωρήσουμε στη σχεδίαση και υλοποίηση της εφαρμογής.

4

Σχεδίαση εφαρμογής

Απώτερος σκοπός της παρούσας εργασίας είναι η δημιουργία μιας εφαρμογής η οποία θα δέχεται από το χρήστη ένα σημείο πάνω στο χάρτη και ένα χρονικό διάστημα και θα υπολογίζει τις ισοχρονικές καμπύλες χρονοαπόστασης οπτικοποιώντας τες πάνω στο χάρτη. Η εφαρμογή αυτή αποφασίστηκε να είναι μία web-εφαρμογή. Στο κεφάλαιο αυτό παρουσιάζεται συνοπτικά η σχεδίασή της. Αρχικά, αιτιολογείται γιατί επιλέχθηκε η υλοποίηση μίας web-εφαρμογής και στην συνέχεια δίνονται οι προδιαγραφές της λειτουργίας της εφαρμογής ώστε στο επόμενο κεφάλαιο να περάσουμε στην υλοποίησή της.

4.1 Γιατί web-εφαρμογή;

Το πρώτο ερώτημα που έπρεπε να απαντήσουμε πριν ξεκινήσουμε την σχεδίαση της εφαρμογής ήταν τι είδους εφαρμογή θα είναι αυτή. Πιο συγκεκριμένα, το ερώτημα ήταν mobile-εφαρμογή, web-εφαρμογή ή εφαρμογή που εγκαθίσταται και τρέχει στον υπολογιστή του πελάτη (ας την πούμε client-εφαρμογή); Στον Πίνακα 4.1 παρουσιάζονται λίγα πλεονεκτήματα και μειονεκτήματα για το κάθε είδος.

Είδος	Πλεονεκτήματα	Μειονεκτήματα
mobile-εφαρμογή	• Υποστήριξη σε κινητές συσκευές. • Εύκολη και γρήγορη εκτέλεση της εφαρμογής για κινούμενους χρήστες.	• Περιορισμός από το μέγεθος της οθόνης. • Περιορισμός στις δυνατότητες πλοήγησης από τις μεθόδους εισόδου που προσφέρει η κινητή συσκευή. • Πιθανή υπερφόρτωση του server που παρέχει την υπηρεσία. • Διακόπτεται η λειτουργία της εφαρμογής αν “πέσει” ο server. • Εφαρμογή ειδική για κάποιο συγκεκριμένο mobile-λειτουργικό.
client-εφαρμογή	• Δυνατότητα εκμετάλλευσης των υπολογιστικών πόρων του client. • Ταχύτερη απόκριση μη εξαρτώμενη από την κατάσταση του δικτύου. • Πλούσιο user-interface. • Δυναμική και άμεσα αποκρίσιμη αλληλεπίδραση του χρήστη με την εφαρμογή. • Δυνατότητα εκτέλεσης εκτός σύνδεσης.	• Πολυπλοκότητα για το deployment της εφαρμογής. • Πολυπλοκότητα στην ανανέωση και βελτίωση της εφαρμογής με νέες εκδόσεις. • Εφαρμογή εκτελέσιμη σε συγκεκριμένη πλατφόρμα οπότε μη μεταφέρσιμη.
web-εφαρμογή	• Υποστήριξη σε όλες τις συσκευές που παρέχουν έναν απλό web-browser. • Εύκολο deployment της εφαρμογής. • Ευκολία στην βελτίωση και ανανέωση με νέες εκδόσεις.	• Εξάρτηση από την κατάσταση του δικτύου. • Αν “πέσει” ο server η εφαρμογή δε μπορεί να εκτελεστεί. • Δυσκολία εκτέλεσης σε περίπτωση υπερφόρτωσης του server.

Πίνακας 4.1

Το βασικό πλεονέκτημα της web-εφαρμογής έναντι των άλλων περιπτώσεων είναι η δυνατότητα εκτέλεσης σε οποιαδήποτε πλατφόρμα με οποιονδήποτε web-browser (cross-platform and cross-browser support). Αυτό, σε συνδυασμό με την ευκολία του deployment της εφαρμογής και της αναβάθμισης σε νέες εκδόσεις μας οδήγησε στην επιλογή της web-εφαρμογής.

4.2 Προδιαγραφές

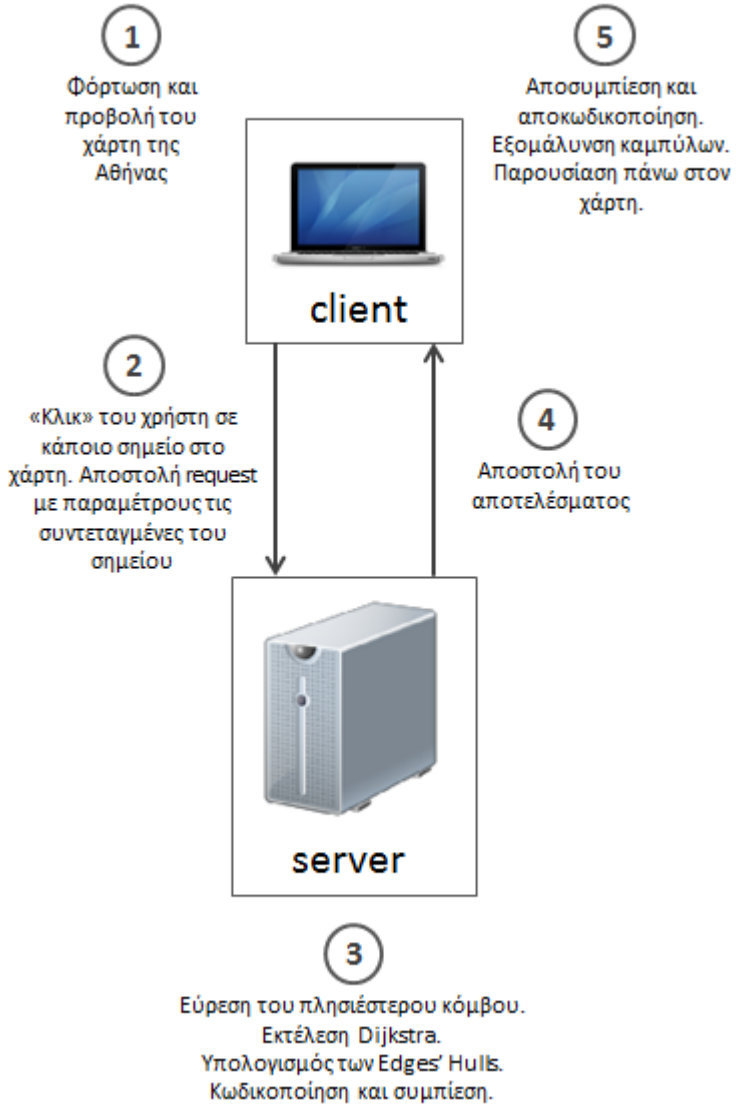
Εφόσον θα υλοποιήσουμε μία web-εφαρμογή θα πρέπει να σχεδιάσουμε τις λειτουργίες που θα αναλάβει ο server και αυτές που θα αναλάβει ο client αλλά και την αλληλεπίδραση μεταξύ τους. Σε γενικές γραμμές, ο server θα αναλάβει το μεγαλύτερο μέρος του υπολογισμού και ο client θα περιοριστεί στην αλληλεπίδραση με τον χρήστη.

Αναλυτικότερα, σχεδιάζουμε το σύστημα έτσι ώστε να λειτουργεί ως εξής:

- Στην αρχική σελίδα της εφαρμογής στον client φορτώνεται ο χάρτης και δίνεται η δυνατότητα στον χρήστη να επιλέξει ένα οποιοδήποτε σημείο στον χάρτη.
- Αφού ο χρήστης επιλέξει ένα σημείο πάνω στον χάρτη, ο client στέλνει ένα request στον server με παραμέτρους τις συντεταγμένες του επιλεγμένου σημείου.
- Μόλις ο server λάβει το request ξεκινάει τον υπολογισμό:
    - Αρχικά, βρίσκει τον πλησιέστερο στο επιλεγμένο σημείο κόμβο του οδικού δικτύου χρησιμοποιώντας το R* tree.
    - Στη συνέχεια εκτελεί τον αλγόριθμο του Dijkstra ξεκινώντας από τον πλησιέστερο κόμβο.
    - Μετά υπολογίζει τα Edges' Hulls για όσα χρονικά διαστήματα έχουν οριστεί να υπολογίζονται.
    - Οι παραχθείσες καμπύλες κωδικοποιούνται με τον αλγόριθμο Encoded Polyline.
    - Τέλος, γίνεται συμπίεση των αποτελεσμάτων και αποστέλλονται πίσω στον client.
- Ο client λαμβάνει τα αποτελέσματα, τα αποσυμπιέζει και τα αποκωδικοποιεί με τον αλγόριθμο αποκωδικοποίησης του Encoded Polyline.
- Αν είναι ενεργοποιημένη η λειτουργία της εξομάλυνσης, ο client προχωρεί στην εξομάλυνση των καμπύλων με τον αλγόριθμο Ramer-Douglas-Peucker.
- Τελικά, ο client παρουσιάζει πάνω στον χάρτη τις ισοχρονικές καμπύλες χρονοαπόστασης.

Παρατηρούμε ότι όλοι οι υπολογισμοί πραγματοποιούνται στον server εκτός από την εξομάλυνση των καμπύλων. Αυτό έχει διπλό όφελος. Πρώτον, είναι εύκολο ο χρήστης να επιλέξει αν θέλει να γίνει η εξομάλυνση ή όχι αφού αυτή γίνεται στον υπολογιστή του. Δεύτερον, γίνεται εκμετάλλευση των αδρανών πόρων του χρήστη για να μην επιβαρύνεται περισσότερο ο server αλλά όχι με βαριά εργασία που θα μπορούσε να δημιουργήσει πρόβλημα στον υπολογιστή του client.

Στο σχήμα 4.1 παρουσιάζεται σχηματικά η σχεδίαση του συστήματος.



Σχήμα 4.1: Η σχεδίαση του συστήματος που θα υλοποιηθεί για την εφαρμογή.

Μετά την σχεδίαση της λειτουργίας του συστήματος και την κατανομή των λειτουργιών ανάμεσα σε server και client είμαστε έτοιμοι να προχωρήσουμε στην υλοποίηση του συστήματος και την παρουσίαση της αρχιτεκτονικής του.

5

Αρχιτεκτονική συστήματος

Αφού είδαμε την γενική σχεδίαση της εφαρμογής προχωράμε στην υλοποίησή της. Στο κεφάλαιο αυτό παρουσιάζεται η αρχιτεκτονική του συστήματος που δημιουργήθηκε για την υλοποίηση της εφαρμογής ξεκινώντας από την περιγραφή των δεδομένων του οδικού δικτύου και συνεχίζοντας με την περιγραφή της βάσης δεδομένων και την ανάλυση της αρχιτεκτονικής του server και του client της εφαρμογής.

5.1 Το οδικό δίκτυο

Το οδικό δίκτυο που χρησιμοποιείται στην εφαρμογή για τον υπολογισμό των ισοχρονικών καμπύλων είναι αυτό της Αθήνας. Περιέχει 92728 κόμβους και 634427 ακμές ενώ υπάρχουν σε αυτό 4226 τομές μη επικοινωνούντων ακμών. Παρακάτω περιγράφουμε από που προέρχονται τα δεδομένα του οδικού δικτύου και πώς τα ανακτήσαμε.

5.1.1 OpenStreetMap

Το OpenStreetMap [OSM] είναι ένα project που αποσκοπεί στη δημιουργία ενός δωρεάν επεξεργάσιμου χάρτη για ολόκληρο τον κόσμο. Οι χάρτες δημιουργούνται χρησιμοποιώντας δεδομένα από κινητές συσκευές GPS, αεροφωτογραφίες και άλλες

δωρεάν πηγές. Κάθε χρήστης μπορεί να βοηθήσει στη βελτίωση των χαρτών με προσθήκες ή διορθώσεις. Το σημαντικό στοιχείο του project και αυτό που κάνει τη διαφορά είναι ότι όλα τα δεδομένα των χαρτών παρέχονται δωρεάν για ανάκτηση.

5.1.2 Η ανάκτηση του οδικού δικτύου

Τα δεδομένα του OpenStreetMap για οδικά δίκτυα από διάφορες περιοχές σε ολόκληρο τον κόσμο παρέχονται δωρεάν από την Cloudmade [Cloudm]. Τα αρχεία που παρέχονται είναι στη μορφή OSM XML [OSMXML]. Τα αρχεία αυτά περιέχουν τρεις βασικές οντότητες: nodes, ways και relations. Τα nodes, που περιλαμβάνουν και τις συντεταγμένες τους, μπορεί να είναι είτε σημεία ενδιαφέροντος, τα οποία σε αυτή τη φάση δε μας ενδιαφέρουν για την εφαρμογή, είτε σημεία των δρόμων του οδικού δικτύου, που μας ενδιαφέρουν. Τα ways είναι διατεταγμένα σύνολα από nodes και μπορεί να αντιπροσωπεύουν περιοχές, οπότε δε μας ενδιαφέρουν, ή δρόμους του οδικού δικτύου. Όταν πρόκειται για δρόμους υπάρχει ένα tag που υποδεικνύει την κατηγορία του δρόμου. Τα relations δε μας ενδιαφέρουν.

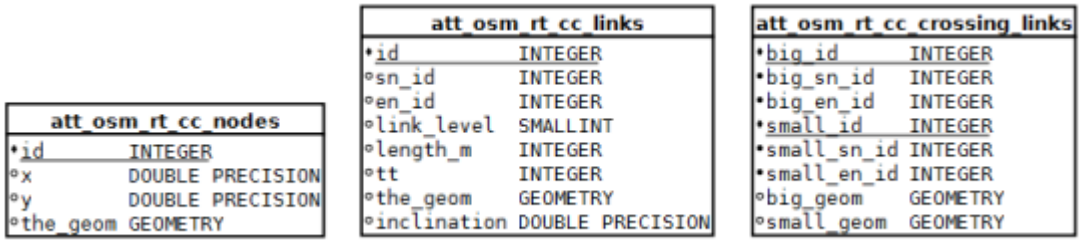
Για να ανακτήσουμε μόνο τα δεδομένα που χρειαζόμαστε, πρώτα υποβάλλουμε το αρχείο σε μία προ-επεξεργασία η οποία αφαιρεί πολλά δεδομένα που μας είναι αχρείαστα όπως τα nodes που αποτελούν σημεία ενδιαφέροντος, τα ways που αντιπροσωπεύουν περιοχές και τα relations. Στη συνέχεια, αφαιρούμε τους κόμβους που αποτελούν ένα δρόμο κρατώντας μόνο αυτούς που είναι σημεία διασταυρώσεων ή άκρες δρόμων. Οπότε τελικά καταλήγουμε με τον γράφο που μας είναι απαραίτητος και αποτελείται από τους κόμβους, τις ακμές και τα βάρη των ακμών όπως περιγράφηκαν στην § 3.1.

5.2 Βάση δεδομένων

Η βάση δεδομένων χρησιμοποιείται για την αποθήκευση και την ανάκτηση των δεδομένων του οδικού δικτύου που χρησιμοποιεί η εφαρμογή για τον υπολογισμό. Το οδικό δίκτυο αναπαρίσταται στη βάση σαν κόμβοι οι οποίοι ενώνονται μεταξύ τους με ακμές. Οι κόμβοι είναι τα σημεία του οδικού δικτύου όπου υπάρχουν διασταυρώσεις ενώ οι ακμές είναι οι δρόμοι του οδικού δικτύου. Έχοντας πληροφορία για τις συντεταγμένες των κόμβων, μπορούμε να εξάγουμε και πληροφορία και για τη θέση των ακμών αν τις θεωρήσουμε σαν ευθύγραμμα τμήματα με άκρα δύο κόμβους. Βέβαια, σε πολλές περιπτώσεις οι δρόμοι είναι καμπύλοι αλλά αυτό δεν επηρεάζει τον υπολογισμό που κάνει ο αλγόριθμος Dijkstra παρά μόνο μπορεί να επηρεάσει σε εξαιρετικές περιπτώσεις αλλά όχι σε σημαντικό βαθμό τον υπολογισμό των ισοχρονικών καμπύλων. Εκτός από τους κόμβους και τις ακμές, στην βάση δεδομένων αποθηκεύεται και η πληροφορία για τις τεμνόμενες ακμές που δεν επικοινωνούν μεταξύ τους, η οποία είναι απαραίτητη για τον υπολογισμό των ισοχρονικών καμπύλων όπως είδαμε στην § 3.5.

5.2.1 Το σχήμα της βάσης δεδομένων

Στο σχήμα 5.1 παρουσιάζονται σε διαγραμματική μορφή οι πίνακες της βάσης δεδομένων που χρησιμοποιούνται από την εφαρμογή.



Σχήμα 5.1: Το σχήμα της βάσης δεδομένων

Ο πρώτος πίνακας με όνομα att_osm_rt_cc_nodes είναι ο πίνακας που αποθηκεύει τους κόμβους του οδικού δικτύου. Αποτελείται από 4 στήλες:

- id - πρόκειται για το αναγνωριστικό και μοναδικό id του κόμβου. Ξεκινάει από 0 και φτάνει μέχρι το n-1 όπου n ο αριθμός των κόμβων.
- x - είναι η πρώτη συντεταγμένη δηλαδή το γεωγραφικό μήκος του κόμβου.
- y - είναι η δεύτερη συντεταγμένη δηλαδή το γεωγραφικό πλάτος του κόμβου.
- the_geom - είναι η γεωγραφική πληροφορία για τον κόμβο που χρειάζεται το σύστημα της βάσης δεδομένων για την πραγματοποίηση χωρικών πράξεων. Το σύστημα της βάσης δεδομένων περιγράφεται στην § 5.3.1.

Ο πίνακας των κόμβων έχει σαν πρωτεύον κλειδί το πεδίο id οπότε και αποθηκεύει τις εγγραφές ταξινομημένες ως προς αυτό το πεδίο. Αυτό μας είναι χρήσιμο καθώς γίνεται ταχύτερη ανάκτηση και αποθήκευση των κόμβων στη μνήμη του προγράμματος της εφαρμογής αφού ο πίνακας δε χρειάζεται επαναταξινόμηση ως προς το id.

Ο δεύτερος πίνακας με όνομα att_osm_rt_cc_links είναι ο πίνακας που αποθηκεύει τις ακμές του οδικού δικτύου. Αποτελείται από 8 στήλες:

- id - πρόκειται για το αναγνωριστικό και μοναδικό id της ακμής. Ξεκινάει από 0 και φτάνει μέχρι το m-1 όπου m ο αριθμός των ακμών.
- sn_id - είναι το id του κόμβου που αποτελεί το σημείο αρχής της ακμής.
- en_id - είναι το id του κόμβου που αποτελεί το σημείο τέλους της ακμής.
- link_level - πρόκειται για μία ιεράρχηση του μεγέθους και της σημαντικότητας των δρόμων (π.χ. από επαρχιακός χωματόδρομος μέχρι εθνική οδός). Ξεκινάει από 1 για τους μεγαλύτερους δρόμους και φτάνει μέχρι 12 για τους μικρότερους.

- length_m - είναι το μήκος του δρόμου που αντιπροσωπεύει η ακμή σε μέτρα.
- tt - είναι ο εκτιμώμενος χρόνος διάσχισης της ακμής σε δέκατα του δευτερολέπτου. Πρόκειται για το πεδίο το οποίο αντιπροσωπεύει το βάρος της ακμής στον υπολογισμό με τον αλγόριθμο Dijkstra.
- the_geom - είναι η γεωγραφική πληροφορία για την ακμή που χρειάζεται το σύστημα της βάσης δεδομένων για την πραγματοποίηση χωρικών πράξεων.
- inclination - πρόκειται για την κλίση της ακμής ως προς ένα θεωρούμενο οριζόντιο άξονα. Ο τρόπος υπολογισμού της κλίσης αναλύεται στην § 5.2.2.

Ο πίνακας των ακμών έχει σαν πρωτεύον κλειδί το πεδίο id. Αυτό μας είναι χρήσιμο για τον ίδιο λόγο που αναφέραμε και στην περίπτωση του πίνακα των κόμβων.

Ο τρίτος και τελευταίος πίνακας με όνομα att_osm_rt_cc_crossing_links είναι ο πίνακας που σε κάθε εγγραφή του περιέχει όσες πληροφορίες χρειάζονται για κάθε μία τομή ακμών που δεν επικοινωνούν. Αποτελείται από 8 στήλες:

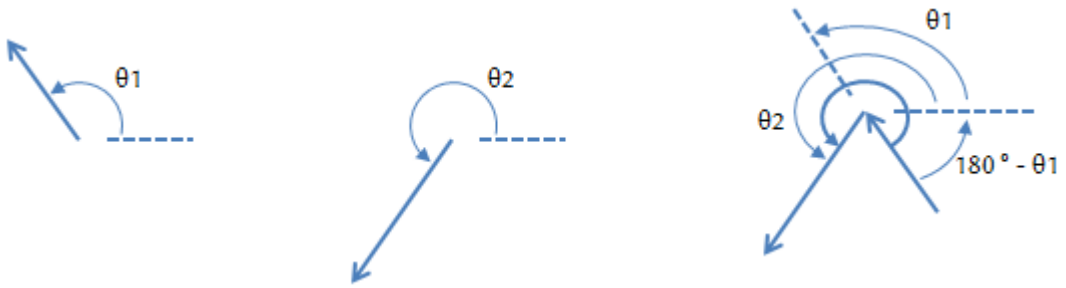
- big_id - είναι το id της μίας από τις δύο ακμές που τέμνονται. Αν υπάρχει διαφορά ιεράρχησης ανάμεσα στις δύο τεμνόμενες ακμές (διαφορά στο link_level) τότε εδώ μπαίνει το id της μεγαλύτερης.
- big_sn_id - είναι το id του κόμβου που αποτελεί το σημείο αρχής της ακμής με id big_id.
- big_en_id - είναι το id του κόμβου που αποτελεί το σημείο τέλους της ακμής με id big_id.
- small_id - είναι το id της μίας από τις δύο ακμές που τέμνονται. Αν υπάρχει διαφορά ιεράρχησης ανάμεσα στις δύο τεμνόμενες ακμές (διαφορά στο link_level) τότε εδώ μπαίνει το id της μικρότερης.
- small_sn_id - είναι το id του κόμβου που αποτελεί το σημείο αρχής της ακμής με id small_id.
- small_en_id - είναι το id του κόμβου που αποτελεί το σημείο τέλους της ακμής με id small_id.
- big_geom - είναι η γεωγραφική πληροφορία για την ακμή με id big_id που χρειάζεται το σύστημα της βάσης δεδομένων για την πραγματοποίηση χωρικών πράξεων.
- small_geom - είναι η γεωγραφική πληροφορία για την ακμή με id small_id που χρειάζεται το σύστημα της βάσης δεδομένων για την πραγματοποίηση χωρικών πράξεων.

Ο πίνακας των τεμνόμενων ακμών έχει σαν πρωτεύον κλειδί τον συνδυασμό των πεδίων [big_id, small_id] ο οποίος είναι μοναδικός για κάθε εγγραφή. Αυτό σημαίνει ότι

οι εγγραφές ταξινομούνται με βάση το big_id αλλά όταν υπάρχουν δύο ή περισσότερες εγγραφές με το ίδιο big_id, αυτές ταξινομούνται με το small_id.

5.2.2 Ο υπολογισμός της κλίσης των ακμών

Όπως έχουμε αναλύσει στην § 3.5 για τον υπολογισμό των ισοχρονικών καμπύλων χρειάζεται υπολογισμός γωνιών που σχηματίζονται μεταξύ των ακμών. Για να είναι αυτός ο υπολογισμός γρήγορος κατά την εκτέλεση της εφαρμογής φροντίζουμε να έχει γίνει ένας προϋπολογισμός των γωνιών που σχηματίζουν όλες οι ακμές ως προς έναν θεωρούμενο οριζόντιο άξονα. Έτσι κατά την εκτέλεση του αλγορίθμου υπολογισμού των καμπύλων δε χρειάζονται πολύπλοκες τριγωνομετρικές πράξεις παρά μόνο προσθαφαιρέσεις για να υπολογισθούν οι γωνίες. Στο σχήμα 5.2 φαίνονται δύο τυχαίες ακμές, οι γωνίες τους ως προς οριζόντιο άξονα και η γωνία που σχηματίζεται μεταξύ τους.



Σχήμα 5.2: Αριστερά φαίνονται δύο ακμές και οι γωνίες κλίσης τους θ1 και θ2. Δεξιά, η γωνία που σχηματίζεται από τις δύο ακμές ισούται με 180° – θ1 + θ2. Παρατηρούμε ότι για τον υπολογισμό της γωνίας δεν χρειάζονται τριγωνομετρικές πράξεις.

Η γωνία σε μοίρες που σχηματίζει μία ακμή με άκρα τα σημεία (x_1, y_1) και (x_2, y_2) με τον οριζόντιο άξονα υπολογίζεται ως εξής:

- Όταν $x_1 \neq x_2$ τότε:
  - Η κλίση της ακμής είναι:
    - $\lambda = (y_2 - y_1) / (x_2 - x_1)$
  - Η αντίστοιχη γωνία σε ακτίνια (rad) είναι:
    - $rad = \arctan(\lambda)$
  - Για να μετατρέψουμε τα ακτίνια σε μοίρες (deg) χρησιμοποιούμε τον τύπο:
    - $deg = rad \times 180 / \pi$
- Όταν $x_1 = x_2$ οπότε δεν μπορεί να υπολογισθεί η κλίση λ τότε η γωνία είναι 90° όταν $y_2 >= y_1$ ή 270° όταν $y_2 < y_1$.

Με τα παρακάτω τέσσερα SQL queries κατά σειρά εκτελείται ο υπολογισμός των γωνιών των ακμών ως προς οριζόντιο άξονα και αποθηκεύεται στην κατάλληλη στήλη. Το πρώτο υπολογίζει τις κλίσεις των ακμών που δεν είναι κατακόρυφες (δηλαδή 90° ή 270°). Το δεύτερο προσθέτει 360° στις κλίσεις με αρνητικό πρόσημο ώστε να υπάρχουν τιμές στο εύρος [0..360) και όχι στο εύρος [-180,180). Το τρίτο βρίσκει τις ακμές με κλίση 90° και τις ενημερώνει ενώ το τέταρτο αυτές με κλίση 270°.

```
WITH edge_inclinations AS
(
  SELECT l.id AS linkid,
         DEGREES(ATAN((n2.y-n1.y)/(n2.x-n1.x))) AS incl
    FROM att_osm_rt_cc_links l,
         att_osm_rt_cc_nodes n1,
         att_osm_rt_cc_nodes n2
   WHERE l.sn_id = n1.id AND l.en_id = n2.id
         AND n2.x - n1.x != 0
)
UPDATE att_osm_rt_cc_links
SET l.inclination = i.incl
FROM att_osm_rt_cc_links l, edge_inclinations i
WHERE l.id = i.linkid
```

```
UPDATE att_osm_rt_cc_links
SET inclination = inclination + 360.0
WHERE inclination < 0
```

```
WITH edge_inclinations AS
(
  SELECT l.id AS linkid, 90.0 AS incl
    FROM att_osm_rt_cc_links l,
         att_osm_rt_cc_nodes n1,
         att_osm_rt_cc_nodes n2
   WHERE l.sn_id = n1.id AND l.en_id = n2.id
         AND n2.x - n1.x = 0 AND n2.y >= n1.y
)
UPDATE att_osm_rt_cc_links
SET l.inclination = i.incl
FROM att_osm_rt_cc_links l, edge_inclinations i
WHERE l.id = i.linkid
```

```
WITH edge_inclinations AS
(
  SELECT l.id AS linkid, 270.0 AS incl
    FROM att_osm_rt_cc_links l,
         att_osm_rt_cc_nodes n1,
         att_osm_rt_cc_nodes n2
   WHERE l.sn_id = n1.id AND l.en_id = n2.id
         AND n2.x - n1.x = 0 AND n2.y < n1.y
)
UPDATE att_osm_rt_cc_links
SET l.inclination = i.incl
FROM att_osm_rt_cc_links l, edge_inclinations i
WHERE l.id = i.linkid
```

5.2.3 Η εύρεση των τεμνόμενων ακμών

Όπως έχουμε διαπιστώσει στην § 3.5, είναι απαραίτητη η πληροφορία για τις τεμνόμενες ακμές που δεν επικοινωνούν έτσι ώστε να υπολογισθούν σωστά οι ισοχρονικές καμπύλες. Επειδή η εύρεση των τομών είναι μία διαδικασία που παίρνει αρκετό χρόνο, ανάλογα και με το πλήθος των ακμών στο διαθέσιμο οδικό δίκτυο, συμφέρει να γίνεται πριν την έναρξη της εφαρμογής και μάλιστα μόνο μία φορά για κάποιο οδικό δίκτυο. Για αυτό το λόγο, εκτελούμε τη διαδικασία μέσα από τη βάση δεδομένων και αποθηκεύουμε σε αυτήν τα αποτελέσματα για να μη χρειαστεί να επανεκτελεστεί.

Το σύστημα βάσης δεδομένων που χρησιμοποιούμε και το οποίο περιγράφεται αναλυτικότερα στην § 5.3.1 μας δίνει τη δυνατότητα της εκτέλεσης χωρικών πράξεων ανάμεσα σε πεδία που είναι τύπου geometry. Μία από αυτές τις πράξεις εκτελεί και η συνάρτηση ST_Crosses. Η συνάρτηση αυτή δέχεται δύο παραμέτρους τύπου geometry και επιστρέφει TRUE αν η τομή τους διασταυρώνεται χωρικά δηλαδή αν τα δύο αντικείμενα έχουν στο χώρο κάποια από τα σημεία τους κοινά αλλά όχι όλα. Σε αντίθετη περίπτωση επιστρέφει FALSE. Έτσι, εφόσον για κάθε ακμή έχουμε το πεδίο the_geom που είναι τύπου geometry και την περιγράφει, μπορούμε δίνοντας δύο ακμές ως παραμέτρους στην ST_Crosses να διαπιστώσουμε αν τέμνονται.

Το παρακάτω SQL query, εκτελούμενο, γεμίζει τον πίνακα att_osm_rt_cc_crossing_links με όλες τις απαραίτητες πληροφορίες για τις τεμνόμενες ακμές.

```
INSERT INTO att_osm_rt_cc_crossing_links(
               big_id, big_sn_id, big_en_id,
               small_id, small_sn_id, small_en_id,
```

```
               big_geom, small_geom)

SELECT l1.id, l1.sn_id, l1.en_id,
       l2.id, l2.sn_id, l2.en_id,
       l1.the_geom, l2.the_geom
FROM att_osm_rt_cc_links l1, att_osm_rt_cc_links l2
WHERE l1.inclination != l2.inclination AND
       ((l1.link_level < l2.link_level) OR
        (l1.link_level=l2.link_level AND l1.id < l2.id)) AND
      ST_Crosses(l1.the_geom, l2.the_geom)
ORDER BY l1.link_level, l1.id
```

5.3 Ο server

Ο server είναι η καρδιά της εφαρμογής. Αυτός κατέχει τα δεδομένα και πραγματοποιεί τους υπολογισμούς όποτε ζητούνται. Αποτελείται από το σύστημα διαχείρισης της βάσης δεδομένων που είναι ο PostgreSQL καθώς και από τον application server που είναι ο Apache Tomcat και πάνω σε αυτόν τρέχει ο κώδικας της εφαρμογής. Στο κεφάλαιο αυτό περιγράφονται τα δύο αυτά συστήματα καθώς και ο τρόπος που συνεργάζονται για τη λειτουργία της εφαρμογής.

5.3.1 PostgreSQL

Για τη διαχείριση της βάσης δεδομένων χρησιμοποιήσαμε τον PostgreSQL [PSQL]. Ο PostgreSQL είναι ένα ισχυρό, αντικειμενο-σχεσιακό σύστημα βάσεων δεδομένων ανοικτού κώδικα. Αναπτύσσεται συνεχώς εδώ και 15 χρόνια και πλέον συγκαταλέγεται ανάμεσα στα κορυφαία συστήματα βάσεων δεδομένων. Είναι διαδεδομένο για την αξιοπιστία του και την ακεραιότητα και ορθότητα των δεδομένων που διαχειρίζεται. Είναι συμβατό με όλα τα κύρια λειτουργικά συστήματα όπως Linux, Unix (συμπεριλαμβανομένου του Mac OS X) και Windows. Είναι πλήρως συμβατό με το μοντέλο ACID[1] και υποστηρίζει

[1] Το ACID (atomicity, consistency, isolation, durability) είναι ένα σύνολο από ιδιότητες οι οποίες εγγυώνται ότι οι δοσοληψίες σε μία βάση δεδομένων εκτελούνται αξιόπιστα:

- **Ατομικότητα**. Εξασφαλίζει ότι οι πράξεις μίας δοσοληψίας είτε θα πετύχουν είτε θα αποτύχουν όλες. Δηλαδή κάθε δοσοληψία είτε ολοκληρώνεται επιτυχώς είτε εμφανίζεται σα να μην έχει ξεκινήσει καν.
- **Συνέπεια**. Εξασφαλίζει ότι μετά από οποιαδήποτε δοσοληψία που πραγματοποιείται η βάση παραμένει σε συνεπή κατάσταση.
- **Απομόνωση**. Εξασφαλίζει ότι καμία δοσοληψία δε μπλέκει με την εκτέλεση μιας άλλης. Κατά συνέπεια, κάθε δοσοληψία συμπεριφέρεται σα να είναι η μόνη που τρέχει στο σύστημα.
- **Μονιμότητα**. Εξασφαλίζει ότι εφόσον μία δοσοληψία ολοκληρωθεί και καταχωρηθεί, δε υπάρχει περίπτωση αναίρεσης της.

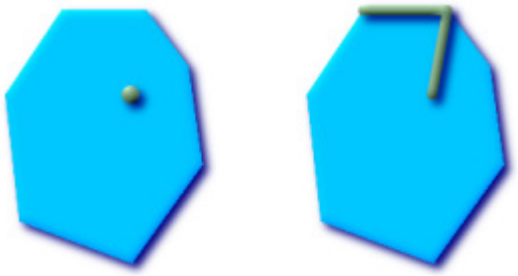
πλήρως foreign keys, joins, views, triggers. Συμπεριλαμβάνει τους περισσότερους τύπους δεδομένων του SQL:2008 όπως INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL και TIMESTAMP. Επιπλέον, υποστηρίζει την αποθήκευση μεγάλων binary αντικειμένων όπως εικόνες, ήχους και βίντεο. Διαθέτει εγγενείς προγραμματιστικές διεπαφές για πολλές διαδεδομένες γλώσσες προγραμματισμού όπως C/C++, Java, .Net, Perl, Ruby, Python. Τέλος, έχει πολύ ισχυρό documentation.

Εκτός από τα παραπάνω που αποδεικνύουν ότι ο PostgreSQL είναι μία πολύ αξιόπιστη και ευέλικτη επιλογή για σύστημα βάσεων δεδομένων και μάλιστα ανοικτού κώδικα, ένας ακόμα και μάλλον ο σημαντικότερος λόγος που μας οδήγησε στην επιλογή του είναι το PostGIS [PGIS]. Το PostGIS είναι ένα ανοικτό project το οποίο προσθέτει στον PostgreSQL υποστήριξη για γεωγραφικά αντικείμενα. Στην πραγματικότητα, το PostGIS προσθέτει τη δυνατότητα στον PostgreSQL server να χρησιμοποιηθεί σαν εργαλείο διαχείρισης χωρικής βάσης δεδομένων σε γεωγραφικά συστήματα πληροφοριών (GIS).

Το PostGIS παρέχει μία αρκετά μεγάλη ποικιλία συναρτήσεων που εξακριβώνουν χωρικές σχέσεις μεταξύ αντικειμένων τύπου geometry. Κατά την εκπόνηση της εργασίας βρήκαμε χρήσιμες αρκετές από αυτές. Ενδεικτικά αναφέρουμε δύο:

- ST_Contains

  Η ST_Contains δέχεται δύο παραμέτρους Α και Β τύπου geometry. Επιστρέφει TRUE όταν κανένα σημείο του αντικειμένου Β δεν βρίσκεται στο εξωτερικό του αντικειμένου Α και τουλάχιστον ένα σημείο του εσωτερικού του Β βρίσκεται στο εσωτερικό του Α. Για παράδειγμα στο σχήμα 5.3 φαίνονται δύο περιπτώσεις όπου επιστρέφεται TRUE ενώ στο σχήμα 5.4 μία περίπτωση όπου επιστρέφεται FALSE. Τα μεγάλα πολύγωνα είναι η παράμετρος Α ενώ τα μικρά σχήματα η παράμετρος Β.



Σχήμα 5.3: Δύο περιπτώσεις όπου η ST_Contains επιστρέφει TRUE. Εικόνα από το documentation της PostGIS

Σχήμα 5.4: Μία περίπτωση όπου η ST_Contains επιστρέφει FALSE. Εικόνα από το documentation της PostGIS

Την ST_Contains την χρησιμοποιήσαμε για να εξάγουμε στατιστικά στοιχεία σχετικά με το πόσοι ισοχρονικοί κόμβοι και πόσοι μη ισοχρονικοί κόμβοι περιέχονται μέσα στο πολύγωνο που σχηματίζουν οι αντίστοιχες ισοχρονικές καμπύλες.

- ST_Crosses

  Η ST_Crosses περιγράφηκε στην § 5.2.3. Χρησιμοποιήθηκε για την εύρεση των τεμνόμενων ακμών. Στο σχήμα 5.5 φαίνεται μία περίπτωση όπου επιστρέφει TRUE.

Σχήμα 5.5: Η κλασική περίπτωση όπου η ST_Crosses επιστρέφει TRUE. Εικόνα από το documentation της PostGIS

Επιπλέον, μέσω του PostGIS παρέχονται συναρτήσεις οι οποίες παράγουν αντικείμενα τύπου geometry. Για παράδειγμα υπάχει η συνάρτηση ST_ConvexHull η οποία δέχεται μία παράμετρο τύπου geometry η οποία μπορεί να είναι ένα σύνολο σημείων στο διδιάστατο επίπεδο και επιστρέφει ένα πολύγωνο το οποίο αποτελεί το Convex Hull (για το οποίο συζητήσαμε στην § 3.4.1) του συνόλου αυτού. Ακόμα, υπάρχει η συνάρτηση ST_ConcaveHull η οποία δέχεται πάλι ένα αντικείμενο τύπου geometry το οποίο μπορεί να είναι ένα σύνολο σημείων και ακόμα μία παράμετρο (καθώς όπως αναλύσαμε στην § 3.4.2 κάθε αλγόριθμος υπολογισμού του Concave Hull είναι παραμετρικός) και υπολογίζει ένα Concave Hull του συνόλου αυτού. Τονίζουμε ότι οι

συναρτήσεις αυτές αναφέρονται μόνο για να δείξουμε τις δυνατότητες που προσφέρει το PostGIS. Για την υλοποίηση και αξιολόγηση των μεθόδων Convex Hull και Concave Hull για τον υπολογισμό των ισοχρονικών καμπυλών χρησιμοποιήσαμε αλγορίθμους υλοποιημένους στη γλώσσα προγραμματισμού Java.

5.3.2 Apache Tomcat

Σαν application server της εφαρμογής μας χρησιμοποιούμε τον Apache Tomcat [TOMC]. Ο Apache Tomcat είναι ένα λογισμικό ανοικτού κώδικα το οποίο υλοποιεί τις τεχνολογίες Java Servlet και JavaServer Pages. Χρησιμοποιείται για πολυάριθμες μεγάλου εύρους web εφαρμογές από πολλές εταιρείες και οργανισμούς. Λειτουργεί σαν web server και servlet container. Παρέχει ένα περιβάλλον HTTP web server όπου μπορεί να τρέξει Java κώδικας.

Επιλέξαμε τον Apache Tomcat σαν application server της εφαρμογής μας επειδή ο κώδικας της εφαρμογής έχει γραφτεί σε Java και ο συγκεκριμένος server είναι ένας από τους πιο διαδεδομένους και εύκολους στη χρήση για το deployment web-εφαρμογών που τρέχουν Java κώδικα. Επιπλέον, υποστηρίζει συμπίεση GZIP για την οποία μιλήσαμε αναλυτικά στην § 3.6.2.

5.3.3 Το Java Servlet της εφαρμογής

Ο κώδικας της εφαρμογής που τρέχει στον server είναι γραμμένος σε Java και αποτελείται από 16 Java κλάσεις. Εδώ αναφέρουμε ποιές είναι αυτές οι κλάσεις και τις περιγράφουμε συνοπτικά:

- **Isochrones**. Είναι η κλάση που κάνει extend το HttpServlet και ουσιαστικά ενορχηστρώνει την εκτέλεση της εφαρμογής. Αποτελείται από δύο μέρη. Αυτό που εκτελείται κατά την εκκίνηση του server και αυτό που εκτελείται όταν έρθει ένα HTTP Servlet Request. Τα βήματα που εκτελούνται σε κάθε περίπτωση περιγράφονται στις επόμενες παραγράφους.
- **RoadNetworkRetriever**. Είναι η κλάση που αναλαμβάνει τη σύνδεση με τη βάση δεδομένων, την ανάκτηση των δεδομένων του οδικού δικτύου δηλαδή κόμβων, ακμών και τεμνόμενων ακμών καθώς και την προ-επεξεργασία του γράφου για την απαλοιφή των τεμνόμενων ακμών (όπως συζητήθηκε στην § 3.5.2).
- **NodeSimple**. Είναι η κλάση των αντικειμένων που αντιπροσωπεύουν ένα κόμβο του οδικού δικτύου. Κρατάει κάποιες απαραίτητες πληροφορίες για τον κόμβο όπως οι συντεταγμένες του, ο δείκτης στην πρώτη ακμή του διανύσματος ακμών που έχει σαν αρχικό κόμβο τον ίδιο (ο τρόπος αναπαράστασης του γράφου αναλύθηκε στην § 3.1.1) και το id του "αδερφού" κόμβου, αν αυτός υπάρχει, ο

οποίος δημιουργείται στην προ-επεξεργασία του γράφου (ανάλυση στην § 3.5.2).

- **EdgeSimple**. Είναι η κλάση των αντικειμένων που αντιπροσωπεύουν μία ακμή του οδικού δικτύου. Οι πληροφορίες που κρατάει για την κάθε ακμή είναι το id της, το id του κόμβου στον οποίο καταλήγει, το βάρος της που είναι ο χρόνος διάσχισής της και η γωνία κλίσης της ως προς ένα θεωρούμενο οριζόντιο άξονα.
- **CrossingSimple**. Είναι η κλάση των αντικειμένων που περιγράφουν μία τομή δύο μη επικοινωνούντων ακμών στο οδικό δίκτυο. Κρατάει πληροφορίες για τις δύο ακμές που τέμνονται και συγκεκριμένα τα id τους και τα id των κόμβων από τους οποίους ξεκινούν και αυτών στους οποίους καταλήγουν.
- **ComputationThread**. Είναι η κλάση που αναλαμβάνει την ενορχήστρωση όλων των διαδικάσιών που πρέπει να εκτελεστούν για τον υπολογισμό των ισοχρονικών καμπύλων όταν έρθει ένα request.
- **DijkstraSearch**. Είναι η κλάση που περιέχει τον κώδικα που εκτελεί τον αλγόριθμο Dijkstra δεδομένου ενός αρχικού σημείου και ενός μέγιστου ορίου χρονοαπόστασης πέρα από το οποίο τερματίζεται. Υπενθυμίζουμε ότι ο αλγόριθμος έχει διαμορφωθεί έτσι ώστε να υπολογίζει και τις ισοχρονικές ακμές (όπως αναλύσαμε στην § 3.3.3).
- **DijkstraSearchFp**. Είναι μία βοηθητική κλάση που βοηθά στην καλύτερη οργάνωση του κώδικα για τον αλγόριθμο Dijkstra. Σε αυτή την κλάση ορίζεται το priority queue που θα χρησιμοποιηθεί κατά την εκτέλεση του Dijkstra.
- **DijkstraSearchResult**. Είναι επίσης μία βοηθητική κλάση για τη διαχείριση του κώδικα του αλγορίθμου Dijkstra.
- **PriorityQueueDial**. Είναι η κλάση που περιέχει όλες τις μεταβλητές και τις μεθόδους που είναι απαραίτητες για τη λειτουργία της priority queue Dial η οποία περιγράφηκε στην § 3.3.4.
- **DijkstraNode**. Είναι η κλάση των αντικειμένων που εισάγει ο αλγόριθμος Dijkstra στο priority queue. Περιλαμβάνει το id του κόμβου που αφορά καθώς και την τρέχουσα εκτίμηση της χρονοαπόστασης του από τον αρχικό κόμβο.
- **IsochroneEdge**. Είναι η κλάση που χρησιμοποιείται για τις ισοχρονικές ακμές. Περιλαμβάνει το id της ακμής, το id του κόμβου στον οποίο καταλήγει, την γωνία της κλίσης ως προς ένα θεωρούμενο οριζόντιο άξονα και, το σημαντικότερο, τον χρόνο διάσχισης της ακμής. Ουσιαστικά, περιέχει όλες τις πληροφορίες που χρειάζεται ο αλγόριθμος του Edges' Hull για τον υπολογισμό των ισοχρονικών καμπύλων. Ο ακριβής χρόνος διάσχισης χρειάζεται σε περίπτωση που ζητείται ο υπολογισμός για πολλαπλά χρονικά διαστήματα, για

παράδειγμα για κάθε 10' έως τη μία ώρα, και όχι για ένα μόνο. Οπότε, σε αυτή την περίπτωση οι ισοχρονικές ακμές για τα 20' είναι μόνο αυτές που έχουν χρόνο διάσχισης μικρότερο ή ίσο με 20'.

- **EdgesHull**. Είναι η κλάση που περιέχει τον κώδικα του αλγορίθμου του Edges' Hull (ο οποίος περιγράφηκε αναλυτικά στην § 3.5). Ουσιαστικά είναι η υπεύθυνη κλάση για τον υπολογισμό των ισοχρονικών καμπύλων.
- **PolylineEncoder**. Είναι η κλάση που αναλαμβάνει την πρώτη συμπίεση των παραχθέντων καμπύλων με τον αλγόριθμο Encoded Polyline (ο οποίος συζητήθηκε στην § 3.6.1).
- **Trackpoint**. Είναι η κλάση που περιγράφει τους κόμβους που αποτελούν τις τελικές καμπύλες όπως τους θέλει ο αλγόριθμος Encoded Polyline.
- **Track**. Είναι η κλάση που μαζεύει όλους τους κόμβους των τελικών καμπύλων (σε μορφή αντικειμένων της κλάσης Trackpoint) σε λίστα.

Επιπλέον, χρησιμοποιούνται 3 βιβλιοθήκες οι οποίες είναι απαραίτητες σε διάφορα σημεία της εφαρμογής. Είναι οι εξής:

- **PostgreSQL JDBC Driver**. Είναι η απαραίτητη βιβλιοθήκη για την αλληλεπίδραση του java servlet με τον server της βάσης δεδομένων (PostgreSQL).
- **ELKI**. Είναι το framework το οποίο μας παρέχει υλοποίηση του R* tree που χρησιμοποιούμε για το indexing των κόμβων του οδικού δικτύου (αναφερθήκαμε αναλυτικά στην § 3.2.1).
- **JSON-Simple**. Είναι η βιβλιοθήκη που χρησιμοποιείται για την κωδικοποίηση σε κείμενο json του τελικού αποτελέσματος που θα αποσταλεί στον client. Η κωδικοποίηση JSON περιγράφεται αναλυτικότερα σε επόμενη παράγραφο που περιγράφει τον client.

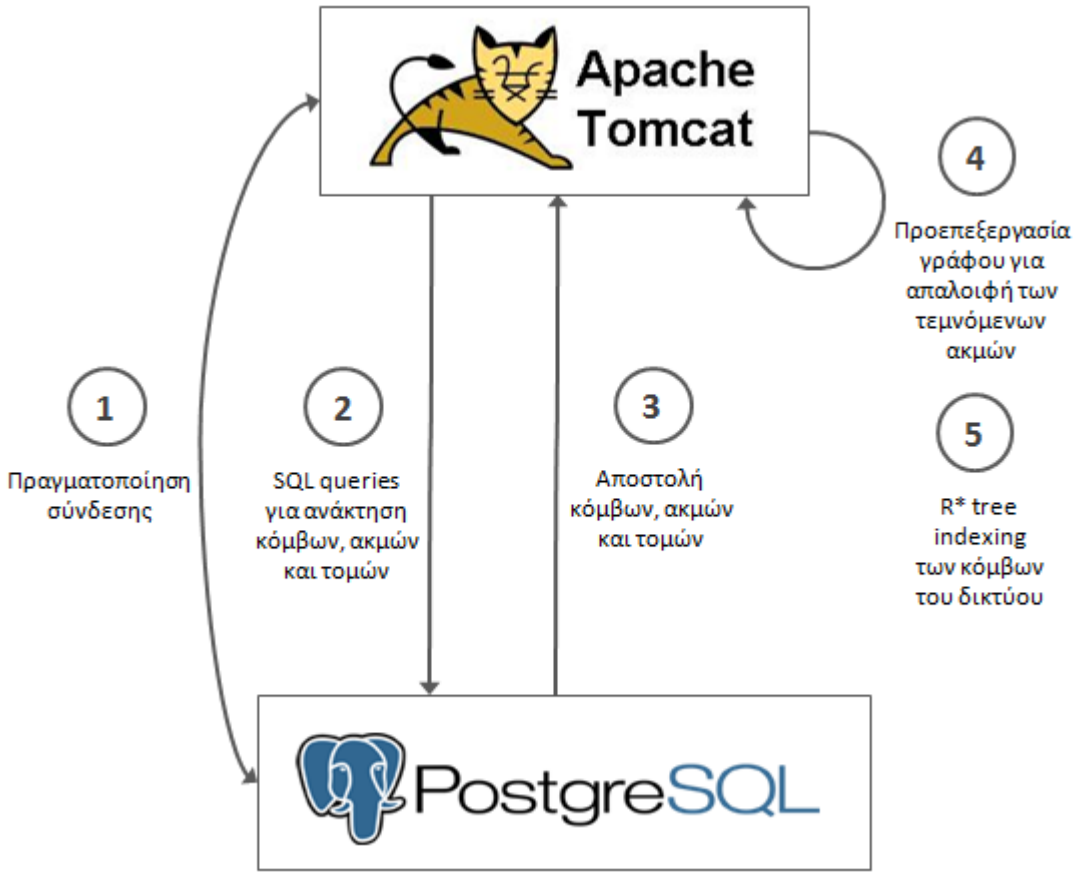
5.3.4 Κατά την έναρξη της εφαρμογής

Κατά την εκκίνηση του server που φιλοξενεί την εφαρμογή πραγματοποιείται η προετοιμασία της εφαρμογής ώστε να είναι έτοιμη να εξυπηρετήσει τα requests που θα ακολουθήσουν. Αυτή η προετοιμασία γίνεται εφικτή μέσω της μεθόδου init() της κλάσης Isochrones που κάνει extend την HttpServlet. Η προετοιμασία περιλαμβάνει τα εξής:

- Αρχικά, πραγματοποιείται η σύνδεση με τον server της βάσης δεδομένων και η προετοιμασία των SQL ερωτημάτων που θα ακολουθήσουν.

- Στη συνέχεια, εκτελούνται τα SQL ερωτήματα τα οποία ανακτούν τα απαραίτητα δεδομένα για το οδικό δίκτυο δηλαδή τον πίνακα των κόμβων, τον πίνακα των ακμών και τον πίνακα των τεμνόμενων ακμών.
- Μετά από αυτό, υπάρχουν όλες οι πληροφορίες που απαιτούνται για να εκτελεστεί η προ-επεξεργασία του γράφου για την απαλοιφή των τεμνόμενων ακμών.
- Μετά την προ-επεξεργασία του γράφου γίνεται το indexing των κόμβων του οδικού δικτύου με ένα R* tree.
- Πλέον, ο server είναι έτοιμος να εξυπηρετήσει γρήγορα τα requests που καταφθάνουν.

Στο σχήμα 5.6 παρουσιάζονται και σε σχηματική μορφή τα βήματα που εκτελούνται κατά την εκκίνηση του server.



Σχήμα 5.6: Η διαδικασία που πραγματοποιείται κατά την εκκίνηση του server οπότε και την εκκίνηση της εφαρμογής

5.3.5 Κατά την άφιξη ενός request

Κάθε request που καταφθάνει στον server περιλαμβάνει τις συντεταγμένες του αρχικού σημείου από το οποίο ζήτησε ο χρήστης να υπολογισθούν οι ισοχρονικές καμπύλες χρονοαπόστασης. Ας θεωρήσουμε ότι ο server είναι ρυθμισμένος να παρέχει τις ισοχρονικές καμπύλες χρονοαπόστασης για 6 χρονικά διαστήματα και συγκεκριμένα ανά 10' μέχρι μία ώρα (10', 20', ... , 60'). Τονίζουμε ότι ο αριθμός και οι τιμές των χρονικών διαστημάτων που υπολογίζονται μπορούν πολύ εύκολα να μεταβληθούν.

Για την απάντηση στο request γίνονται κατά σειρά τα εξής:

- Δεδομένων των συντεταγμένων του σημείου που επέλεξε ο χρήστης σαν αρχικό σημείο, χρησιμοποιείται το R* tree για να βρεθεί ο κόμβος του οδικού δικτύου ο οποίος βρίσκεται στη μικρότερη απόσταση από το σημείο.
- Εκτελείται ο αλγόριθμος Dijkstra με αρχικό σημείο τον πλησιέστερο κόμβο. Ο αλγόριθμος τερματίζεται όταν ξεπεραστεί το μέγιστο χρονικό διάστημα δηλαδή τα 60'. Το αποτέλεσμα του αλγορίθμου είναι το σύνολο των ισοχρονικών ακμών με τους χρόνους διάσχισης τους. Επίσης, όπως έχουμε αναφέρει στην § 3.5.1, κατά την εκτέλεση του Dijkstra υπολογίζεται για κάθε χρονικό διάστημα ο ισοχρονικός κόμβος με την μεγαλύτερη συντεταγμένη x ή αλλιώς το δεξιότερο ισοχρονικό σημείο.
- Στη συνέχεια αναλαμβάνει δράση ο αλγόριθμος του Edges' Hull ο οποίος εκτελείται μία φορά για κάθε χρονικό διάστημα. Για κάθε χρονικό διάστημα ξεκινάει από τον δεξιότερο ισοχρονικό κόμβο και υπολογίζει το περίγραμμα των ισοχρονικών ακμών που έχουν χρόνο διάσχισης μικρότερο ή ίσο με το αντίστοιχο χρονικό διάστημα.
- Μετά τον υπολογισμό των πολυγώνων, γίνεται η κωδικοποίησή τους με τον αλγόριθμο Encoded Polyline.
- Τα έξι strings που προκύπτουν από τον αλγόριθμο Encoded Polyline κωδικοποιούνται σε JSON κείμενο για την αποστολή στον client. Το πώς γίνεται η κωδικοποίηση περιγράφεται στην επόμενη παράγραφο.
- Τέλος, το JSON κείμενο αποστέλλεται από τον server στον client αφού πραγματοποιηθεί και GZIP compression εφόσον ο client την υποστηρίζει.

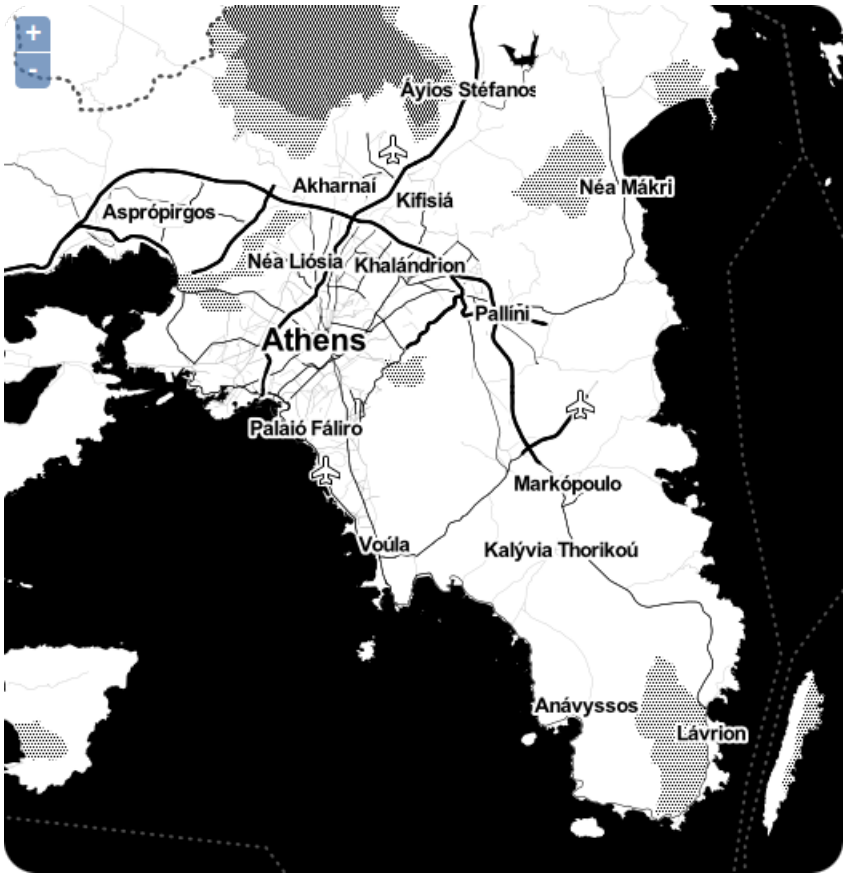
Αφού παρακάτω περιγραφεί και ο client υπάρχει σχήμα που περιγράφει και σχηματικά όλα όσα συμβαίνουν σε client και server όταν δημιουργείται ένα νέο request.

5.4 Ο client

Ο client της εφαρμογής μπορεί να είναι ένας απλός web-browser που μπορεί να εκτελεί κώδικα JavaScript. Αυτό ήταν κι ένα από τα πλεονεκτήματα που μας οδήγησαν στην επιλογή της web-εφαρμογής. Παρακάτω περιγράφεται η λειτουργία του client της εφαρμογής δηλαδή η φόρτωση του χάρτη, η δυνατότητα στο χρήστη για επιλογή σημείου αρχής, η αποστολή του request προς τον server και η οπτικοποίηση των ισοχρονικών καμπυλών που έρχονται σαν απάντηση. Όλα αυτά πραγματοποιούνται με κώδικα JavaScript.

5.4.1 Φόρτωση του χάρτη

Ο client της εφαρμογής χρησιμοποιεί δεδομένα από το OpenStreetMap (για το οποίο μιλήσαμε στην § 5.1.1) για να απεικονίσει τον χάρτη της Αθήνας στην αρχική σελίδα. Για την απεικόνιση έχει χρησιμοποιηθεί ειδικό rendering ώστε ο χάρτης να είναι μόνο ασπρόμαυρος και τελικά τα πολύγωνα των ισοχρονικών καμπύλων να φαίνονται έντονα. Στο σχήμα 5.7 φαίνεται ο αρχικός χάρτης όπως παρουσιάζεται όταν φορτωθεί η σελίδα της εφαρμογής.



Σχήμα 5.7: Ο χάρτης της Αθήνας όπως φορτώνεται στην αρχική σελίδα

Είναι πολύ εύκολο ο χρήστης να περιηγηθεί στον χάρτη της Αθήνας κάνοντας zoom και "σέρνοντας" τον προς οποιαδήποτε κατεύθυνση. Για παράδειγμα, στο σχήμα 5.8 φαίνεται στο χάρτη μία περιοχή στη Νέα Σμύρνη όπου έχει κάνει zoom ο χρήστης.

Σχήμα 5.8: Μία περιοχή στη Νέα Σμύρνη μετά από περιήγηση και zoom του χρήστη στον χάρτη

5.4.2 Αποστολή του request και JSON

Ο χρήστης έχει τη δυνατότητα να κάνει "κλικ" πάνω σε οποιοδήποτε σημείο στο χάρτη της Αθήνας. Όταν το κάνει, ο client αναλαμβάνει να στείλει, χρησιμοποιώντας την τεχνολογία AJAX που επιτρέπει την ασύγχρονη αποστολή και ανάκτηση δεδομένων, ένα HTTP request στον server και συγκεκριμένα στο java servlet της εφαρμογής. Το request αυτό περιλαμβάνει δύο παραμέτρους, x και y που είναι οι συντεταγμένες του σημείου του χάρτη όπου έγινε το "κλικ".

Μετά από λίγο έρχεται η απάντηση από τον server. Αρχικά, αν έχει εφαρμοστεί GZIP compression γίνεται το decompression. Στη συνέχεια, ο client έχει τα έξι strings που αντιπροσωπεύουν τις έξι ισοχρονικές καμπύλες κωδικοποιημένες με τον αλγόριθμο Encoded Polyline σε μορφή JSON [JSON]. Το JSON (JavaScript Object Notation) είναι

ένα απλό format κειμένου για την παρουσίαση δεδομένων. Είναι πλέον πολύ διαδεδομένο για τη χρήση σε web-εφαρμογές και είναι εύκολα διαχειρίσιμο σε πολλές διαδεδομένες γλώσσες προγραμματισμού όπως C, C++, C#, Java, JavaScript, Perl, Python. Στην περίπτωσή μας το JSON κείμενο που αποστέλλεται από των server και περιέχει τις έξι ισοχρονικές καμπύλες είναι ως εξής:

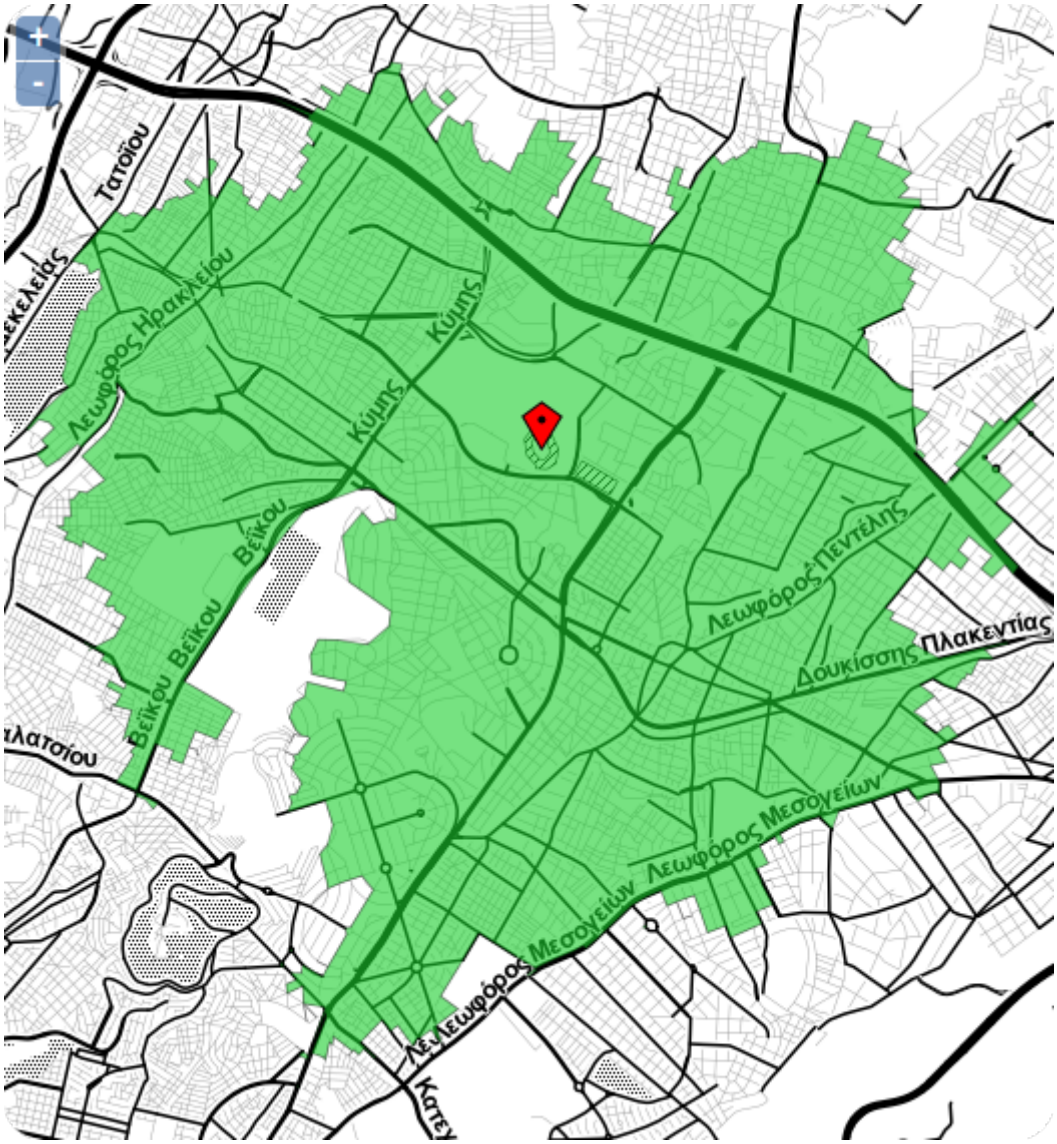
```
{
  "isochrone1" : "EncodedPolyline_string_for_isochrone_1",
  "isochrone2" : "EncodedPolyline_string_for_isochrone_2",
  "isochrone3" : "EncodedPolyline_string_for_isochrone_3",
  "isochrone4" : "EncodedPolyline_string_for_isochrone_4",
  "isochrone5" : "EncodedPolyline_string_for_isochrone_5",
  "isochrone6" : "EncodedPolyline_string_for_isochrone_6"
}
```

Ο client λαμβάνει αυτά τα έξι strings και τα αποκωδικοποιεί με τον αλγόριθμο αποκωδικοποίησης του Encoded Polyline. Οπότε, τελικά, έχει στη διάθεσή του τα έξι πολύγωνα που αντιστοιχούν στα χρονικά διαστήματα των 10', 20', ... , 60'.

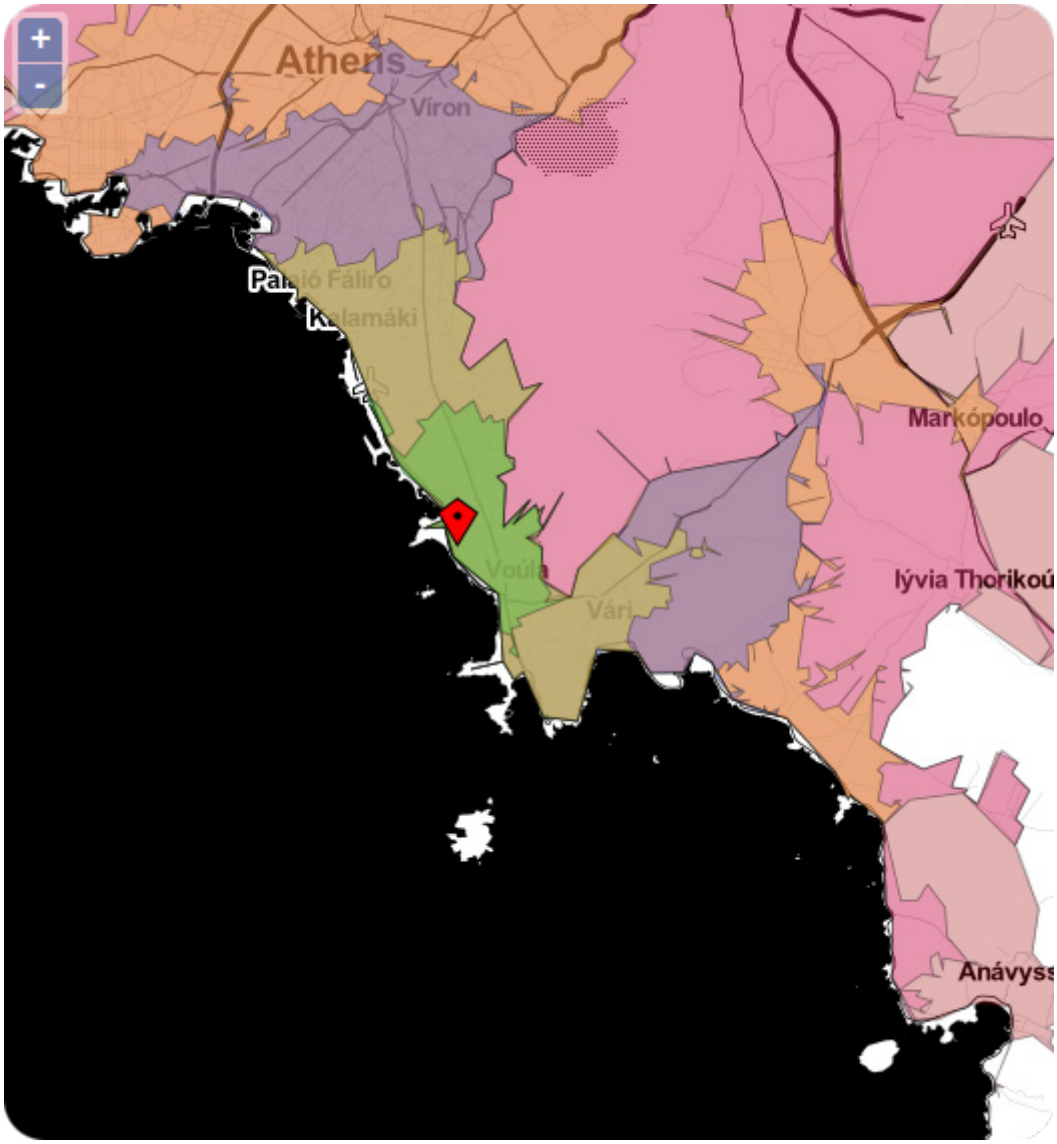
5.4.3 Παρουσίαση ισοχρονικών καμπύλων

Αυτό που μένει είναι η παρουσίαση των ισοχρονικών καμπυλών επάνω στον χάρτη της Αθήνας. Δίνεται η δυνατότητα στον χρήστη μέσω ενός checkbox να επιλέξει αν θέλει να γίνει εξομάλυνση της καμπύλης πριν αυτή παρουσιαστεί όπως συζητήθηκε στην § 3.7. Αν το ζητούμενο είναι η απόλυτη ακρίβεια τότε αυτό δεν είναι επιθυμητό.

Οι ισοχρονικές καμπύλες για τα διάφορα χρονικά διαστήματα παρουσιάζονται πάνω στον χάρτη της Αθήνας σαν πολύγωνα με διαφορετικά χρώματα γεμίσματος το καθένα. Στα σχήματα 5.9 και 5.10 φαίνονται δύο περιπτώσεις παρουσίασης ισοχρονικών καμπύλων.



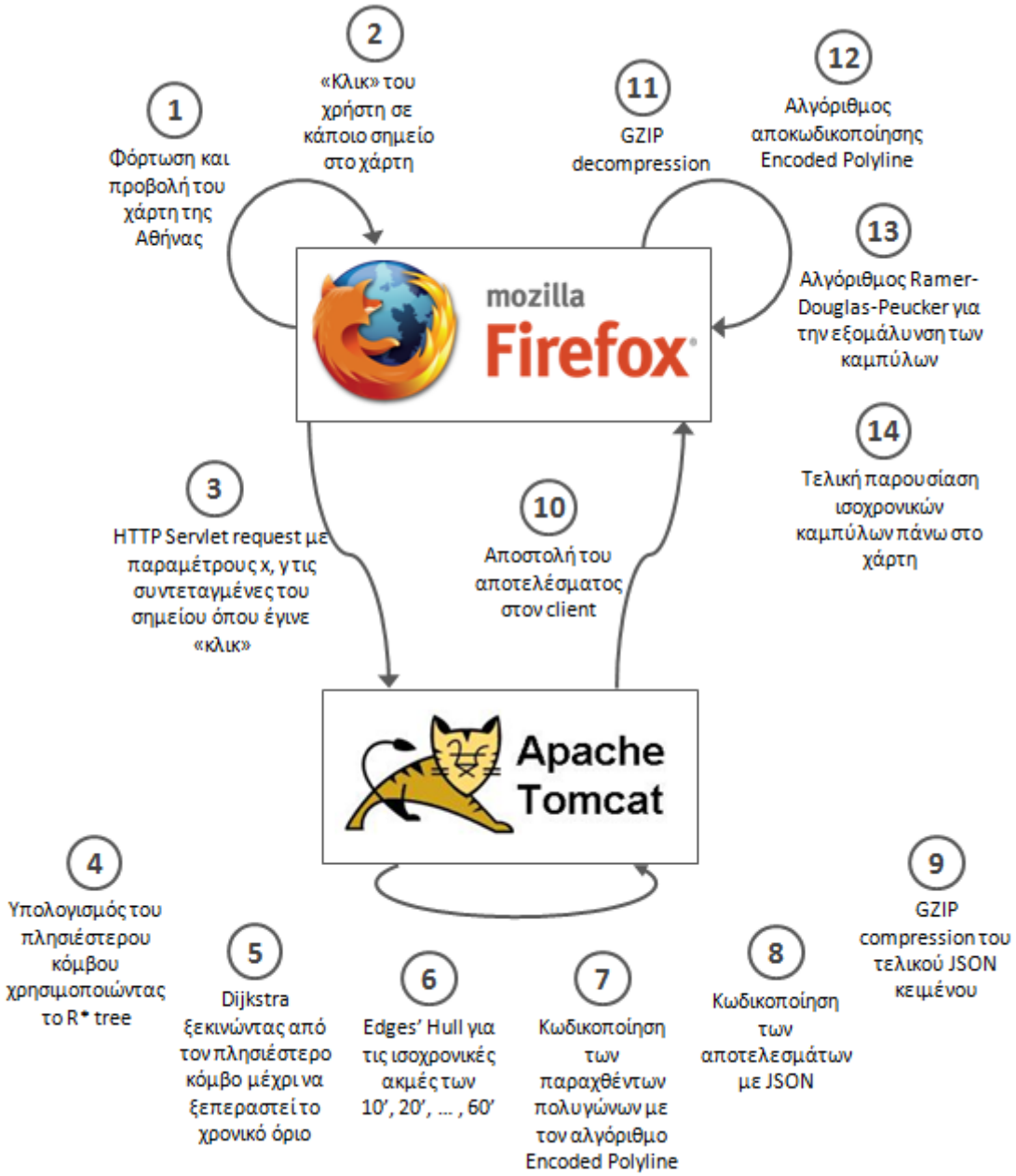
Σχήμα 5.9: Οι ισοχρονικές καμπύλες χρονοαπόστασης 5' από το Ολυμπιακό Αθλητικό Κέντρο Αθηνών (ΟΑΚΑ) χωρίς εξομάλυνση



Σχήμα 5.10: Οι ισοχρονικές καμπύλες χρονοαπόστασης 5', 10', 15', 20', 25', 30' από το νοσοκομείο "Ασκληπιείο Βούλας" με εξομάλυνση

5.5 Η πορεία ενός request

Στο σχήμα 5.11 παρουσιάζονται σχηματικά όλα τα βήματα που συμβαίνουν σε client και server από τη στιγμή που ο χρήστης θα επιλέξει ένα αρχικό σημείο πάνω στο χάρτη μέχρι τη στιγμή που θα απεικονιστούν οι αντίστοιχες ισοχρονικές καμπύλες.



Σχήμα 5.11: Η διαδικασία που πραγματοποιείται σε client και server κατά τη δημιουργία ενός request

Μετά την ανάλυση της αρχιτεκτονικής του συστήματος που υλοποιεί την εφαρμογή, αυτό που μένει είναι η πειραματική ανάλυση και αξιολόγηση του αποτελέσματος.

6

Πειραματική ανάλυση και αξιολόγηση

Στο κεφάλαιο αυτό αξιολογούμε την επίδοση της εφαρμογής. Αφού πρώτα περιγράψουμε το υπολογιστικό περιβάλλον στο οποίο έγινε η πειραματική ανάλυση, στη συνέχεια δίνουμε συγκεκριμένους χρόνους εκτέλεσης για κάθε βήμα που πραγματοποιείται κατά την εκτέλεση της εφαρμογής. Αρχικός μας στόχος ήταν η εφαρμογή να είναι όσο το δυνατόν αποδοτικότερη λαμβάνοντας υπόψη δύο κριτήρια, την ταχύτητας εκτέλεσης και την ακρίβεια του υπολογισμού. Ο αλγόριθμος υπολογισμού του Edges' Hull έδωσε αρκετά μεγάλη ακρίβεια στον υπολογισμό των ισοχρονικών καμπύλων. Η ταχύτητα εκτέλεσης αναλύεται στις παρακάτω παραγράφους.

6.1 Το περιβάλλον εκτέλεσης

Για την εκτέλεση των πειραμάτων, στην πλευρά του server χρησιμοποιήθηκε ένα μηχάνημα με διπύρηνο, 64-bit επεξεργαστή Intel® Core™2 Duo CPU E6550 @ 2.33GHz και 2 GB μνήμη. Στο μηχάνημα τρέχει λειτουργικό Ubuntu 12.04 LTS. Σαν database server χρησιμοποιήθηκε ο PostgreSQL 9.1. Σαν web server χρησιμοποιήθηκε ο Apache Tomcat 7.0.29. Το σύστημα έχει εγκατεστημένη τη Java Version 1.7. Από την πλευρά του client χρησιμοποιήθηκε ένα απλό μηχάνημα με Mozilla Firefox 15.0.1.

Υπενθυμίζουμε ότι χρησιμοποιήθηκε το οδικό δίκτυο της Αθήνας. Αυτό περιλαμβάνει 92728 κόμβους και 634427 ακμές. Ο αριθμός των τομών μη επικοινωνούντων ακμών, οι οποίες υπολογίστηκαν όπως αναλύεται στην § 5.2.3, ανέρχεται στις 4226.

6.2 Χρόνος εκκίνησης της εφαρμογής

Όταν εκκινείται ο web server τότε εκτελούνται και οι εργασίες που χρειάζονται για την εκκίνηση της εφαρμογής. Η ταχύτητα εκκίνησης της εφαρμογής δεν είναι κρίσιμη. Αυτό όμως, δε σημαίνει ότι θα ήταν ανεκτό να διαρκούσε κάμποσα λεπτά. Ωστόσο, αν διαρκεί μόνο λίγα δευτερόλεπτα δεν υπάρχει κανένα πρόβλημα.

Στον Πίνακα 6.1 παρουσιάζονται οι χρόνοι που διαρκεί κατά μέσο όρο η κάθε εργασία στην εκκίνηση της εφαρμογής.

Εργασία	Χρόνος
Σύνδεση με βάση	0.18 sec
Μέτρηση μεγέθους πινάκων	0.69 sec
Φόρτωση κόμβων	1.24 sec
Φόρτωση ακμών	2.16 sec
Φόρτωση τομών	0.01 sec
Προ-επεξεργασία γράφου	0.33 sec
R* tree indexing	5.23 sec
ΣΥΝΟΛΙΚΑ	***9.85 sec***

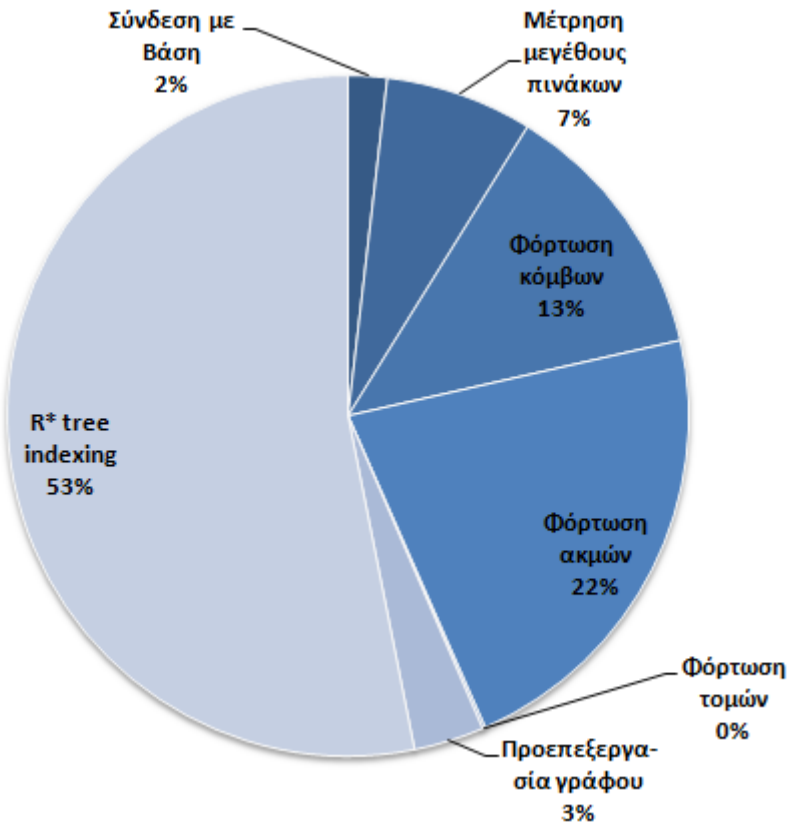
Πίνακας 6.1

Για να γίνει σαφές το τι γίνεται σε κάθε βήμα, περιγράφουμε πολύ συνοπτικά το καθένα:

- **Σύνδεση με βάση**. Πραγματοποιείται η σύνδεση του servlet το οποίο εκτελείται στον application server με τον server της βάσης δεδομένων.
- **Μέτρηση μεγέθους πινάκων**. Με τρία απλά SQL queries ανακτάται από τη βάση το πλήθος των κόμβων, των ακμών και των τομών.
- **Φόρτωση κόμβων**. Ανακτώνται όλοι οι κόμβοι με τα χρήσιμα χαρακτηριστικά τους από τη βάση.

- **Φόρτωση ακμών**. Ανακτώνται όλες οι ακμές με τα χρήσιμα χαρακτηριστικά τους από τη βάση.
- **Φόρτωση τομών**. Ανακτάται ο πίνακας που περιέχει όλη τη χρήσιμη πληροφορία σχετικά με τις τεμνόμενες ακμές του οδικού δικτύου.
- **Προ-επεξεργασία γράφου**. Εκτελείται η προ-επεξεργασία του γράφου του οδικού δικτύου η οποία οδηγεί στην απαλοιφή όλων των τομών των ακμών του.
- **R* tree indexing**. Πραγματοποιείται το indexing των κόμβων του δικτύου για την γρήγορη εύρεση του πλησιέστερου κόμβου σε ένα δεδομένο σημείο πάνω στο χάρτη.

Στο σχήμα 6.1 παρουσιάζεται η κατανομή του χρόνου εκκίνηση της εφαρμογής στις διάφορες εργασίες που πραγματοποιούνται.



Σχήμα 6.1: Η κατανομή του χρόνου εκκίνησης της εφαρμογής

6.3 Χρόνος απάντησης ενός request

Το κρίσιμο τμήμα της εφαρμογής ως προς τις απαιτήσεις μας στην ταχύτητα είναι η διαδικασία της απάντησης σε ένα request. Υπενθυμίζουμε ότι θέλουμε η εφαρμογή να έχει

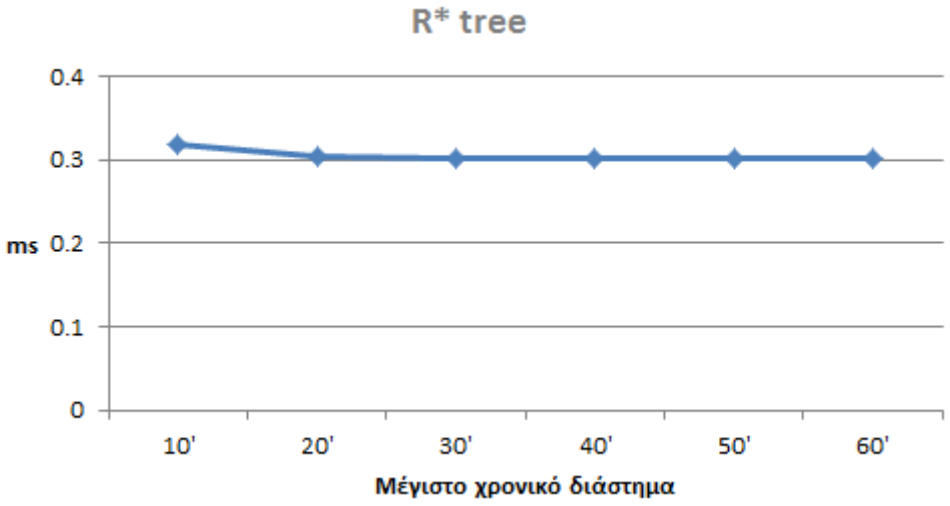
άμεση απόκριση στα requests του χρήστη δηλαδή, όσο αυτό είναι δυνατό, να βλέπει τα αποτελέσματα στην οθόνη του αμέσως μετά το "κλικ" της επιλογής του σημείου. Στην παράγραφο αυτή αναλύουμε τον χρόνο που κάνει ο server για να απαντήσει σε ένα request. Επίσης, αναλύουμε την κατανομή του συνολικού χρόνου στις επιμέρους διαδικασίες που πραγματοποιούνται.

Για το σκοπό αυτό, πήραμε μετρήσεις χρόνων εκτέλεσης για πολλές χιλιάδες requests. Συγκεκριμένα, για κάθε ένα από 9200 περίπου αρχικά σημεία πραγματοποιήσαμε 6 requests. Το κάθε request αποτελούνταν από διαφορετικά χρονικά διαστήματα:

- Το μέγιστο χρονικό διάστημα για τα 6 requests ήταν αντίστοιχα 10', 20', 30', 40', 50', 60'.
- Κάθε request αποτελούνταν από 6 ισοκατανεμημένα χρονικά διαστήματα.
- Για παράδειγμα, αυτό με μέγιστο τα 30' αποτελούνταν από τα χρονικά διαστήματα 5', 10', 15', 20', 25', 30' ενώ αυτό με μέγιστο τα 60' από τα 10, 20', 30', 40', 50', 60'.

6.3.1 Χρόνος αναζήτησης του πλησιέστερου κόμβου μέσω του R* tree

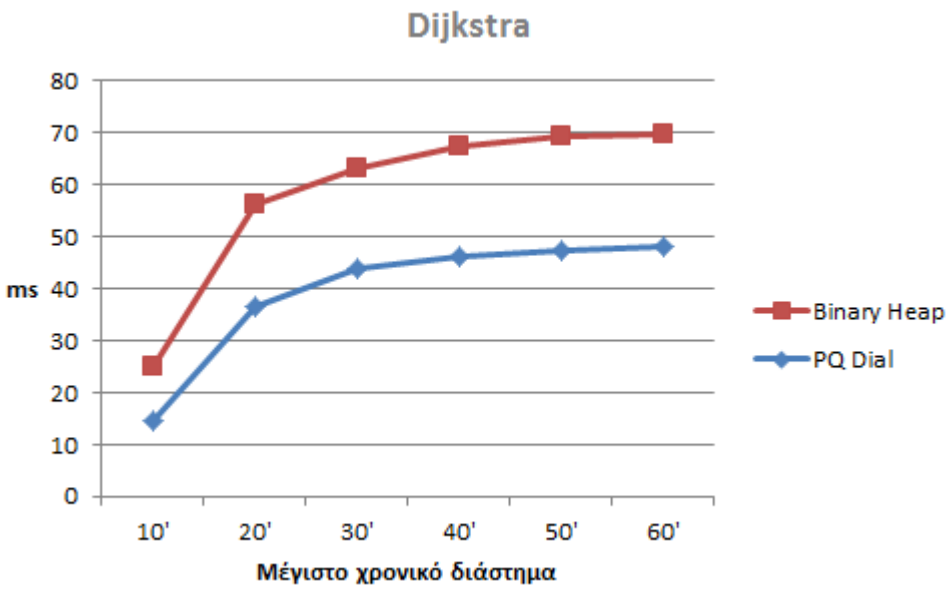
Το indexing των κόμβων σε ένα R* tree με βάση τις συντεταγμένες τους δίνει πολύ μεγάλη αποδοτικότητα στην εύρεση του πλησιέστερου κόμβου από το σημείο που επιλέγει ο χρήστης. Συγκεκριμένα, ο χρόνος της αναζήτησης, όπως φαίνεται και στο σχήμα 6.2, περιορίζεται κατά μέσο όρο στα 0,3 msec, χρόνος που είναι ελάχιστος και δεν επηρεάζει καθόλου την συνολική ταχύτητα απόκρισης της εφαρμογής.



Σχήμα 6.2: Οι χρόνοι εύρεσης του πλησιέστερου κόμβου μέσω του R* tree

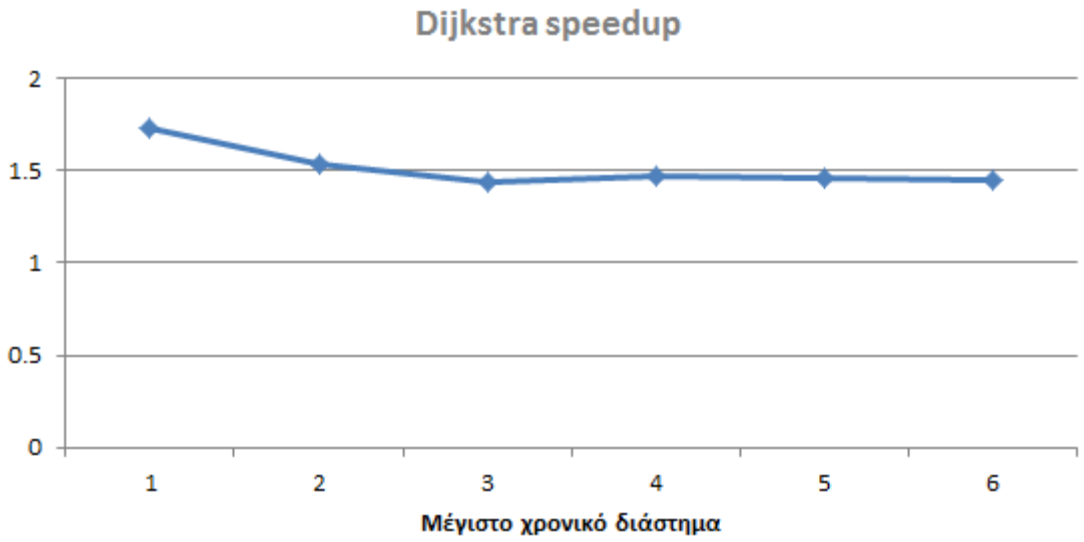
6.3.2 Χρόνος εκτέλεσης του Dijkstra

Το κρισιμότερο από άποψη ταχύτητας απόκρισης τμήμα της εξυπηρέτησης ενός request είναι η εκτέλεση του αλγορίθμου Dijkstra καθώς αυτή παίρνει το μεγαλύτερο χρόνο. Για αυτό το λόγο προσπαθήσαμε να επιταχύνουμε την εκτέλεση του αλγορίθμου χρησιμοποιώντας την ουρά προτεραιότητας του Dial η οποία αναλύθηκε στην § 3.3.4. Στο σχήμα 6.3 παρέχεται ένα διάγραμμα σύγκρισης των χρόνων που επιτυγχάνονται για την εκτέλεση του Dijkstra χρησιμοποιώντας την ουρά του Dial ή χρησιμοποιώντας ένα binary heap που ήταν η αρχική προσέγγιση.



Σχήμα 6.3: Οι χρόνοι εκτέλεσης του αλγορίθμου Dijkstra χρησιμοποιώντας binary heap ή την ουρά προτεραιότητας του Dial

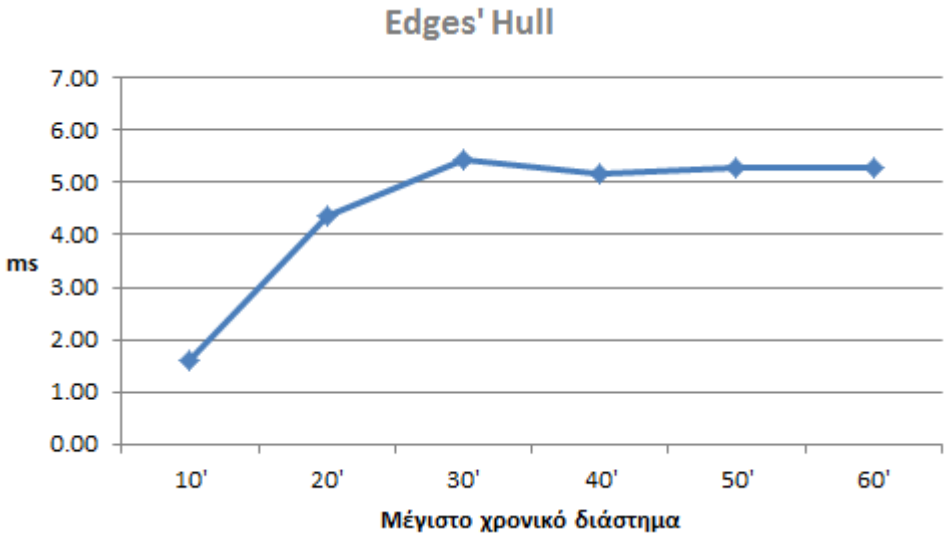
Όπως παρατηρούμε η ταχύτητα της εκτέλεσης του Dijkstra βελτιώνεται σημαντικά. Για την ακρίβεια, ο χρόνος εκτέλεσης του Dijkstra στην εφαρμογή πέφτει κάτω από τα 50 msec ενώ με το binary heap ήταν περίπου στα 70 msec. Στο σχήμα 6.4 παρουσιάζεται και ένα διάγραμμα του speedup που επιτυγχάνεται με τη χρήση της ουράς προτεραιότητας του Dial. Όπως βλέπουμε, αυτό είναι γύρω στο 1,5.



Σχήμα 6.4: Το speedup που επιτυγχάνεται χρησιμοποιώντας την ουρά προτεραιότητας του Dial

6.3.3 Χρόνος εκτέλεσης του αλγορίθμου για το Edges' Hull

Όπως έχουμε αναφέρει, το Edges' Hull είναι μία πολύ καλή λύση για τις ισοχρονικές καμπύλες χρονοαπόστασης τόσο από άποψη ακρίβειας όσο και από άποψη ταχύτητας. Ο υπολογισμός των Edges' Hulls στην εφαρμογή μας διαρκεί μόνο μερικά milliseconds. Στο σχήμα 6.5 φαίνονται οι χρόνοι που χρειάστηκαν στα πειράματα που εκτελέσαμε.



Σχήμα 6.5: Οι χρόνοι εκτέλεσης του αλγορίθμου υπολογισμού του Edges' Hull

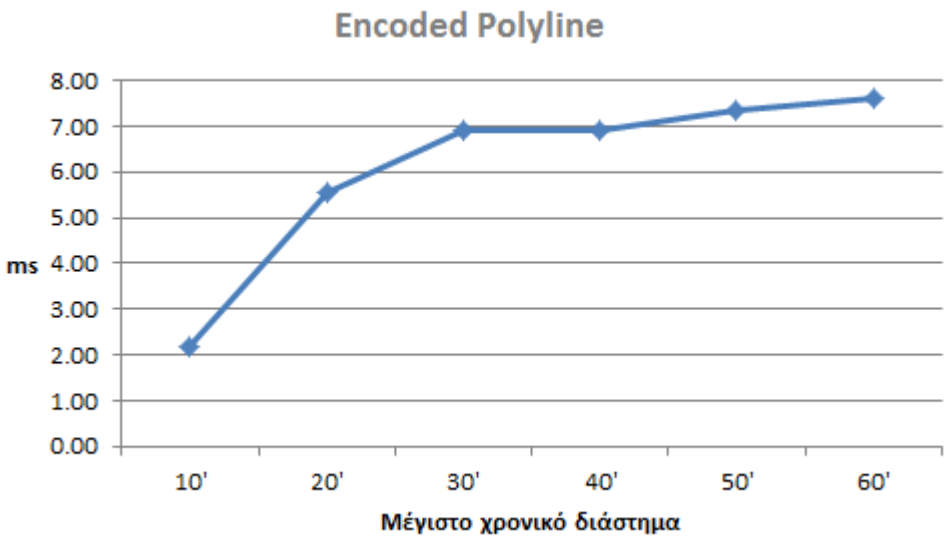
Μία πρώτη παρατήρηση είναι ότι όταν το μέγιστο χρονικό διάστημα είναι μικρότερο, ο χρόνος υπολογισμού είναι επίσης μικρότερος. Αυτό είναι λογικό καθώς με μικρότερο χρονικό διάστημα έχουμε λιγότερες ισοχρονικές ακμές και το Edges' Hull είναι

μικρότερο και αποτελείται από λιγότερες ακμές. Παρατηρούμε, όμως, ότι μετά τα 30' λεπτά δεν υπάρχει σαφής αύξηση του χρόνου υπολογισμού. Αυτό εξηγείται, διότι το οδικό δίκτυο της Αθήνας αραιώνει αρκετά μετά τα 30' αφού βγαίνει από την κύρια αστική περιοχή οπότε και το πλήθος των ακμών του Edges' Hull δεν αυξάνεται αισθητά.

Συνολικά, ο χρόνος υπολογισμού των Edges' Hulls στην εφαρμογή περιορίζεται στα 5 msec περίπου οπότε μπορούμε να πούμε ότι δεν επηρεάζει αισθητά την απόκριση της εφαρμογής.

6.3.4 Χρόνος εκτέλεσης του αλγορίθμου Encoded Polyline

Ο αλγόριθμος Encoded Polyline πραγματοποιεί κωδικοποίηση, και μέσω αυτής συμπίεση, των καμπύλων έτσι ώστε να μικρύνει ο όγκος των δεδομένων προς αποστολή μέσω του δικτύου και άρα να γίνει ταχύτερα η μεταφορά και με λιγότερη επιβάρυνση του δικτύου. Όμως, αυτό είναι συμφέρον μόνο αν η εκτέλεση του αλγορίθμου δεν διαρκεί αρκετά ώστε να υπερκαλύπτει τα προηγούμενα οφέλη. Στο σχήμα 6.6 παρουσιάζεται το διάγραμμα με τους χρόνους εκτέλεσης του αλγορίθμου στα πειράματα.

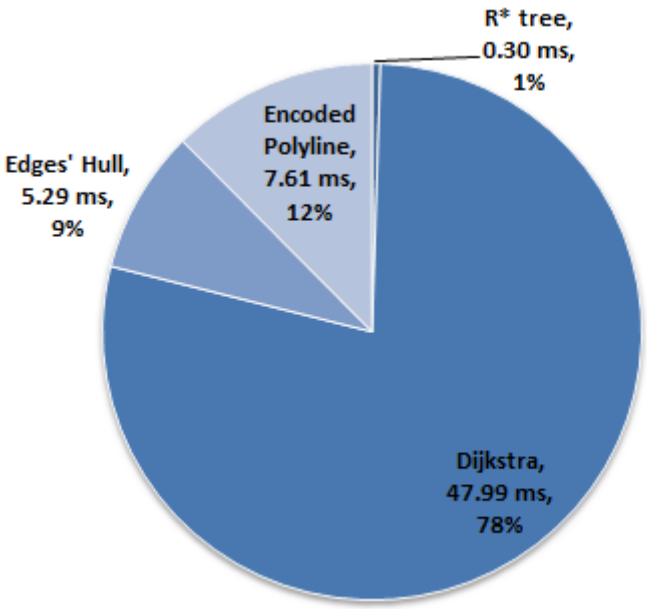


Σχήμα 6.6: Οι χρόνοι εκτέλεσης του αλγορίθμου Encoded Polyline

Παρατηρούμε ότι όσο αυξάνεται το χρονικό διάστημα οπότε αυξάνεται και το μέγεθος των Edges' Hulls αυξάνεται και ο χρόνος εκτέλεσης του αλγορίθμου, πράγμα φυσιολογικό αφού αυξάνονται τα δεδομένα προς κωδικοποίηση. Ο αλγόριθμος είναι αρκετά αποδοτικός δεδομένου ότι δεν ξεπερνάει τα 8 msec οπότε για χάρη της συμπίεσης που επιτυγχάνει και η οποία παρουσιάστηκε στην § 3.6.1 είναι συμφέρουσα η χρήση του.

6.3.5 Συνολικός χρόνος

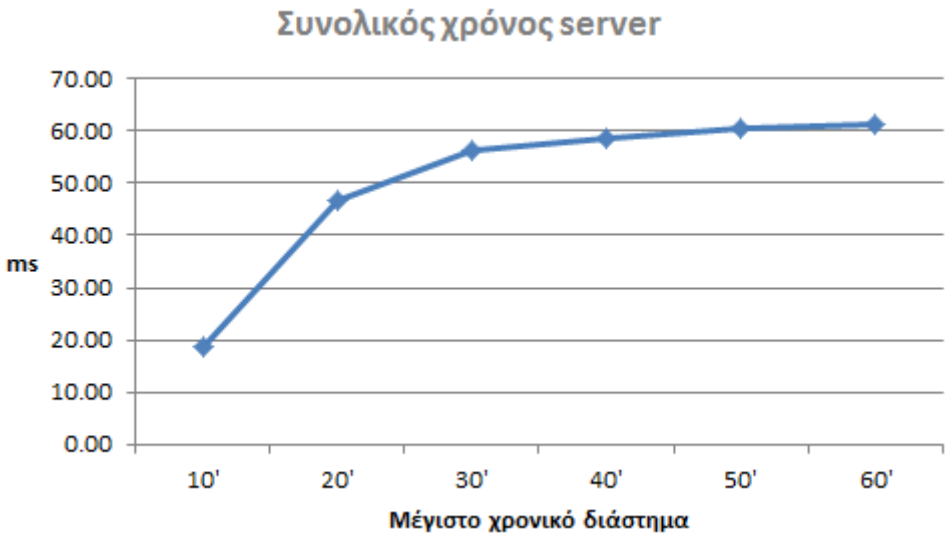
Ο συνολικός χρόνος που χρειάζεται ο server για να απαντήσει σε ένα request είναι και το μέτρο της ταχύτητας στην απόκριση της εφαρμογής μας. Αρχικά, στο σχήμα 6.7 βλέπουμε την ποσοστιαία επιβάρυνση των διαφόρων διαδικασιών που αναλύσαμε μία προς μία προηγουμένως στον συνολικό χρόνο εξυπηρέτησης.



Σχήμα 6.7: Η κατανομή του χρόνου εξυπηρέτησης ενός request ανάμεσα στις διάφορες διαδικασίες

Αυτό που γίνεται φανερό είναι πως η εκτέλεση του αλγορίθμου Dijkstra είναι πράγματι η πιο αργή λειτουργία που πραγματοποιείται στον server και αυτή είναι υπεύθυνη για περισσότερο από τα 3/4 του χρόνου εξυπηρέτησης ενός request. Όποτε, η βελτίωση που πραγματοποιήθηκε μέσω του priority queue Dial ήταν πραγματικά καθοριστική για την αύξηση της ταχύτητας απόκρισης της εφαρμογής.

Στο σχήμα 6.8 φαίνονται και οι συγκεκριμένοι χρόνοι που χρειάζεται ο server για να απαντήσει σε κάποιο request.



Σχήμα 6.8: Οι συνολικοί χρόνοι εξυπηρέτησης ενός request

Όπως βλέπουμε, η καμπύλη ακολουθεί τη μορφή της καμπύλης του διαγράμματος για τον αλγόριθμο Dijkstra αφού αυτός είναι υπεύθυνος για το μεγαλύτερο ποσοστό του συνολικού χρόνου. Από εκεί και πέρα παρατηρούμε ότι στην δυσκολότερη περίπτωση δηλαδή για μέγιστο χρονικό διάστημα 60' ο server χρειάζεται περίπου 60 msec συνολικά για να απαντήσει στο request. Αυτό επιβεβαιώνει ότι η εφαρμογή είναι, πράγματι, άμεσα αποκρίσιμη στα requests του χρήστη όπως παρατηρείται και στην πράξη.

6.4 Χρόνος εξομάλυνσης και οπτικοποίησης

Αφού έρθει η απάντηση στο request του client μένει η εξομάλυνση της καμπύλης, αν έχει ενεργοποιηθεί η δυνατότητα αυτή, και η οπτικοποίηση των τελικών πολυγώνων.

Η εκτέλεση του αλγορίθμου Ramer-Douglas-Peucker που πραγματοποιεί την εξομάλυνση τρέχει στον υπολογιστή του client και διαρκεί μόνο ελάχιστα milliseconds ώστε δεν επιβαρύνει την αποκρισιμότητα της εφαρμογής. Μάλιστα, επειδή μειώνει σε σημαντικό ποσοστό τις ακμές των τελικών πολυγώνων κάνει την διαδικασία της οπτικοποίησης ταχύτερη και στις περισσότερες περιπτώσεις οι ισοχρονικές καμπύλες παρουσιάζονται ταχύτερα όταν είναι ενεργοποιημένη η εξομάλυνση παρά όταν δεν είναι.

Στον Πίνακα 6.2 παρουσιάζονται κάποια στατιστικά που δείχνουν κατά πόσο μειώνονται οι ακμές των τελικών πολυγώνων κατά την εξομάλυνση.

χρονικό όριο	μέσος #σημείων πριν την εξομάλυνση	μέσος #σημείων μετά την εξομάλυνση	ποσοστό μείωσης
10’	2581	217	91,6 %
20’	3801	507	86,7 %
30’	3141	547	82,6 %
40’	3331	642	80,8 %
50’	3376	662	80,4 %
60’	3148	602	80,9 %

Πίνακας 6.2

6.5 Συμπέρασμα

Το μεγαλύτερο μέρος του υπολογισμού γίνεται στον server, όμως υπάρχει και ένα κομμάτι, η εξομάλυνση, που πραγματοποιείται στον client εκμεταλλευόμενο τους αδρανείς πόρους του client. Η εργασία που πραγματοποιείται στον client καταφέρνει να αποφορτίσει τον server χωρίς να δημιουργεί κίνδυνο υπερφόρτωσης του client καθώς είναι ελαφριά και διαρκεί μόλις λίγα milliseconds.

Το συμπέρασμα από την πειραματική αξιολόγηση της εφαρμογής είναι ότι σε γενικές γραμμές έχουμε μία εφαρμογή η οποία είναι αρκετά γρήγορη και άμεσα αποκρίσιμη στον χρήστη.

7

Επίλογος

Το αποτέλεσμα της παρούσας εργασίας ήταν η δημιουργία μίας web-εφαρμογής η οποία υπολογίζει γρήγορα και με αρκετά μεγάλη ακρίβεια τις ισοχρονικές καμπύλες χρονοαπόστασης από οποιοδήποτε σημείο στο οδικό δίκτυο της Αθήνας.

7.1 Τα κύρια σημεία

Ο κορμός της εφαρμογής στον οποίο οφείλεται και το συντριπτικά μεγαλύτερο ποσοστό του χρόνου υπολογισμού είναι η εκτέλεση του αλγορίθμου Dijkstra. Παρόλο που τελικά πετύχαμε έναν αρκετά μικρό χρόνο υπολογισμού, με τη χρήση και της ουράς προτεραιότητας του Dial, που κάνει την εφαρμογή άμεσα αποκρίσιμη στην επιλογή του αρχικού σημείου από τον χρήστη, περαιτέρω βελτίωση του χρόνου εκτέλεσης θα ήταν ευπρόσδεκτη και θα αύξανε την αποδοτικότητα της εφαρμογής.

Το ουσιαστικότερο, όμως, βήμα της εφαρμογής είναι ο υπολογισμός των ισοχρονικών καμπύλων. Μέσω του Edges' Hull καταφέραμε να έχουμε έναν αρκετά ακριβή υπολογισμό των καμπύλων και μάλιστα με έναν αλγόριθμο ο οποίος είναι αρκετά αποδοτικός και δεν επιβαρύνει την ταχύτητα της εφαρμογής.

Με το indexing των κόμβων και τη δημιουργία του R* tree μπορούμε να βρίσκουμε τον πλησιέστερο στο επιλεγμένο σημείο κόμβο πρακτικά σε μηδενικό χρόνο.

Επίσης, με τον αλγόριθμο Encoded Polyline και το GZIP compression έχουμε καταφέρει να μειώσουμε τον όγκο των δεδομένων που διακινούνται στο δίκτυο κατά ένα ποσοστό μεγαλύτερο του 90% φτάνοντας σε ένα μέγεθος περίπου 30 KB το οποίο είναι ασήμαντο ακόμα και για αργές συνδέσεις.

Τέλος, με την εξομάλυνση των καμπύλων επιτυγχάνεται όχι μόνο ένα καλύτερο αισθητικά αποτέλεσμα αλλά λόγω της μείωσης των ακμών των πολυγώνων κατά περισσότερο από 80% επιτυγχάνεται και μείωση του χρόνου οπτικοποίησης αρκετή ώστε ο συνολικός χρόνος απόκρισης να είναι μικρότερος σε σχέση με την περίπτωση που δε γίνεται εξομάλυνση.

7.2 Μελλοντικές επεκτάσεις

Μελλοντικά, η εφαρμογή θα μπορούσε να εμπλουτιστεί ώστε να μπορεί να δέχεται live δεδομένα από την κίνηση στους δρόμους της πόλης και να πραγματοποιεί τον υπολογισμό σύμφωνα με αυτά, προσαρμόζοντας τα βάρη των ακμών του γράφου. Με αυτό τον τρόπο ο χρήστης θα μπορεί να βλέπει τις ισοχρονικές καμπύλες όπως αυτές διαμορφώνονται αν ξεκινήσει εκείνη τη στιγμή από το αρχικό σημείο καθώς και πώς αυτές μεταβάλλονται κατά τη διάρκεια της ημέρας.

Για το παραπάνω αρκεί να αλλάζουν τα βάρη των ακμών του γράφου χωρίς να αλλάξει κάτι στους αλγορίθμους ή την υπόλοιπη εφαρμογή. Με την ίδια λογική, αλλάζοντας δηλαδή τα βάρη των ακμών, είναι δυνατό με την ίδια εφαρμογή να γίνεται ο υπολογισμός των ισοχρονικών καμπύλων χρονοαπόστασης για άλλα μέσα μετακίνησης όπως το ποδήλατο ή το περπάτημα.

Μία άλλη σκέψη θα ήταν η ενσωμάτωση στην εφαρμογή σημείων ενδιαφέροντος όπως μνημεία, ξενοδοχεία, πολυκαταστήματα ή και νοσοκομεία. Με αυτόν τον τρόπο, μέσω της εφαρμογής θα μπορούν να απαντώνται ενδιαφέροντα ερωτήματα όπως «Πόσα σημεία τουριστικού ενδιαφέροντος είναι σε ακτίνα 10’ από ένα ξενοδοχείο;». Ή αν αντιστρέψουμε το γράφο δηλαδή κάνουμε ανάποδη την κατεύθυνση των ακμών μπορούν να απαντηθούν ακόμα πιο ενδιαφέροντα ερωτήματα όπως «Από ποιες γειτονιές είναι προσβάσιμο μέσα σε 10’ ένα πολυκατάστημα ή ένα νοσοκομείο;».

8

Βιβλιογραφία

[BGL+08] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, I. Timko, Computing isochrones in multi-modal, schedule-based transport networks, GIS '08, Nov. 2008

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, SIGMOD '90, 1990

[CGR93] B. Cherkassky , A. V. Goldberg , T. Radzik, Shortest Paths Algorithms: Theory And Experimental Evaluation, Aug. 1993

[Cloudm] http://downloads.cloudmade.com/

[CLRS09] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, MIT Press, 2009

[Dia69] R. B. Dial, Algorithm 360: shortest-path forest with topological ordering, Communications of the ACM Volume 12 Issue 11, Nov. 1969

[dkuCH] http://user.dankook.ac.kr/~bitl/html/dkuCH.html

[DP06] D. Douglas and T. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Cartographica: The International Journal for Geographic Information and Geovisualization, University of Toronto Press

[ELKI] http://elki.dbs.ifi.lmu.de/

[EncPo] https://developers.google.com/maps/documentation/utilities/polylinealgorithm

[GBCI11] J. Gamper, M. Boehlen, W. Cometti, M. Innerebner, Defining isochrones in multimodal spatial networks, CIKM '11, 2011

[Isok] http://www.isokron.com/nos-produits/isokron-maps

[JSON] http://www.json.org/

[Mapn] http://www.mapnificent.net/

[MCCH] http://www.algorithmist.com/index.php/Monotone_Chain_Convex_Hull

[MG10] S. Marciuska and J. Gamper, Determining objects within isochrones in spatial network databases, ADBIS'10, 2010

[MS08] K. Mehlhorn & P. Sanders, Algorithms and Data Structures - The Basic Toolbox, Springer, 2008

[OSM] http://www.openstreetmap.org/

[OSMXML] http://wiki.openstreetmap.org/wiki/OSM_XML

[PGIS] http://postgis.refractions.net/

[PO12] Jin-Seo Park and Se-Jong Oh, A New Concave Hull Algorithm and Concaveness Measure for n-dimensional Datasets, Journal of Information Science and Engineering, Vol. 28 No. 3, pp. 587-600, May 2012

[PSQL] http://www.postgresql.org/

[RFC1952] RFC 1952 – GZIP file format specification version 4.3, http://tools.ietf.org/html/rfc1952

[TOMC] http://tomcat.apache.org/