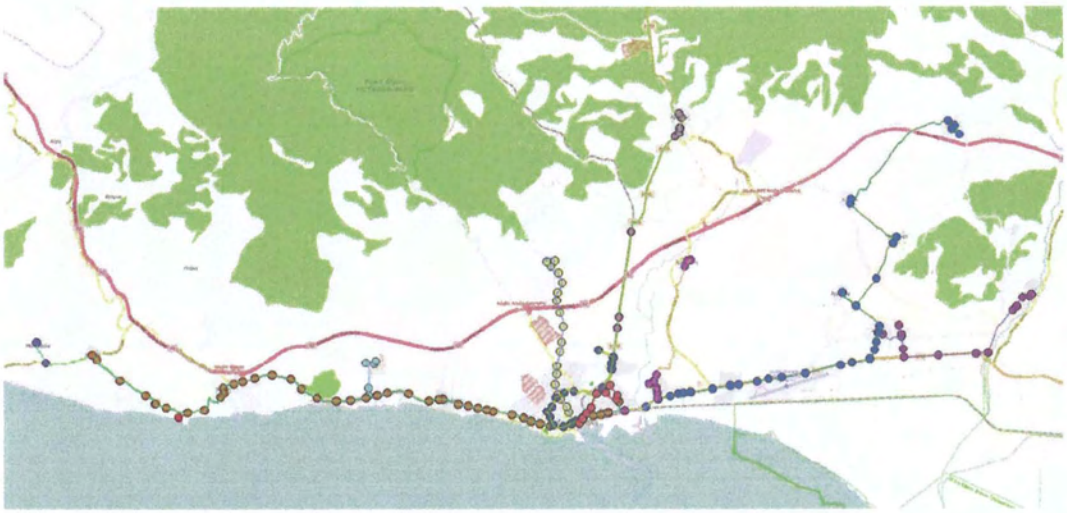
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Το Πρόβλημα του Σχεδιασμού Ταξιδιού σε Δίκτυο Πολυτροπικών Μετακινήσεων και η Επίλυσή του»



Επίθετο: Δημούτσης

Όνομα: Ευάγγελος

Α.Μ. 891

Επιβλέπων Καθηγητής: Σαχαρίδης Γιώργος

ΒΟΛΟΣ 2016

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Δημήτριος Παντελής- Επ. Καθηγητής, Στοχαστικά Πρότυπα Επιχειρησιακής Έρευνας στη Βιομηχανική Διοίκηση

Γεώργιος Λυμπερόπουλος- Καθηγητής, Στοχαστικές Μέθοδοι στην Διοίκηση Παραγωγής

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Γεώργιος Σαχαρίδης- Επ. Καθηγητής, Μέθοδοι Επιχειρησιακής Έρευνας στη Βιομηχανική Διοίκηση

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να αναφέρω και να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν στην διεκπεραίωση όχι μόνο της διπλωματικής μου εργασίας, αλλά και των ακαδημαϊκών μου σπουδών.

Αρχικά τον Επιβλέπων καθηγητή μου κ. Σαχαρίδη Γεώργιο, για την καθοδήγηση που μου προσέφερε και την εμπιστοσύνη που έδειξε στο πρόσωπό μου. Επίσης τον συμφοιτητή μου Δημήτρη Ριζόπουλο που μου προσέφερε πολύτιμες γνώσεις στο αντικείμενο του προγραμματισμού.

Ένα μεγάλο ευχαριστώ ιδιαιτέρως στον συνάδελφο μου Απόστολο Γκούντα, καθώς και τους δικούς μου ανθρώπους Ανδρέα Κ., Γιώργο Π., Ζαφείρη Υ. και Ευαγγελία Α. που ήταν δίπλα μου σε όλη αυτή την πορεία. Επίσης στον Αλέξανδρο και στον Γιώργο Δημούτση που με βοήθησαν ηθικά και πρακτικά σε όλη την διάρκεια της πραγματοποίησης της διπλωματικής μου εργασίας και όχι μόνο.

Τέλος, οφείλω να ευχαριστήσω τους γονείς μου Παντελή και Ράνια, τον αδερφό μου Θανάση για την κατανόηση και την απεριόριστη στήριξή τους όλα αυτά τα χρόνια, στους οποίους και αφιερώνω την εργασία αυτή.

"Σαν βγεις στον πηγαιμό για την Ιθάκη, να εύχεσαι να 'ναι
μακρύς ο δρόμος, γεμάτος περιπέτειες, γεμάτος γνώσεις."

Κ.Π. Καβάφης, 1863-1933

Περίληψη

Η διπλωματική εργασία η οποία εκπονήθηκε έχει θέμα «Το Πρόβλημα του Σχεδιασμού Ταξιδιού σε Δίκτυο Πολυτροπικών Μετακινήσεων και η Επίλυσή του».

Αρχικά και στο 1ο κεφάλαιο, ασχοληθήκαμε με την γενικότερη έννοια του όρου, με τον τρόπο λειτουργίας των σχεδιαστών ταξιδιού και με την σημασία που αποκτά με το πέρασμα του χρόνου στις υπηρεσίες μετακινήσεων. Ένας σχεδιαστής ταξιδιού παρέχει ένα σύνολο υπηρεσιών με σκοπό την μετακίνηση. Οι υπηρεσίες μπορούν να σχετίζονται με απλές πληροφορίες (π.χ. δρομολόγια), αλλά και με ποιο σύνθετα ερωτήματα (π.χ. ποιος είναι ο βέλτιστος τρόπος να μετακινηθώ από ένα σημείο Α, σε ένα σημείο Β).

Οι πληροφορίες που παρέχουν οι σχεδιαστές ταξιδιού βασίζονται και σχετίζονται άμεσα με ένα δίκτυο μεταφορών. Στο 2ο κεφάλαιο λοιπόν γίνεται μία ανάλυση της έννοιας των δικτύων, καθώς και κάποιων βασικών στοιχείων που τα χαρακτηρίζουν. Επίσης γίνεται η παρουσίαση των δικτύων που απασχόλησαν την παρούσα διπλωματική εργασία, ο τρόπος που αναπτύχθηκαν σαν πρωτογενή δεδομένα, η αρχική τους ψηφιοποίησή και η μετατροπή τους σε μορφή αρχείων που θα είναι πιο εύκολο να χρησιμοποιηθούν για ένα λογισμικό παρόμοιο με αυτό ενός σχεδιαστή ταξιδιού.

Όπως αναφέρθηκε στο 1ο κεφάλαιο ο σχεδιαστής ταξιδιού βασίζεται στην λειτουργία και την επίλυση γράφων. Στο 3ο κεφάλαιο αναλύεται η γενική έννοια των γράφων και ο τρόπος παραγωγής γράφων. Ο στόχος της παρούσας διπλωματικής εργασίας είναι η προσέγγιση του χτισίματος των γράφων με δύο διαφορετικούς τύπους γραφημάτων. Ο τρόπος λειτουργίας και τα χαρακτηριστικά των δύο αυτών τύπων (time-expanded graph builder και time-dependent graph builder) συμπεριλαμβάνεται στο 3ο κεφάλαιο.

Στο 4ο κεφάλαιο γίνεται με μεγαλύτερη εξειδίκευση η παρουσίαση της λειτουργίας του σχεδιαστή ταξιδιού. Ουσιαστικά πρόκειται για έναν αλγόριθμο, ο οποίος δίνει επίλυση στο πρόβλημα του σχεδιασμού ταξιδιού (journey planning), με τον τρόπο της εξεύρεσης του συντομότερου μονοπατιού (shortest path).Ο αλγόριθμος αντλεί τα δεδομένα σε μορφή γράφων. Επίσης στο συγκεκριμένο κεφάλαιο παρουσιάζονται αναλυτικά οι δύο κώδικες οι οποίοι σχεδιάστηκαν στα πλαίσια της διπλωματικής εργασίας, και αφορούν τους δύο τρόπους χτισίματος γράφων που αναφέρθηκαν στο 3ο κεφάλαιο.

Τέλος στα πλαίσια του 5ου κεφαλαίου "τρέξαμε" τους δύο κώδικες, οι οποίοι έβγαλαν αποτελέσματα για τον κάθε τύπο graph builder. Τα αποτελέσματα χρησιμοποιήθηκαν από τον ίδιο αλγόριθμο και οι τελικοί χρόνοι υπολογισμού του συντομότερου μονοπατιού κρίθηκαν και σχολιάστηκαν στα συμπεράσματα της διπλωματικής μας εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

Πίνακας περιεχομένων

ΠΕΡΙΕΧΟΜΕΝΑ ΕΙΚΟΝΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ ΠΙΝΑΚΩΝ

Κεφάλαιο 1ο: Σχεδιαστές Ταξιδιού και Συνδυασμένη Μεταφορά

1.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει μία παρουσίαση του θέματος που θα απασχολήσει την παρούσα διπλωματική εργασία, των βασικών αρχών και της κύριας λειτουργίας ενός σχεδιαστή ταξιδιού (journey planner), την εξέλιξή του και τα οφέλη που μπορεί να προσφέρει στην καθημερινότητα των ανθρώπων. Επίσης θα αναδειχθεί η σημασία της διατροπικότητας των μετακινήσεων (intermodal transport) και σε συνέπεια, το αντίκτυπο που έχουν οι συνδυασμένες μεταφορές (co-modal transport) στην πορεία και εξέλιξη των επιβατικών μετακινήσεων.

1.2 Σχεδιαστές ταξιδιών (journey planners)

Τελευταία, και με την αλματώδη εξέλιξη της τεχνολογίας, η ανάπτυξη των υπολογιστικών συσκευών και συστημάτων που σχετίζονται με αυτές, σε συνδυασμό με τις τεχνολογίες εντοπισμού θέσης, έχουν ωθήσει στην δημιουργία και εξέλιξη ολοένα και περισσότερων συστημάτων πλοήγησης και συστημάτων προσδιορισμών θέσης, καθώς και την συνεχή προσαρμοστικότητα αυτών στην καθημερινότητα των ανθρώπων. Κατά κανόνα, τα συστήματα αυτά αφορούν το οδικό δίκτυο και τις μετακινήσεις που γίνονται εντός οδικού δικτύου. Όμως όπως είναι φυσιολογικό και επακόλουθο έχει υπάρξει εξέλιξη και σε συστήματα που αφορούν την μετακίνηση πεζή, με ποδήλατο, ή ακόμη και με τις σιδηροδρομικές και αεροπορικές μεταφορές.

Συστήματα του υπολογιστή που έχουν αναπτυχθεί και μπορούν να παρέχουν σε ένα ταξιδιώτη μία διαδρομή για ένα ταξίδι μεταφορών είναι τα συστήματα σχεδιαστών ταξιδιού (journey planners). Το ταξίδι μπορεί να είναι μία μονοτροπική μεταφορά, ή διατροπική. Το σύστημα μπορεί να παρέχει χρονοδιαγράμματα δρομολόγησης και άλλες ταξιδιωτικές πληροφορίες. Ένα απλό ταξίδι, όπως αναφέρθηκε παραπάνω, μπορεί να χρησιμοποιήσει μία σειρά από δρομολόγια μέσων μεταφοράς, πράγμα που σημαίνει ότι το σύστημα είναι ενημερωμένο για τις υπηρεσίες δημόσιων μεταφορών και σχετικά με τα δίκτυα μεταφορών.

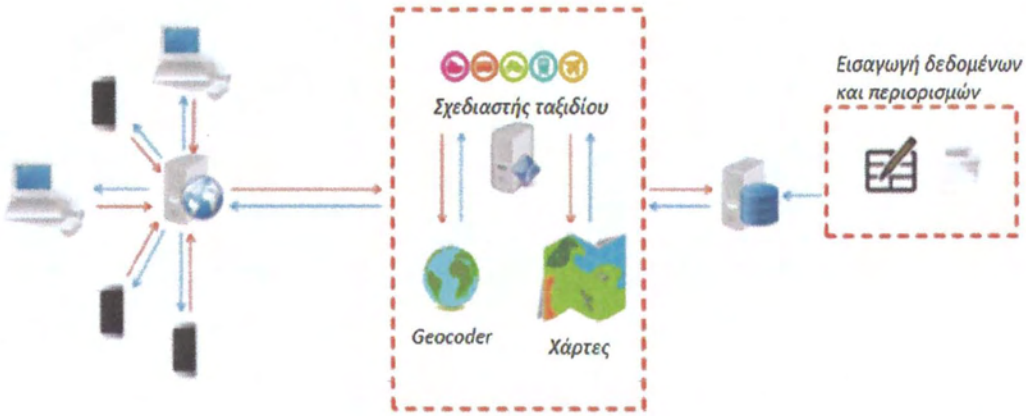
Ένας σχεδιαστής ταξιδιού παρέχει ένα σύνολο υπηρεσιών με σκοπό την μεταφορά απαντώντας στο κατ' ελάχιστο ερώτημα «Πώς μπορώ να μετακινηθώ από την θέση Α στην θέση Β», εντός χρονικών δεδομένων άφιξης και αναχώρησης. Οι θέσεις Α και Β, καθώς και τα χρονικά όρια άφιξης/αναχώρησης είναι τα δεδομένα που παρέχει ο χρήστης στον σχεδιαστή ταξιδιού.

Το ερώτημα θα μπορούσε να περιέχει πιο πολύπλοκα επιπρόσθετα κριτήρια όπως είναι «ο γρηγορότερος τρόπος», «ο φθηνότερος τρόπος», «η μεταφορά χωρίς αλλαγές μέσων», κ.λπ. Ο

χρήστης μπορεί να δώσει τις δικές του λύσεις ταξιδιού, λαμβάνοντας υπ' όψη διάφορες πληροφορίες, ή την λύση μπορεί να του την δώσει ο σχεδιαστής ταξιδιού.

Ο κάθε άνθρωπος έχει τις δικές του προτιμήσεις για τον τρόπο μετακίνησής του, οι οποίες επηρεάζονται και προσαρμόζονται καθημερινά από διάφορους παράγοντες. Επίσης η μετακίνηση από έναν προορισμό σε άλλο, εντός οδικού δικτύου, συχνά περιλαμβάνει επιλογές μεταξύ διαφορετικών διαδρομών για το ίδιο ζεύγος αφετηρίας-προορισμού. Ο σχεδιαστής ταξιδιού έχει την δυνατότητα γρήγορης εύρεσης βέλτιστων διαδρομών λαμβάνοντας υπ' όψη τα κριτήρια που θέτει ο ίδιος ο χρήστης και που κατά την γνώμη του είναι τα πιο σημαντικά (π.χ. γρηγορότερη διαδρομή σε χρόνο).

Ένας σχεδιαστής ταξιδιού παρέχει την δυνατότητα να δημιουργήσει ο χρήστης και να σώσει στην συσκευή που τον χρησιμοποιεί, ένα ταξίδι με πολλαπλούς προορισμούς, έτσι ώστε να μπορεί ο χρήστης να χρησιμοποιήσει την διαδρομή για πιθανό μελλοντικό ταξίδι. Επίσης ένας σχεδιαστής ταξιδιού δίνει τις απαραίτητες πληροφορίες και επιτρέπει στο χρήστη να προσαρμόσει το ταξίδι του σύμφωνα με τα δικά του δεδομένα. Για παράδειγμα να βελτιώσει την σειρά των προορισμών του, να ορίσει την διάρκεια στάσης στον κάθε προορισμό, καθώς και να θέσει την ώρα αναχώρησης από έναν προορισμό που επιτρέπονται βάσει της ώρας άφιξης στον συγκεκριμένο προορισμό.



Εικόνα 1: Αρχιτεκτονική πλατφόρμας σχεδιαστή ταξιδιού [πηγή: http://www.giftsmartways.com]

1.3 Η εξέλιξη των σχεδιαστών ταξιδιού (journey planners)

Όσο περνάει ο καιρός, αυξάνεται ο ρυθμός της εξέλιξης της τεχνολογίας και σε αποτέλεσμα αυτού του φαινομένου επηρεάζονται πολλοί τομείς της καθημερινότητας. Η απήχηση αυτής

της εξέλιξης θα μπορούσαμε να πούμε ότι έχει θετικές επιδράσεις στην πλειοψηφία των τομέων που επηρεάζονται. Οι τηλεπικοινωνίες που σχετίζονται με τεχνολογίες εντοπισμού θέσης, τεχνολογίες ενημέρωσης και τεχνολογίες άμεσης παροχής πληροφοριών είναι ο τομέας με τους ίσως μεγαλύτερους ρυθμούς εξέλιξης. Η εξέλιξη και η βελτίωση των τεχνολογιών αυτών επηρεάζουν και την ποιότητα των σχεδιαστών ταξιδιών (journey planners) με αποτέλεσμα κάθε έκδοση να είναι πιο προσιτή και πιο χρήσιμη για το ευρύ κοινό. Ο κόσμος εξοικειώνεται όλο και περισσότερο με την τεχνολογία και κατ' επέκταση με συστήματα όπως είναι οι σχεδιαστές ταξιδιού.

Το παγκόσμιο σύστημα εντοπισμού θέσης (Global Positioning System) με την πάροδο του χρόνου και σε συνδυασμό με την βελτίωση της ακρίβειας του συστήματος, καθώς και με την μείωση του κόστους απόκτησής του, επεκτάθηκε και σε εφαρμογές άλλων τομέων, όπως είναι οι σχεδιαστές ταξιδιών (journey planners). Ένα μεγάλο κοινό λοιπόν έχει αρχίσει να χρησιμοποιεί στις μετακινήσεις του GPS (κυρίως για μετακινήσεις με το αυτοκίνητο), οπότε και έχει κάνει την χρήση και του σχεδιαστή ταξιδιού πιο προσιτή και πιο οικεία.

Εφαρμογές των σχεδιαστών ταξιδιού είχαν αρχίσει να εμφανίζονται και πριν την εξάπλωση του GPS, κυρίως σε συνδυασμό με την χρήση Ηλεκτρονικών Υπολογιστών και οδήγησαν στην δημιουργία αντίστοιχων αλγορίθμων (π.χ. MS Auto Route), όμως η απήχηση που είχαν στο κοινό δεν ήταν ιδιαίτερα ευρεία. Από την μία οι γνώσεις για την χρησιμοποίηση των Η/Υ δεν είχαν γίνει τόσο κοινές για την πλειοψηφία της κοινωνίας και από την άλλη, τα αποτελέσματα που παρείχαν στους χρήστες τα συστήματα δεν ήταν τόσο ελκυστικά και αποτελεσματικά, αλλά ούτε και τόσο αξιόπιστα.

Η χρήση των σχεδιαστών ταξιδιού για το κοινό αυξάνεται ολοένα και περισσότερο και είναι λογικό. Από την στιγμή που κάποιος χρήστης χρησιμοποιεί έναν σχεδιαστή ταξιδιού η διαδρομή βελτιώνεται ανάλογα με τα φίλτρα και τους περιορισμούς που έχει θέσει ο χρήστης για το ταξίδι του. Μπορεί το ταξίδι να γίνει πιο γρήγορο αν χρησιμοποιηθούν οι πιο άμεσοι οδοί, μία διαδρομή να γίνει πιο όμορφη αν χρησιμοποιηθούν δρόμοι που παράπλευρα αυτών βρίσκονται αξιοθέατα ή πεδία με φυσική ομορφιά, ή πιο οικονομικό αν χρησιμοποιηθεί συνδυασμός δρόμων που δεν περιέχει διόδια.

Επίσης όμως οι σχεδιαστές ταξιδιού έχουν επηρεάσει θετικά εκτός από τους χρήστες και συνολικότερα το σύστημα μεταφορών. Ως σύστημα μεταφορών νοείται το σύνολο των μέσων μαζικής μεταφοράς και γενικότερα κάθε εμπλεκόμενο μέσο που σχετίζεται με την μετακίνηση ενός επιβάτη από έναν προορισμό σε έναν άλλον. Η χρήση των σχεδιαστών ταξιδιού, έχει φέρει πιο κοντά το κοινό με τα μέσα μαζικής μεταφοράς, διότι η ενημέρωση και οι υπηρεσίες που παρέχουν τα δεύτερα έχουν γίνει πιο ευέλικτες, πιο αξιόπιστες και πιο προσιτές προς τον

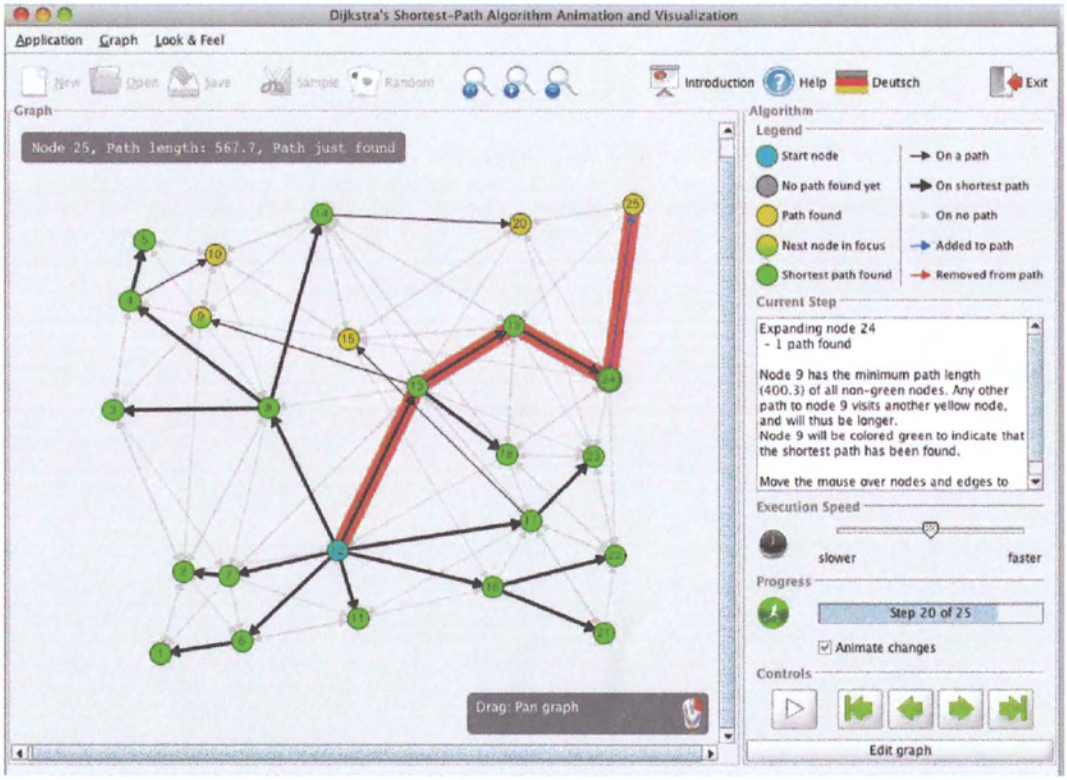
χρήστη. Η χρήση έχει γίνει πιο άνετη και για τις δύο πλευρές και έχει δοθεί χώρος για την μεγαλύτερη αύξηση του φάσματος των υπηρεσιών που μπορούν να παρασχεθούν.

Τα μέσα μαζικής μεταφοράς εξελίσσονται με το πέρασμα του χρόνου, αυξάνοντας τις υπηρεσίες που μεμονωμένα παρέχουν στους χρήστες. Παρόλα αυτά η έλλειψη επικοινωνίας και συνεργασίας μεταξύ των μέσων καθιστά δυσκολότερη την αποδοτικότερη αξιοποίηση αυτών από τους χρήστες. Η εξάπλωση όμως των σχεδιαστών ταξιδιού και η γοργή εξέλιξη αυτών συνεισφέρει σημαντικά στην ολοένα αυξανόμενη χρήση των μέσων μαζικής μεταφοράς. Βοηθά τον χρήστη να σχεδιάσει το βέλτιστο, αξιόπιστο και ευνοϊκότερο για τα δικά του δεδομένα, ταξίδι με την χρήση ενός μέσου, ή συνδυασμό μέσων.

1.4 Λειτουργία ενός σχεδιαστή ταξιδιού

Τα συστήματα που αναλύθηκαν παραπάνω δέχονται κάποια ελάχιστα δεδομένα από τον χρήστη (όπως είναι το σημείο αναχώρησης και το σημείο προορισμού) και έπειτα λειτουργούν και αποδίδουν τα αποτελέσματα. Η λειτουργία του σχεδιαστή ταξιδιού (journey planner) βασίζεται στην επίλυση ενός γράφου, στον οποίο κόμβοι μπορεί να είναι π.χ. οι διασταυρώσεις, τα σημεία ενδιαφέροντος, οι σταθμοί των μέσων μετακίνησης (σταθμοί ΚΤΕΛ, αεροδρόμια, κ.λπ.) ενώ σύνδεσμοι μπορεί να είναι τα δίκτυα πάνω στα οποία κινούνται τα μέσα μετακίνησης (δρόμοι, σιδηρόδρομοι, ακτοπλοϊκές γραμμές, κ.λπ.).

Ο κάθε σύνδεσμος έχει την ανάλογη βαρύτητα, η οποία ορίζεται από το σύστημα ή από τον χρήστη. Η βαρύτητα μπορεί να ορίζεται ποσοτικά (π.χ. πόση είναι η απόσταση της διαδρομής, πόση ώρα θα διαρκέσει, πόσο κοστίζει το εισιτήριο, κ.λπ.), ή ποιοτικά (π.χ. πόσο ασφαλές είναι το μέσο μετακίνησης, αν διέρχεται η διαδρομή από διόδια, κ.λπ.). Οι σχεδιαστές ταξιδιών κάνουν χρήση διάφορων αλγόριθμων επίλυσης (π.χ. Dijkstra, A* [A-Star], κ.α.) έτσι ώστε να υπολογίζουν την βέλτιστη λύσει, βάσει των περιορισμών και των δεδομένων που ορίζει ο χρήστης στο σύστημα. Ένα παράδειγμα ζητούμενης αναζήτησης από τον σχεδιαστή ταξιδιού είναι το εξής: εύρεση ταξιδιού με χρήση Υπεραστικού Κτελ, με περιορισμό του κόστους ταξιδιού σε συνδυασμό με την ελάχιστη δυνατή αλλαγή λεωφορείων.



Εικόνα 2:Επίλυση ενός γράφου με τον αλγόριθμο του Dijkstra, με χρήση λογισμικού που οπτικοποιεί την διαδικασία επίλυσης κατά βήμα. [πηγή: http://www.uweschmidt.org.]

Πέρα από το σύστημα υπολογισμού της βέλτιστης λύσης μέσω του σχετικού γράφου, ένας σχεδιαστής ταξιδιού (journey planner) αποτελείται και από κάποιο περιβάλλον, το οποίο δίνει την δυνατότητα στον χρήστη να εισάγει τα δεδομένα και τους περιορισμούς που θέλει, αλλά και να δεχτεί τα δεδομένα εξόδου. Τέτοια περιβάλλοντα είναι ο υπολογιστής ή το κινητό τηλέφωνο κ.α. Επίσης ο σχεδιαστής ταξιδιού (journey planner) εξελίσσεται και συνδυάζεται με την χρήση διάφορων geolocators για την εύρεση διευθύνσεων και σημείων ενδιαφέροντος, ενώ μπορεί να παρέχει υπηρεσίες που κάνουν το ταξίδι πιο εύκολο και προσιτό, όπως είναι η έκδοση εισιτηρίου και η παροχή πληροφόρησης για όλα τα μεταφορικά μέσα που αφορούν ένα δρομολόγιο σε πραγματικό χρόνο.

Η εξέλιξη των σχεδιαστών ταξιδιών είναι πλήρως εξαρτώμενη με την εξέλιξη της τεχνολογίας, παλαιότερα τον συνδυασμό μεμονωμένων τεχνολογιών, αλλά και την εξοικείωση της τεχνολογίας με την καθημερινότητα του ανθρώπου. Οι σχεδιαστές ταξιδιών ολοένα και ενσωματώνουν νέες δυνατότητες στα προγράμματά τους, είτε οι ίδιοι ενσωματώνονται σε άλλες περισσότερες υπηρεσίες (π.χ. ταξιδιωτικά portals). Η εξέλιξη των σχεδιαστών ταξιδιών γίνεται προς όφελος των χρηστών, όμως έχει αντίκτυπο και σε άλλες παραμέτρους όπως είναι το περιβάλλον και η κοινωνία γενικότερα. Η αποτελεσματικότερη λειτουργία ενός σχεδιαστή

ταξιδιού είναι άρρηκτα συνδεδεμένη με την ποιότητα των δεδομένων που χρησιμοποιεί. Όπου ποιότητα εννοείται η μορφή των δεδομένων, η εγκυρότητα, η πληρότητα, κ.α.

1.5 Οι γράφοι και η σημασία τους

Παραπάνω και στο αντίστοιχο κεφάλαιο που αναλύθηκε η λειτουργία του σχεδιαστή ταξιδιού (journey planner) αναφέρθηκε πως η λειτουργία του συστήματος βασίζεται στην επίλυση ενός γράφου. Τα συστήματα όπως αυτό που σχετίζεται με τους σχεδιαστές ταξιδιού αναπαρίστανται με την χρήση γράφων. Προβλήματα όπως η αναζήτηση της βέλτιστης διαδρομής ανάγονται σε προβλήματα επίλυσης γράφων.

Ο γράφος [2] στον απλούστερο ορισμό του είναι η οπτική αναπαράσταση των σχέσεων που αναπτύσσουν ορισμένες ποσότητες, σχεδιασμένες σε σχέση με ένα σύνολο αξόνων. Ένας άλλος ορισμός που κινείται στο ίδιο εννοιολογικό πλαίσιο της οπτικής αναπαράστασης αναγνωρίζει τον γράφο ως απεικόνιση αποτελούμενη από ένα σύνολο σημείων (κορυφών ή κόμβων) που συνδέονται με γραμμές (ακμές). Στους κατευθυνόμενους ή προσανατολισμένους γράφους οι ακμές απεικονίζονται διανυσματικά.

Σε μία άλλη εκδοχή είναι ένα σύνολο από κόμβους (κορυφές) που ενώνονται μεταξύ τους με ακμές και ορίζεται από τον τρόπο με τον οποίο συνδέονται οι κορυφές (κόμβοι). Αν οι ακμές προσανατολίζονται οριζόμενες από διατεταγμένα ζεύγη κόμβων, τότε ο γράφος αποκαλείται κατευθυνόμενος. Αν οι ακμές δεν προσανατολίζονται, οριζόμενες απλώς από διμελή σύνολα και όχι διατεταγμένα ζεύγη, τότε αποκαλείται μη κατευθυνόμενος. Επιπλέον στοιχεία για τον ορισμό ενός γράφου είναι η σύνδεση των ακμών του με κάποια αξία, οπότε αποκαλείται σταθμισμένος. Με τη σειρά του πλήρης αποκαλείται ο γράφος που περιέχει ακμές για κάθε ζεύγος κόμβων, αραιός εκείνος που περιέχει λίγες ακμές ή αντίστροφα πυκνός.

Η χρησιμοποίηση των γράφων, ως δομή, γίνεται κυρίως με σκοπό να μοντελοποιηθούν σύνολα περίπλοκων δεδομένων. Εδώ και πολλά χρόνια έχουν αναπτυχθεί πάρα πολλοί αλγόριθμοι που αποτέλεσμα έχουν να δίνουν απαντήσεις σε ερωτήσεις για δεδομένα οργανωμένα σε γράφους. Μία σύνηθες ερώτηση που απαντάται είναι «Ποιος είναι ο πιο βέλτιστος δρόμος μεταξύ δύο κόμβων».

Τα τελευταία χρόνια και με την σημαντική εξέλιξη εφαρμογών Σημασιολογικού Ιστού (όπως είναι τα κοινωνικά δίκτυα, τα δίκτυα από blogs, κ.α.), έχουν προκύψει νέες κατηγορίες ερωτήσεων για δεδομένα οργανωμένα σε γράφους.

Επιπλέον, το μέγεθος των γράφων αυτών των εφαρμογών κρίνεται αρκετά μεγάλο. Στην πλειοψηφία τους γνωστοί αλγόριθμοι γράφων που έχουν αναπτυχθεί μπορούν να εφαρμοστούν αποδοτικά, είτε σε μορφή γράφων μικρού μεγέθους (δηλαδή σε γράφους που είναι δυνατό να αποθηκευτούν στην κύρια μνήμη) είτε σε μορφή μη-μεταβαλλόμενων γράφων (δηλαδή σε

γράφους στους οποίους ο αριθμός των κόμβων, των ακμών αλλά και των βαρών των ακμών παραμένει σταθερός).

1.6 Λειτουργία αλγορίθμων επίλυσης

Παραπάνω και στο σημείο που αναλύθηκε ο τρόπος λειτουργίας των σχεδιαστών ταξιδιών, αναφέρθηκε επίσης ότι η λειτουργία τους βασίζεται στην χρήση διάφορων αλγόριθμων επίλυσης.

Ο γενικότερος όρος αλγόριθμος χρησιμοποιείται για να δηλώσει μεθόδους που εφαρμόζονται για την επίλυση προβλημάτων.

Κάθε αλγόριθμος απαραίτητα ικανοποιεί τα επόμενα κριτήρια [3]:

- Είσοδος (input). Καμία, μία ή και περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια των συναρτήσεων παραγωγής τυχαίων αριθμών, ή με την βοήθεια άλλων απλών επιλογών,
- Έξοδος (output). Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν άλλο αλγόριθμο,
- Καθοριστικότητα (definiteness). Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής του. Λόγου χάριν, μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέτης λαμβάνει τη μηδενική τιμή,
- Περατότητα (finiteness). Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά υπολογιστική διαδικασία (computational procedure),
- Αποτελεσματικότητα (effectiveness). Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μία εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη.

Οι αλγόριθμοι μπορούν να υλοποιηθούν από προγράμματα ηλεκτρονικών υπολογιστών, παρότι συχνά σε περιορισμένες μορφές.

Κατά την διάρκεια σύνταξης και σχεδιασμού ενός αλγόριθμου "δίνονται" κάποιες εντολές από τον χρήστη προκειμένου να εκτελεστούν από το πρόγραμμα για την υλοποίηση του αλγόριθμου.

Κάποιες βασικές εντολές που χρησιμοποιούνται είναι οι εξής: [2]

Δομή ακολουθίας. Η δόμηση των διαδικασιών σε τέτοια μορφή, έτσι ώστε οι διαδικασίες να εκτελούνται με την σειρά από τον υπολογιστή.

Δομή επιλογής. Η προγραμματιστική δομή που περικλείει τον έλεγχο μιας συνθήκης και μία ή δύο ομάδες εντολών. Από τις ομάδες των εντολών εκτελείται πρώτη, αν ισχύει η συνθήκη, ή αν υπάρχει και δεύτερη ομάδα εντολών εκτελείται η δεύτερη αν δεν ισχύει η συνθήκη. Με τον όρο συνθήκη εννοούμε δύο όρους ίδιου τύπου και ανάμεσα τους ένας τελεστής σύγκρισης. Με τον όρο τελεστή σύγκρισης εννοούμε ένα από τα σύμβολα <, >, <>, <=, >=, =. Το αποτέλεσμα της σύγκρισης των όρων (νοείται εφόσον οι όροι έχουν κάποια τιμή, δηλαδή δεν περιέχουν την τιμή null) είναι η αλγεβρική τιμή Αληθής (True) ή Ψευδής (False). Οι όροι μπορεί να είναι μεταβλητές ή σταθερές.

Δομή επανάληψης. Η προγραμματιστική δομή που περικλείει τον συνεχή έλεγχο μίας συνθήκης και μία ομάδα εντολών. Οι εντολές εκτελούνται ανάλογα με την δομή της επανάληψης. Υπάρχουν τριών ειδών επαναλήψεις.

- Επανέλαβε εφόσον. Σε αυτή την δομή επανάληψης ελέγχεται πρώτα η συνθήκη και εφόσον ισχύει (δίνει τιμή αληθή) εκτελείται η ομάδα εντολών,
- Επανέλαβε μέχρι. Σε αυτή την δομή επανάληψης εκτελείται η ομάδα εντολών, στην συνέχεια ελέγχεται αν ισχύει η συνθήκη και εφόσον ΔΕΝ ισχύει (δίνει τιμή ψευδής) εκτελείται ξανά η ομάδα εντολών,
- Για Ν φορές επανέλαβε. Σε αυτή την δομή επανάληψης εκτελείται η ομάδα εντολών Ν φορές όπου Ν είναι αριθμός θετικός ακέραιος.

Υπάρχουν διάφοροι τρόποι για την αναπαράσταση ενός αλγορίθμου:

- με ελεύθερο κείμενο (free text), που αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Έτσι μπορεί εύκολα να οδηγήσει σε μη εκτελέσιμη παρουσίαση παραβιάζοντας το τελευταίο χαρακτηριστικό των αλγορίθμων, δηλαδή την αποτελεσματικότητα,
- με διαγραμματικές τεχνικές (diagramming techniques), που συνιστούν ένα γραφικό τρόπο παρουσίασης αλγορίθμου. Η πιο γνωστή από αυτές είναι το διάγραμμα ροής (flowchart). Ωστόσο η χρήση διαγραμμάτων ροής για την παρουσίαση αλγορίθμων δεν αποτελεί την καλύτερη λύση γι' αυτό και εμφανίζονται όλο και σπανιότερα στη βιβλιογραφία και πράξη,
- με φυσική γλώσσα (natural language) κατά βήματα. Στην περίπτωση αυτή χρειάζεται προσοχή, γιατί μπορεί να παραβιασθεί το τρίτο βασικό χαρακτηριστικό ενός αλγορίθμου, όπως προσδιορίστηκε προηγουμένως δηλαδή το κριτήριο του καθορισμού,

- με κωδικοποίηση (coding), δηλαδή με ένα πρόγραμμα που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

1.7 Διατροπικές μετακινήσεις (intermodal transport) και πολυτροπικές μετακινήσεις (multimodal transport)

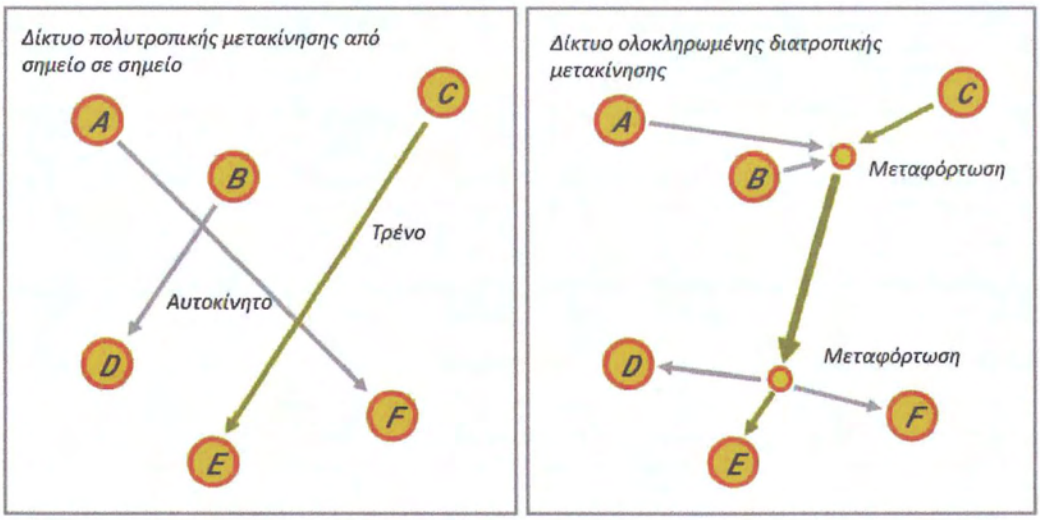
Σε προηγούμενο υποκεφάλαιο έγινε αναφορά στον όρο της διατροπικότητας. Η λειτουργία του σχεδιαστή ταξιδιών (journey planner) είναι συνδεδεμένη με την διατροπικότητα στις μετακινήσεις. Θα μπορούσαμε να πούμε ότι για μία μονοτροπική μετακίνηση δεν θα υπήρχε ουσιαστικό νόημα χρησιμοποίησης ενός σχεδιαστή ταξιδιού, διότι θα υπήρχε μόνο μία επιλογή σαν αποτέλεσμα ανεξαρτήτως δεδομένων και περιορισμών.

Οι όροι «πολυτροπικότητα» και «διατροπικότητα» [4] χρησιμοποιούνται για να περιγράψουν το συνδυασμό των διαφόρων μέσων μεταφοράς. Ο όρος «πολυτροπικότητα» αναφέρεται στη χρήση διαφορετικών μέσων μετακίνησης για διαφορετικές μεταβάσεις ενώ η «διατροπικότητα» αναφέρεται στην αδιάλειπτη χρήση αρκετών διαφορετικών μέσων μεταφοράς για μια μετάβαση (όπως είναι η κλασσική περίπτωση του park & ride – στις εγκαταστάσεις στάθμευσης για μετεπιβίβαση).

Ο όρος "διατροπικός" έχει χρησιμοποιηθεί σε πολλές εφαρμογές που περιλαμβάνουν τις μεταφορές επιβατών. Ο όρος "πολυτροπικός" είναι ένας περιγραφικότερος όρος για αυτή την διαδικασία.

Το διατροπικό δίκτυο μεταφορών είναι ένα σύστημα που χρησιμοποιεί δύο ή περισσότερα μεταφορικά μέσα κατά την διάρκεια μίας μετακίνησης μεταξύ ενός σημείου εκκίνησης (σημείο Α) και ενός σημείου τερματισμού (σημείο Β). Το πολυτροπικό δίκτυο μεταφορών είναι ένα σύνολο τρόπων μεταφορών που προσφέρουν συνδέσεις μεταξύ ενός σημείου εκκίνησης (σημείο Α) και ενός σημείου τερματισμού (σημείο Β).

Ακολουθεί η διαγραμματική επεξήγηση των δύο αναλυθέντων εννοιολογικών προσεγγίσεων:



Εικόνα 3: Πολυτροπική και διατροπική μεταφορά. [πηγή: Dr. Jean-Paul Rodrigue and Dr. Claude Comtois, Intermodal transportation]

Όπως μπορούμε να παρατηρήσουμε ένα δίκτυο μεταφοράς αν είναι διατροπικό, θα είναι και πολυτροπικό, αλλά δεν ισχύει απαραίτητα το αντίθετο.

Η εξέλιξη των μέσων μεταφοράς που χρησιμοποιεί σήμερα ένας επιβάτης για την μετακίνησή του και η άνεση που του παρέχεται στην πλειονότητα αυτών, δεν συνιστούν άμεση ανησυχία για την επιλογή τρόπου μετακίνησης. Η μεγαλύτερη ανησυχία που πιθανώς να επηρεάσει στην επιλογή τρόπου μετακίνησης αφορά τις συνδυασμένες υπηρεσίας, το επίπεδο υπηρεσίας, το κόστος, οι δρομολογήσεις και οι χρόνοι εξυπηρέτησης.

Για τους λόγους αυτούς, τα δίκτυα μεταφορών που υπάρχει η δυνατότητα επισκόπησής τους με διάφορους τρόπους, εξετάζονται υπό δύο διαφορετικές εννοιολογικές προοπτικές οι οποίες και αναλύθηκαν.

1.8 Η συνδυασμένη μεταφορά (co-modal transport)

Οι σχεδιαστές ταξιδιών παλαιότερα παρουσίαζαν μία διαδρομή χωρίς φίλτρα. Όσο τα χρόνια περνάνε και όσο εξελίσσονται οι δυνατότητες των εν λόγω συστημάτων προωθούν και δημιουργούν κίνητρα χρήσης τους για διατροπικές μετακινήσεις. Ένας σχεδιαστής ταξιδιού πλέον έχει φτάσει σε σημείο να προτείνει την καλύτερη επιλογή ανάμεσα σε πολλά δρομολόγια, αλλά και διάφορα μέσα μεταφοράς.

Η συνδυασμένη μετακίνηση είναι ένας όρος που αναφέρεται στην έξυπνη χρήση δύο ή και παραπάνω τρόπων (μέσων) μετακίνησης. Στόχος είναι η χρησιμοποίηση μεμονωμένων ή και συνδυασμό μέσων, με τρόπο τέτοιο για να πάρει ο επιβάτης το μέγιστο όφελος από κάθε ένα μέσο, ώστε το σύνολο του ταξιδιού να είναι το βέλτιστο. Η βελτιστοποίηση αυτή αφορά

οικονομικά δεδομένα, περιβαλλοντικά, κοινωνικά, δεδομένα κινητικότητας ή και συνδυασμό αυτών.

Το 2006, η Ευρωπαϊκή Επιτροπή εισήγαγε μία νέα έννοια: συν-τροπικότητα (co-modality) η οποία αναφέρεται στην χρήση διαφορετικών μέσων, μεμονωμένων ή σε συνδυασμό σε αναζήτηση, με στόχο την βέλτιστη και πιο ουσιαστική χρησιμοποίηση των πόρων. Με αυτή την έννοια, σημαίνει την εύρεση μίας βέλτιστης αναζήτησης, εντός σχετικού πεδίου ορισμού, από διαφορετικά μέσα μετακίνησης (συμπεριλαμβάνοντας ατομικά και δημόσια μέσα), αλλά και συνδυασμό αυτών, με τρόπο τέτοιο ώστε δεδομένα που θέτει ο επιβάτης να λαμβάνονται υπ' όψη. Σε ένα σύστημα πληροφοριών συνδυασμένης μετακίνησης, παρέχονται στον χρήστη πληροφορίες ταξιδιού για διάφορα μέσα μετακίνησης, όπως η καλύτερη διαδρομή ή το καλύτερο χρονοδιάγραμμα, με τον υπολογισμό του βέλτιστου τρόπου όπως αναλύθηκε παραπάνω.

Οι αποκεντρωμένες λειτουργίες των δεδομένων που χαρακτηρίζουν το κατανεμημένο σύστημα πληροφοριών. Γενικά, μόνο τα υποσυστήματα παίρνουν τον πλήρη έλεγχο των δεδομένων τους. Όταν είναι απαραίτητο, οι αναλυτικές πληροφορίες για μία διαδρομή, όπως είναι το πρόγραμμα και η διαθεσιμότητα, διανέμονται μέσω του συστήματος και το υποσύστημα παρέχει τις τοπικές πληροφορίες που κατέχει.

Ένα ολόκληρο ταξίδι μπορεί να περιλαμβάνει διάφορα μέσα μετακίνησης, άρα και διαφορετικά υποσυστήματα. Το κύριο σύστημα έχει την ευθύνη να καλύψει πληροφορίες παγκοσμίως μέσω υποσυστημάτων που επεξεργάζονται τις εκάστοτε τοπικές πληροφορίες. Το κατανεμημένο σύστημα πληροφοριών αποφεύγει την συντήρηση και την ανανέωση των δεδομένων που προέρχονται από κάθε πάροχο πληροφοριών, και για λόγους διακριτικότητας προς τις εταιρίες είναι περισσότερο αποδεκτό.

Ουσιαστικά, το σημείο αναχώρησης, το σημείο προορισμού και ο σχετικός χρόνος είναι απαραίτητα δεδομένα για την έναρξη. Για προχωρημένες επιλογές, περιορισμοί όπως οι προτιμήσεις στα μέσα μετακίνησης και στα χρονοδιαγράμματα επιτρέπουν στο σύστημα την επιστροφή περισσότερο κατάλληλων αποτελεσμάτων σύμφωνα με τις απαιτήσεις που τέθηκαν.

Το σύστημα της συνδυασμένης μετακίνησης είναι ταχέως αναδυόμενο σαν μία από τις πιο δυναμικές τεχνολογίες για την ανάπτυξη μεγάλης - κλίμακας κατανεμημένων συστημάτων το οποίο ασχολείται με την αβεβαιότητα σε ένα δυναμικό περιβάλλον, λόγω της αυτονομίας του, αλλά και της αντιδραστικής και προληπτικής του φύσης. Σε αυτό το δυναμικό πρόβλημα, τα υποσυστήματα είναι ανεξάρτητα το ένα από το άλλο, όμως οι επιμέρους διαδρομές θα αλληλοεπιδρούν μεταξύ τους με σκοπό να σχηματίζουν μία ολοκληρωμένη διαδρομή, κάτω από ένα πρωτόκολλο επικοινωνίας.[5]

Σε μία διατροπική μετακίνηση [6], η αρχική συνολική διαδρομή προορισμού που επιλέγεται, απαρτίζεται από διάφορα τμήματα (ή διαδρομές) και κάθε ένα από αυτά εξασφαλίζεται από έναν τρόπο μετακίνησης (π.χ. με αμάξι, τρένο, κ.λπ). Αντίθετα, στο συνδυασμένο σύστημα μετακίνησης, ένα τμήμα (ή διαδρομή) της αρχικής συνολικής διαδρομής προορισμού που επιλέγεται, μπορεί να εξυπηρετηθεί από παραπάνω από έναν τρόπους μετακίνησης. Σημειώνεται ότι όπου τρόπους μετακίνησης θεωρούνται δημόσιοι και ιδιωτικοί. Οι τρόποι μετακίνησης διαγωνίζονται μεταξύ τους, αφού λάβουν υπ' όψη τις απαιτήσεις του επιβάτη. Το σύστημα συνδυασμένης μετακίνησης θα βρει τον πιο συμβατό και κατάλληλο τρόπο μετακίνησης για κάθε κομμάτι της διαδρομής και θα σχηματίσει την βέλτιστη συνολική διαδρομή προορισμού.

1.9 Βασικά οφέλη από την συνδυασμένη μεταφορά (co-modal transport)

Η συνδυασμένη μεταφορά περιλαμβάνει τη φυσική υποδομή, την μετακίνηση, καθώς και οδηγούς πληροφοριών. Η συνδυασμένη μετακίνηση έχει επέλθει κυρίως λόγω της εξέλιξης της τεχνολογίας. Όπως αναφέρθηκε και παραπάνω ο συνδυαστικός τρόπος μεταφοράς για έναν επιβάτη σημαίνει το να χρησιμοποιήσει τουλάχιστον δύο διαφορετικά μέσα μεταφοράς για την αποπεράτωση ενός ταξιδιού, δηλαδή η μετακίνηση του επιβάτη από ένα σημείο Α σε ένα σημείο Β. Η συνδυασμένη μεταφορά και η διατροπικότητα στις μετακινήσεις, μπορούν να μετατρέψουν ένα σύστημα μεταφορών, όπου το σύνολο των παρεχόμενων υπηρεσιών έχει γίνει παραγωγικότερο, αποδοτικότερο και πιο προσιτό.

Ένα σύστημα συνδυασμένης μετακίνησης σύμφωνα και με τον ορισμό του έχει ως στόχο την αξιοποίηση των πλεονεκτημάτων του κάθε μέσου μεταφοράς, έτσι ώστε να βελτιστοποιηθούν τα αποτελέσματα του κάθε ταξιδιού, όπως θα τα ορίσει σε κάθε περίπτωση ο χρήστης. Με την αξιοποίηση των επιμέρους ιδιαίτερων ποιοτικών χαρακτηριστικών του κάθε μέσου και τον συνδυασμό αυτών σε ένα ενιαίο σύστημα είναι προφανές ότι προκύπτουν βασικά οφέλη.

Κάποια από αυτά είναι:

- το κοινό απέκτησε αμεσότητα και αποδοτικότητα στις μετακινήσεις του,
- οι μετακινήσεις έγιναν πιο αξιόπιστες, πιο ασφαλείς και οι υπηρεσίες πιο ολοκληρωμένες,
- η προσβασιμότητα στα μέσα (ίσα δικαιώματα και ευκαιρίες για όλους),
- η διευκόλυνση στην αλλαγή μεταφορικού μέσου και μείωση του συνολικού χρόνου ταξιδιού για χρήστες που ούτως ή άλλως θα χρησιμοποιούσαν παραπάνω από ένα μέσο μεταφοράς,
- η μείωση της κυκλοφοριακής συμφόρησης μέσω της αναζήτησης ευφυέστερων τρόπων ταξιδιού,

- η αύξηση της ανταγωνιστικότητας μεταξύ των μέσων μεταφορών, με αποτέλεσμα την βελτίωση των παρεχόμενων υπηρεσιών,
- η αύξηση της χρήσης των μέσων μαζικής μεταφοράς, με αντίκτυπο οικονομικό, περιβαλλοντικό και κοινωνικό,
- η καλύτερη εξυπηρέτηση του κοινού σε συνδυασμό με την άμεση παροχή πληροφοριών, την βελτίωση των υπαρχουσών πληροφοριών και την άμεση ανταλλαγή δεδομένων.

1.10 Σύνοψη κεφαλαίου

Οι σχεδιαστές ταξιδιών (journey planners), των οποίων η λειτουργία θα μας απασχολήσει και στα επόμενα κεφάλαια, αποκτούν ουσία στις διατροπικές μετακινήσεις και αποκτούν μεγαλύτερο υπόβαθρο όταν οι μετακινήσεις είναι συνδυασμένες (co-modal transport).Η λειτουργία του συστήματος ενός σχεδιαστή ταξιδιού βασίζεται στην επίλυση ενός γράφου και κάνοντας χρήση διάφορων αλγόριθμων επίλυσης. Η ανάπτυξη των γράφων και των αλγορίθμων επίλυσης είναι καθοριστικής σημασίας για την λειτουργία των σχεδιαστών ταξιδιού και για τον λόγο αυτό μας απασχόλησαν στο κεφάλαιο που προηγήθηκε.

Κεφάλαιο 2ο: Εισαγωγή στα Μεταφορικά Δίκτυα που Μελετήσαμε

2.1 Εισαγωγή

Στο παρόν κεφάλαιο γίνεται η ανάλυση των δικτύων που μελετώνται στην παρούσα διπλωματική εργασία, καθώς και του συνόλου των δεδομένων που συλλέχθηκαν και θα χρησιμοποιηθούν για την προσομοίωση του κώδικα που θα παρουσιαστεί στα επόμενα κεφάλαια. Τα δεδομένα που θα παρατεθούν, ποιοτικά, θα είναι αυτά που θα προσδιορίζουν την βαρύτητα του κάθε κόμβου του δικτύου σε κάθε περίπτωση και ειδικότερα θα είναι αυτά που θα χρησιμοποιούνται για την εξεύρεση της βέλτιστης διαδρομής, σύμφωνα με τις ανάγκες του κάθε χρήστη.

2.2 Η έννοια των δικτύων

Η έννοια του δικτύου είναι άρρηκτα συνδεδεμένη με την έννοια του γράφου, η οποία αναφέρθηκε και σε προηγούμενο κεφάλαιο, στα πλαίσια του ότι στην λειτουργία των γράφων βασίζεται και η λειτουργία των σχεδιαστών ταξιδιού. Η ανάλυση του δικτύου μεταφορών χρησιμοποιείται για τον προσδιορισμό της ροής οχημάτων (ή ατόμων), τυπικά χρησιμοποιώντας την μαθηματική θεωρία γραφημάτων. Μπορεί να συνδυάζονται και διαφορετικά μέσα μεταφοράς, με το μοντέλο να αποτελείται από συνδυασμένες μετακινήσεις.

Ένας κατευθυνόμενος γράφος με βάρη είναι ένας κατευθυνόμενος γράφος στον οποίο έχουμε συσχετίσει την κάθε ακμή ένα βάρος. [2] Ένας κατευθυνόμενος γράφος με βάρη ονομάζεται δίκτυο. Ανάλογα με το είδος της εφαρμογής το βάρος μπορεί να είναι κόστος, χωρητικότητα, απόσταση κ.λπ. Οι γράφοι, όπως αναφέρθηκε προσφέρουν μία χρήσιμη μέθοδο για την διατύπωση και λύση πολλών προβλημάτων, όπως είναι η επιλογή δρομολογίων μέσω ενός χάρτη. Το πρόβλημα αυτό απασχολεί και την παρούσα διπλωματική εργασία.

Τα δίκτυα συνήθως αποτελούνται από δύο τύπους στοιχείων, ένα είναι το σύνολο των σημείων (κόμβοι) και ένα είναι το σύνολο των γραμμών (ακμών). Οι ακμές ή σύνδεσμοι συνδέουν το σύνολο των κόμβων μεταξύ τους και προκύπτει ο μαθηματικός ορισμός του δικτύου. Τα σύνολα αυτά μπορεί να χαρακτηρίζονται από μία τιμή (βάρος), η οποία μπορεί να επηρεάζει τις επιμέρους συνδέσεις των κόμβων και τις επιμέρους διαδρομές. Σε συνέπεια αυτού τα βάρη επηρεάζουν και την συνολική διαδρομή που θα ακολουθήσει ο γράφος. Από τι ορίζεται η τιμή του βάρους καθορίζεται σε κάθε περίπτωση από την φύση του προβλήματος.

Οι κόμβοι αντιπροσωπεύουν το σημείο σύνδεσης των ακμών. Στους κόμβους της παρούσας διπλωματικής εργασίας, όπως και στην επίλυση κάθε γράφου, είναι δυνατόν να περάσεις ακολουθώντας ένα μονοπάτι-διαδρομή (path). Μονοπάτι, θεωρείται το σύνολο των κόμβων

και η σειρά προσέγγισης αυτών εντός γραφήματος που ακολουθείται για την ένωση του αρχικού κόμβου με τον κόμβο προορισμού.

Ο όρος δίκτυο μπορεί να χρησιμοποιηθεί για να περιγράψει μία δομή που μπορεί να είναι είτε φυσική (π.χ. δρόμοι και διασταυρώσεις, τηλεφωνικές γραμμές κ.λπ) είτε εννοιολογική (π.χ. γραμμές πληροφοριών, τηλεοπτικοί σταθμοί κ.λπ). Ένα δίκτυο μεταφορών είναι μία υλοποίηση ενός χωροταξικού δικτύου το οποίο περιγράφει την κίνηση οχημάτων ή την ροή της κίνησης. Παραδείγματα είναι το δίκτυο των δρόμων, των σιδηρόδρομων, κ.λπ.

Εικόνα 4: Δίκτυο Αττικού Μετρό [πηγή: http://el.wikipedia.org/wiki/Μετρό Αθήνας]

Στην παρούσα διπλωματική εργασία συλλέχθηκαν, μελετήθηκαν και χρησιμοποιήθηκαν δίκτυα μετακινήσεων (transportation networks) και πιο συγκεκριμένα δίκτυα Μέσων Μαζικής Μεταφοράς.

Παρακάτω αναλύονται περιγραφικά τα κύρια είδη δικτύων, τα οποία είναι τα εξής:[7]

<u>Δίκτυα αυτοκινητοδρόμων</u>

Τα δίκτυα αυτοκινητοδρόμων παρέχουν μία αναπαράσταση αυτόνομων δεσμών μεταξύ των διασταυρώσεων, μέσω της χρήσης των ακμών και των κόμβων του δικτύου. Οι ακμές συνήθως χαρακτηρίζονται από το μήκος, την χωρητικότητα, τον αριθμό γραμμών, της επιτρεπόμενης

ταχύτητας κ.α. Τα βάρη των ακμών που αναφέρθηκαν δύναται σε πολλές περιπτώσεις να υπολογίζονται μέσω κάποιων συναρτήσεων.

Δίκτυα μετακινήσεων

Τα δίκτυα μετακινήσεων αποτελούνται επίσης από ακμές και κόμβους. Οι κόμβοι συνήθως αποτελούνται από στάσεις λεωφορείου ή τρένου, οι οποίες είναι χωροθετημένες κατά μήκος των διαδρομών της μετακίνησης και χωρικά προσδιορίζονται από γεωγραφικές συντεταγμένες μήκους και πλάτους. Διάφορα γνωρίσματα της κάθε στάσης μπορεί να συνδέονται με έναν κόμβο, για να περιγράφει τα χαρακτηριστικά της κάθε στάσης. Τέτοια χαρακτηριστικά μπορούν να είναι τυπικοί χρόνοι παραμονής και συχνότητα άφιξης οχημάτων στην συγκεκριμένη στάση. Οι ακμές σε αυτού του είδους τα δίκτυα συνδέουν συνεχόμενες στάσεις. Περεταίρω πληροφορίες για κάθε διαδρομή μπορεί να περιλαμβάνουν τιμές ναύλων, χρονοδιαγράμματα και χρόνους μετάβασης.

Άλλα στοιχεία χωρικών δεδομένων που σχετίζονται με τα δίκτυα μετακίνησης περιλαμβάνουν πληροφορίες που εξυπηρετούν τις διατροπικές και τις συνδυασμένες μετακινήσεις.

Άλλα είδη δικτύων

Μη μηχανοκίνητα μέσα, όπως είναι το περπάτημα και το ποδήλατο τις περισσότερες φορές δεν παρουσιάζονται σε μοντέλα πρόβλεψης ταξιδιών. Ως εκ τούτου, πληροφορίες για αυτούς τους τρόπους μετακίνησης δεν συμπεριλαμβάνονται στα δίκτυα που αναλύθηκαν παραπάνω. Ο κύριος λόγος για αυτό τον αποκλεισμό είναι διότι τα περισσότερα μοντέλα αναπτύχθηκαν για να αντιμετωπίσουν ζητήματα που αφορούν τον προορισμό, ή το μέγεθος της διαδρομής και το επίπεδο των συγκεκριμένων δικτύων δεν επηρεάζεται σε ιδιαίτερο βαθμό από αυτά τα ζητήματα. Παρόλα αυτά όμως, μια αναδυόμενη περιοχή ενδιαφέροντος περιλαμβάνει την χρήση ποδηλάτου, της οποίας το δίκτυο μπορεί να περιλαμβάνει ειδικά χαρακτηριστικά που επηρεάζουν την επιλογή της διαδρομής (όπως είναι η κλίση του δρόμου).

Τα εμπορευματικά δίκτυα μπορεί να περιλαμβάνουν τόσο τις μεταφορές και τις εγκαταστάσεις φορτηγών όσο και των σιδηροδρομικές μεταφορές.

Τα σούπερ δίκτυα (super networks) αποτελούν ένα ολοκληρωμένο πολυτροπικό δίκτυο με την πρόθεση να προσφέρει μια συνολική ματιά σε διαδρομές που αφορούν όλα τα διαθέσιμα μεταφορικά μέσα και τα πρότυπα πρόσβασης/εξόδου. Τα στοιχεία εμφανίζονται σε πολλαπλά στρώματα και συνδέονται τεχνητά με συνδέσεις μεταφοράς. Ένα στοιχείο μπορεί να αντιπροσωπεύει μία ποικιλία εξόδων.

2.3 Τοπολογία και τυπολογία των σημείων των δικτύων μετακινήσεων

Η τοπολογία (topology) είναι μία διαδικασία που έχει ως σκοπό την καταγραφή των χωρικών σχέσεων μεταξύ των στοιχείων ενός ψηφιακού χάρτη.

Τα δίκτυα μετακινήσεων, όπως και πολλά δίκτυα γενικά ενσωματώνουν ένα σύνολο από προορισμούς (κόμβους) και μία σειρά από διαδρομές (ακμές) που εκπροσωπούν τις συνδέσεις μεταξύ αυτών των προορισμών. Η διάταξη και η συνδεσιμότητα ενός δικτύου αφορούν την τοπολογία του και έτσι κάθε δίκτυο μεταφοράς έχει συγκεκριμένη τοπολογία. Τα πιο βασικά στοιχεία μιας τέτοιας δομής είναι η γεωμετρία του δικτύου και το επίπεδο συνδεσιμότητας. Τα δίκτυα μεταφορών μπορούν να ταξινομηθούν σε συγκεκριμένες κατηγορίες ανάλογα με μία σειρά από τοπολογικές ιδιότητες που τα περιγράφουν. Είναι δυνατό λοιπόν να καθοριστεί μία βασική τυπολογία των δικτύων μετακινήσεων που θα σχετίζεται με τα γεωγραφικά στοιχεία, καθώς και με την συνδυαστικότητα στους τρόπους μετακίνησης.

Η γεωγραφική απεικόνιση ενός δικτύου ποικίλλει σε συνάφεια ανάλογα με τον τρόπο μετακίνησης που το χαρακτηρίζει. Οι δρόμοι και οι σιδηροδρομικές γραμμές απαρτίζονται από υποδομές που αφορούν οχήματα, ενώ οι θαλάσσιες και αεροπορικές μετακινήσεις παραμένουν πιο αόριστες, λόγω της υψηλότερης χωρικής ευελιξίας τους.

Ως εκ τούτου υπάρχουν δύο τύποι των φυσικών χώρων, οι οποίοι μπορούν να καθορίσουν τα δίκτυα μετακινήσεων και όπου ο κάθε ένας αντιπροσωπεύει μία συγκεκριμένη λειτουργία της χωρικής κατανομής. Οι δύο τύποι είναι οι εξής:[8]

- Σαφώς καθορισμένοι και οριοθετημένοι χώροι. Ο χώρος που καταλαμβάνεται από το δίκτυο μετακινήσεων στις περιπτώσεις αυτές, αυστηρά προορίζεται για αποκλειστική χρήση από το μέσο και μπορεί να εντοπιστεί και να οριοθετηθεί σε ένα χάρτη. Η ιδιοκτησία των χώρων αυτών δύναται να καθοριστεί επίσης με σαφήνεια. Χαρακτηριστικά παραδείγματα αποτελούν οι οδοί, τα κανάλια και οι σιδηροδρομικές γραμμές.
- Αόριστα καθορισμένοι χώροι. Ο χώρος αυτών των δικτύων μπορεί να μοιραστεί με άλλα μέσα και δεν είναι αντικείμενο κάποιας συγκεκριμένης ιδιοκτησίας, αλλά μόνο των δικαιωμάτων διέλευσης. Παραδείγματα αποτελούν οι αεροπορικές γραμμές και οι ακτοπλοϊκές.

Τα δίκτυα γενικότερα παρέχουν ένα επίπεδο υπηρεσιών μετακινήσεων που έχει σχέση, επηρεάζεται και εξαρτάται με το κόστος αυτών. Ένα ιδανικό δίκτυο μετακινήσεων θα ήταν ένα δίκτυο το οποίο θα εξυπηρετούσε όλους τους πιθανούς προορισμούς, αλλά οι υπηρεσίες του θα είχαν πολύ υψηλό πάγιο και λειτουργικό κόστος. Οι υποδομές των μέσων μεταφορών έχουν εγκατασταθεί πάνω σε ασυνεχή δίκτυα, δεδομένου ότι τα περισσότερα δίκτυα δεν

κατασκευάστηκαν ταυτόχρονα, από τον ίδιο φορέα ή με την ίδια τεχνολογία. Ως εκ τούτου, τα λειτουργικά δίκτυα σπάνια εξυπηρετούν όλα τα τμήματα της επικράτειας και όλους τους πιθανούς προορισμούς. Μερικές φορές πρέπει να βρίσκεται ένας συμβιβασμός μεταξύ κάποιων εναλλακτικών επιλογών, λαμβάνοντας υπόψη μία ποικιλία συνδυασμών και επιπέδων υπηρεσιών. Τα δίκτυα επίσης χαρακτηρίζονται ανάλογα με τις συνολικές τους ιδιότητες με τον εξής τρόπο:[8]

- Κανονικά δίκτυα. Ένα δίκτυο όπου όλοι οι κόμβοι έχουν τον ίδιο αριθμό ακμών. Στο ίδιο πνεύμα και αντίστοιχα ένα τυχαίο δίκτυο είναι ένα δίκτυο που σχηματίζεται από τυχαίες διαδικασίες. Ενώ τα κανονικά δίκτυα τείνουν να συνδέονται με υψηλά επίπεδα χωρικής οργάνωσης (π.χ. με ένα πλέγμα πόλης), τα τυχαία δίκτυα τείνουν να συνδέονται με καιροσκοπικές ευκαιρίες ανάπτυξης, όπως είναι η προσωρινή πρόσβαση σε έναν πόρο.
- Δίκτυα μικρόκοσμου. Ένα δίκτυο με πυκνές συνδέσεις μεταξύ των κοντινών γειτονικών κόμβων και λίγες, αλλά αρκετά σημαντικές συνδέσεις μεταξύ μακρινούς κόμβους. Τέτοια δίκτυα είναι ιδιαίτερα ευάλωτα στο να αποτυγχάνουν καταστροφικά σε περιπτώσεις που σχετίζονται με μεγάλους γεωγραφικούς άξονες.
- Δίκτυα χωρίς κλίμακα. Ένα δίκτυο που έχει μία ισχυρή ιεραρχική διάσταση, με λίγους κόμβους και πολλές ακμές. Τέτοια δίκτυα εξελίσσονται μέσα από την δυναμική των προτιμήσεων, με την οποία νέοι κόμβοι θα προστεθούν στο δίκτυο μόνο αν έχουν αυξημένη προτίμηση και όχι τυχαία.

2.4 Ο σχεδιασμός ταξιδιού στα δίκτυα μετακίνησης Μέσων Μαζικής Μεταφοράς

Αυτή η ενότητα [9] εξετάζει την σύνδεση του σχεδιασμού ταξιδιού με το δίκτυο δημόσιας συγκοινωνίας. Σε αυτή την περίπτωση τα δεδομένα εισόδου δίνονται από ένα δρομολόγιο. Σε γενικές γραμμές, κάποιος πεπερασμένος αριθμός στάσεων αποτελούν ένα δρομολόγιο (όπως στάσεις λεωφορείων, σταθμοί τρένων), ένα σύνολο διαδρομών (όπως γραμμές του λεωφορείου ή του τρένου) και ένα σύνολο ταξιδιών. Τα ταξίδια αντιστοιχούν σε μεμονωμένα οχήματα που επισκέπτονται τις στάσεις κατά μήκος μιας συγκεκριμένης διαδρομής και σε κάποια συγκεκριμένη ώρα της ημέρας. Τα ταξίδια μπορούν να υποδιαιρεθούν περαιτέρω σε ακολουθίες στοιχειωδών συνδέσεων, κάθε μία η οποία θα δίνεται ως ένα ζεύγος από στάσεις (προέλευσης/ προορισμού) και ώρες (αναχώρησης/άφιξης) μεταξύ των οποίων το όχημα ταξιδεύει χωρίς διακοπή.

Τα δημόσια δίκτυα διέλευσης από την φύση τους εξαρτώνται πλήρως από τον χρόνο, εν αντιθέσει με τα οδικά δίκτυα, εφόσον από σημεία του δικτύου τα μέσα μπορούν να περάσουν μόνο σε συγκεκριμένα και διακριτά χρονικά σημεία. Ως εκ τούτου, η κύρια διαφορά αφορά

την μοντελοποίηση του δρομολόγιο με κατάλληλο τρόπο, ώστε να καταστεί δυνατός ο κατάλληλος υπολογισμός των διαδρομών. Για τα προβλήματα που αφορούν τα οδικά δίκτυα μία μοντελοποίηση απλού τύπου, η οποία θα αναζητά την βέλτιστη διαδρομή (συνήθως την πιο γρήγορη από άποψη χρόνου) είναι συχνά επαρκής. Στα δίκτυα μαζικής μεταφοράς δεν ισχύει το ίδιο καθώς είναι σημαντικό να αναζητούνται λύσεις για περισσότερα προβλήματα, λαμβάνοντας υπόψη συχνά διάφορα κριτήρια βελτιστοποίησης. Τέτοιο είδος μοντελοποίησης θα μας απασχολήσει και θα αναλυθεί σε επόμενα κεφάλαια της παρούσας διπλωματικής εργασίας.

Η επιτάχυνση της επίλυσης των προβλημάτων για αποτελεσματικούς σχεδιαστές ταξιδιού αποτελεί ένα μακροχρόνιο ζήτημα. Ένας μεγάλος αριθμός αλγορίθμων έχει αναπτυχθεί μέχρι σήμερα, όχι μόνο για να απαντήσει βασικά ερωτήματα σε μικρό χρόνο, αλλά επίσης για να ασχοληθεί με εκτεταμένα σενάρια που θα ενσωματώνουν τις καθυστερήσεις, θα υπολογίζουν τα σημαντικότερα ταξίδια, ή θα βελτιστοποιούν αθροιστικά κριτήρια που θα θέτονται, όπως είναι τα κόστη με τους χρόνους.

2.5 Η ανάπτυξη ενός δικτύου

Σε αυτή την ενότητα θα αναλυθεί ο τρόπος αναπτύσσεται ένα δίκτυο μετακινήσεων και παράλληλα ο τρόπος με τον οποίο επετεύχθη η ανάπτυξη των δικτύων των Μέσων Μαζικής Μεταφοράς, τα οποία αναλύονται παρακάτω, και τα δεδομένα των οποίων χρησιμοποιήθηκαν για την μοντελοποίηση στην παρούσα διπλωματική εργασία.

Η διαδικασία μετατροπής γεωγραφικών δεδομένων που απεικονίζονται σε ένα χάρτη σε ψηφιακή μορφή ονομάζεται ψηφιοποίηση. Ένας θεματικός χάρτης ονομάζεται επικάλυψη (coverage). Τα σημεία (points), οι γραμμές (lines) και οι επιφάνειες (areas) ενός επιθέματος του ψηφιακού χάρτη με την ψηφιοποίηση αποκτούν γεωγραφικές συντεταγμένες χ και ψ.

Μέσω της χρήσης Γεωγραφικών Συστημάτων Πληροφοριών (Geographic Information Systems), δημιουργήθηκαν δίκτυα, τα οποία κατασκευάστηκαν έπειτα από την συλλογή δεδομένων και μέσω δεδομένων χαρτογράφησης μέσω δορυφόρου. Η αποτύπωση είναι ακριβής σε επίπεδο μερικών ιντσών.

Το σύστημα γεωγραφικών πληροφοριών (ΣΓΠ) [2], γνωστό ευρέως και ως G.I.S. (Geographic Information Systems), είναι ένα σύστημα διαχείρισης χωρικών δεδομένων (spatialdata) και συσχετισμένων ιδιοτήτων. Στην πιο αυστηρή μορφή του είναι ένα ψηφιακό σύστημα, ικανό να ενσωματώσει, αποθηκεύσει, προσαρμόσει, αναλύσει και παρουσιάσει γεωγραφικά συσχετισμένες (geographically - referenced) πληροφορίες. Σε πιο γενική μορφή, ένα ΓΣΠ είναι ένα εργαλείο "έξυπνου χάρτη", το οποίο επιτρέπει στους χρήστες του να αποτυπώσουν μία περίληψη του πραγματικού κόσμου, να δημιουργήσουν διαδραστικά ερωτήσεις χωρικού ή

περιγραφικού χαρακτήρα, να αναλύσουν τα χωρικά δεδομένα, να τα προσαρμόσουν και να τα αποδώσουν σε αναλογικά μέσα ή σε ψηφιακά μέσα.

Η τεχνολογία των Συστημάτων Γεωγραφικών Πληροφοριών (ΣΓΠ/GIS) άρχισε να αναπτύσσεται κατά την δεκαετία του 1980 και σήμερα αποτελεί ένα από τα πλέον σημαντικά ερευνητικά πεδία των γεωεπιστημών και της επιστήμης των υπολογιστών. Τα GISεμφανίστηκαν στην αρχή ως ένα λογισμικό (GISystem), το οποίο σχεδιάστηκε για την αναπαράσταση γεωγραφικής πληροφορίας και απευθυνόταν κυρίως σε εξειδικευμένους χρήστες. Στη συνέχεια εξελίχθηκε σε ένα αυτοτελές επιστημονικό πεδίο (GISience) η έρευνα του οποίου εστίαζε στην ανάπτυξη και περαιτέρω βελτίωση των Συστημάτων Γεωγραφικών Πληροφοριών μέσα από την αξιοποίηση της άλγεβρας χαρτών (mapalgebra) και τη μοντελοποίηση των χωρικών λειτουργιών (spatialoperations). Σήμερα συνιστούν πλέον συστήματα παροχής «γεωγραφικών» υπηρεσιών (GIServices) (π.χ. πλοήγηση) σε διάφορους χρήστες του παγκόσμιου ιστού και της κινητής τηλεφωνίας. [18]

Τα συστήματα ΣΓΠ αποτυπώνουν χωρικά δεδομένα σε γεωγραφικό ή χαρτογραφικό ή καρτεσιανό σύστημα συντεταγμένων. Βασικό χαρακτηριστικό των ΣΓΠ είναι ότι τα χωρικά δεδομένα συνδέονται και με περιγραφικά δεδομένα, π.χ. μία ομάδα σημείων που αναπαριστούν θέσεις πόλεων συνδέεται με ένα πίνακα όπου κάθε εγγραφή εκτός από τη θέση περιέχει πληροφορίες όπως ονομασία, πληθυσμός κ.λπ.

Τα συστήματα ΣΓΠ είναι πληροφοριακά συστήματα (Information Systems) που παρέχουν την δυνατότητα συλλογής, διαχείρισης, αποθήκευσης, επεξεργασίας, ανάλυσης και οπτικοποίησης σε ψηφιακό περιβάλλον, των δεδομένων που σχετίζονται με τον χώρο. Τα δεδομένα αυτά συνήθως λέγονται γεωγραφικά ή χαρτογραφικά ή χωρικά (spatial) και μπορεί να συσχετίζονται με μία σειρά από περιγραφικά δεδομένα τα οποία και τα χαρακτηρίζουν μοναδικά. [2]

Για την ψηφιοποίηση και ανάπτυξη των δικτύων που μελετήθηκαν στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το λογισμικό QGIS (Quantum GIS). Το λογισμικό αυτό είναι ένα φιλικό προς τον χρήστη λογισμικό Γεωγραφικών Συστημάτων Πληροφοριών Ανοιχτού Κώδικα και διέπεται από την άδεια General Public License (GNU). Το QGIS αποτελεί επίσημο πρόγραμμα του Open Source Geospatial Foundation (OSGeo). Τρέχει σε Linux, Unix, Mac OS X και Windows και υποστηρίζει πολλές μορφές γεωγραφικών δεδομένων (διανυσματικών και ψηφιδωτών) και μορφές βάσεων δεδομένων. Έχει αρκετές λειτουργικές δυνατότητες που αυξάνονται με τις νεότερες εκδόσεις που είναι συνεχείς. Το QGIS είναι διαθέσιμο από την ιστοσελίδα http://www.qgis.org και στην δικιά μας περίπτωση χρησιμοποιήθηκε η έκδοση για Windows.

Απαραίτητη προϋπόθεση για τη δημιουργία του δικτύου είναι η εγκατάσταση ενός θεματικού χάρτη, ο οποίος εγκαταστάθηκε στην γεωγραφική βάση δεδομένων. Στην περίπτωσή μας ο θεματικός χάρτης που χρησιμοποιήθηκε είναι το πρόσθετο "OpenStreetMap" (OSM).

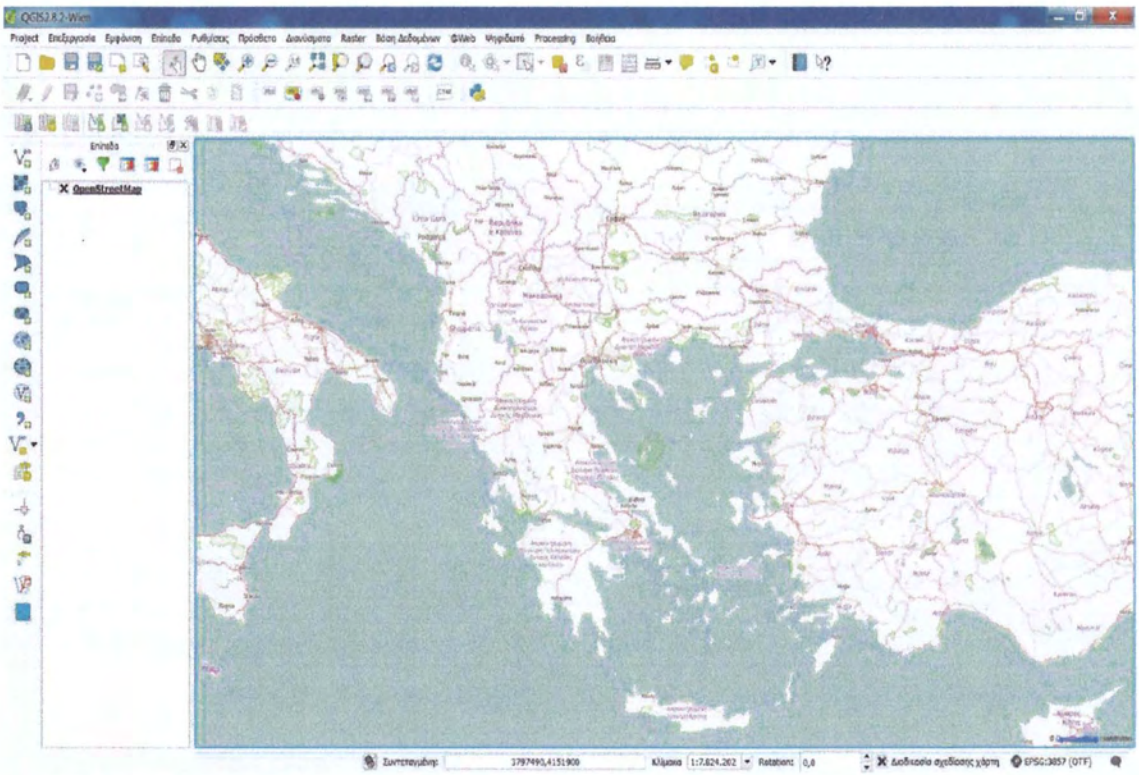
Το OpenStreetMap (OSM) είναι ένας χάρτης με ελεύθερη άδεια ο οποίος αναπτύσσεται από μία κοινότητα εθελοντών που συνεισφέρουν και διατηρούν δεδομένα σχετικά με δρόμους, μονοπάτια, καφετέριες, σιδηροδρομικούς σταθμούς και πολλά περισσότερα σε όλο τον κόσμο. Οι συνεισφέροντες χρησιμοποιούν αεροφωτογραφίες, συσκευές GPS και τοπικούς χάρτες χαμηλής τεχνολογίας για να σιγουρευτούν πως το OSMείναι ακριβής και ενημερωμένο στο μεγαλύτερο δυνατό επίπεδο.

Το OpenStreetMap έχει ελεύθερα δεδομένα που διατίθενται με άδεια Open Data Commons Open Database License (ODbL), τα οποία μπορεί να χρησιμοποιήσει κανείς για οποιονδήποτε σκοπό, εφόσον μνημονευθεί το OpenStreetMap και οι συνεισφέροντές του.

Τα δεδομένα των δικτύων μετακίνησης τείνουν να παραμένουν σχετικά σταθερά κατά την διάρκεια του χρόνου. Τα περισσότερα μοντέλα έχουν αποθηκευμένα τα δεδομένα των δικτύων, τα οποία απλά ενημερώνονται, ώστε να εξελίσσονται σε ειδικές περιπτώσεις, όπως είναι η κατασκευή νέων δρόμων, η ικανότητα του οδοστρώματος (π.χ. προσθήκη λωρίδων), κ.α.

Τα δεδομένα που συλλέχθηκαν αφορούν τα απαραίτητα στοιχεία που συντάσσουν μία διαδρομή. Τα στοιχεία αυτά είναι οι στάσεις (κόμβοι) και οι συνδέσεις (ακμές) των στάσεων αυτών. Το κάθε χωρικό στοιχείο που συλλέχθηκε και καταγράφηκε σαν δίκτυο στο λογισμικό QGISαποδόθηκε με ένα ξεχωριστό χρώμα και ομαδοποιήθηκαν σε επίπεδα (layers) σύμφωνα με τις διαδρομές που ομαδοποιούνται σε κάθε δίκτυο.

Τα στοιχεία του OSM χρησιμοποιούνται στα πλαίσια της παρούσας διπλωματικής εργασίας για την δημιουργία γράφου και το routing με όλα τα μέσα από την σχετική μηχανή.



Εικόνα 5: Περιβάλλον του λογισμικού QGIS με εγκατεστημένο τον χάρτη OpenStreetMap

2.6 Παρουσίαση των δικτύων που μελετήθηκαν

Στην παρούσα διπλωματική εργασία συλλέχθηκαν πρωτογενή δεδομένα για συνολικά 3 δίκτυα, τα οποία μελετήθηκαν και χρησιμοποιήθηκαν για την προσομοίωση του κώδικα που αναλύεται σε επόμενα κεφάλαια. Τα δεδομένα συλλέχθηκαν έπειτα από τηλεφωνικές επικοινωνίες με τους φορείς, επί τόπου επισκέψεις, αλλά και από τις επίσημες ιστοσελίδες αυτών. Τα δεδομένα επεξεργάστηκαν αργότερα με βάση την εμπειρική γνώση των δικτύων. Έπειτα ψηφιοποιήθηκαν με τον τρόπο που παρουσιάστηκε σε προηγούμενη ενότητα.

Το κομμάτι της συλλογής των δεδομένων ήταν πολύ σημαντικό και έγινε με ιδιαίτερη προσοχή γιατί ακόμα και αν βρεθεί και υλοποιηθεί ο καλύτερος αλγόριθμος για την εξαγωγή αποτελεσμάτων, αυτά θα είναι χωρίς αντίκρισμα αν τα δεδομένα με τα οποία λειτουργεί δεν είναι αξιόπιστα, πλήρη, ακριβή και κατάλληλα δομημένα.

Τα δεδομένα που συλλέχθηκαν καταρχάς είναι πλήρη, με την έννοια ότι τα στοιχεία αφορούν το σύνολο των γραμμών και όχι μέρος αυτών. Επίσης διαθέτουν μεταξύ τους συνάφεια, είτε εσωτερική (μεταξύ των ίδιων των στοιχείων), είτε με άλλα στοιχεία που χρησιμοποιούνται (όπως π.χ. με τους χρόνους διεξαγωγής των δρομολογίων).

Τα δίκτυα των οποίων τα δεδομένα μας απασχόλησαν είναι τα εξής:

- Δίκτυο δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης. [10] Το Αστικό Κ.Τ.Ε.Λ. Αλεξανδρούπολης δημιουργήθηκε από την δεκαετία του 1980. Την εποχή που ξεκίνησε να λειτουργεί για τις μετακινήσεις των επιβατών χρησιμοποιούσε αρκετά σύγχρονα λεωφορεία για την εποχή. Η μορφή που είχε τότε ονομάζεται σήμερα το "παλιό αστικό" και τα δρομολόγια που διεξάγονταν αφορούσαν μόνο κοντινές αποστάσεις. Σήμερα ο στόλος έχει γίνει σύγχρονος και διαρκώς εξελίσσεται προσφέροντας άνετη μετακίνηση εντός πόλεως, αλλά και σε όλα τα χωριά του Δήμου Αλεξανδρουπόλεως.
- Δίκτυο δρομολογίων Αστικού Κ.Τ.Ε.Λ. Κομοτηνής. [12] Το Αστικό Κ.Τ.Ε.Λ. Κομοτηνής δραστηριοποιείται στον χώρο των μεταφορών από το 1989 και παρέχει ποιοτικές μετακινήσεις στους κατοίκους, επισκέπτες, αλλά και στους φοιτητές της Κομοτηνής. Ο στόλος της εταιρίας αποτελείται από δέκα εννιά (19) οχήματα. Έντεκα (11) οχήματα δώδεκα μέτρων, ένα (1) όχημα δέκα μέτρων, έξι (6) οχήματα μίνι μπας και ένα (1) αρθρωτό όχημα.
- Δίκτυο δρομολογίων Αστικού Κ.Τ.Ε.Λ. Ξάνθης. [14] Το Αστικό Κτελ Ξάνθης εμφανίστηκε το 1992 μετά από διάσπαση του από το Υπεραστικό ΚΤΕΛ Ν. Ξάνθης κατέχοντας τότε στην δύναμή του έξι (6) αστικά λεωφορεία. Στην αρχή ξεκίνησε να εξυπηρετεί τις γύρω αστικές περιοχές.

  Το 1996 προστέθηκαν άλλα δύο (2) αστικά λεωφορεία και το 1998 άλλα τρία (3) αστικά λεωφορεία πυκνώνοντας έτσι όλο και περισσότερο την συχνότητα των δρομολογίων και καλύπτοντας μεγαλύτερο εύρος των αναγκών του επιβατικού κοινού.

  Τον Αύγουστο του 2008 το ΚΤΕΛ ΑΣΤΙΚΩΝ ΓΡΑΜΜΩΝ ΞΑΝΘΗΣ αλλάζει μορφή σύμφωνα με τον νόμο 2963/2001 και γίνεται Αστικό ΚΤΕΛ ΞΑΝΘΗΣ Α.Ε.

  Αμέσως μετά την αλλαγή της εταιρικής μορφής του ΑΣΤΙΚΟ ΚΤΕΛ Α.Ε. προχώρησε στην αύξηση του στόλου η οποία σήμερα φθάνει τα δεκαεπτά (17) αστικά λεωφορεία συνεχίζοντας για την καλύτερη δυνατή εξυπηρέτηση του αστικού επιβατικού κοινού.

Τα δεδομένα που συλλέχθηκαν αφορούν το σύνολο των δρομολογίων των φορέων που αναφέρθηκαν παραπάνω. Δηλαδή, τα δίκτυα που αναπτύχθηκαν και ψηφιοποιήθηκαν απαρτίζονται από το σύνολο των προορισμών στους οποίους εκτελούν δρομολόγια τα Αστικά Κτελ των παραπάνω πόλεων.

Αναλυτικά οι διαδρομές που δημιουργούν τα 3 επιμέρους δίκτυα είναι οι εξής:

Για το Αστικό Κ.Τ.Ε.Λ. Αλεξανδρούπολης:

- Γραμμή 1 Νοσοκομείο
- Γραμμή 2 Μαΐστρος
- Γραμμή 3 Μάκρη

- Γραμμή 4 Παλαγία
- Γραμμή 5 Λουτρά
- Γραμμή 6 Νίψα
- Γραμμή 7 Μεσημβρία
- Γραμμή 8 Άβαντας
- Γραμμή 9 Αμφιτρίτη
- Γραμμή 10 Αισύμη
- Γραμμή 11 Αγία Παρασκευή
- Γραμμή 12 Δίκελλα
- Γραμμή 14 Νέα Νεκροταφεία
- Γραμμή 21 Τέρμα Άβαντος

Για το Αστικό Κ.Τ.Ε.Λ. Κομοτηνής:

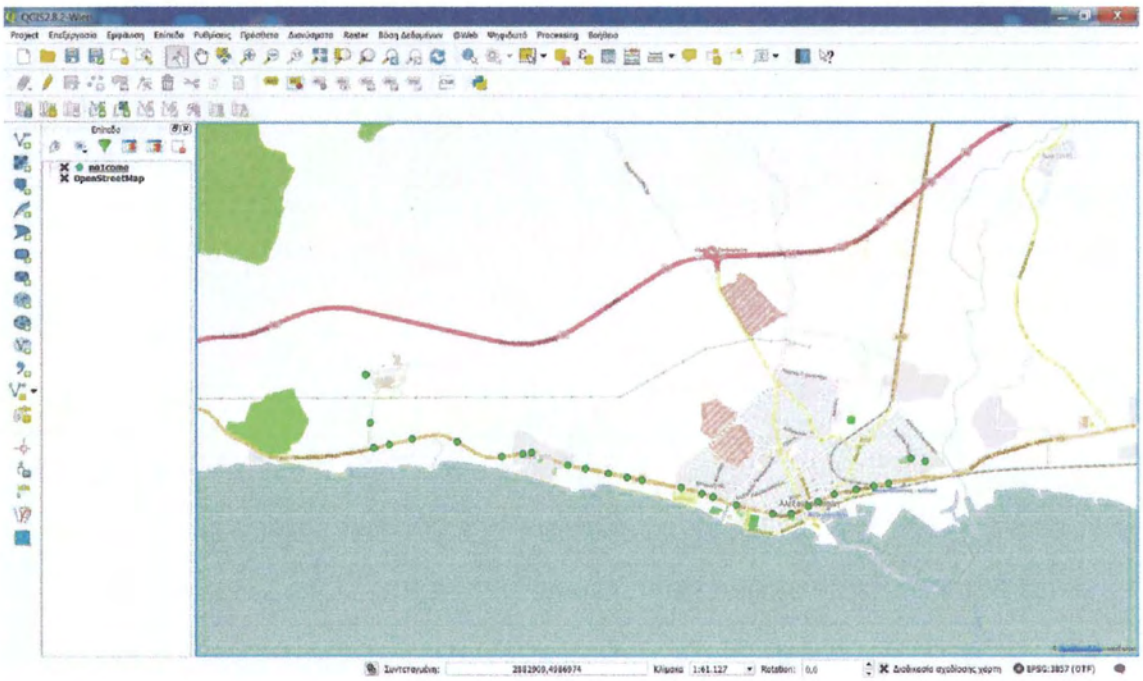
- Γραμμή 1 Περιφέρεια Α.Μ.Θ. - 1ο ΕΠΑΛ
- Γραμμή 2Α Νέα Μοσυνούπολη, Κολυμβητήριο
- Γραμμή 2Β Κολυμβητήριο, Νέα Μοσυνούπολη
- Γραμμή 3 Στρατώνες - ΟΣΕ
- Γραμμή 4 Πανεπιστημιούπολη
- Γραμμή 5 Μεταλλουργική
- Γραμμή 6 Πάμφορο
- Γραμμή 7 Σώστης
- Γραμμή 8 Ασώματος
- Γραμμή 9 Άγιοι Θεόδωροι
- Γραμμή 10 Πάνδροσος
- Γραμμή 11 Νέα Ανδριανή

Για το Αστικό Κ.Τ.Ε.Λ. Ξάνθης:

- Γραμμή 1 Χρύσα
- Γραμμή 2 Καλλιθέα - Νοσοκομείο
- Γραμμή 3 Κιμμέρια
- Γραμμή 4 Μαγικό
- Γραμμή 5 Φελώνη
- Γραμμή 6 Εύμοιρο - Μορσίνη
- Γραμμή 7 Πηγάδια

Τα πρωτογενή δεδομένα που συλλέχθηκαν και ψηφιοποιήθηκαν αρχικά αφορούν τις στάσεις του κάθε δρομολογίου, οι οποίες αποτελούν και τους κλάδους του εκάστοτε δικτύου. Η καταγραφή των ακριβών συντεταγμένων της κάθε στάσης έγινε είτε έπειτα από στοιχεία που μας παρείχαν οι εταιρίες, είτε χειροκίνητα πάνω σε χάρτες αρχικά. Έπειτα και όπως αναφέρθηκε παραπάνω ψηφιοποιήθηκαν με την βοήθεια του λογισμικού QGIS και με την χρήση του χάρτη - πρόσθετου OpenStreetMap (OSM). Τα δεδομένα που προέκυψαν έπειτα της ψηφιοποίησης είναι σε μορφή shapefile (.shp).

Οι απεικονίσεις γίνονται πάνω στον χάρτη και τα δεδομένα (συντεταγμένες) του OpenStreetMap (OSM) και στο περιβάλλον του λογισμικού QGIS. Ενδεικτικά παρατίθεται η γεωγραφική απεικόνιση των κόμβων της Γραμμής 1 «Νοσοκομείο», του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης.



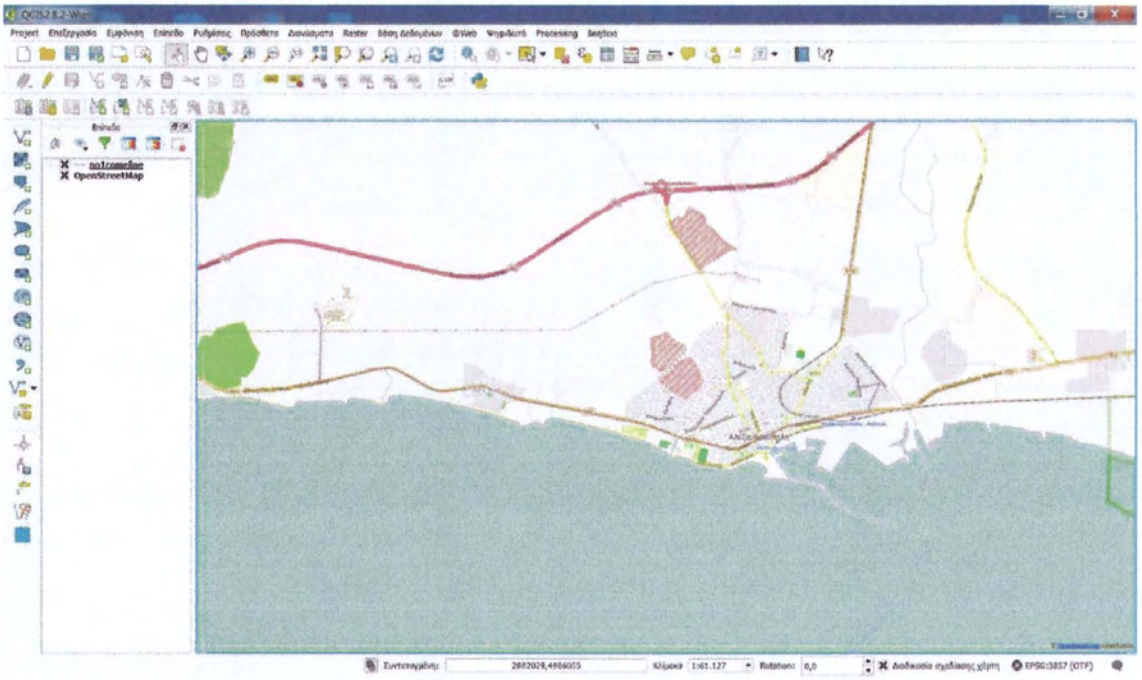
Εικόνα 6: Γεωγραφική απεικόνιση των κόμβων της γραμμής 1 του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης με την χρήση του λογισμικού QGIS με OpenStreetMap και Openlayers

Στον κάθε κόμβο (στάση) του κάθε δρομολογίου, και για τον αξιόπιστο διαχωρισμό αυτών, έχει δοθεί ένα όνομα (περιγραφή). Το κάθε όνομα για την κάθε στάση είναι μοναδικό και δεν έχει δοθεί αυθαίρετα, αφού χρησιμοποιήθηκαν τα ίδια ονόματα με αυτά που χρησιμοποιούν για την κάθε στάση οι επιμέρους φορείς (Αστικά Κ.Τ.Ε.Λ.).

Οι κόμβοι του κάθε δικτύου συνδέονται μέσω γραμμών (ακμές δικτύου/routes) οι οποίες επίσης αποτελούν πρωτογενή δεδομένα στα πλαίσια της παρούσας διπλωματικής εργασίας, συλλέχθηκαν και ψηφιοποιήθηκαν με την βοήθεια του λογισμικού QGIS και με την χρήση του

χάρτη - πρόσθετου OpenStreetMap (OSM).Τα δεδομένα που προέκυψαν έπειτα της ψηφιοποίησης είναι σε μορφή shapefile (.shp).

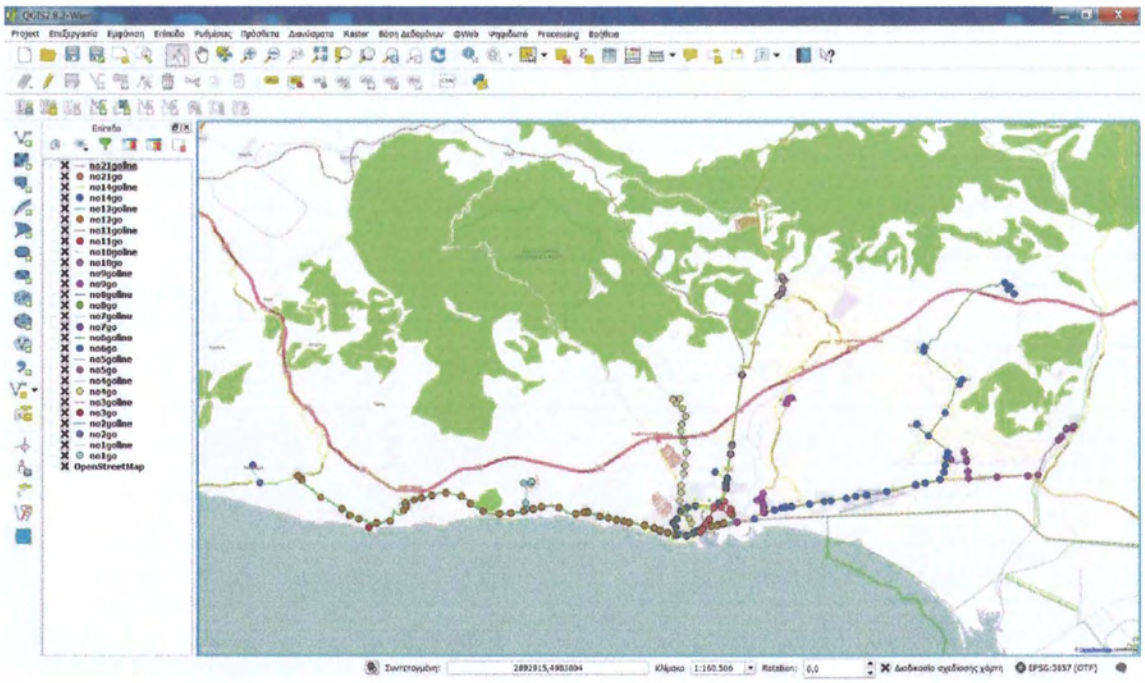
Οι απεικονίσεις γίνονται πάνω στον χάρτη OpenStreetMap (OSM) και στο περιβάλλον του λογισμικού QGIS. Ενδεικτικά παρατίθεται η γεωγραφική απεικόνιση των γραμμών της Γραμμής 1 «Νοσοκομείο», του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης.



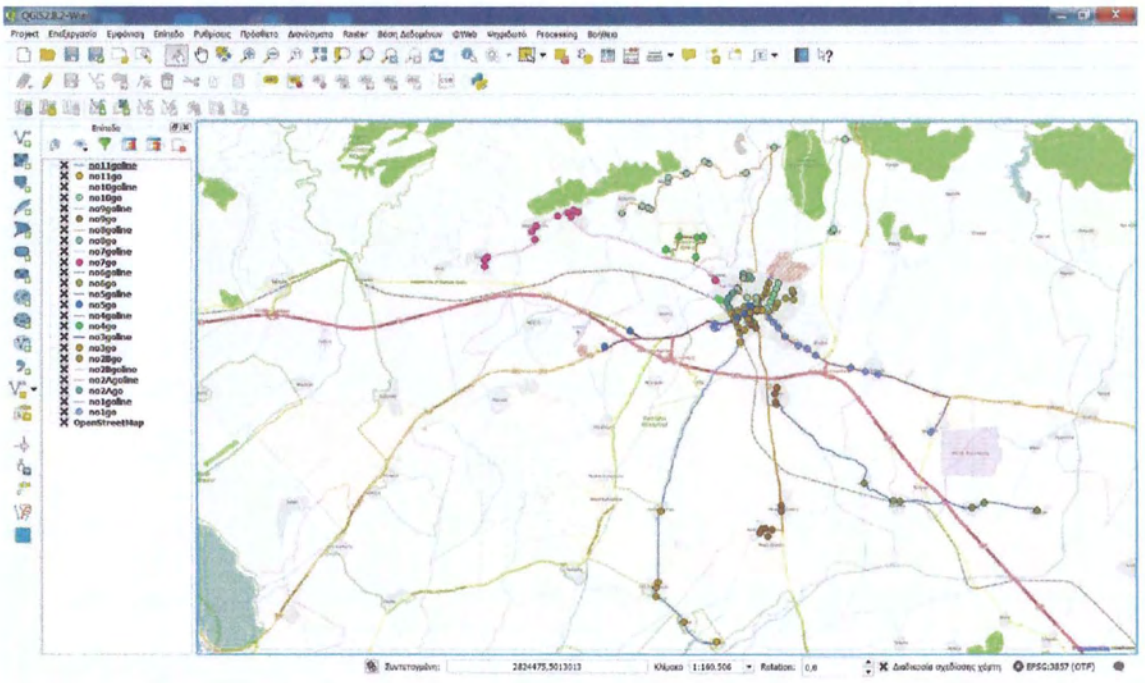
Εικόνα 7: Γεωγραφική απεικόνιση των γραμμών της γραμμής 1 του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης με την χρήση του λογισμικού QGIS με OpenStreetMap και Openlayers

Η πλειοψηφία των επιμέρους γραμμών των δικτύων σαν συνολικό δρομολόγιο, όπως παρουσιάζεται και στις γεωγραφικές απεικονίσεις, γεωμετρικά δεν φαίνονται σαν κυκλική γραμμή, όμως σαν λειτουργία υπολογίζεται σαν κυκλική. Δηλαδή, ένας κόμβος (στάση) ορίζεται σαν θεωρητικός ή και πρακτικός τερματισμός - αρχή και οι γραμμές διαχωρίζονται σε αυτές που πηγαίνουν προς αυτόν (go) και σε αυτές που έρχονται από αυτόν (come). Το σκεπτικό για αυτή την θεώρηση είναι ότι αφού οι επιβάτες, ανάλογα με την κατεύθυνση που θέλουν να μετακινηθούν θα χρησιμοποιήσουν τις αντίστοιχες στάσεις από τις δύο πλευρές του δρόμου, χωρίς να κάνουν τον περιττό κύκλο.

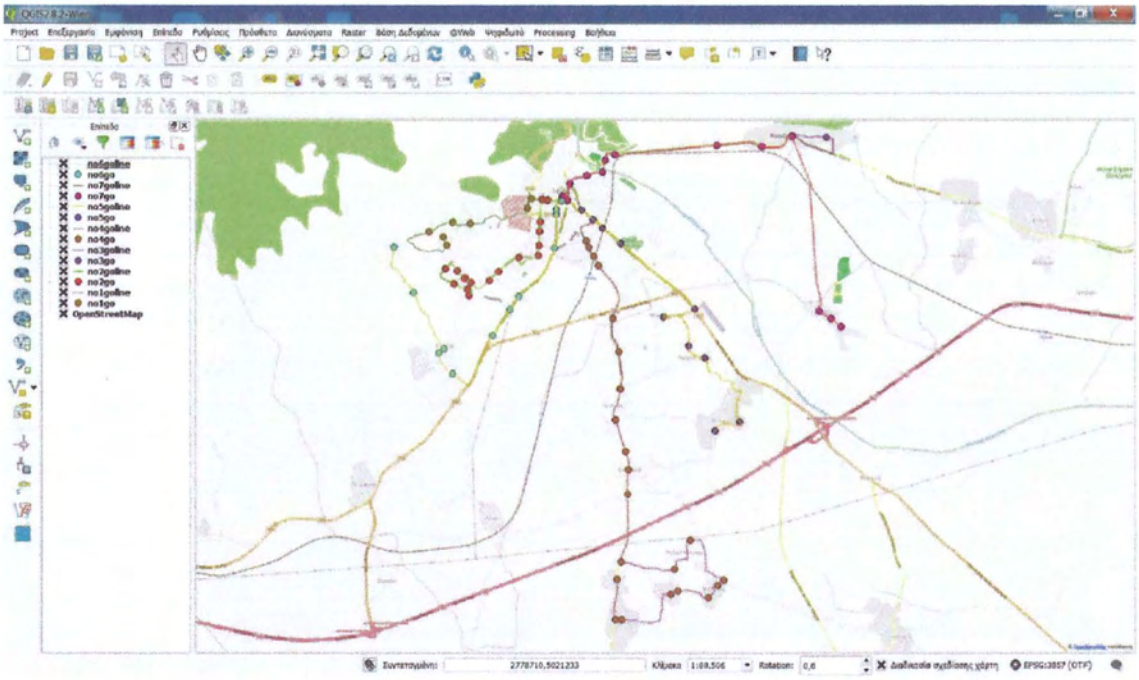
Παρακάτω παρατίθενται οι απεικονίσεις του κάθε δικτύου στο σύνολό του, ψηφιοποιημένο με την χρήση του λογισμικού QGIS και πάνω στον εγκατεστημένο χάρτη OpenStreetMap.



Εικόνα 8: Γεωγραφική απεικόνιση του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης στο σύνολό του με την χρήση του λογισμικού QGIS με εγκατεστημένο τον χάρτη OpenStreetMap



Εικόνα 9: Γεωγραφική απεικόνιση του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Κομοτηνής στο σύνολό του με την χρήση του λογισμικού QGIS με εγκατεστημένο τον χάρτη OpenStreetMap



Εικόνα 10: Γεωγραφική απεικόνιση του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Ξάνθης στο σύνολό του με την χρήση του λογισμικού QGIS με εγκατεστημένο τον χάρτη OpenStreetMap

Τα δίκτυα που παρουσιάστηκαν και τα στοιχεία των οποίων θα χρησιμοποιηθούν στον αλγόριθμο που θα αναλυθεί σε επόμενο κεφάλαιο αφορούν Μέσα Μαζικής Μεταφοράς. Επίσης σε προηγούμενο κεφάλαιο έγινε γνωστό ότι ο κώδικα που θα σχεδιαστεί στην παρούσα διπλωματική εργασία σχετίζεται με την προσομοίωση ενός σχεδιαστή ταξιδιού, ο οποίος θα δύναται να χρησιμοποιηθεί και για συνδυασμένη μετακίνηση. Αυτό σημαίνει ότι μία συνολική διαδρομή μπορεί να συμπεριλαμβάνει δύο ή περισσότερα δρομολόγια από αυτά που παρατέθηκαν παραπάνω. Η χρήση αυτή και ο συσχετισμός της εργασίας με την φύση των Μέσων Μαζικής Μεταφοράς δίνει μεγαλύτερο ρόλο και σημασία αρχικά στην ύπαρξη χρονικών στοιχείων και έπειτα στην αξιοπιστία αυτών.

Στα πρωτογενή δεδομένα που συλλέχθηκαν και θα χρησιμοποιηθούν για τους λόγους που αναλύθηκαν παραπάνω ανήκουν και τα αναλυτικά χρονοδιαγράμματα όλων των δρομολογίων του κάθε δικτύου. Τα χρονοδιαγράμματα αποτελούνται από τις ώρες αναχώρησης και τις ώρες άφιξης της κάθε διαδρομής που εκτελείται καθημερινά, για τα εκάστοτε δρομολόγια που απαρτίζουν το κάθε δίκτυο.

Τα χρονοδιαγράμματα των δρομολογίων συνεισφέρουν κυρίως στα πλαίσια συνδυασμένων μετακινήσεων στον συνδυασμό δύο ξεχωριστών δρομολογίων, ώστε να βελτιστοποιηθεί μία συνολική διαδρομή. Όμως τα χρονοδιαγράμματα σαν ξεχωριστά δεδομένα μπορούν να

παρέχονται σαν επιπλέον πληροφορίες στους χρήστες ενός σχεδιαστή ταξιδιού εξελίσσοντας τις παρεχόμενες υπηρεσίες.

Ενδεικτικά παρατίθεται το χρονοδιάγραμμα όλων των ημερήσιων δρομολογίων της Γραμμής 2 του δικτύου του Αστικού ΚΤΕΛ Αλεξανδρούπολης.

Καθημερινές			
go		come	
Ώρα εκκίνησης	Ώρα Τερματισμού	Ώρα εκκίνησης	Ώρα Τερματισμού
07.25	07.37	07.53	08.05
08.30	08.42	08.42	08.54
09.30	09.42	09.42	09.54
10.30	10.42	10.42	10.54
11.30	11.42	11.42	11.54
12.30	12.42	12.58	13.10
13.40	13.52	13.52	14.04
14.40	14.52	15.08	16.10
17.30	17.42	17.42	17.54
18.30	18.42	18.42	18.54
19.30	19.42	19.42	19.54
20.30	20.42	20.42	20.54

Σάββατο			
go		come	
Ώρα εκκίνησης	Ώρα Τερματισμού	Ώρα εκκίνησης	Ώρα Τερματισμού
07.25	07.37	07.53	08.05
08.30	08.42	08.42	08.54
10.30	10.42	10.42	10.54
12.30	12.42	12.58	13.10
14.40	14.52	15.08	16.10
17.30	17.42	17.42	17.54
18.30	18.42	18.42	18.54
20.30	20.42	20.42	20.54

Κυριακή			
go		come	
Ώρα εκκίνησης	Ώρα Τερματισμού	Ώρα εκκίνησης	Ώρα Τερματισμού
08.00	08.12	08.12	08.24
11.30	11.42	11.42	11.54
18.30	18.42	18.42	18.54
20.30	20.42	20.42	20.54

Πίνακας 1: Πίνακας χρονοδιαγραμμάτων δρομολογίων της γραμμής 2 του δικτύου δρομολογίων Αστικού Κ.Τ.Ε.Λ. Αλεξανδρούπολης

2.7 Τελική μορφή δεδομένων για journey planning

Τα δεδομένα που συλλέχθηκαν και ψηφιοποιήθηκαν με τον τρόπο που παρουσιάστηκε παραπάνω δεν δύναται να χρησιμοποιηθούν στην μορφή που βρίσκονται για την χρήση σε πληροφοριακά συστήματα, τα οποία χρησιμοποιούνται για σχεδιασμό ταξιδιού. Χρειάζονται περαιτέρω τροποποίηση η οποία θα μετατρέψει τα δεδομένα μορφής shapefile σε δεδομένα μορφής GTFS (General Transit Feed Specification).Ο graph builder, η έννοια και η λειτουργία του οποίου, θα αναπτυχθεί σε επόμενο κεφάλαιο θα χρησιμοποιεί δεδομένα για τις αστικές συγκοινωνίες που στηρίζονται πάνω στο πρότυπο GTFS.

Το GTFS [21] γενικά ορίζει μία κοινή μορφή για τα χρονοδιαγράμματα των δημόσιων συγκοινωνιών και τις γεωγραφικές πληροφορίες που σχετίζονται με αυτά. Δηλαδή συνδυάζει τα χρονοδιαγράμματα με τα γεωχωρικά δεδομένα. Τα αρχεία GTFS επιτρέπουν στους οργανισμούς μαζικής μεταφοράς να δημοσιεύουν στοιχεία για τις μετακινήσεις των μέσων τους και επίσης επιτρέπουν σε προγραμματιστές να γράψουν εφαρμογές που τροφοδοτούνται με τα δεδομένα με διαλειτουργικό τρόπο. Η ιδέα για το GTFS ξεκίνησε από την Portland TriMet, που είναι εταιρία αστικών συγκοινωνιών στο Portland, Oregon των ΗΠΑ. Αναπτύχθηκε σε συνεργασία με την Google η οποία και το χρησιμοποιεί για την υπηρεσία δρομολόγησης στους χάρτες της (GoogleMaps).

Κατ' επέκταση το πρότυπο GTFS μπορεί να χρησιμοποιηθεί στο σχεδιασμό ταξιδιών, στον προγραμματισμό δρομολογίων, αλλά και σε μεγάλο αριθμό εφαρμογών που σχετίζονται με τις κάθε είδους πληροφορία των δρομολογίων των Μέσων Μαζικής Μεταφοράς.

Το GTFS – real time [21] αποτελεί μία επέκταση του προτύπου GTFS και επιτρέπει στα πρακτορεία συγκοινωνιών να παρέχουν ενημερώσεις που σχετίζονται με τα δρομολόγια τους σε πραγματικό χρόνο. Το πρότυπο αυτό σχεδιάστηκε ώστε να διευκολύνει τη διαλειτουργικότητα του GTFS προτύπου και να εστιάσει στην άμεση και έγκαιρη ενημέρωση του επιβάτη. Υποστηρίζει τις αλλαγές που αφορούν ένα ταξίδι όπως καθυστερήσεις, ακυρώσεις και αλλαγές, τις αλλαγές σε μία υπηρεσία όπως την μεταφορά θέσης μία στάσης, καθώς και τοποθεσίες των οχημάτων.

Μία συλλογή δεδομένων σε πρότυπο GTFS αποτελείται από μία σειρά από αρχεία κειμένου (txt) συγκεντρωμένα σε ένα φάκελο και πιο συγκεκριμένα σε ένα συμπιεσμένο αρχείο (zip). Όλα αυτά τα αρχεία αντιστοιχούν σε έναν πίνακα, τα πεδία του οποίου διαχωρίζονται με κόμμα.

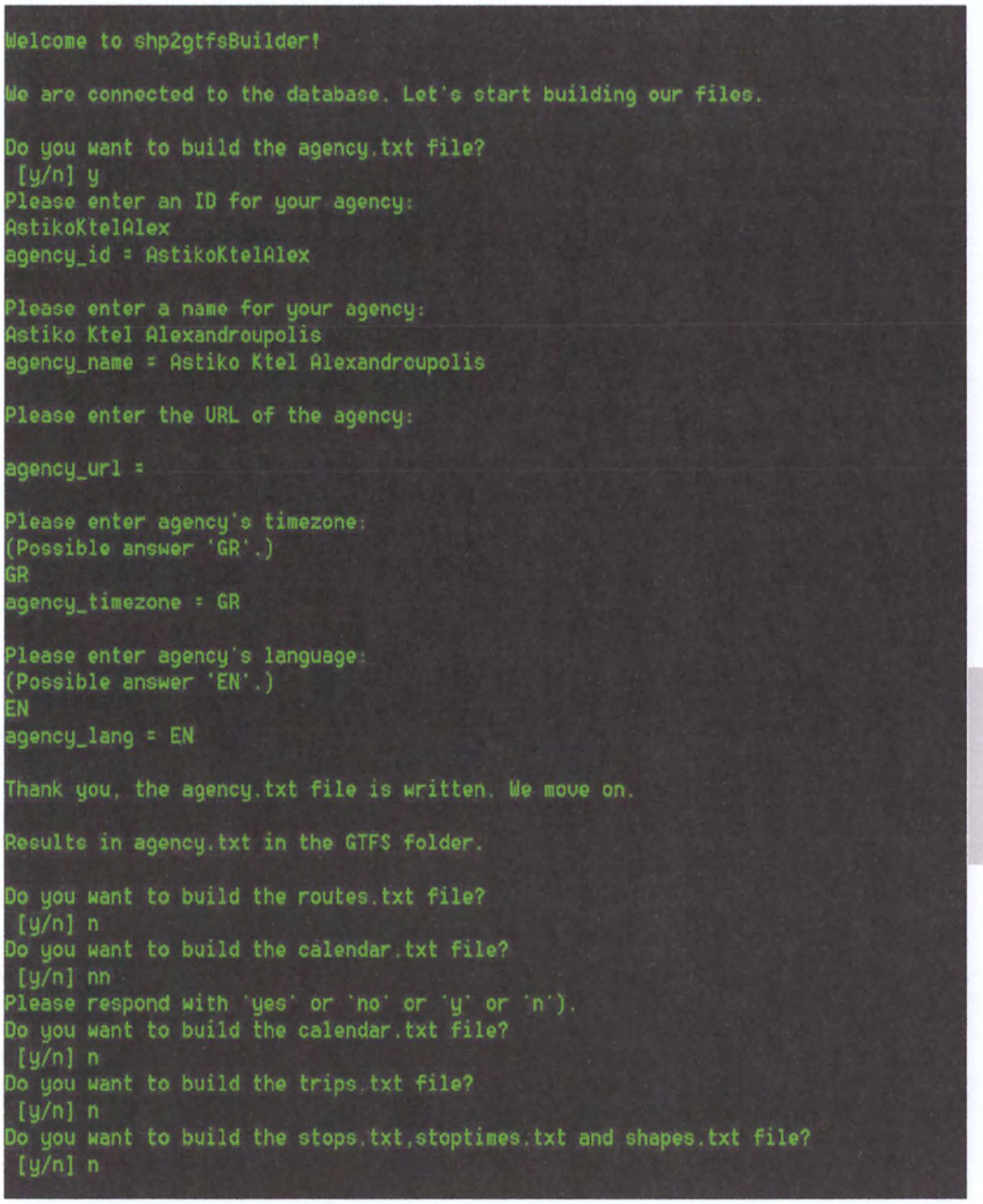
Κάθε αρχείο κειμένου μοντελοποιεί μία διαφορετική πτυχή του συνόλου της πληροφορίας: στάσεις, διαδρομές, δρομολόγια, χρονοδιαγράμματα κ.λπ. Ακολουθώντας τους οδηγούς δημιουργίας ενός GTFS πακέτου, μία εταιρία συγκοινωνιών μπορεί να δημοσιεύσει τα

δεδομένα της και να δώσει τη δυνατότητα στους προγραμματιστές να συμπεριλάβουν τα δρομολόγιά της στις εφαρμογές τους. [21]

Τα αρχεία που παράγονται στην περίπτωσή μας, συνοπτικά είναι τα εξής:

- "agency.txt", το οποίο περιέχει πληροφορίες για το ποιός οργανισμός παρέχει τα δρομολόγια,
- "calendar.txt", το οποίο παρέχει πληροφορίες για το πότε τα δρομολόγια κάθε γραμμής είναι διαθέσιμα,
- "routes.txt", το οποίο παρέχει μία λίστα σχετικά με τις επιμέρους γραμμές,
- "shapes.txt", το οποίο παρέχει πληροφορίες για το σχήμα της κάθε διαδρομής πάνω στον χάρτη,
- "stops.txt", το οποίο παρέχει πληροφορίες για τις γεωγραφικές συντεταγμένες της κάθε στάσης,
- "trips.txt", το οποίο παρέχει πληροφορίες - λίστα με όλα τα δρομολόγια που εκτελούνται για κάθε επιμέρους γραμμή.

Ειδικά, στη δική μας περίπτωση, ο συνδυασμός και η μετατροπή των δεδομένων από γεωχωρικά (shape file layers) σε μορφή GTFS γίνεται με το λογισμικό shp2GTFSπου έχει αναπτυχθεί από τον Επίκουρο Καθηγητή του Πανεπιστημίου Θεσσαλίας κ. Σαχαρίδη Γιώργο και τους συνεργάτες του. Μέσα από το πρόγραμμα δώσαμε σαν δεδομένα εισόδου τα shapefiles που δημιουργήσαμε κατά τη ψηφιοποίηση των δεδομένων και τα χρονοδιαγράμματα και το πρόγραμμα μετά από κατάλληλη επεξεργασία μας έδωσε τα ανάλογα GTFS αρχεία. Παρατίθενται δύο εικόνες στις οποίες αρχικά παρουσιάζεται το περιβάλλον και η εντολές που δόθηκαν για την δημιουργία ενός αρχείου GTFS, καθώς και τα αποτελέσματα (αρχείο .txt) που μας προσέφερε το πρόγραμμα. Στο συγκεκριμένο παράδειγμα δημιουργείται το αρχείο agency.txt για το Αστικό Κτελ Αλεξανδρούπολης.



Εικόνα 11: Το περιβάλλον του προγράμματος shp2GTFS και η διαδικασία μετατροπής ενός αρχείου από μορφή shapefile σε GTFS

Εικόνα 12: Αρχείο agency.txt τύπου GTFS για το Αστικό Κτελ Αλεξανδρούπολης, το οποίο δημιουργήθηκε με το πρόγραμμα shp2GTFS

2.8 Σύνοψη κεφαλαίου

Στο παρόν κεφάλαιο έγινε η εννοιολογική ανάλυση γενικότερα των δικτύων, καθώς και η παρουσίαση των δικτύων που απασχόλησαν την παρούσα διπλωματική εργασία. Τα εν λόγω δίκτυα αποτελούνταν από πρωτογενή δεδομένα τα οποία συλλέχθηκαν. Επίσης αναπτύχθηκαν και οι μετατροπές που υπέστησαν τα πρωτογενή δεδομένα (ψηφιοποίηση, μετατροπή σε GTFS), ώστε να βρίσκονται σε κατάλληλη μορφή για να αποτελέσουν δεδομένα εισόδου για τους graph builders που θα αναπτυχθούν στο επόμενο κεφάλαιο.

Κεφάλαιο 3ο: Η Έννοια, η Παρουσίαση και η Παραγωγή Γράφων

3.1 Εισαγωγή

Στο παρόν κεφάλαιο αναλύεται η έννοια των γράφων και παρουσιάζονται κάποια στοιχεία που αφορούν την λειτουργία τους. Μία πρώτη αναφορά στους γράφους έγινε και στο 1ο Κεφάλαιο, στα πλαίσια της παρουσίασης του τρόπου λειτουργίας των σχεδιαστών ταξιδιού (journey planners). Η λειτουργία του σχεδιαστή ταξιδιού, και γενικότερα τα προβλήματα αναζήτησης βέλτιστης διαδρομής, βασίζεται στην επίλυση ενός γράφου. Επίσης στην συνέχεια του κεφαλαίου γίνεται η παρουσίαση και η επεξήγηση των δύο τύπων graph builders οι οποίοι επιλέχθηκαν και χρησιμοποιήθηκαν για την μετατροπή των αρχείων gtfs που συλλέχθηκαν και τροποποιήθηκαν με τον τρόπου που αναλύθηκε στο προηγούμενο κεφάλαιο, σε μορφή γράφων. Οι δύο τύποι αυτοί είναι ο time-dependent graph builder και ο time-expanded graph builder.

Στην αρχή του κεφαλαίου αναλύεται το εννοιολογικό πλαίσιο σχετικά με την γενικότερη έννοια του γράφου και των δομικών του στοιχείων. Έπειτα, περιλαμβάνεται η παρουσίαση των κύριων κατηγοριών ταξινόμησης των γράφων, η οποίες εξαρτώνται από τα χαρακτηριστικά τους, και η περιγραφή των βασικών ιδιοτήτων τους. Στην συνέχεια, θα μας απασχολήσουν ζητήματα σχετικά με την δομή και την μορφή των δεδομένων τα οποία χρησιμοποιούνται για την αναπαράσταση ενός γράφου σε Η/Υ. Σε τελικό στάδιο θα εξεταστεί το πρόβλημα της μοντελοποίησης δικτύων με την βοήθεια γράφων μέσα από την παρουσίαση αντιπροσωπευτικών περιπτώσεων.

3.2 Βασικοί ορισμοί - Εννοιολογικό πλαίσιο των γράφων

Οι βάσεις της θεωρίας των γράφων ήρθαν στο προσκήνιο μέσω της επιστήμης των διακριτών μαθηματικών στις αρχές του 18ου αιώνα. Η συνεχής εξέλιξη και ανάπτυξη αυτού του πεδίου των μαθηματικών έκανε τον γράφο μία δομή, η οποία, όσο περνάει ο καιρός, χρησιμοποιείται από διάφορους επιστημονικούς κλάδους καθώς θεωρείται η πλέον κατάλληλη για την αναπαράσταση οποιουδήποτε είδους δικτύων και συστημάτων, τα στοιχεία των οποίων έχουν κάποιου είδους αλληλεπίδραση και σύνδεση.

Για την ιστορία της θεωρίας γράφων θεωρείται σημαντική η μελέτη του Λέοναρντ Όιλερ για τις Επτά Γέφυρες του Κένιγκσμπεργκ το 1736.Πρόκειται για πρόβλημα που τέθηκε το 1735 από τον Ελβετό μαθηματικό και φυσικό Leonard Euler, θέτοντας τις αρχές της θεωρίας Γράφων, και προοιωνίζοντας την ιδέα της Τοπολογίας. Η εργασία του δημοσιεύτηκε με τον τίτλο «Solutio problematis ad geometriam situs pertinentis» («Η επίλυση ενός προβλήματος σε

σχέση με την γεωμετρία της θέσης») το 1741, στο «Commentari i academia escientiarum Petropolitanae».

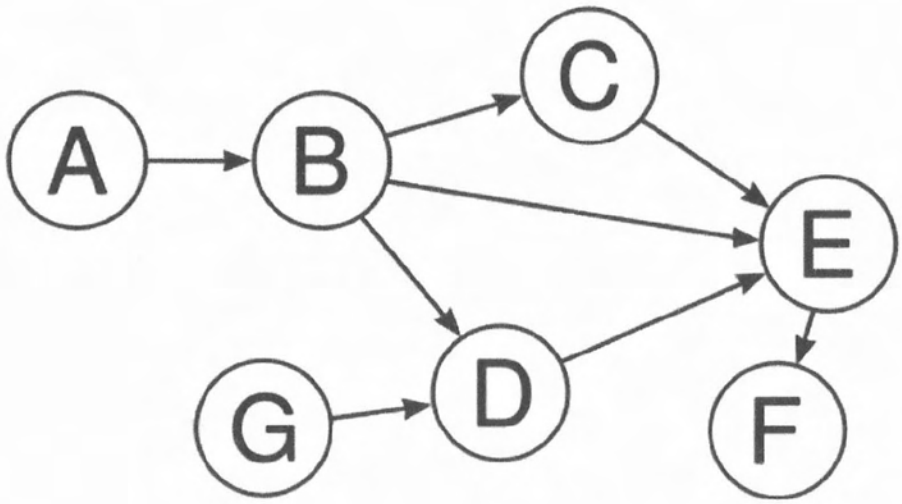
Είναι ένα από τα προβλήματα στα οποία ο Ελβετός μαθηματικός έδωσε λύση. Υπήρξε μια εποχή που ο Όιλερ ζούσε στο Κένιγκσπεργκ, που βρισκόταν σε πρωσικό έδαφος. Την πόλη διασχίζει ο ποταμός Πρέγκελ, που δημιουργεί δυο νησίδες στο κέντρο της πόλης. Οι κάτοικοι είχαν κατασκευάσει επτά γέφυρες για να υπάρχει συγκοινωνία με διάφορά μέρη. Το πρόβλημα αν μπορούσε κάποιος να περιηγηθεί στην πόλη, περνώντας από κάθε γέφυρα μία μόνο φορά και να επιστρέψει στο ίδιο σημείο από όπου είχε ξεκινήσει, ήταν ένας γρίφος που βασάνιζε για πολλά χρόνια τους κατοίκους. Το 1735, ήρθε ο Όιλερ να δώσει τη λύση, αποδεικνύοντας ότι κάτι τέτοιο ήταν αδύνατο. Στην απόδειξη του Όιλερ, σημασία έχει το δίκτυο των συνδέσεων μεταξύ των διαφόρων τμημάτων της πόλης και όχι η συγκεκριμένη θέση τους ή οι αποστάσεις μεταξύ τους. Ο χάρτης του Μετρό του Λονδίνου είναι ένα αντίστοιχο παράδειγμα. [1], [2]

Η φύση του προβλήματος μπορεί να χαρακτηριστεί παρόμοια με την φύση προβλημάτων που τίθενται σήμερα και σχετίζονται με δίκτυα και την επίλυση ζητημάτων που εντοπίζονται σε αυτά. Ενδεικτικά προβλήματα που σχετίζονται με τέτοιου είδους ζητήματα θεωρούνται, το πρόβλημα εύρεσης βέλτιστης διαδρομής μεταξύ δύο κόμβων ενός δικτύου, το πρόβλημα του πλανόδιου πωλητή, η αναζήτηση πιθανών διαδρομών που συνδέουν μεταξύ τους δύο ή περισσότερους κόμβους κ.α..Η μελέτη προβλημάτων που σχετίζονται με χωρικά αντικείμενα, το σύνολο των οποίων αποτελεί ένα δίκτυο βασίζεται στις σχέσεις συνδεσιμότητας που αναπτύσσονται μεταξύ των επιμέρους στοιχείων του δικτύου.

Το αντικείμενο της θεωρίας γράφων είναι η μελέτη των γράφων, μαθηματικών δομών δηλαδή, που χρησιμοποιούνται για την μοντελοποίηση και την αναπαράσταση των σχέσεων που υφίστανται μεταξύ ζευγών αντικειμένων. Σύμφωνα με τον ορισμό που προέρχεται από τα μαθηματικά, ένας γράφος G = (V,E) συνιστά μία αναπαράσταση ενός συνόλου αντικειμένων που συνδέονται μεταξύ τους. Τα αντικείμενα καλούνται κόμβοι ή κορυφές (vertices) του γράφου και συμβολίζονται με V και οι συνδέσεις που υφίστανται ανάμεσα σε ένα ζεύγος κόμβων ονομάζονται ακμές (edges) του γράφου και συμβολίζονται με E. Οι Aldous and Wilson ορίζουν το γράφο ως ένα διάγραμμα αποτελούμενο από σημεία που ονομάζονται κόμβοι, τα οποία συνδέονται μεταξύ τους με γραμμές που ονομάζονται ακμές, ενώ κάθε ακμή συνδέει ακριβώς δύο κόμβους. [16]

Κάθε ακμή του γράφου συνδέεται με ένα ζεύγος κόμβων που βρίσκονται στα άκρα της και οι οποίοι καλούνται άκρα της ακμής. Κάθε κόμβος του γράφου χαρακτηρίζεται από το βαθμό του, ο οποίος ορίζεται ίσος με τον αριθμό των ακμών που συνδέονται με το συγκεκριμένο κόμβο. Δύο ακμές που συνδέονται στον ίδιο κόμβο καλούνται γειτονικές ακμές, ενώ δύο κόμβοι που συνδέονται μεταξύ τους μέσω μίας κοινής ακμής καλούνται γειτονικοί κόμβοι αντίστοιχα. Το

σύνολο των ακμών ενός γράφου υπάρχει πιθανότητα να είναι κενό, ενώ το σύνολο των κόμβων είναι απαραιτήτως μη κενό. Οι ακμές ενός γράφου μπορεί να είναι κυκλικές (loops), δηλαδή να καταλήγουν στον ίδιο κόμβο από τον οποίο ξεκινούν. Στην περίπτωση που μία ακμή συνδέεται με έναν κόμβο, ο οποίος με τη σειρά του συνδέεται μέσω ακμής με κάποιο άλλο κόμβο του γράφου, καλείται τυφλή ακμή, ενώ μία ακμή, η οποία δε συνδέεται με άλλη ακμή και τα άκρα της είναι βαθμού ένα, καλείται αιωρούμενη ακμή. Ακμές που συνδέουν ακριβώς το ίδιο ζεύγος κόμβων καλούνται πολλαπλές ακμές. Τέλος, θα πρέπει να αναφερθεί ότι ένας κόμβος δύναται να ανήκει σε ένα γράφο, χωρίς απαραίτητα να ανήκει ταυτόχρονα σε κάποια από τις ακμές του γράφου. [16], [17]



Εικόνα 13: Σχηματική αναπαράσταση γράφου. [πηγή: el.wikipedia.org]

3.3 Τύποι γράφων

Οι γράφοι εμφανίζονται με διαφορετικούς βαθμούς γενικότητας μεταξύ τους, ανάλογα με την ιδιαίτερη δομή και τα χαρακτηριστικά τους. Έτσι λοιπόν, διαχωρίζονται σε επιμέρους κατηγορίες στη βάση ενός αριθμού κριτηρίων που τίθενται κατά περίπτωση, όπως για παράδειγμα η ανάθεση ή μη φοράς στις ακμές του γράφου, η συνδεσιμότητα του γράφου, η ανάθεση βαρών στις ακμές του γράφου κ.λπ.

Ανάλογα με την δομή και τα χαρακτηριστικά τους, οι γράφοι χρησιμοποιούνται για την μοντελοποίηση διαφορετικής φύσεως προβλημάτων. Η δομή ενός κατευθυνόμενου γράφου για παράδειγμα, κρίνεται καταλληλότερη για την μοντελοποίηση δικτύων ενώ η δομή ενός απλού μη κατευθυνόμενου γράφου, θεωρείται καταλληλότερη για την αναπαράσταση των χημικών δεσμών που υφίστανται μεταξύ των ατόμων ενός μορίου. Κάθε κατηγορία γράφων χαρακτηρίζεται από τον τρόπο διαγραμματικής αναπαράστασης των γράφων που ανήκουν σ'

αυτή, τα χαρακτηριστικά αυτών και τις ιδιότητες των δομικών στοιχείων των γράφων. Επίσης διαφοροποιούνται και βάσει των κανόνων που τίθενται από τα μαθηματικά που ορίζουν τους γράφους κάθε κατηγορίας.

Στην συνέχεια παρατίθενται οι βασικότεροι τύποι στους οποίους διακρίνονται οι γράφοι σύμφωνα με τον κύριο ορισμό τους.

Όπως προαναφέρθηκε, σε διάφορα πλαίσια μπορεί να είναι χρήσιμο να προσδιοριστεί ο όρος γράφος με διαφορετικούς βαθμούς γενικότητας. Οπότε είναι απαραίτητο να καθοριστεί μία αυστηρή διάκριση. Πιο συχνά, στα σύγχρονα κείμενα στην θεωρία γράφων, εκτός αν ορίζεται διαφορετικά, γράφος [2] σημαίνει "απλός, πεπερασμένος, μη κατευθυνόμενος γράφος".

Μη κατευθυνόμενος γράφος [2] είναι εκείνος στον οποίο οι ακμές δεν έχουν προσανατολισμό. Η ακμή (α,β) είναι ταυτόσημη με την άκρη (β,α), δηλαδή, δεν υπάρχουν διατεταγμένα ζεύγη, αλλά σύνολα {u,v} των κορυφών.

Κατευθυνόμενος γράφος (Directed Graph) [2] ή δίγραφος είναι ένα διατεταγμένο ζεύγος D = (V, A) όπου V, είναι ένα σύνολο του οποίου τα στοιχεία λέγονται κορυφές ή κόμβοι και A, είναι μία σειρά από διατεταγμένα ζεύγη κορυφών, τα οποία ονομάζονται τόξα, διατεταγμένες ακμές ή βέλη. Ένα τόξο a = (x, y) θεωρείται ότι κατευθύνεται από το x στο y. ονομάζεται η αρχή και x το τέλος του τόξου. y λέγεται ότι είναι άμεσος διάδοχος του x, και το x λέγεται ότι είναι ο άμεσος προκάτοχος του y. Αν ένα μονοπάτι οδηγεί από το x στο y, τότε το y λέγεται ότι είναι διάδοχος του x και προσβάσιμο από το x, και το x λέγεται ότι είναι προκάτοχος του y. Το τόξο (y, x) ονομάζεται (x, y) ανεστραμμένο.

Ένας κατευθυνόμενος γράφος D λέγεται συμμετρικός αν για κάθε τόξο στο D, το αντίστοιχο αντεστραμμένο τόξο ανήκει και αυτό στον D. Ένας συμμετρικός χωρίς επαναλήψεις κατευθυνόμενος γράφος D = (V, A) είναι ισοδύναμος με έναν απλό μη-κατευθυνόμενο γράφο G = (V, E), όπου τα ζευγάρια του αντίστροφου τόξου στο A αντιστοιχούν 1- προς 1 με τις ακμές στο E. Έτσι ο αριθμός |E| = |A|/2, ή το μισό του αριθμού των τόξων στο D. Μία παραλλαγή αυτού του ορισμού είναι ο προσανατολισμένος γράφος, στο οποίο δεν μπορούν να υπάρχουν παραπάνω από ένα από τα (x, y) και (y, x) τα οποία μπορούν να είναι τόξα.

Μικτός γράφος (Mixed Graph) [2] είναι ένας γράφος G στον οποίο μερικές ακμές μπορεί να είναι κατευθυνόμενες και κάποιες μπορεί να είναι μη κατευθυνόμενες. Ο γράφος είναι γραμμένος ως διατεταγμένος τριπλός G = (V, E, A) με V, Εκαι Αόπως ορίζεται ανωτέρω. Οι κατευθυνόμενοι και οι μη κατευθυνόμενοι γράφοι είναι ειδικές περιπτώσεις.

Πολύγραφος [2]. Ένας βρόγχος είναι μία ακμή (κατευθυνόμενη ή μη), η οποία αρχίζει και τελειώνει στην ίδια κορυφή. Αυτό μπορεί να επιτρέπεται ή να μην επιτρέπεται ανάλογα με την εφαρμογή. Στο πλαίσιο αυτό, μία ακμή με δύο διαφορετικές άκρες ονομάζεται μία σύνδεση. Ο

όρος «πολύγραφος» είναι γενικά αποδεκτό ότι εννοεί ότι επιτρέπονται πολλαπλές ακμές (και μερικές φορές βρόγχοι). Σε περίπτωση που οι γράφοι ορίζονται έτσι ώστε να επιτρέπουν βρόγχους και πολλαπλές ακμές, ένας πολύγραφος ορίζεται συχνά για να σημαίνει ένας γράφος χωρίς βρόγχους, ωστόσο, όπου οι γράφοι ορίζονται έτσι ώστε να μην επιτρέπονται βρόγχοι και πολλαπλές ακμές, ο όρος ορίζεται συχνά για να σημαίνει ένας «γράφος», ο οποίος μπορεί να έχει και πολλές ακμές και βρόγχους, αν και πολλοί χρησιμοποιούν τον όρο «ψευδογράφος» για αυτή την έννοια.

Απλός γράφος (Simple Graph) [2]. Σε αντίθεση με τον πολύγραφο, ένας απλός γράφος είναι ένα μη-κατευθυνόμενο γράφημα που δεν έχει βρόγχους και έχει όχι περισσότερες από μία ακμή ανάμεσα σε δύο διαφορετικές κορυφές. Σε ένα απλό γράφο οι ακμές του αποτελούν ένα σύνολο (και όχι πολυσύνολο) και κάθε ακμή είναι ένα ξεχωριστό ζευγάρι κορυφών. Σε ένα απλό γράφο με nκορυφές, κάθε κορυφή έχει ένα βαθμό που είναι μικρότερος από n(το αντίστροφο, όμως, δεν είναι αλήθεια - υπάρχουν και μη- απλοί γράφοι με nκορυφές στους οποίους κάθε κορυφή έχει βαθμό μικρότερο από το n).

Κάθε απλός γράφος είναι πολύγραφος. Όμως όλοι οι πολύγραφοι δεν είναι απλοί γράφοι.

Σταθμισμένος γράφος (Weighted Graph) [2]. Ένας γράφος είναι σταθμισμένος γράφος αν ένας αριθμός (βάρος) έχει ανατεθεί σε κάθε ακμή. Οι τιμές των βαρών θα μπορούσαν να αντιπροσωπεύουν, για παράδειγμα, κόστη, μήκη ή ικανότητες, κ.λπ. ανάλογα με το πρόβλημα κάθε φορά.

Τυπικός γράφος (Regular Graph) [2]Ένας τυπικός γράφος είναι ένας γράφος, όπου κάθε κορυφή έχει τον ίδιο αριθμό γειτόνων, δηλαδή, κάθε κορυφή έχει τον ίδιο βαθμό ή σθένος. Ένα τυπικός γράφος με κορυφές kβαθμού καλείται K-τυπικός γράφος ή τυπικός γράφος βαθμού k.

Πλήρες γράφος (Complete Graph) [2] καλείται ο γράφος όπου μεταξύ κάθε ζεύγους κόμβων υφίσταται οπωσδήποτε μία σύνδεση. Οι γράφοι αυτής της κατηγορίας, περιλαμβάνουν τον μέγιστο δυνατό αριθμό ακμών.

Κυκλικός γράφος (Cycle Graph)[2]. Ένας γράφος καλείται κυκλικός όταν αποτελείται από έναν κύκλο, δηλαδή έναν κόμβο και μία κυκλική ακμή όπου η αρχή και το πέρας της είναι ο μοναδικός κόμβος του γράφου. Επίσης, ένας κυκλικός κόμβος ορίζεται και ως μία κλειστή «αλυσίδα» ακμών που συνδέονται μεταξύ τους, ενώ ο κόμβος αφετηρίας της πρώτης ακμής είναι ο ίδιος με τον κόμβο πέρατος της τελευταίας ακμής.

3.4 Αναπαράσταση γράφων

Υπάρχουν διάφοροι τρόπου αναπαράστασης γράφων στον υπολογιστή, κάθε ένας με τα πλεονεκτήματά του και τα μειονεκτήματά του. Σε κάποιες περιπτώσεις ή αλγόριθμους που

θέλουμε να τρέξουμε με γράφους σαν είσοδο, προτιμάται η μία αναπαράσταση και σε άλλες περιπτώσεις άλλη. Η επιλογή εξαρτάται τις περισσότερες φορές από την δομή των δεδομένων σε συνάρτηση με το είδος του προβλήματος που επιζητά επίλυση.

Γενικά όμως υπάρχουν δύο καθιερωμένοι τρόποι αναπαράστασης ενός στατικού γράφου G = (V, E) και είναι οι εξής:

- υπό την μορφή μιας συλλογής από λίστες γειτνίασης και
- υπό την μορφή ενός πίνακα γειτνίασης.

Και οι δύο αναπαραστάσεις μπορούν να εφαρμοστούν τόσο σε κατευθυνόμενους όσο και σε μη κατευθυνόμενους γράφους.

Η αναπαράσταση μέσω λιστών γειτνίασης (adjacency list) ενός γράφου G = (V, E) συνίσταται σε μία συστοιχία Adj από |V| καταλόγους, έναν για κάθε κόμβο του V. Για κάθε u □V, ο κατάλογος γειτνίασης Adj [u] περιέχει όλους τους κόμβους v για τους οποίους υπάρχει ακμή (u, v)□ E. Δηλαδή, ο Adj [u] αποτελείται από όλους τους κόμβους που γειτνιάζουν με τον u στο G. Η διάταξη των κόμβων σε κάθε κατάλογο γειτνίασης είναι κατά κανόνα αυθαίρετη. Στις παρακάτω εικόνες που παρατίθενται φαίνεται αρχικά η αναπαράσταση ενός μη κατευθυνόμενου γράφου μέσω καταλόγων γειτνίασης αρχικά και έπειτα η αναπαράσταση ενός κατευθυνόμενου γράφου μέσω των καταλόγων γειτνίασης.

Στην περίπτωση ενός κατευθυνόμενου γράφου με βάρη, η λίστα ενός κόμβου περιλαμβάνει το σύνολο των γειτονικών του κόμβων, που είναι προσβάσιμοι από τον αρχικό, συνοδευόμενων από το βάρος που αντιστοιχεί στην εκάστοτε παρεμβαλλόμενη ακμή.

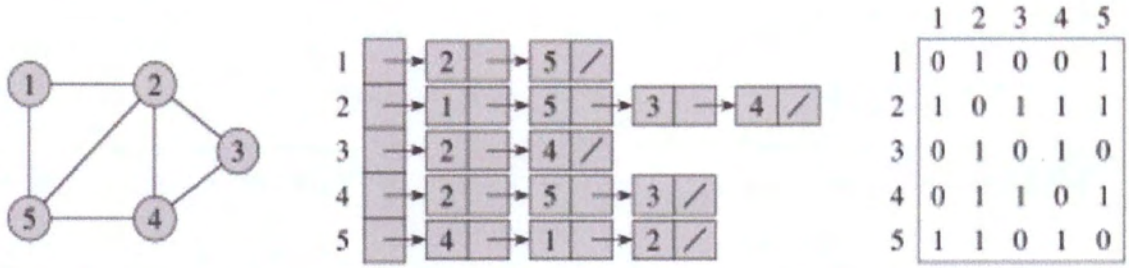
Ο πρώτος τρόπος αναπαράστασης χρησιμοποιείται συνήθως για την αποθήκευση αραιών και σχετικά μικρών γράφων (sparse graphs), όπου ο αριθμός των ακμών τους |E| είναι κατά πολύ μικρότερος από τον αριθμός του τετραγώνου των κόμβων τους $|V^2|$. [18], [19]

Ο δυσδιάστατος πίνακας συνιστά την απλούστερη δομή που χρησιμοποιείται για την αναπαράσταση ενός γράφου. Οι γραμμές και οι στήλες του πίνακα περιλαμβάνουν τους κόμβους του γράφου, κάθε κόμβος δηλαδή περιγράφεται από μία γραμμή και μία στήλη. Στην περίπτωση ενός μη - κατευθυνόμενου γράφου, τα κελιά του πίνακα παίρνουν τιμές 1 ή 0, ανάλογα με το εάν υφίσταται ή όχι σύνδεση μεταξύ δύο κόμβων, του κόμβου γραμμής και του κόμβου στήλης. Ο πίνακας στην περίπτωση αυτή είναι συμμετρικός ως προς τα στοιχεία που περιλαμβάνονται στην κύρια διαγώνιο και είναι ίσα με 1. Η ίδια λογική ακολουθείται και κατά τη διαδικασία αναπαράστασης ενός κατευθυνόμενου γράφου, με τη διαφορά ότι στους κόμβους που περιλαμβάνονται στις γραμμές του πίνακα ανατίθεται ο ρόλος «από», ενώ στους κόμβους που περιλαμβάνονται στις στήλες του πίνακας ανατίθεται ο ρόλος «προς». Στην περίπτωση

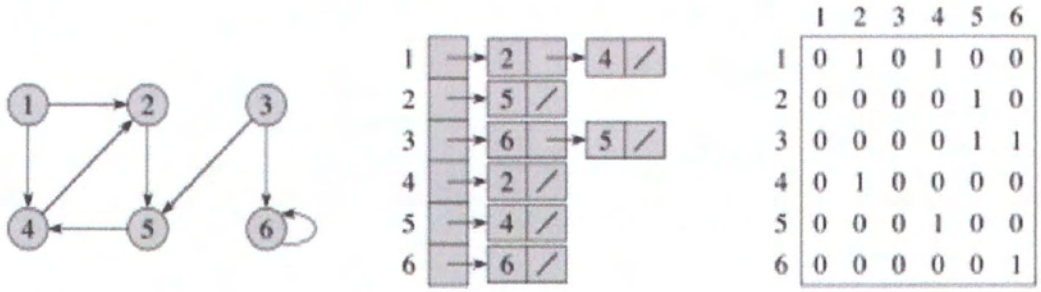
αναπαράστασης σταθμισμένου γράφου με δυδιάστατο πίνακα, τα πεδία του πίνακα περιλαμβάνουν τα βάρη των αντίστοιχων ακμών, ενώ στην περίπτωση που μεταξύ δύο κόμβων δεν υφίσταται ακμή, στο αντίστοιχο κελί του πίνακα ανατίθεται τιμή ίση με το (∞).

Ο δεύτερος αυτός τρόπος αναπαράστασης χρησιμοποιείται συνήθως για την αναπαράσταση πυκνών γράφων όπου ο αριθμός των ακμών |E| του γράφου είναι περίπου ίσος με τον αριθμός του τετραγώνου των κόμβων του $|V^2|$.[17], [19], [20]

Στις παρακάτω εικόνες παρατηρούμε τους πίνακες γειτνίασης ενός μη κατευθυνόμενου και ενός κατευθυνόμενου γράφου.



Εικόνα 14: Αναπαράσταση ενός μη κατευθυνόμενου γράφου μέσω καταλόγων γειτνίασης. [πηγή:www.hawaii.edu]



Εικόνα 15: Αναπαράσταση ενός κατευθυνόμενου γράφου μέσω καταλόγων γειτνίασης. [πηγή:www.hawaii.edu]

Αξίζει να σημειώσουμε ότι με μία μικρή τροποποίηση του καταλόγου και του πίνακα γειτνίασης, οι συγκεκριμένες μέθοδοι μπορούν να χρησιμοποιηθούν και για την αναπαράσταση εμβαρών γράφων. Το βάρος w (u,v) της ακμής (u,v) □ E μπορεί απλώς να αποθηκευτεί μαζί με τον κόμβο v στον κατάλογο γειτνίασης του u.

Επίσης, όταν απαιτείται απάντηση σε μικρό χρονικό διάστημα σε ερωτήματα του τύπου εάν υπάρχει μία ακμή που να συνδέει δύο οποιουσδήποτε κόμβους ενός γράφου, ο δεύτερος τρόπος αναπαράστασης (πίνακας) είναι προτιμότερος, κυρίως ως δομή καθώς εγγυάται ταχύτερο "σκανάρισμα" των στοιχείων που περιέχονται σε αυτόν, σε αντίθεση με τη λίστα, η οποία

πρέπει να προσπελάσει τα δεδομένα αυτής ένα προς ένα ώστε να δώσει απάντηση στο ερώτημα που έχει ζητηθεί.

Η αναπαράσταση των εξελισσόμενων γράφων γίνεται με διαφορετική λογική από ότι οι στατικοί. Τα δεδομένα και τα στοιχεία των εξελισσόμενων γράφων δύναται να τροποποιηθούν όσο περνάει ο χρόνος. Υπάρχει πιθανότητα δηλαδή να προστεθούν ή να αφαιρεθούν στοιχεία (όπως είναι κόμβοι ή ακμές). Για την αναπαράσταση των εξελισσόμενων γράφων, εν αντιθέσει με τους στατικούς, γίνεται μοντελοποίηση διαφορετικών καταστάσεων στα επιμέρους χρονικά βήματα που ορίζονται, τα οποία συνήθως εξαρτώνται από την συχνότητα που λαμβάνονται δείγματα από τα στοιχεία. Μέχρι στιγμής έχουν οριστεί δύο μοντέλα αναπαράστασης εξελισσόμενων γράφων (Giatsoglou & Vakali, 2012), το Στιγμιότυπο Γράφου (Graph Snapshot) και η Ροή Αλλαγών(Change Stream).

3.5 Εισαγωγή στην έννοια του Graph builder

Ένα δίκτυο μπορεί να κατασκευαστεί σε μοντέλο με την χρήση ενός απλού γράφου που αποτελείται από κορυφές που παριστάνουν τις στάσεις στην περίπτωσή μας και από μη κατευθυνόμενες ακμές που παριστάνουν τις διαδρομές. Στην περίπτωση του απλού γράφου κάθε ακμή συνδέει δύο ξεχωριστές κορυφές και δεν υπάρχουν δύο ακμές που να συνδέουν το ίδιο ζεύγος κορυφών.

Όταν υπάρχουν πολλαπλές επιλογές για τη μεταφορά μεταξύ δύο σημείων του δικτύου, τότε ίσως υπάρχουν πολλαπλές ακμές μεταξύ των κορυφών αυτών. Δηλαδή αντίστοιχα στην περίπτωσή μας, πολλές διαδρομές για το ίδιο ζεύγος στάσεων. Οι απλοί γράφοι δεν αρκούν για την κατασκευή μοντέλων δικτύων μέσων μαζικής μεταφοράς. Αντίθετα χρησιμοποιούνται πολυγράφοι, που αποτελούνται από κορυφές και από κατευθυνόμενες ακμές μεταξύ αυτών των κορυφών, όπου επιτρέπονται πολλαπλές ακμές μεταξύ ζευγών κορυφών.

Στην περίπτωση που εξετάζεται από την παρούσα διπλωματική εργασία το ίδιο ζεύγος κορυφών μπορεί να ενώνεται με παραπάνω από μία ακμές. Για τον λόγο αυτό θα χρησιμοποιηθεί ο τύπος πολύγραφου.

Στις μέρες μας, οι πλατφόρμες και οι διαδικτυακές εφαρμογές επιβάλλουν το συστηματοποιημένο «χτίσιμο» των δικτύων (στην περίπτωσή μας δίκτυο μαζικής μεταφοράς) στην μνήμη του υπολογιστή. Για αυτό τον σκοπό έχουν αναπτυχθεί τα προγράμματα λογισμικού graphbuilders, βασική λειτουργία των οποίων είναι να παίρνουνε ως είσοδο δεδομένα, να τα επεξεργάζονται και να δημιουργούνε ψηφιακά αντικείμενα στην μνήμη του υπολογιστή, τα οποία έχουν τις ιδιότητες των γράφων. Όπως έχουμε αναφερθεί με αυτό τον τρόπο μπορούμε να καλούμε προγραμματιστικές μεθόδους και λειτουργίες, όπως και να κάνουμε υπολογισμούς πολύ πιο γρήγορα και εύκολα.

Ανάλογα με τονgraph builder κάθε φορά, ο παραγόμενος γράφος έχει διαφορετικές ιδιότητες. Από την μία υπάρχουνε graph builders οι οποίοι θέλουνε πολύ χρόνο για να επεξεργαστούν τα δεδομένα, αλλά παράγουνε δομές δεδομένων πάνω στις οποίες μπορούμε να τρέξουμε πολύ γρήγορους αλγόριθμους. Από την άλλη υπάρχουν άλλες κατηγορίες από graph builders, οι οποίοι κατασκευάζου τις παραγόμενες δομές δεδομένων πιο γρήγορα, αλλά και παράλληλα πιο «πρόχειρα», με την έννοια ότι δεν μας επιτρέπουν να τρέξουμε σύγχρονους και γρήγορους αλγόριθμους.

Μία ακόμα ιδιότητα που μπορεί να έχει ένας graph builder είναι η γλώσσα προγραμματισμού στην οποία έχει υλοποιηθεί. Κάποιες γλώσσες είναι καταλληλότερες για τη συγκεκριμένη ενέργεια αφού μας επιτρέπουν να αποθηκεύουμε τους γράφους πιο εύκολα ή να επεκτείνουμε τις δυνατότητες τους πιο απλά και αξιόπιστα,.

Στη συνέχεια, θα αναφερθούμε στον τρόπο με τον οποίο προσδώσαμε ανάλογες ιδιότητες στους δικούς μας graph builders.

Αρχικά, επιλέξαμε να χρησιμοποιήσουμε τη γλώσσα προγραμματισμού JAVA. Η γλώσσα προγραμματισμού JAVAείναι σχεδιασμένη, έτσι ώστε να δημιουργηθεί μία γλώσσα που θα ήταν ανεξάρτητης πλατφόρμας, δηλαδή θα αναπτύσσει εφαρμογές που θα «τρέχουν» εύκολα παντού.Ένα από τα βασικά χαρακτηριστικά που μας οδήγησε στην επιλογή της γλώσσας αυτής, είναι ότι η συγκεκριμένη έναντι της πλειοψηφίας των άλλων γλωσσών είναι ανεξάρτητη από άποψη λειτουργικού συστήματος και πλατφόρμας.

Στη συνέχεια αξίζει να αναφερθούμε στο γεγονός ότι επιδιώξαμε να υλοποιήσουμε δύο τύπους graphbuildersμε σκοπό να τους συγκρίνουμε ως προς τις επιδόσεις τους. Αυτοί οι τύποι είναι οι time-expanded και time-dependent graph builders.

Όσον αφορά τους πρώτους, κύριο χαρακτηριστικό τους είναι ότι μοντελοποιούν το γράφο σύμφωνα με τα γεγονότα ή events, όπως θα αναφερόμαστε σε αυτά και στη συνέχεια. Δηλαδή, οι κόμβοι του γράφου δεν αντιπροσωπεύουνε γεωγραφικά σημεία, αλλά γεγονότα στο χρόνο. Για τους συνδέσμους μεταξύ των κόμβων αυτών, ως βάρη χρησιμοποιήθηκαν ο χρόνος που απαιτείται για την διαδοχική εκτέλεση των γεγονότων και την διαδοχική πρόσβαση στους επιμέρους κόμβους. Απεναντίας το time-dependent graph μοντελοποιεί τους κόμβους ως χωρικά σημεία και τα βάρη που ορίστηκαν στις ακμές των γράφων αυτών είναι οι αποστάσεις μεταξύ των διαδοχικών κόμβων που ενώνουν οι επιμέρους ακμές.

Στο πρόβλημα του προγραμματισμού που απασχολεί την παρούσα διπλωματική εργασία, ίσως η μεγαλύτερη πρόκληση είναι η διαμόρφωση της μορφής του χρονοδιαγράμματος και των δεδομένων, προκειμένου να καταστεί δυνατή η χρησιμοποίηση αυτών και κατ' επέκταση ο υπολογισμός της βέλτιστης διαδρομής από τον αλγόριθμο.

Εν κατακλείδι, στην περίπτωσή μας ο graph builder είναι ένας κώδικας σε γλώσσα προγραμματισμού JAVA, ο οποίος θα μετατρέπει την ψηφιοποιημένη μορφή των δεδομένων, μορφή gtfs, σε μορφή που θα είναι κατάλληλη για χρήση από τον αλγόριθμο, μορφή γράφου, που θα αναπτυχθεί σε επόμενα κεφάλαια.

Να σημειωθεί ότι στην βιβλιογραφία θα δούμε αναφορές στα δεδομένα που παίρνει ως είσοδο ο graph builder, που αφορούν αποκλειστικά τα χρονοδιαγράμματα. Αυτό συμβαίνει γιατί μία κοινή πρακτική που ακολουθείται από πολλούς οργανισμούς είναι η συλλογή μικρού εύρους γεωχωρικών δεδομένων και παράλληλα μεγαλύτερη έμφαση στην συλλογική επεξεργασία δεδομένων που αφορούν τα χρονοδιαγράμματα. Στην περίπτωσή μας δόθηκε εξίσου ανάλογη βάση σε όλα τα δεδομένα, δεδομένα όπως είναι οι στάσεις, οι γραμμές, αλλά και τα χρονοδιαγράμματα.

Ο αλγόριθμος που θα χρησιμοποιηθεί στην δική μας περίπτωση, όπως θα αναλυθεί και σε επόμενα κεφάλαια χρειάζεται ως δεδομένα εισόδου γράφους τύπου G = (V, A) οι οποίοι θα αντιπροσωπεύουν τα χρονοδιαγράμματα και γενικότερα όλα τα δεδομένα. Στις επόμενες ενότητες θα εξεταστούν δύο προσεγγίσεις, οι οποίες μπορούν να κάνουν την μετατροπή αυτή (συγκεκριμένα την μετατροπή από αρχεία τύπου gtfs σε γράφους). Οι δύο προσεγγίσεις αυτές αποτελούν τους δύο graph builders της διπλωματικής εργασίας και όπως αναφέρθηκε και παραπάνω είναι ο time-expanded και time-dependent graph builder.

3.6 Ο time-expanded Graph builder

Βασιζόμενοι γενικά στην λογική ότι κάθε ένα δεδομένο (χρονοδιάγραμμα, συντεταγμένες στάσης, γραμμής), στην πράξη αποτελεί συμβάν/ γεγονός(π.χ. ένα όχημα φτάνει ή φεύγει σε μία στάση αποτελεί ένα χρονικό συμβάν) που εξαρτάται από τον χρόνο και συμβαίνει σε συγκεκριμένα χρονικά σημεία, η ιδέα του time-expanded τύπου [9] είναι να χτίσει ένα γράφο χώρου-χρόνου, που συχνά ονομάζεται και γράφος συμβάντος (event graph), και ο οποίος θα ¨ξετυλίγει¨ στον χρόνο. Σε γενικές γραμμές, ο τύπος αυτός δημιουργεί έναν κόμβο για κάθε περίπτωση δεδομένου και χρησιμοποιεί τόξα για να συνδέσει στα επόμενα γεγονότα, τα οποία βρίσκονται στην κατεύθυνση που ορίζει η ροή του χρόνου. Για παράδειγμα το στοιχείο μίας στάσης συμπεριλαμβάνει τα συμβάντα της άφιξης στην στάση, της αναμονής στην στάση και της αναχώρησης από την στάση. Σύμφωνα με το πρότυπο time-expanded δημιουργείται ένας κόμβος για κάθε ένα από αυτά τα συμβάντα (3 στο σύνολο), τα οποία ενώνονται με τόξα μεταξύ τους.

Μία βασική εκδοχή του τύπου, που είναι και αυτή η οποία αναπτύσσουμε στη συνέχεια, περιέχει έναν κόμβο για κάθε γεγονός αναχώρησης, άφιξης και αναμονής, τα οποία συνδέονται μεταξύ τους με τόξα, στα οποία αντιστοιχεί και συγκεκριμένο βάρος. Στη συνέχεια, και πριν

αναφερθούμε στο τρόπο με τον οποίο παράγονται αυτά τα τόξα, θα ήταν σκόπιμο να αναφερθούμε επιγραμματικά σε μερικά σημαντικά στοιχεία γύρω από τους κόμβους και πως αυτοί συνδέονται με τα γεγονότα στο χρόνο. Παίρνοντας ως δεδομένο ότι κάθε δρομολόγιο των αρχείων GTFS αποτελείται από υπογεγονότα, όπως αφίξεις, αναχωρήσεις και αναμονές στους εκάστοτε κόμβους, πρέπει αν δημιουργήσουμε τα κατάλληλα αντικείμενα μέσα στον γράφο έτσι ώστε να αντιπροσωπεύουμε όσο το δυνατόν πιο παραστατικά και αποτελεσματικά αυτές τις πληροφορίες. Ειδικότερα, κάθε 'πέρασμα' του δρομολογίου από κάθε στάση ανάγεται στις τρεις αυτές επιλογές (και πράξεις) που έχει και μπορεί να πραγματοποιήσει ο επιβάτης. Δηλαδή, παίρνοντας μία γραμμή, μπορεί πάντα με αρχή το ότι υπάρχει η άφιξη στη στάση, μπορεί είτε να συνεχίσει στο ίδιο δρομολόγιο, είτε να περιμένει στον κόμβο για να αλλάξει δρομολόγιο. Έτσι λοιπόν για κάθε φορά που περνά ο χρήστης/επιβάτης από μία στάση εμείς δημιουργούμε τον κόμβο της άφιξης και τον συνδέουμε τον κόμβο της αναχώρησης όταν συνεχίζει με το ίδιο δρομολόγιο, ή με τον κόμβο της αλλαγής δρομολογίου και της αναγκαστικής αναμονής έστω για δύο λεπτά (μέσος χρόνος αποβίβασης και επιβίβασης). Επίσης, έμφαση δίνεται στους κόμβους αναμονής και τη διασύνδεσή τους με άλλα γεγονότα που επακολουθούν στο χρόνο. Η σύνδεση αυτή μεταξύ των κόμβων αναμονής, είναι αυτή που μας επιτρέπει τη μεταφορά από ένα μέσο ή ένα δρομολόγιο σε ένα άλλο.

Όσον αφορά τα βάρη των τόξων, για τη μεταφορά από κόμβο άφιξης κάθε γεγονότος προς κόμβο αναχώρησης το κόστος είναι μηδέν. Αυτό συμβαίνει γιατί συνήθως κάθε γεγονός αφορά πάντα ένα δρομολόγιο και ο χρήστης δεν χρειάζεται να ξοδέψει χρόνο για αυτή τη μετάβαση. Σχετικά, τώρα, με τη μεταφορά από κόμβο άφιξης προς κόμβο αναμονής, αναθέτουμε ένα κόστος 2 λεπτών, όπου λειτουργεί ως ένα μέσο χρόνο μετάβασης από ένα όχημα ενός δρομολογίου σε ένα άλλο. Τέλος, στα time-expanded graphs, συνδέουμε κάθε κόμβο αναμονής με τον επόμενο κόμβο αναμονής στο χρόνο, και ασφαλώς στην ίδια στάση, με το επόμενο κόμβο αναχώρησης στον ίδια στάση και τελικώς τον κάθε κόμβο αναχώρησης με τον επόμενο αντίστοιχο κόμβο άφιξης όχι στο χρόνο, αλλά ανάλογα με το δρομολόγιο στο οποίο ανήκει και στη σειρά των στάσεων του δρομολογίου αυτού.

Μερικά στοιχεία από τη βιβλιογραφία που βρήκαμε και στα οποία πρέπει να αναφερθούμε είναι ότι οι Müller-Hannemann και Weihe επέκτειναν το μοντέλο για τον διαχωρισμό των τρένων (για να βελτιστοποιήσουν τον αριθμό των μεταβιβάσεων που λαμβάνονται κατά την διάρκεια ερωτήσεων στην βάση δεδομένων), με υποδιαίρεση του κάθε τόξου σύνδεσης με ένα νέο κόμβο και στην συνέχεια με την διασύνδεση όλων των κόμβων του κάθε ταξιδιού.

Επίσης οι Pyrga e tal. Και Müller-Hannemann and Schnee επέκτειναν ακόμα περισσότερο τονtime-expandedτύπο, ώστε να ενσωματώσει τις ελάχιστες δυνατές χρονικές αλλαγές (που

δίνονται από τα δεδομένα εισόδου) και οι οποίες παίζουν το ρόλο του ρυθμιστικού χρόνου (buffer time), κατά την αλλαγή ενός δρομολογίου ή και μέσου σε ένα σταθμό.

Ο τύπος που αναλύθηκε αποκαλείται «ρεαλιστικό μοντέλο» και γενικά έχει εισάγει έναν πρόσθετο κόμβο μεταφοράς ανά συμβάν αναχώρησης και συνδέει κάθε κόμβο άφιξης με τον πρώτο κόμβο μεταφοράς, ο οποίος υπακούει στον περιορισμό για την ελάχιστη χρονική μεταβολή.

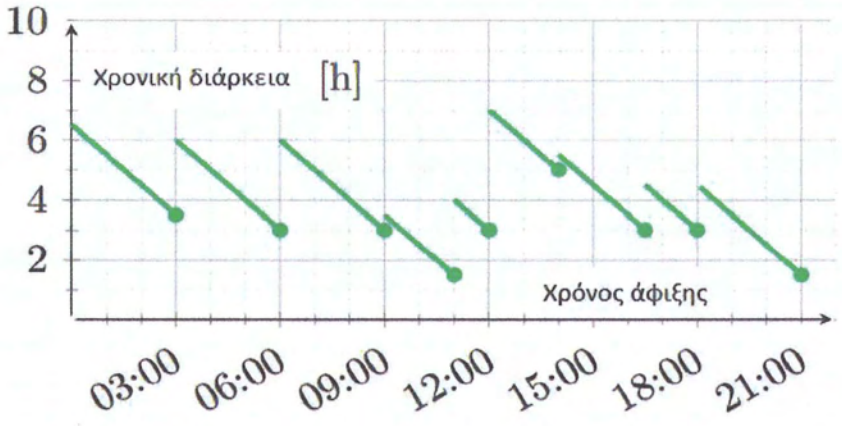
3.7 Ο time-dependent Graph builder

Το κύριο χαρακτηριστικό, το οποίο παράλληλα κρίνεται ως το βασικό μειονέκτημα του προηγούμενου τύπου των graph builders, είναι ότι τα αποτελέσματα που προκύπτουν από τύπους time-expanded graph builders είναι αρκετά μεγάλου μεγέθους γράφοι. Αντίθετα, βάσει της προσέγγισης του τύπου time-dependent [9], παράγονται σημαντικά μικρότεροι γράφοι (όσον αφορά τον αριθμό των κορυφών και των τόξων). Αυτό επιτυγχάνεται με την επιλογή να μην ξετυλίγεται το χρονοδιάγραμμα. Αντί αυτού, οι εξαρτήσεις χρόνου κωδικοποιούνται από κάποιες συναρτήσεις χρόνου που προσδιορίζουν το κάθε τόξο, το άθροισμα των οποίων υπολογίζει την συνολική χρονική διάρκεια ταξιδιού για κάθε ταξίδι.

Αξιολογώντας το κόστος ενός τόξου στη συνέχεια, αυτό εξαρτάται από τον χρόνο στον οποίο διασχίζεται. Αποδεικνύεται ότι προβλήματα παρόμοιας φύσεως με αυτό που ασχολείται η παρούσα διπλωματική εργασία, μπορούν να λυθούν αποτελεσματικά αν οι συναρτήσεις που υπολογίζουν την χρονική διάρκεια είναι μη αρνητικές.

Στην παρούσα διπλωματική εργασία και όπως έχει επεξηγηθεί σε προηγούμενα κεφάλαια εξετάζονται οι μαζικές μετακινήσεις και μεταφορές. Οι τύποι time-dependent graph builders παράγουν γράφους, στα μοντέλα των οποίων οι κόμβοι αναπαριστούν τις στάσεις και ένα τόξο προστίθεται μεταξύ δύο στάσεων, εάν υπάρχει τουλάχιστον μία στοιχειώδης σύνδεση, η οποία εξυπηρετεί με οποιονδήποτε τρόπο τις αντίστοιχες στάσεις.

Τα ακριβή χρονικά σημεία μεταξύ των αφίξεων και των αναχωρήσεων μεταξύ δύο κόμβων, βρίσκονται κωδικοποιημένα από την συνάρτηση της χρονικής διάρκειας που αναλύθηκε παραπάνω. Η συνάρτηση αυτή συνδέεται πλήρως με το αντίστοιχο τόξο μεταξύ των δύο στάσεων. Παράδειγμα συνάρτησης διάρκειας ταξιδιού παρουσιάζεται στην εικόνα που παρατίθεται παρακάτω.

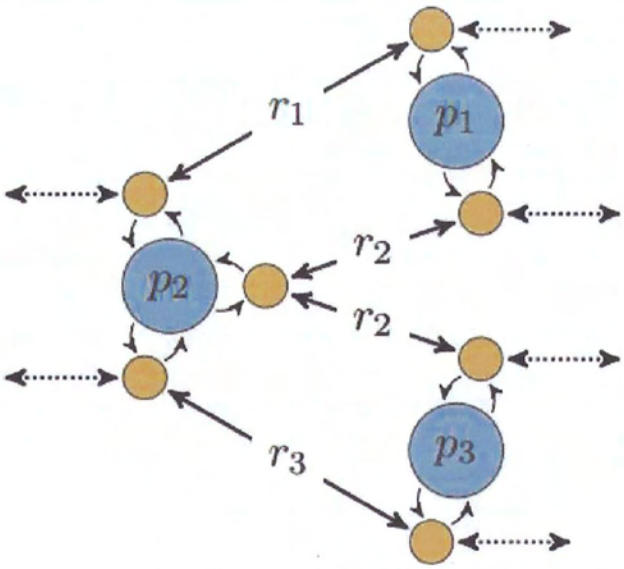


Εικόνα 16: Συνάρτηση χρονικής διάρκειας άφιξης σε κόμβο στην περίπτωση παραγωγής γράφου από time-dependent graph builder [πηγή: Route Planning in Transportation Networks, Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthiaw Muller - Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck, January 8, 2014]

Επόμενη εξέλιξη των time dependent graph builders ήταν η επέκταση του βασικού μοντέλου, με τρόπο τέτοιο που είχε ως σκοπό να καταστεί ικανός ο υπολογισμός των μικρότερων χρονικών αλλαγών. Αυτό επετεύχθη δημιουργώντας για κάθε στάση p και κάθε γραμμή που εξυπηρετεί την στάση p, ένα σύνολο κόμβων αφιερωμένων στην p. Οι κόμβοι αυτοί είναι συνδεδεμένοι σε ένα κοινό σύνολο από τόξα, τα οποία είναι προσδιορισμένα με σταθερά κόστη που απεικονίζουν την ελάχιστη τιμή χρονικής διάρκειας για την στάση p.

Τα τόξα γραμμών συνδέονται με τις κορυφές της γραμμής που επακολουθούν. Η σκιαγράφηση των ταξιδιών αποτελείται από τόξα που ενώνουν τις κορυφές που επακολουθούν η μία την άλλη.

Στην εικόνα που παρατίθεται παρακάτω παρουσιάζεται ο τρόπος ανάπτυξης και σύνδεσης των δεδομένων που έχουν τροποποιηθεί με graph builder τύπου time-expanded. Όπως παρατηρεί κανείς με αυτό τον τύπο graph builder στον γράφο που δημιουργείται κάθε στάση, έστω η στάση p_1, δημιουργεί κόμβους (ή κορυφές), τόσους όσοι είναι και οι πιθανοί κόμβοι με τους οποίους μπορεί να συνδεθεί η συγκεκριμένη στάση. Στην περίπτωση της εικόνας οι πιθανοί κόμβοι που μπορεί να συνδεθεί η στάση p_1, άρα και σε συνέπεια οι κόμβοι που δημιουργούνται για την συγκεκριμένη στάση από τον graph builder, είναι τρείς (3). Επίσης παρατηρείται αυτό που αναφέρθηκε και παραπάνω, ότι κάθε κόμβος που δημιουργείται ενώνεται με αντίστοιχο τόξο, το οποίο χαρακτηρίζεται με το αντίστοιχο βάρος, στην περίπτωσή μας το βάρος ισούται με την χρονική διάρκεια μετάβασης.



Εικόνα 17: Μοντέλο γράφου που προκύπτει από time-dependent graph builder [πηγή: Route Planning in Transportation Networks, Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthiaw Muller - Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck, January 8, 2014]

Τέλος αξίζει να σημειωθεί, ότι αντίστοιχα με τον time-expanded graph builder, όπου αναφέρεται ως δεδομένο εισόδου η στάση (π.χ. η στάση p_1), εννοείται το αντίστοιχο αρχείο μορφής gtfs.

Επίσης, έχει αναπτυχθεί μοντέλο του τύπου time-expanded graph builder, ο οποίος θα μπορεί να εξετάσει περίπτωση που επιτρέπει αυθαίρετη και συνεχή αλλαγή των χρονικών διαρκειών μεταξύ δύο κόμβων που αντιπροσωπεύουν δύο στάσεις.

3.8 Σύνοψη κεφαλαίου

Στα πλαίσια των εργασιών που αναλύθηκαν στο παρόν κεφάλαιο έλαβε χώρα η μετατροπή των ψηφιοποιημένων δεδομένων που παρουσιάστηκαν στο 2° κεφάλαιο σε μορφή γράφων. Η εργασία αυτή, όπως αναφέρθηκε και παραπάνω είχε στόχο την δημιουργία γράφων για τον εξής λόγο: Η μορφή των γράφων είναι αυτή η μορφή στην οποία τα δεδομένα που συλλέχθηκαν μπορούν να αποτελέσουν δεδομένα εισόδου για τον αλγόριθμο που θα εξεταστεί σε επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 4ο: Παρουσίαση Προβλήματος και Αλγορίθμου Υπολογισμού της Λύσης του

4.1 Εισαγωγή

Στο κεφάλαιο που ακολουθεί λαμβάνει χώρα η παρουσίαση του προβλήματος που απασχόλησε την παρούσα διπλωματική εργασία και δεν είναι άλλο από το πρόβλημα σχεδιασμού ταξιδιού (journey planning problem). Επίσης γίνεται αναφορά και ανάλυση του αλγορίθμου επίλυσης του προβλήματος, καθώς και όλα τα σχετικά με την δομή αυτού. Να σημειωθεί για ακόμη μία φορά ότι η πτυχιακή εργασία δεν αφορά την εφαρμογή καλύτερου αλγορίθμου πάνω στα δεδομένα, αλλά τον καλύτερο τρόπο με τον οποίο αυτά μπορούν να παρασταθούν στη μνήμη του υπολογιστή (τον καλύτερο τύπο graph builder).

Για αυτό, λοιπόν, διερευνήσαμε πολλαπλούς αλγορίθμου και επιλέξαμε ένα από αυτούς για χρήση με τα δεδομένα μας. Τα κριτήρια μας για αυτή την επιλογή ήταν δύο, πόσο διαδομένος και αναγνωρίσιμος είναι ο αλγόριθμος στη βιβλιογραφία και δεύτερον πόσο εύκολο ήταν να τον βρούμε και να τον συνδυάσουμε με τον ψευδοκώδικα που γράψαμε για τη σύνθεση των δεδομένων.

4.2 Η έννοια του προβλήματος βελτιστοποίησης

Την λέξη πρόβλημα την συναντάμε πολύ συχνά στην καθημερινότητά μας και σε όλους τους τομείς που μπορεί να μας απασχολούν. Καθημερινά λοιπόν βρίσκουμε λύσεις για προβλήματα σε απλά θέματα, αλλά και για πιο σύνθετα. Πολλά προβλήματα αποπερατώνονται σύντομα και εύκολα, άλλα θεωρούνται πιο δύσκολα στο να επιλυθούν και πιο πολύπλοκα. Επίσης υπάρχουν και προβλήματα που βάσει των γνώσεων που έχουν αναπτυχθεί δεν δύναται να επιλυθούν, αλλά και προβλήματα που έχει αποδειχθεί ότι δεν έχουν λύση.

Αρχικά για τον προσδιορισμό και έπειτα την επίλυση ενός προβλήματος, όλα ξεκινάνε από την συλλογή και την καταγραφή των δεδομένων. Επίσης ο προσδιορισμός του προβλήματος εξαρτάται από τον ορισμό των ζητουμένων. Τα ζητούμενα είναι η απάντηση στο "ερώτημα" που έχει τεθεί από το πρόβλημα. Τέλος η διαδικασία μέσω της οποίας δίνεται η απάντηση στα επιμέρους προβλήματα που συναντώνται ονομάζεται επίλυση του προβλήματος.

Οι δομές του προβλήματος βελτιστοποίησης που απασχόλησε την παρούσα διπλωματική εργασία, δηλαδή συλλογή και καταγραφή των δεδομένων (δίκτυα Μ.Μ.Μ.), ορισμός των ζητουμένων και το ερώτημα που θα τεθεί στο πρόβλημα (πως θα επιτευχθεί πιο γρήγορα η επιλογή της βέλτιστης διαδρομής μεταξύ δύο στάσεων), παρουσιάστηκαν και αναλύθηκαν σε προηγούμενα κεφάλαια.

Παρακάτω θα γίνει ανάλυση τριών βασικών προβλημάτων. Αρχικά θα παρουσιαστεί το πρόβλημα σχεδιασμού ταξιδιού πολυτροπικών μετακινήσεων (Multimodal Journey Planning Problem), παραλλαγή του οποίου επιλύεται και στην παρούσα διπλωματική εργασία. Επίσης παρουσιάζεται το πρόβλημα της συντομότερης διαδρομής (Shortest Path Problem) και το πρόβλημα συντομότερης άφιξης (Earliest Arrival Problem).Τα προβλήματα αυτά έχουν αναπτυχθεί από καιρό και πλέον σήμερα είναι βάση για την λογική εξεύρεσης λύσεων σε προβλήματα που αφορούν την βελτιστοποίηση, δηλαδή προβλημάτων παρόμοιας φύσεως με αυτό που αφορά και εμάς και δεν είναι άλλο από το πρόβλημα σχεδιασμού ταξιδιού.

Αντικείμενο της βελτιστοποίησης αξίζει να αναφερθεί είναι να βρίσκει την καλύτερη λύση ανάμεσα σε σύνολο λύσεων. Για παράδειγμα ένα πρότυπο προβλήματος βελτιστοποίησης είναι το εξής. Έστω ζεύγος <S, n>, όπου S είναι ένα πεπερασμένο σύνολο λύσεων και n: S→R μία συνάρτηση που προσδίδει κάποιο βάρος στις λύσεις (όπου βάρος εννοείται κόστος, χρόνος, απόσταση, κ.α.). Το βάρος είτε βέλτιστα θα μεγιστοποιείται, είτε βέλτιστα θα ελαχιστοποιείται. Ζητείται λύση s* □ S τέτοια ώστε:

$$f(s^*) \leq f(s), \forall\ s \square S$$

αν πρόκειται για πρόβλημα ελαχιστοποίησης ή

$$f(s^*) \geq f(s), \forall\ s \in S$$

αν πρόκειται για πρόβλημα μεγιστοποίησης.

Σε γενικότερο πλαίσιο, θα δούμε πως τα τρία προβλήματα που παρουσιάζονται, ενώ αποτελούν ξεχωριστές και πολύ σημαντικές έννοιες, συνδυάζονται κατάλληλα έτσι ώστε να γίνει απλούστερα η κατανόηση και η επίλυση του προβλήματος του σχεδιασμού ταξιδιού.

Έχοντας το σχεδιασμό ταξιδιού ως βασικό άξονα, τα προβλήματα του συντομότερου μονοπατιού και της συντομότερης άφιξης θα χρησιμοποιηθούν ως εργαλεία. Τα εργαλεία αυτά θα μας βοηθήσουν σε πρώτη φάση (και πιο σημαντική) να προσδιορίσουμε το πως θα δομηθούν τα δεδομένα στην μνήμη του υπολογιστή. Επίσης το πως θα χρησιμοποιηθούν οι μέθοδοι και οι αλγόριθμοι που έχουν αναπτυχθεί πάνω στα δεδομένα για να πάρουμε τα αποτελέσματα που επιδιώκουμε στον συντομότερο χρόνο.

Αυτή η αναγωγή του προβλήματος σχεδιασμού ταξιδιού στα δύο υποπροβλήματα, δεν είναι τυχαία καθώς έχει αντληθεί από την βιβλιογραφία. Αρχικά το πρόβλημα του σχεδιασμού ταξιδιού ήταν ταυτόσημο με αυτό της συντομότερης διαδρομής, αλλά τελευταία συναντώνται και άλλες εκδοχές. Εκτενέστερη αναφορά επί του συγκεκριμένου θέματος και περαιτέρω επί των εργασιών που θα εκτελεστούν για το πρόβλημα σχεδιασμού ταξιδιού, θα γίνει και στο

τελευταίο κομμάτι της εργασίας. Αξίζει να αναφερθεί όμως ότι ο κατακερματισμός του προβλήματος σχεδιασμού ταξιδιού σε μικρότερα προβλήματα είναι το βασικό εργαλείο για την κατανόησή του.

4.3 Πρόβλημα σχεδιασμού ταξιδιού πολυτροπικών μετακινήσεων (Multimodal Journey Planning Problem)

Σε αυτή την ενότητα θα παρουσιαστεί το πρόβλημα του σχεδιασμού ταξιδιού σε πολυτροπικές μετακινήσεις [9]. Θεωρούμε τον σχεδιασμό ταξιδιού (journey planning) σε σενάριο πολυτροπικών μεταφορών. Σε αυτή την περίπτωση, το γενικό πρόβλημα είναι να υπολογιστεί το ταξίδι που συνδυάζει σε λογικά πλαίσια διαφορετικούς τρόπους μετακινήσεων (πολυτροπικές μεταφορές), με μία προσέγγιση ολιστικά αλγοριθμική. Εδώ, ο αλγόριθμος, όχι μόνο εξετάζει κάθε μέσο μεταφοράς μεμονωμένα, αλλά βελτιστοποιεί την επιλογή (και επιπτώσεις που μπορεί να έχει η επιλογή αυτή) μέσου μετακίνησης, με ολοκληρωμένη σκοπιά.

Τρόποι και μέσα μετακίνησης τυπικά θεωρούνται και συμπεριλαμβάνονται το περπάτημα (απεριόριστα), η μετακίνηση με ιδιωτικό αμάξι (απεριόριστα), μετακίνηση με μέσα μαζικής μεταφοράς (τοπικά, αλλά και μεγάλων αποστάσεων), δίκτυα μετακινήσεων από αέρος και δίκτυα ενοικίασης ποδηλάτων. Γενικά δίνεται έμφαση στον ορισμό ότι οι πολυτροπικές μετακινήσεις απαιτούν κάποια ποικιλομορφία στα μέσα μετακινήσεων, π.χ. τόσο τα απεριόριστα και τα προγραμματισμένα μέσα μετακινήσεων θα λαμβάνονται υπ' όψη από τον αλγόριθμο. Για παράδειγμα, ταξίδια που χρησιμοποιούν αποκλειστικά ένα μέσο μεταφοράς δεν αποτελούν πραγματικά πολυτροπικές μετακινήσεις (σύμφωνα με τον ορισμό), εφόσον αυτά τα μέσα μεταφοράς μπορούν να παρουσιαστούν και σαν απλά χρονοδιαγράμματα δρομολογίων δημόσιων μέσων μαζικής μεταφοράς.

Στην πραγματικότητα, η εξέταση του τρόπου εκτέλεσης των μεταφορών αποκλειστικά από τον αλγόριθμο, είναι ζωτικής σημασίας στην πράξη, αφού οι λύσεις που υπολογίζονται πρέπει να είναι εφικτές, συμπεριλαμβάνοντας και τις επιπτώσεις της επιλογής του κάθε μέσου μεταφοράς (π.χ. δεν μπορεί να είναι λύση η χρησιμοποίηση ιδιωτικού αμαξιού ενδιάμεσα από δρομολόγια τρένων). Ιδανικά, ακόμα και η γνώμη του χρήστη θα έπρεπε να λαμβάνεται υπ' όψη. Για παράδειγμα κάποιοι χρήστες προτιμούν την χρήση ταξί, αντί της χρήσης μαζικών μέσων μεταφοράς.

Μία γενική προσέγγιση για την κατασκευή ενός πολυτροπικού δικτύου, είναι το να χτιστεί αρχικά ένα μεμονωμένο γράφημα για κάθε μέσο μετακίνησης, και μετά να συμπτυχθούν σε ένα ενιαίο πολυτροπικό γράφημα με τόξα σύνδεσης (ή ακμές), τα οποία θα προστεθούν για την ενεργοποίηση του δικτύου μέσων μετακινήσεων.

Για να τεθεί ως στόχος τα ταξίδια που προκύπτουν σαν αποτελέσματα, να συνδυάζουν λογικά τα διάφορα μέσα μετακίνησης και τις διαδρομές αυτών, μπορεί κανείς να χρησιμοποιήσει κυρώσεις που θα συνδυαστούν με την αντικειμενική συνάρτηση του αλγορίθμου. Οι κυρώσεις αυτές μπορεί να έχουν διαφόρων τύπων μορφές, οι οποίες εξαρτώνται κυρίως από τον χρήστη και τον τύπο της βελτιστοποίησης που επιθυμεί να πετύχει. Τυπικά είδη κυρώσεων είναι ο χρόνος, το κόστος, η άνεση κ.α. Οι κυρώσεις αυτές συχνά θεωρούνται ως γραμμικός συνδυασμός με πρωταρχικό στόχο την βελτιστοποίηση της διαδρομής (συνήθως με τον χρόνο ως κριτήριο).

4.4 Το πρόβλημα της συντομότερης διαδρομής (Shortest Path Problem)

Το πρόβλημα της συντομότερης διαδρομής (Shortest Path Problem) είναι ίσως το πιο διαδεδομένο πρόβλημα βελτιστοποίησης, το οποίο και συναντάται σε πολλές εφαρμογές. Η επίλυση του προβλήματος αυτού με χρήση διαφόρων τύπων αλγορίθμων συναντάται σε αρκετές πρακτικές εφαρμογές στην καθημερινότητά μας και σε «εργαλεία» των οποίων η χρήση είναι συνηθισμένοι. Ο λόγος της ευρείας χρήσης του προβλήματος της συντομότερης διαδρομής έχει οδηγήσει και στο γεγονός πως έχουν αναπτυχθεί πολλών τύπων και διάφοροι αλγόριθμοι. Οι αλγόριθμοι αυτοί αποτελούν σε πολλές περιπτώσεις την βάση για την επίλυση άλλων προβλημάτων που αντικειμενικό σκοπό έχουν την εύρεση της συντομότερης λύσης.

Ο ορισμός του προβλήματος της συντομότερης διαδρομής [2] (Shortest Path Problem) αναφέρει πως: Το πρόβλημα της συντομότερης διαδρομής μπορεί να οριστεί για παραστάσεις γράφων είτε κατευθυνόμενων, είτε μη κατευθυνόμενων, η μικτών. Εδώ δίνεται ο ορισμός για μη κατευθυνόμενους γράφους, διότι για κατευθυνόμενους γράφους ο ορισμός της διαδρομής απαιτεί ότι οι συνεχόμενες κορυφές συνδέονται με μία κατάλληλη κατευθυνόμενη ακμή. Δύο κορυφές είναι γειτονικές, όταν προσπίπτουν και οι δύο σε μία κοινή ακμή. Μία διαδρομή σε έναν μη κατευθυνόμενο γράφο είναι μία ακολουθία κορυφών (ή κόμβων) τύπου $P = (v_1, v_2, \ldots, v_n) \square V \times V \times \ldots \times V$ με τρόπο τέτοιο ώστε το v_i να είναι γειτονικό του v_{i+1} για $1 \leq i \leq n$. Τέτοιο μονοπάτι όπως το P ονομάζεται μονοπάτι διαδρομής n-1 από τον v_1 μέχρι τονv_n. (Οι κόμβοι v_iείναι μεταβλητές, η αρίθμησή τους εδώ σχετίζεται με την θέση τους στη σειρά και δεν χρειάζεται να αφορά οποιαδήποτε κανονική σήμανση κορυφών).

Έστω $e_{i,j}$ είναι η προσπίπτουσα ακμή και στο v_iκαι στο v_j. Λαμβάνοντας υπ' όψη μία πραγματική - που προσδίδει βάρη συνάρτηση τύπου $f: E \rightarrow \mathbb{R}$, και έναν μη κατευθυνόμενο γράφο G, η συντομότερη διαδρομή από το νστο ν'είναι το μονοπάτι $P = (v_1, v_2, \ldots\ldots, v_n)$ (όπου $v_1=v$ και $v_1=v'$), στο οποίο όλες οι πιθανές επιλογές ελαχιστοποιούν το άθροισμα

$$\sum_{i=1}^{n-1} f(e_{i,i+1})$$

Όταν κάθε ακμή του γράφου έχει μία ενότητα βάρους ή f: E→ {1}, αυτό είναι ίσο με την εύρεση του μονοπατιού με τις λιγότερες ακμές.

Το πρόβλημα αυτό επίσης μερικές φορές ονομάζεται το πρόβλημα εύρεσης της συντομότερης διαδρομής μεταξύ ενός κόμβου αφετηρίας και ενός κόμβου προορισμού σε ένα γράφο, για να διαχωριστεί από τις ακόλουθες παραλλαγές:

- Το πρόβλημα της εύρεσης συντομότερης διαδρομής μεταξύ ενός κόμβου αφετηρίας και όλων των υπόλοιπων κόμβων του γράφου (single – source shortest path problem). Στο πρόβλημα αυτό πρέπει να βρούμε τι συντομότερο μονοπάτι από v κόμβους, οι οποίοι προέρχονται από μία απλή πηγή, προς όλους τους κόμβους του γράφου,
- Το απλό πρόβλημα εύρεσης συντομότερης διαδρομής μεταξύ ενός κόμβου προορισμού και των υπόλοιπων κόμβων του γράφου (single-destination shortest path problem). Στο πρόβλημα αυτό πρέπει να βρούμε τα συντομότερα μονοπάτια από όλους τους κόμβους σε ένα κατευθυνόμενο γράφο, προς μία απλής διαδρομής κορυφή v. Αυτό το πρόβλημα μπορεί να αναχθεί σε πρόβλημα της συντομότερης διαδρομής, απλής πηγής, αν αντιστραφούν τα τόξα του κατευθυνόμενου γράφου,
- Το πρόβλημα εύρεσης συντομότερης διαδρομής μεταξύ του συνόλου των ζευγών κόμβων που περιλαμβάνει ο γράφος (all-pairs shortest path problem). Στο πρόβλημα αυτό πρέπει να βρούμε το συντομότερο μονοπάτι μεταξύ κάθε ζευγαριού κόμβων v, v' στον γράφο.

Αυτές οι γενικότητες έχουν ιδιαιτέρως πιο αποτελεσματικούς αλγόριθμους από την απλοϊκή προσέγγιση του «τρεξίματος» του αλγορίθμου που λύνει το πρόβλημα της συντομότερης διαδρομής απλών ζευγών, ο οποίος θα πρέπει να εξετάσει όλα τα σχετικά ζεύγη κόμβων.

Οι πιο σημαντικοί αλγόριθμοι που έχουν αναπτυχθεί για την επίλυση τέτοιου είδους προβλημάτων είναι οι εξής:

- Αλγόριθμος του Ντάικστρα, (Dijkstra's algorithm), ο οποίος λύνει το πρόβλημα της εύρεσης συντομότερης διαδρομής μεταξύ ενός κόμβου αφετηρίας και όλων των υπόλοιπων κόμβων του γράφου,
- Αλγόριθμος του Μπέλμαν – Φορντ (Bellman-Ford algorithm), ο οποίος λύνει το πρόβλημα της εύρεσης συντομότερης διαδρομής μεταξύ ενός κόμβου αφετηρίας και

όλων των υπόλοιπων κόμβων του γράφου, εάν τα βάρη των ακμών μπορεί να είναι αρνητικά,

- Αλγόριθμος Α* αναζήτησης (A* search algorithm), ο οποίος λύνει το πρόβλημα εύρεσης συντομότερης διαδρομής μεταξύ του συνόλου των ζευγών κόμβων που περιλαμβάνει ο γράφος χρησιμοποιώντας ευρετικούς κανόνες,
- Αλγόριθμος του Φλόυντ – Γουορσολ (Floyd-Warshall algorithm), ο οποίος λύνει το πρόβλημα εύρεσης συντομότερης διαδρομής μεταξύ του συνόλου των ζευγών κόμβων που περιλαμβάνει ο γράφος,
- Αλγόριθμος του Τζονσον (Johnson's algorithm), ο οποίος λύνει το πρόβλημα εύρεσης συντομότερης διαδρομής μεταξύ του συνόλου των ζευγών κόμβων που περιλαμβάνει ο γράφος, και πιθανώς πιο γρήγορα από τον αλγόριθμο του Floyd-Warshall σε αραιούς γράφους,
- Αλγόριθμος Βιτερμπι (Viterbi algorithm), ο οποίος λύνει το πρόβλημα της συντομότερης στοχαστικής διαδρομής (the shortest stochastic path problem), με ένα επιπλέον πιθανοτικό βάρος σε κάθε κόμβο.

4.5 Πρόβλημα συντομότερης άφιξης (Earliest Arrival Problem)

Για την εξήγηση του προτύπου του προβλήματος της συντομότερης άφιξης αρκεί να θεωρήσουμε ως δεδομένο ένα δίκτυο με συγκεκριμένη χωρητικότητα κόμβων και χρονικά δεδομένα μεταφοράς πάνω στα τόξα, ένα υποσύνολο μίας πηγής η οποία περιέχει στοιχεία και μία δεξαμενή με κόμβους προορισμού. Ο στόχος είναι να σταλούν τα στοιχεία από την πηγή έως την δεξαμενή το συντομότερο δυνατόν (as quickly as possible).

Αλγόριθμοι που επιλύουν τέτοιου είδους προβλήματα έχουν αναπτυχθεί ως επί των πλείστων για να χρησιμοποιηθούν σε εφαρμογές που σχετίζονται με την εκκένωση κτιρίων. Θεωρητικά η πιο γρήγορη ροή πάνω στον χρόνο έχει σκοπό να εξασφαλίσει ότι όλα τα στοιχεία (στην περίπτωση του κτιρίου όλα τα άτομα) θα έχουν φτάσει στο τέλος της διαδρομής το συντομότερο δυνατό.

Στο πρότυπο της λύσης του προβλήματος εξασφαλίζεται η εύρεση της βέλτιστης άφιξης, ανεξάρτητα από τον χρόνο που δίνεται ως διαθέσιμος για την ολοκλήρωση της διαδρομής από όλα τα στοιχεία. Η λογική πάνω στην οποία βασίζεται η επίλυση του προβλήματος συντομότερης άφιξης (Earliest Arrival Problem) αποδεικνύει ότι η πρακτική αυτή είναι η βέλτιστη για ζητήματα σαν και το θέμα της εκκένωσης κτιρίου.

Παρόλα αυτά όμως δεν ενδείκνυται για όλες τις περιπτώσεις, γιατί πρακτικά υπάρχουν είδη δικτύων που δεν εξασφαλίζουν λύση για το πρόβλημα αυτό.

Έχει αποδειχθεί ότι σε περιπτώσεις που υπολογίζονται τα δεδομένα με την ύπαρξη δικτύου μόνο μίας πηγής και μόνο μίας δεξαμενής (single source-single-sink network) υπάρχει πάντα λύση. Το ίδιο δεν ισχύει για δίκτυα που αποτελούνται από πολλαπλές πηγές ή/και πολλαπλές δεξαμενές (multiple-sources-single sink network).

Για να επεξηγήσουμε πιο εύκολα το πρότυπο του προβλήματος υπολογισμού της συντομότερης άφιξης (Earliest Arrival Problem) θεωρούμε δίκτυο N = (V, A) στο οποίο δίνεται μία συνάρτηση του ρυθμού ροής με τον χρόνο. Γενικά το πρότυπο της επίλυσης του συνολικού προβλήματος για ένα εκτεταμένο δίκτυο, αποτελείται από επιμέρους επιλύσεις μικρότερων προβλημάτων ίδιου είδους. Για παράδειγμα για να υπολογίσει ένας αλγόριθμος το μονοπάτι με την συντομότερη άφιξη, πρέπει να διερευνήσει τους χρόνους κάθε δυνατού μονοπατιού και στο τέλος να τα συγκρίνει.

Οι ροές πάνω στον χρόνο (flow over time) παρέχουν ένα πολύτιμο εργαλείο για την μοντελοποίηση αλγορίθμων που επιλύουν το πρόβλημα της συντομότερης άφιξης. Στο πρόβλημα συντομότερης άφιξης (Earliest Arrival Problem), στόχος είναι να βρεθεί η πιο απλή ροή πάνω στον χρόνο (flow over time), η οποία ταυτόχρονα θα μεγιστοποιεί και το μέγεθος της ροής. Μία ροή πάνω στο χρόνο που ικανοποιεί αυτή την απαίτηση λέγεται πως έχει επιτυγχάνει την πιο σύντομη άφιξη και λέγεται πως είναι η ροή με την πιο σύντομη άφιξη (earliest arrival flow). Έχει αποδειχθεί πως μία ροή με αυτά τα χαρακτηριστικά υπάρχει σε κάθε δίκτυο, ασχέτως όμως αν αυτή ικανοποιεί ή όχι τις απαιτήσεις του χρήστη του δικτύου.

Οι επιστήμονες αρχικά που δημιούργησαν αλγόριθμους που λύνουν το πρόβλημα της συντομότερης άφιξης είναι οι Μινιέκα και Γουίλκινσον (Minieka and Wilkinson), των οποίων οι αλγόριθμοι βασίζονται κυρίως στους αλγόριθμους που είχαν αναπτυχθεί για την λύση του προβλήματος της συντομότερης διαδρομής. Επίσης οι Χοπ και Τάρντος (Hoppe and Tardos) παρουσίασαν πλήρως έναν αλγόριθμο που επιλύει το πρόβλημα αυτό. Ο αλγόριθμος των δεύτερων βασίζεται σε έναν έξυπνο κανόνα.

Συγκεκριμένα, σύμφωνα με την προσέγγιση των Hoppe και Tardos [15], συνυπολογίζεται ότι σε ένα δίκτυο με διάφορες πηγές και δεξαμενές, με δεδομένες παροχές και ζητήσεις, οι ροές πάνω στον χρόνο (flows over time), οι οποίες δίνουν την συντομότερη άφιξη μπορεί να μην υπάρχουν απαραίτητα. Για την περίπτωση των πολλαπλών πηγών με μοναδική δεξαμενή, η συντομότερη άφιξη υπάρχει πάντα. Αυτή η περίπτωση στην συνέχεια ονομάστηκε ως ύπαρξη της μέγιστης ροής σε δίκτυα τύπου time-expanded. Σήμερα για αυτό το πρόβλημα αναφερόμαστε και ως μεταφόρτωση του προβλήματος της συντομότερη άφιξης (earliest arrival transshipment problem).

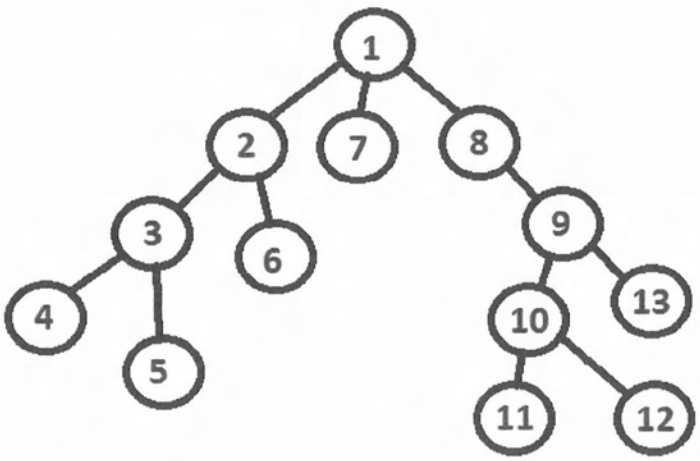
4.6 Οι αλγόριθμοι διάσχισης γράφων (tree traversal algorithms)

Κάποια γενικά στοιχεία για την έννοια και την δομή για τους αλγόριθμους με την ευρεία έννοια παρουσιάστηκαν στο πρώτο κεφάλαιο και στα πλαίσια της ανάλυσης του τρόπου λειτουργίας των σχεδιαστών ταξιδιού. Στο κεφάλαιο αυτό παραθέτονται στοιχεία σχετικά με τον αλγόριθμο που θα χρησιμοποιήσουμε για την επίλυση του προβλήματός μας.

Σε προηγούμενα κεφάλαια είδαμε πως τα πρωτογενή δεδομένα αρχικά ψηφιοποιήθηκαν και στην συνέχεια μετατράπηκαν σε μορφή γράφων. Οι γράφοι αυτοί θα χρησιμοποιηθούν και με την βοήθεια του αλγόριθμου που θα παρουσιαστεί στην συνέχεια θα δίνουν την βέλτιστη λύση για το πρόβλημα που έχει τεθεί. Μία πολύ σημαντική κατηγορία αλγορίθμων, στην οποία εντάσσεται και ο αλγόριθμος που θα χρησιμοποιήσουμε είναι οι αλγόριθμοι διάσχισης γράφων (tree traversal algorithms).

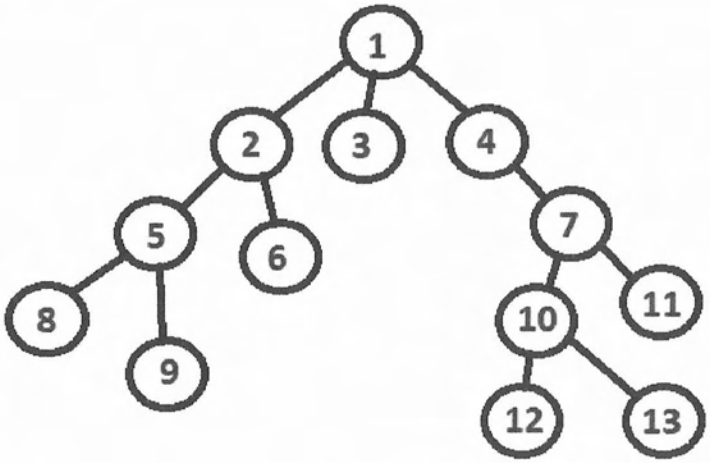
Στην επιστήμη των υπολογιστών, οι αλγόριθμοι διάσχισης γράφων είναι ένας τύπος που αναφέρεται στην διαδικασία επίσκεψης (απλού ελέγχου ή/και ενημέρωσης της κατάστασης) κάθε κόμβου σε ένα γράφο. Η διάσχιση ταξινομείται ανάλογα με την σειρά κατά την οποία επισκέπτονται οι κόμβοι. Υπάρχουν δύο τρόποι διάσχισης ενός γράφου που παρουσιάζονται παρακάτω.

Ο πρώτος τρόπος είναι η αναζήτηση κατά βάθος (depth-first search, DFS), κατά την οποία η αναζήτηση εμβαθύνει όσο είναι δυνατό περισσότερο σε κάθε "παιδί" του κόμβου πριν συνεχίσει προς τους υπόλοιπους "συγγενείς". Δηλαδή, ξεκινώντας από τον αρχικό κόμβο επισκεπτόμαστε τον πρώτο γειτονικό του κόμβο και προχωράμε μέχρι να φτάσουμε σε κόμβο-κορυφή, δηλαδή σε κόμβο που δεν έχει γειτονικά. Μετά επιστρέφουμε στον πρώτο κόμβο που δεν έχουμε επεξεργαστεί και κάνουμε το ίδιο.



Εικόνα 18: 1ος τρόπος διάσχισης γράφων (depth-first search)

Σύμφωνα με τον δεύτερο τρόπο διάσχισης των γράφων, η μέθοδος αναφέρεται ως αναζήτηση κατά πλάτος (breadth - first search, BFS). Σύμφωνα με αυτό τον τρόπο αναζήτησης ο αλγόριθμος ξεκινάει από τον αρχικό κόμβο και επισκέπτεται κάθε κόμβο στο αμέσως χαμηλότερο επίπεδο, προτού συνεχίσει σε επόμενο επίπεδο. Δηλαδή ο αλγόριθμος επισκέπτεται πρώτα την πλησιέστερη γειτονιά του αρχικού κόμβου και έπειτα επισκέπτεται το σύνολο του γράφου.



Εικόνα 19: 2ος τρόπος διάσχισης γράφων (depth-first search)

4.7 Οι αλγόριθμοι βελτιστοποίησης (optimization algorithms)

Στην παρούσα ενότητα παρουσιάζεται με πιο ειδικά στοιχεία η έννοια των αλγορίθμων βελτιστοποίησης, του είδους δηλαδή που όπως θα αναπτυχθεί και στην συνέχεια μας βοηθάει στην επίλυση του προβλήματος που έχει τεθεί στα πλαίσια της παρούσας διπλωματικής εργασίας.

Οι αλγόριθμοι βελτιστοποίησης, οι οποίοι προσπαθούν να βρουν ελάχιστες ή μέγιστες τιμές μαθηματικών συναρτήσεων που τους δίνονται είναι παντού στην μηχανική. Ένας τρόπος για να επιλυθεί ένα δύσκολο πρόβλημα βελτιστοποίησης είναι πρώτα να μειωθεί σε ένα σχετικό, αλλά πολύ πιο απλό πρόβλημα. Έπειτα θα προστεθεί σταδιακά η πολυπλοκότητα του αρχικού προβλήματος και η χρησιμοποίηση της λύσης του κάθε προβλήματος χρησιμοποιείται ως δεδομένο για το επόμενο πρόβλημα, έως ότου καταλήξουμε στο αρχικό.

Οι αλγόριθμοι βελτιστοποίησης, προσδιορίζουν και υπολογίζουν χαρακτηριστικά και τις αντιδράσεις ενός πληθυσμού στοιχείων (population-based algorithms).

Οι αλγόριθμοι που υπολογίζουν τις βέλτιστες λύσεις μίας συνάρτησης είναι επαναληπτικοί αλγόριθμοι. Δηλαδή για να καταλήξει στην βέλτιστη τιμή, ή στο βέλτιστο μονοπάτι, ο

αλγόριθμος εκτελεί πολλαπλές αξιολογήσεις της συνάρτησης, για όχι πάντα συγκεκριμένο αριθμό τιμών των μεταβλητών εισόδου. Κατά την διάρκεια μίας επανάληψης αξιολογούνται όλα τα στοιχεία που αποτελούν τον πληθυσμό και ορίζεται εκ νέου η θέση τους ή η κατάστασή τους. Στην αρχή τα στοιχεία του πληθυσμού τοποθετούνται σε τυχαίες θέσεις ή καταστάσεις. Στόχος, της λειτουργίας των αλγορίθμων βελτιστοποίησης είναι να χρησιμοποιήσουν τις επιμέρους αξιολογήσεις που εκτελούνται, με στόχο να καταλήξουν στην βέλτιστη τιμή που ζητείται. Σε περιπτώσεις που ο αριθμός των αξιολογήσεων κρίνεται υπερβολικά μεγάλου μεγέθους, το τέλος της εργασίας ορίζεται με βάση έναν συγκεκριμένο αριθμό επαναλήψεων.

Σημαντικό είναι η ομοιομορφία των στοιχείων που αποτελούν τον πληθυσμό. Στις περισσότερες φορές για να κρίνονται όλα τα στοιχεία του πληθυσμού κατάλληλα, επιδιώκεται ο χαρακτηρισμός του συνόλου αυτών στην ίδια μονάδα μέτρησης που χρησιμοποιείται και για την συνάρτηση.

Γενικά για προβλήματα βελτιστοποίησης, υπάρχει μία πιο συγκεκριμένη ταξινόμηση των αλγορίθμων. Οι αλγόριθμοι βελτιστοποίησης, βάσει της μεθόδου επίλυσής του προβλήματος και της μεθόδου σχεδίασης χαρακτηρίζονται και κατηγοριοποιούνται σε γραμμικού προγραμματισμού (linear programming), σε δυναμικού προγραμματισμού (dynamic programming), σε άπληστους αλγόριθμους (greedy method algorithms) και σε αλγορίθμους ευρετικής μεθόδου (heuristic method).

Στον γραμμικό προγραμματισμό [2], η αναζήτηση των βέλτιστων λύσεων σε μία γραμμική συνάρτηση βασίζεται σε μία γραμμική ισότητα και σε ανισότητα των περιορισμών. Οι περιορισμοί του προβλήματος μπορούν να χρησιμοποιηθούν απευθείας για την παραγωγή των βέλτιστων λύσεων. Οι αλγόριθμοι που μπορεί να χρησιμοποιηθούν μπορεί να είναι εξειδικευμένοι για την εξεύρεση της λύσης ή αλγόριθμοι που βρίσκουν κατά προσέγγιση την βέλτιστη λύση. Αυτό εξαρτάται από την φύση και την δυσκολία του προβλήματος.

Όταν σε ένα πρόβλημα η βέλτιστη λύση μπορεί να προκύψει από τον συνδυασμό των βέλτιστων λύσεων σε κάποια υποπροβλήματα, ή αν τα ίδια υποπροβλήματα χρησιμοποιούνται για να δώσουν λύσεις σε πολλές παραλλαγές του προβλήματος μία γρηγορότερη προσέγγιση αποτελεί ο δυναμικός προγραμματισμός [2]. Στον δυναμικό προγραμματισμό αποφεύγεται ο εκ νέου υπολογισμός λύσεων που έχουν ήδη υπολογιστεί. Στην περίπτωση αυτή τα υποπροβλήματα θεωρούνται ανεξάρτητα και δεν υπάρχει επανάληψη και έτσι μειώνεται η εκθετική φύση πολλών προβλημάτων για πολυωνυμική πολυπλοκότητα. Παρόλα αυτά με ο δυναμικός προγραμματισμός δεν είναι η λύση για όλα τα πολύπλοκα προβλήματα.

Ένας άπληστος αλγόριθμος [2] μοιάζει πολύ με έναν αλγόριθμο δυναμικού προγραμματισμού, με την έννοια ότι και αυτός λειτουργεί με την εξέταση επιμέρους υποδομών, αλλά σε αυτή την

περίπτωση όχι του προβλήματος, αλλά μίας δεδομένης απάντησης. Τέτοιου είδους αλγόριθμοι ξεκινάνε με μία λύση, η οποία μπορεί να δίνεται ή κατασκευάζεται επί τόπου με κάποιο τρόπο, και την βελτιώνουν κάνοντας σιγά-σιγά μικρές μορφοποιήσεις. Σε κάποια προβλήματα υπολογίζεται η βέλτιστη λύση, αλλά σε κάποια άλλα δίνεται μία λύση η οποία δεν δύναται να βελτιωθεί περαιτέρω, αλλά δεν είναι η βέλτιστη.

Σε προβλήματα βελτιστοποίησης, οι αλγοριθμοι ευρετικής μεθόδου [2] μπορούν να χρησιμοποιηθούν για να βρεθεί μία λύση, κοντινή της βέλτιστης. Η μέθοδος αυτή χρησιμοποιείται σε περιπτώσεις που η εύρεση της βέλτιστης λύσης είναι μη πρακτική. Τέτοιοι αλγόριθμοι έχουν την λογική ότι φτάνουν πιο κοντά στην βέλτιστη λύση, όσο περισσότερο λειτουργούνε, με την αρχή ότι αν τρέχουν για άπειρο χρόνο θα βρεθεί η βέλτιστη λύση. Το πλεονέκτημά τους είναι ότι μπορούν να βρουν μία λύση πολύ κοντά στη βέλτιστη σε ένα σχετικά σύντομο χρονικό διάστημα.

Η περίπτωση που μελετάται και παρουσιάζεται στην παρούσα διπλωματική εργασία σχετίζεται αποκλειστικά το πρόβλημα της βέλτιστης διαδρομής. Η βέλτιστη διαδρομή όπως αναλύθηκε και παραπάνω, αφορά την ελαχιστοποίηση ή μεγιστοποίηση των δεδομένων εξόδου, κατά την διάσχιση των ακμών ή/και το πέρασμα από τους κόμβους. Τα εν λόγω δεδομένα στην περίπτωσή μας μπορεί να είναι η απόσταση, ο χρόνος, το κόστος, η άνεση, κ.α.

Ο πιο δημοφιλής αλγόριθμος για την εξεύρεση λύσεων σε προβλήματα τέτοιου τύπου είναι ο αλγόριθμος του Ντάικστρα (Dijkstra) [2]. Ο αλγόριθμος του Ντάικστρα (Dijkstra) πήρε το όνομά του από τον Ολλανδό Έντσγκερ Ντάικστρα, ο οποίος τον επινόησε το 1956 και τον δημοσίευσε το 1959. Πρόκειται για έναν αλγόριθμο εύρεσης συντομότερων διαδρομών (single-source shortest path problem) από κοινή αφετηρία σε έναν (κατευθυνόμενο ή μη) γράφο με μη αρνητικά βάρη στις ακμές.

Στην περίπτωση που μελετάται στην παρούσα διπλωματική εργασία, και όπως θα παρουσιαστεί σε επόμενες ενότητες, ο αλγόριθμος που θα υπολογίζει την λύση στο πρόβλημα που έχει τεθεί έχει ως βάση τον αλγόριθμο Ντάικστρα. Για τον λόγο αυτό στην ενότητα αναφέρονται και παρουσιάζονται εκτενέστερα ιστορικά στοιχεία, καθώς και στοιχεία για την δομή και τον τρόπο λειτουργίας του.

Επίσης ο αλγόριθμος που χρησιμοποιείται περισσότερο για τον υπολογισμό βέλτιστης διαδρομής είναι ο Αλγόριθμος A*, ο οποίος είναι μία επέκταση του Αλγορίθμου του Ντάικστρα. Χρησιμοποιεί ευρετικους κανόνες και έτσι επιτυγχάνει καλύτερους χρόνους. Ο αλγόριθμος αυτός χρησιμοποιεί την best-first μέθοδο.

Ο αλγόριθμος A* είναι μία εξαιρετικά επιτυχημένη εφαρμογή της best-first αναζήτησης σε οδικά δίκτυα. Χρησιμοποιεί ευρετική συνάρτηση τύπου «το καλύτερο πρώτα (best-first)» και υπολογίζει το μονοπάτι της βέλτιστης διαδρομής από έναν αρχικό κόμβο, σε έναν κόμβο που ορίζει ο χρήστης. Υπάρχει η δυνατότητα για τον χρήστη να ορίζει περισσότερους του ενός τελικούς κόμβους. Η αναζήτηση τύπου «το καλύτερο πρώτα» διεισδύει στον γράφο ακολουθώντας την διαδρομή από έναν κόμβο που είναι η πιο πολλά υποσχόμενη για να δώσει την επιθυμητή λύση.

4.8 Ο αλγόριθμος Ντάικστρα (Dijkstra's algorithm)

Ο αλγόριθμος του Ντάικστρα (Dijkstra's algorithm) [2] είναι ένας αλγόριθμος που υπολογίζει το συντομότερο μονοπάτι μεταξύ κόμβων σε έναν γράφο, το οποίο μονοπάτι μπορεί να αντιπροσωπεύει για παράδειγμα, ένα δίκτυο δρόμων. Η ιδέα παρουσιάστηκε από τον Ολλανδό επιστήμονα Edsger W. Dijkstra το 1956, όπως έχει αναφερθεί και παραπάνω και δημοσιεύθηκε τρία χρόνια αργότερα.

Ο Ντάικστρα σκέφτηκε το πρόβλημα του συντομότερου μονοπατιού όταν δούλευε στο Μαθηματικό κέντρο του Άμστερνταμ, σαν προγραμματιστής ο οποίος εκείνο τον καιρό προσπαθούσε να αποδείξει τις ικανότητες ενός νέου υπολογιστή. Το αντικείμενό του ήταν να επιλέξει ένα πρόβλημα, μαζί με την λύση του (το οποίο πρόβλημα θα παραγόταν από υπολογιστή), την οποία λύση όμως κανένας άνθρωπος δεν θα μπορούσε να υπολογίσει. Σχεδίασε λοιπόν τον αλγόριθμο του συντομότερου μονοπατιού και αργότερα τον εφάρμοσε στον υπολογιστή τον οποίο μελετούσε.

Αξίζει να αναφερθεί ότι ο εν λόγω αλγόριθμος είναι λειτουργικός και σε κατευθυνόμενους και σε μη κατευθυνόμενους γράφους, όμως έχει κάποιους περιορισμούς στην λειτουργία του. Αρχικά όλες ακμές πρέπει να έχουν μη αρνητικά βάρη. Επίσης οι κόμβοι στο σύνολό τους πρέπει να είναι συνδεμένοι μεταξύ τους.

Ο αλγόριθμος συναντάται σήμερα πλέον σε πολλές παραλλαγές. Η αρχική έκδοση του αλγόριθμου, όπως αναφέρθηκε έβρισκε το συντομότερο μονοπάτι μεταξύ δύο κόμβων, αλλά μία πιο κοινή εκδοχή ενώνει έναν απλό κόμβο με μία πηγή κόμβων και βρίσκει το συντομότερο μονοπάτι από την πηγή προς όλους τους κόμβους του γράφου, δημιουργώντας το δέντρο των συντομότερων μονοπατιών (shortest - path tree).

Ο αλγόριθμος του Dijkstra είναι "άπληστος". Δηλαδή, σε κάθε βήμα επιλέγει την τοπικά βέλτιστη λύση, ώσπου στο τελευταίο βήμα συνθέτει μια συνολικά βέλτιστη λύση.

Η τυποποιημένη περιγραφή του αλγόριθμου σε βήματα είναι η εξής: [2]

1. Ο αλγόριθμος δίνει σε κάθε κόμβο μία τιμή απόστασης. Δίνει την τιμή 0 στον αρχικό κόμβο και τιμή άπειρο σε όλους τους υπόλοιπους κόμβους,
2. Χαρακτηρίζει όλους τους κόμβους ως κόμβους που δεν έχουν επισκεφθεί, εκτός του αρχικού. Ο αρχικός κόμβος χαρακτηρίζεται ως τρέχων,
3. Υπολογίζει για όλους τους γειτονικούς του τρέχοντα κόμβου το συνολικό άθροισμα της προσωρινής απόστασης τους από τον κόμβο αφετηρίας. Αν αυτή η προσωρινή απόσταση είναι μικρότερη από την προηγούμενη καταγεγραμμένη, τότε την αντικαθιστά,
4. Όταν ολοκληρωθεί η εξέταση όλων των γειτονικών κόμβων, σημειώνεται ο τρέχων κόμβος ως επεξεργασμένος. Η απόσταση που καταγράφεται είναι η ελάχιστη, δηλαδή η τελευταία προσωρινή,
5. Ο επόμενος τρέχων κόμβος θα είναι αυτός που δεν έχει επεξεργαστεί και έχει την μικρότερη απόσταση,
6. Αν όλοι οι κόμβοι έχουν επεξεργαστεί προχωράει στον συνολικό υπολογισμό, αλλιώς επιστρέφει στο βήμα 3,
7. Αν όλοι οι κόμβοι έχουν επισκεφτεί, τότε η εκτέλεσή του ολοκληρώνεται. Σαν συνολικό αποτέλεσμα αθροίζει και παρουσιάζει (ξεκινώντας από τον τελευταίο τρέχων κόμβο), όλες τις αποστάσεις με την αντίθετη σειρά των κόμβων που υπολογίστηκαν ως τρέχοντες.

4.9 Παρουσίαση αλγορίθμου που χρησιμοποιήθηκε στην εργασία

Στην ενότητα που ακολουθεί γίνεται η παρουσίαση του αλγόριθμου που χρησιμοποιήθηκε για την εφαρμογή των δύο γράφων που δημιουργήθηκαν με τρόπο που αναλύθηκε σε προηγούμενα κεφάλαια. Επίσης γίνεται αναφορά στον λόγο που επιλέχθηκε ο συγκεκριμένος τύπος αλγορίθμου για να "τρέξει" τα δεδομένα μας.

Στην παρούσα διπλωματική εργασία, και αφού αναπτύχθηκαν τα κομμάτια του λογισμικού για το χτίσιμο των γράφων, τρέξαμε μερικά παραδείγματα για να δούμε την αποτελεσματικότητα των δύο διαφορετικών τύπων του λογισμικού μας. Το "τρέξιμο" των παραδειγμάτων αυτών απαιτεί την χρήση ενός αλγορίθμου εύρεσης του κοντινότερου μονοπατιού. Ο αλγόριθμος που θα χρησιμοποιηθεί θα ανακτά δεδομένα από τους γράφους μας. Κατά βάση εξετάστηκαν ως πιθανές επιλογές οι αλγόριθμοι οι οποίοι είναι πιο συνηθισμένοι και χρησιμοποιούνται ως επί των πλείστων για την επίλυση προβλημάτων βελτιστοποίησης και πιο συγκεκριμένα για την επίλυση των τριών προβλημάτων που παρουσιάστηκαν παραπάνω. Καταλήξαμε επίσης στο συμπέρασμα ότι θα είναι πιο αξιόπιστο να χρησιμοποιηθεί ένας αλγόριθμος για τα παραδείγματα μας, ο ίδιος και για τις δύο περιπτώσεις graph builder, καθώς η βάση αναφοράς των αποτελεσμάτων θα πρέπει να είναι κοινή.

Έπειτα από σχετική διερεύνηση και από μελέτη της βιβλιογραφίας, αποφασίσαμε ότι ο αλγόριθμος του Ντάικστρα (Dijkstra's algorithm) είναι ο καταλληλότερος για αυτές τις δοκιμές. Το γεγονός ότι είναι ο πιο διαδομένος στη βιβλιογραφία αλλά και στα δημοφιλέστερα συστήματα σχεδιασμού ταξιδιού(GoogleMaps, Bing Maps, κ.α.) ήταν τα δύο βασικότερα κριτήρια στα οποία στηριχθήκαμε για την απόφαση μας αυτή. Σημαντικό στοιχείο το οποίο πρέπει να δώσουμε έμφαση είναι οι περισσότεροι αλγόριθμοι journey planning είναι παράγωγα του Dijkstra. Χαρακτηριστικό παράδειγμα ο αλγόριθμος A*.

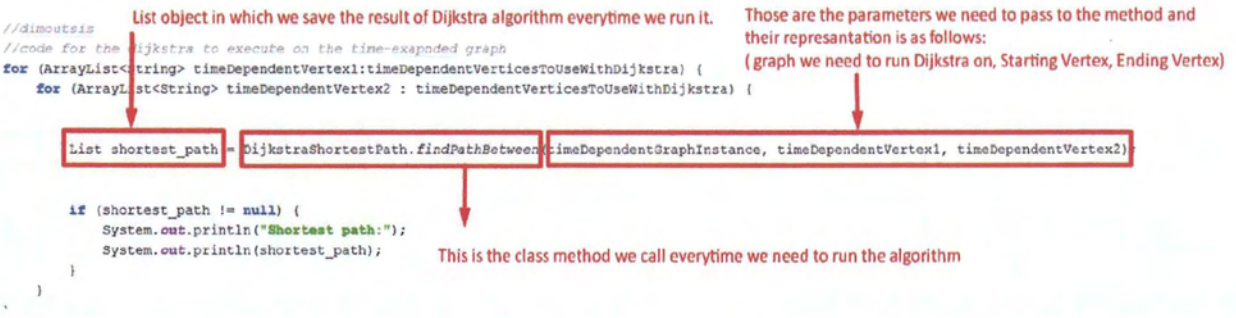
Τώρα θα αναφερθούμε σε ειδικότερα θέματα που αφορούν την εργασία μας και την χρησιμοποίηση του Dijkstra. Επίσης θα παρουσιάσουμε επιγραμματικά στον τρόπο με τον οποίο λειτουργεί και θα αναγάγουμε τις δομές λειτουργίας του, που αναφέρθηκαν σε προηγούμενη ενότητα, με το πρόβλημα που μελετήθηκε στην εργασία μας.

Ο Dijkstra, στην αρχική και πιο "καθαρή" εκδοχή του, λύνει όπως έχουμε πει το πρόβλημα εύρεσης συντομότερου μονοπατιού μέσα από ένα γράφο. Παίρνοντας ως είσοδο ένα γράφο, και δύο κόμβους, στην περίπτωση του προβλήματός μας ένας για την αρχική θέση του επιβάτη και ένας για την τελική θέση του επιβάτη, ξεκινά να αναλύει (ή λύνει) τους κόμβους γύρω από τον αρχικό, έως ότου φτάσει στο τελικό. Συνδυάζοντας κατάλληλα τους λυμένους κόμβους και τις συνδέσεις τους (επιμέρους διαδρομές των γραμμών) δημιουργεί το συντομότερο μονοπάτι (shortest path).

Είναι εμφανές ότι ο Dijkstra λειτουργεί με ευρετικό τρόπο, δηλαδή το πρώτο μονοπάτι που θα βρει να συνδέει τον αρχικό κόμβο με τον τελικό, "βαφτίζεται" ως βέλτιστο. Είναι σημαντικό να αναφέρουμε ότι υπάρχουν άλλες εκδοχές του αλγορίθμου που είναι σε περιπτώσεις πολύ πιο αποτελεσματικές. Ειδικά όταν συνδέονται με τεχνικές προεπεξεργασίας και προσέγγισης των αληθινών στοιχείων, τότε μπορούν να αποδώσουν πολύ καλύτερα και πιο αξιόπιστα από ότι ο αρχικός Dijkstra. Για παράδειγμα ο A*, που αναφερθήκαμε και προηγουμένως, προϋπολογίζει για κάθε κόμβο το κόστος του να φτάσεις σε κάποιον άλλον και λύνει τους κόμβους στη φάση του αλγορίθμου απλά ψάχνοντας τους κόμβους αυτούς που οι συνδέσεις τους έχουν το μικρότερο κόστος. Αυτή η τεχνική δίνει κατεύθυνση στην αναζήτηση των κόμβων και αναδεικνύει γρηγορότερα, καλύτερα αποτελέσματα.

Όπως και για το χτίσιμο των γράφων, και στο κομμάτι του αλγορίθμου, η δουλειά μας στηρίχθηκε στη βιβλιοθήκη jgrapht [13]. Ενώ αποτελεί μία από τις πιο διαδεδομένες βιβλιοθήκες λογισμικού για χρήση δομών δεδομένων γράφων σε γλώσσα προγραμματισμού Java, ταυτόχρονα, η απλή δομή της και ο καλογραμμένος ψευδοκώδικας της, την κάνουν ειδική για το γνωστικό επίπεδο και το χρόνο που χρειαζόμασταν για την εκπόνηση της διπλωματικής εργασίας.

Ουσιαστικά αφού επεκτείναμε τις ήδη υπάρχουσες κλάσεις του project, προσθέτοντας λειτουργίες για το χτίσιμο γράφων για χρήση στον σχεδιασμό ταξιδιού, χρησιμοποιήσαμε την ήδη υπάρχουσα μέσα στο projectυλοποίηση του αλγορίθμου Dijkstra, χωρίς να κάνουμε καμία αλλαγή σε αυτόν. Η αναγνωσιμότητα και κοινή αποδοχή που αντιμετωπίζει το project jgrapht, λειτούργησαν ως εγγυητικός παράγοντας για την αξιοπιστία του λογισμικού αυτού. Παρακάτω παραθέτουμε ένα screenshot με τον τρόπο με τον οποίο καλούμε τον αλγόριθμο πάνω στη δομή δεδομένων που παρήγαγε το λογισμικό το οποίο γράψαμε. Σημειώνεται ότι ο αλγόριθμος Dijkstra καλείται με τον τρόπο που παρατίθεται, εφόσον έχει γίνει import στο πρόγραμμα intellij [11] (το οποίο χρησιμοποιήσαμε) και στην δομή του project μας από το jgrapht.



Εικόνα 20: Τρόπος με τον οποίο καλούμε τον αλγόριθμο Dijkstra πάνω στην δομή δεδομένων που παρήγαγε το λογισμικό το οποίο γράψαμε

4.10 Παρουσίαση των κωδίκων που αναπτύχθηκαν για την δημιουργία των δύο λογισμικών graph builder

Περνώντας στο κύριο κομμάτι της εργασίας μας, αναπτύξαμε δύο διαφορετικές εκδοχές για το λογισμικό graph builder. Υπενθυμίζουμε, ότι το λογισμικό graph builder, ουσιαστικά συνθέτει τα δεδομένα από τα αρχεία GTFS και παράγει δομές δεδομένων οι οποίες είναι καταλληλότερες για το τρέξιμο των αλγορίθμων συντομότερου μονοπατιού και κατ' επέκταση σχεδιασμού ταξιδιού.

Αρχικά, και σχετικά με τις δύο εκδοχές, ασχοληθήκαμε με τους time-expanded και time-dependent τύπους γράφων. Για τους πρώτους, η έννοια του κόμβου ταυτίζεται με ένα γεγονός στο χρόνο και οι ακμές τους συνδέουν κατάλληλα εξασφαλίζοντας τη συνέχεια του μοντέλου του γράφου. Από την άλλη, οι time-dependent γράφοι μοντελοποιούν τον κάθε κόμβο ως ένα γεωγραφικό σημείο και την έννοια του γεγονότος, δηλαδή των διαφορετικών αφίξεων και αναχωρήσεων, προσπαθούν να την εκφράσουν μέσω των ακμών. Παρακάτω θα αναλυθούν οι δύο διαφορετικές προσεγγίσεις και οι υλοποιήσεις που κάναμε για τους δύο τύπους των γράφων.

Να ξεκαθαρίσουμε ότι τα κύρια μέλη του κώδικα είναι τρία αρχεία-κλάσεις. Η main.java, η TimeExpandedGraph.java και ηTimeDependentGraph.java. Η πρώτη κλάση είναι η βασική κλάση η οποία τρέχει και τα test μας, ενώ οι TimeDependentGraph και TimeExpandedGraph είναι αυτές οι οποίες χτίζουν τους γράφους με τον τρόπο που έχουμε αναλύσει, επεκτείνοντας τις ήδη υπάρχουσες δυνατότητες της δημιουργίας πιο απλών γράφων του jgraphtproject.

Στο σημείο αυτό να σημειώσουμε ότι όταν λέμε επεκτείνουμε, μέσω τις εντολής "extends", εννοούμε ότι όλες οι μέθοδοι που συμπεριλαμβάνονται στις κλάσεις αυτές και που έχουν δημιουργηθεί από τους προγραμματιστές του λογισμικού ανοικτού κώδικα jgrapht, μπορούν να χρησιμοποιηθούν και σε αυτήν που δημιουργήσαμε εμείς. Αυτό ήταν ένα πολύ βασικό βήμα για την υλοποίηση μας και στις δύο περιπτώσεις, time-expanded και time-dependent, γιατί μπορέσαμε να χρησιμοποιήσουμε τον ήδη υπάρχοντα κώδικα για την εισαγωγή κόμβων και ακμών. Τώρα, ο κώδικας που προσθέσαμε σε κάθε περίπτωση ήταν αυτός που έλεγε πότε και πως προσθέτουμε κόμβους και ακμές αντίστοιχα.

Ξεκινώντας με την time-expanded προσέγγιση, αυτό που κάναμε μέσα από τον κώδικα είναι να εξασφαλίσουμε δημιουργώντας τις επιπλέον μεθόδους για κάθε κλάση, την εύκολη δημιουργία του γράφου από τα GTFS. Αρχικά δημιουργήσαμε τη κλάση TimeExpandedGraph.java η οποία στην ουσία επεκτείνει την κλάση DirectedWeightedMultiGraph και υλοποιεί (implements) την κλάση WeightedGraph. Να τονίσουμε ότι οι δύο αυτές κλάσεις ανήκουν στο jgraphtproject και δεν τροποποιήθηκαν καθόλου.

Στο time-expanded κομμάτι της δουλειάς μας, και καθώς ο ψευδοκώδικας διαβάζει τα αρχεία GTFS για κάθε γραμμή του αρχείου stop_times.txt παράγουμε ένα event. Αυτό επιτεύχθηκε με τη δημιουργία της μεθόδου addEvent. Η μέθοδος addEvent είναι υπεύθυνη για τη δημιουργία των τριών κόμβων για κάθε γραμμή του stop_times.txt, όπως έχουμε εξηγήσει και παραπάνω, έναν για την άφιξη, έναν για την αναμονή και ένα για την αναχώρηση. Στη συνέχεια για την κατάλληλη σύνδεση αυτών τον κόμβων δημιουργήσαμε την addEdges η οποία καλείται όταν έχουν πλέον δημιουργηθεί οι κόμβοι που αντιπροσωπεύουν τα events. Ο τρόπος που ενώνονται οι κόμβοι είναι αυτός ο οποίος έχει παρουσιαστεί και σε προηγούμενο κεφάλαιο.

Επίσης, στην κλάση TimeExpandedGraph θα συναντήσουμε και άλλες πέντε μεθόδους (utilities), τέσσερις εκ των οποίων χρησιμοποιούνται για κάποιες βασικές διαδικασίες που θέλουμε να εκτελούμε όπως ο η μετατροπή της ώρας από μορφή hh:mm:ss (hours, minutes,seconds) σε μορφή περασμένων δευτερολέπτων από την αρχή της ημέρας (χρειάστηκε για υπολογισμούς κόστους μεταφοράς). Οι άλλες χρήσεις των δύο από τις τρεις αυτές μέθοδοι, είναι για το επαναληπτικό διάβασμα των αρχείων και τη δημιουργία του κόμβου σαν αντικείμενο (object) στη μνήμη. Τέλος, η πέμπτη και τελευταία μέθοδος είναι η

initiateTimeExpandedGraph.java, η οποία συνδυάζει όλες τις υπόλοιπες με κατάλληλο τρόπο έτσι ώστε να προκύπτει ο γράφος μας στη main κλάση (ουσιαστικά "δεσμεύει" χώρο για την δημιουργία του γράφου).

Σαν βάρος για τη μετάβαση από τον ένα κόμβο στον άλλο χρησιμοποιήθηκε ο χρόνος μετάβασης από τον ένα στον άλλο.

Τελικώς, να επισημάνουμε ότι επειδή ακριβώς κάθε κόμβος αποτελεί ένα γεγονός στο χρόνο και επειδή έπρεπε να αποθηκευτεί με κάποιο τρόπο στη μνήμη του υπολογιστή ο χρόνος που συμβαίνει, ο κάθε κόμβος αποθηκεύτηκε σαν λίστα στη μνήμη του υπολογιστή κρατώντας από τη μία το όνομα της στάσης και τον τύπο του κόμβου και από την άλλη το χρόνο στον οποίο πραγματοποιήθηκε.

Περνώντας πλέον στους time-dependent γράφους, σε αυτό το κομμάτι του κώδικα χρησιμοποιήσαμε έξι μεθόδους στην κλάση μας, αλλά σε αυτή τη περίπτωση ήταν πολύ πιο απλή η λειτουργία τους. Επειδή αυτή η προσέγγιση δεν είναι event-based, δεν είχαμε μέθοδο addEvent. Αντί αυτού χρησιμοποιήσαμε τις κλάσεις addVertices, createTimeDependentArray και addPreQueryEdges. Η πρώτη κάνει αυτό που λέει το όνομα της, δηλαδή προσθέτει τους κόμβους στο γράφο ανάλογα, αυτή τη φορά με το αρχείο stop.txt. Οι άλλες δύο έχουν να κάνουν με τη φιλοσοφία μας σχετικά τη time-dependent προσέγγιση.

Αυτό που γίνεται ουσιαστικά σε αυτή τη δεύτερη τεχνοτροπία την οποία αναπτύξαμε είναι να μην προσθέτουμε τις ακμές εξ' αρχής αλλά να το κάνουμε κάθε φορά πριν το τρέξιμο του Dijkstra και ανάλογα με το χρόνο αναχώρησης από την αρχική στάση. Αναμενόμενο αυτής της ιδιότητας που προσδώσαμε στο γράφο είναι ότι χρειάζεται εμφανέστατα πολύ μικρότερο χρόνο για να τρέξει. Οπότε με την timeDependentArray προσπαθήσαμε να κάνουμε αυτή την on-the-fly προετοιμασία του γράφου (δημιουργία συνδέσεων μόνο για τους κόμβους που μας απασχολούνε) όσο πιο απλή γίνεται. Δημιουργήσαμε για κάθε ζεύγος στάσεων, μία λίστα με τα events που συμβαίνουν στον χρόνο και μετά τα ταξινομήσαμε σε αύξουσα σειρά με βάση το πόσο μακριά συμβαίνουν από το την αρχή της ημέρας (00:00:00). Αυτό μας επιτρέπει να χρησιμοποιούμε, για την εύρεση του κατάλληλου χρόνου για ανάθεση ως κόστος σε ακμή, αλγόριθμους για ταξινομημένους πίνακες που συνήθως είναι πολύ πιο γρήγοροι. Χρησιμοποιήσαμε στη περίπτωση μας τον αλγόριθμο binarysearch, τον οποίο τον χρησιμοποιήσαμε ως έχει επίσης από την βιβλιοθήκη jgrapht.

Περνώντας, τώρα, στο πως χρησιμοποιούμε τη μέθοδο addPreQueryEvents, ουσιαστικά ήταν η πιο χρήσιμη μέθοδος γιατί τρέχει κάθε φορά πριν το Dijkstra επιστρέφοντας τις σωστές ακμές για το γράφο. Με την addEdges τα προσθέταμε αναλόγως.

Τελικώς, έχουμε την composeSimpleVertex που χρησιμοποιείται για να συνθέσει τους κόμβους μας που είναι απλές μεταβλητές String, μιας και δεν χρειαζόταν κάτι πιο πολύπλοκο για να αντιπροσωπεύσουμε τους κόμβους. Και στο time-dependentgraph χρειαστήκαμε μια μέθοδο, η οποία λειτουργεί σαν ομπρέλα για όλες τις υπόλοιπες και χρησιμοποιείται για τη σύνθεση του γράφου. Την ονομάσαμε initiateTimeDependentGraph, η οποία επίσης και αντίστοιχα με την περίπτωση του initiateTime-expandedgraph συνδυάζει όλες τις υπόλοιπες με κατάλληλο τρόπο έτσι ώστε να προκύπτει ο γράφος μας στη main κλάση (ουσιαστικά "δεσμεύει" χώρο για την δημιουργία του γράφου).

Τέλος, έχουμε την κλάση main, η οποία δεν κάνει κάτι παραπάνω από το να καλεί τις άλλες κλάσεις δημιουργώντας τα αντικείμενα στη μνήμη του υπολογιστή και μετά να τρέχει τον αλγόριθμο. Μέσα σε αυτή τρέχουμε παράλληλα με τα άλλα tasks και μερικούς timers οι οποίοι είναι μεταβλητές που ασχολούνται με το χρόνο τρεξίματος του προγράμματος. Σε αυτή την κλάση τρέχει και ο Dijkstra σε κάθε περίπτωση.

Η χρήση της main, δεν έγινε για κάποιο συγκεκριμένο λόγο. Απλά ακολουθήσαμε τη μεθοδολογία που ακολουθούν πολλοί προγραμματιστές Java, έχοντας μία main κλάση ως βασική. Ιστορικά, αυτό συμβαίνει στη Java τα τελευταία χρόνια χωρίς να έχει την επιλογή να τρέξει κάποια άλλη κλάση. Αυτό γίνεται γιατί η Java είναι μία γλώσσα η οποία αφορά μεγάλα κυρίως projects που απαιτούν συνδυασμό πολλαπλών αρχείων σε ένα.

Η υλοποίηση των προγραμμάτων έλαβε χώρα στο εργαστήριο Οργάνωσης Παραγωγής της Πολυτεχνικής σχολής του Πανεπιστημίου Θεσσαλίας, που βρίσκεται το τμήμα Μηχανολόγων Μηχανικών. Οι κώδικες που αναπτύχθηκαν στο σύνολό τους επισυνάπτεται στο Παράρτημα της παρούσας διπλωματικής εργασίας.

4.11 Σύνοψη κεφαλαίου

Στο κεφάλαιο που προηγήθηκε παρουσιάστηκε μεθοδολογία που χρησιμοποιήθηκε στο σύνολο της παρούσας διπλωματικής εργασίας. Έγινε αρχικά ο προσδιορισμός του προβλήματος που μας απασχόλησε, και έπειτα και η ανάπτυξη του αλγορίθμου που το επιλύει. Επιπρόσθετα έγινε πλήρης παρουσίαση των δύο κωδίκων, οι οποίοι χρησιμοποιήθηκαν για το χτίσιμο των γράφων.

Στο επόμενο και τελευταίο κεφάλαιο της διπλωματικής εργασίας γίνεται η παρουσίαση των αποτελεσμάτων από το "τρέξιμο" του αλγόριθμου με τα δεδομένα από τους δύο graph builder. Τα αποτελέσματα αυτά, θα σχολιαστούν και θα αναλυθεί ποιος κρίνεται ο βέλτιστος τρόπος, από άποψη ταχύτητας, αρχικά κατασκευής γράφου και έπειτα υπολογισμού λύσεων για το συγκεκριμένο πρόβλημα που έχει τεθεί.

Κεφάλαιο 5ο: Αποτελέσματα - Συμπεράσματα

Στα πλαίσια της παρούσας διπλωματικής εργασίας και όπως παρουσιάστηκε σε προηγούμενα κεφάλαια έγινε η συλλογή, η ψηφιοποίηση των δεδομένων και η μετατροπή τους σε μορφή γράφων. Η μετατροπή σε μορφή γράφων έγινε με την χρησιμοποίηση graph builder. Χρησιμοποιήθηκαν δύο τύποι χτισίματος γράφου, ο ένας ήταν τύπου time-expanded και ο δεύτερος τύπου time-dependent. Αναλυτική παρουσίαση των χαρακτηριστικών των δύο graph builders έγινε στο αντίστοιχο κεφάλαιο.

Έπειτα, οι δύο γράφοι που κατασκευάστηκαν έτρεξαν πάνω σε κοινό αλγόριθμο. Τα αποτελέσματα που θα κρίνουμε στη συνέχεια στα πλαίσια του πέμπτου κεφαλαίου εξαρτώνται από τον χρόνο που χρειάστηκε ο κάθε graph builder να χτίσει τους γράφους και ο χρόνος που χρειάστηκε ο αλγόριθμος στην κάθε περίπτωση που έτρεξε για να μας γυρίσει πίσω αποτελέσματα εύρεσης του συντομότερου μονοπατιού.

Όσον αφορά την εύρεση των συντομότερων μονοπατιών δεν αρκεστήκαμε στον υπολογισμό του χρόνου μεμονωμένων μονοπατιών, αλλά και στην εύρεση του χρόνου μεταξύ ομάδων σημείων. Χωρίζοντας τα σημεία σε ομάδες και τρέχοντας τον αλγόριθμο πιο μαζικά πήραμε ένα πιο αντιπροσωπευτικό δείγμα της ταχύτητας κάθε τύπου γράφου.

Στην συγκεκριμένη διπλωματική εργασία ο τελικός αλγόριθμος που τρέχει και μας δίνει τα τελικά αποτελέσματα δεν αναπτύχθηκε από την αρχή, αλλά χρησιμοποιήθηκε έτοιμος από την βιβλιογραφία. Οπότε και δεν κρίνεται ως προς την αποτελεσματικότητά του αυτόνομα αυτός, ούτε και ως προς την ταχύτητα των αποτελεσμάτων του.

Αντίθετα, αναπτύχθηκε ένας κώδικας, σκοπός του οποίου είναι να χτίσει γράφους με δύο διαφορετικούς τρόπους. Οι γράφοι αυτοί θα χρησιμοποιηθούν ως δεδομένα εισόδου για τον τελικό αλγόριθμο.

Ουσιαστικά τα στοιχεία που θα παρατεθούν και παρακάτω στα πλαίσια του 5ου και τελευταίου κεφαλαίου θα συγκρίνουν τις δύο προσεγγίσεις χτισίματος γράφων. Αρχικά θα τις συγκρίνουν ως προς τον χρόνο της αυτής καθ' ε αυτής της λειτουργίας τους αλλά και ως προς την αποτελεσματικότητά τους ώστε να παρέχουν με τον καλύτερο δυνατό τρόπο, μία μορφή δεδομένων για να τρέχει πιο γρήγορα ο τελικός αλγόριθμος.

Κατά την διάρκεια των παραδειγμάτων δηλαδή, η διαδικασία ξεκινούσε από την εισαγωγή των δεδομένων στον graph builder. Η διαδικασία ολοκληρωνόταν την στιγμή που ο τελικός

αλγόριθμος dijkstra μας παρέχει την λύση στο πρόβλημα του συντομότερου μονοπατιού (shortest path).

Επιλέξαμε ως δείγμα να τρέξουμε τα δεδομένα που συλλέχθηκαν και για τα τρία δίκτυα που αναπτύξαμε, δηλαδή για το Αστικό ΚΤΕΛ Αλεξανδρούπολης, για το Αστικό ΚΤΕΛ Κομοτηνής και για το Αστικό ΚΤΕΛ Ξάνθης. Αυτό έγινε για την εξασφάλιση της αντικειμενικότητας μεταξύ των αποτελεσμάτων. Δηλαδή, έγινε για να αποφευχθεί η περίπτωση που η γεωγραφικότητα χτισίματος του αντίστοιχου γράφου, ενός δικτύου επηρεάσει τους χρόνους παροχής αποτελεσμάτων.

Συνοπτικά αναφέρουμε ότι ο τρόπος χτισίματος των γράφων μας αποδείχτηκε αρκετά χρονοβόρος αν τον συγκρίνουμε με άλλες σύγχρονες εφαρμογές open-source και μη. Οι χρόνοι που θα παρουσιάσουμε και παρακάτω αφορούν και τις διαδικασίες του χτισίματος αλλά και κυρίως τον χρόνο που έκανε ο αλγόριθμος Dijkstra να τρέξει εφόσον ο γράφος ήταν πλήρως στημένος. Ο χαμηλός χρόνος που χρειάστηκε ο Dijkstra για να τρέξει μας δίνει σημάδι ότι το στήσιμο των γράφων, δηλαδή το πλήθος των κόμβων και των ακμών, ήταν αρκετό καλό δεδομένου των πληροφοριών που έπρεπε να αναπαραστήσουμε. Σε μελλοντικές δουλειές, οι ερευνητές θα πρέπει να επικεντρωθούν στο πως να αναπτύξουν την ίδια δομή στο γράφο σε λιγότερο χρόνο. Αυτό μπορεί να συμβεί με πολλούς τρόπους όπως παραλληλοποίηση των διαδικασιών στους διάφορους πυρήνες του επεξεργαστή ή με τη χρήση εξυπνότερων δομών δεδομένων κατά τη διάρκεια του χτισίματος. Χωρίς περεταίρω καθυστέρηση να περάσουμε στη παρουσίαση των αποτελεσμάτων που παρουσιάζει το λογισμικό όταν το τρέξαμε σε μοντέρνο υπολογιστικό σύστημα με τα παρακάτω χαρακτηριστικά.

HP Pavilion system with specifications:
Processor: Intel i7 6700HQ, frequency 2.6Ghz
RAM Memory: 8 Gigabytes, frequency 1666Mhz
Hard Drive speed: 5400rpm

Υπενθυμίζουμε ξανά, ότι το τρέξιμο αφορούσε και τις τρεις πόλεις και έπαιρνε συγκεκριμένους χρόνους οι οποίοι κρίθηκαν σκόπιμοι για τη σύγκριση των γράφων. Τονίζουμε, ξανά σε αυτό το σημείο ότι δε συλλέχθηκαν δεδομένα που αφορούν τη γεωγραφικότητα των δικτύων. Δεν μας αφορούσαν στοιχεία όπως η συνεκτικότητα του δικτύου ή γεωγραφικότητα των στάσεων. Τα δεδομένα χρησιμοποιήθηκαν καθαρά ως ένα μέτρο σύγκρισης για το λογισμικό που αναπτύξαμε. Έτσι λοιπόν, τυχαία επιλέξαμε 10 σημεία για κάθε πόλη ως σημεία εκκίνησης και 10 ως σημεία τερματισμού. Όπου αναφέρονται σημεία, για την χρήση των graph builders θεωρούνται τα αντίστοιχα αρχεία GTFS. Για κάθε ζεύγος κρατήσαμε και ένα αντίστοιχο χρόνο εκκίνησης του ταξιδιού.

Ξεκινώντας με τα time-dependent graphs βρήκαμε ακριβώς τα shortest paths και τους αντίστοιχους χρόνους που χρειάστηκαν για να υπολογιστούν. Αμέσως μετά τρέξαμε για τα ίδια σημεία/γεγονότα που υπολογίστηκαν ως από τα time-dependent το χρόνο που χρειάζεται να τα υπολογίσει ο time-expanded. Έτσι πήραμε ακριβώς τους χρόνους που χρειαζόμασταν για τον υπολογισμό του ίδιου μονοπατιού και στις δύο περιπτώσεις.

Περνώντας στη παρουσίαση των αποτελεσμάτων να αναφερθούμε στο γεγονός ότι μπορεί να υπάρχει μία μικρή απόκλιση στους χρόνους που θα παρουσιάσουμε, καθώς για λόγους ευκολότερης και συντομότερης συγγραφής του κώδικα θυσιάστηκε ελάχιστη ακρίβεια κρατώντας τα ακέραια μέρη των αποτελεσμάτων που ήταν σε millisecond. Στις ώρες και τα λεπτά διατηρήσαμε ακρίβεια ενός δεκαδικού ψηφίου.

Συνολικός χρόνος τρεξίματος του προγράμματος:

Χρόνος σε χιλιοστά του δευτερόλεπτα	10098.581 χιλιοστά του δευτερόλεπτα
Χρόνος σε δευτερόλεπτα	168.3096833333333 δευτερόλεπτα
Χρόνος σε λεπτά	2.805161388888888 λεπτά

Πίνακας 2: Συνολικός χρόνος τρεξίματος του προγράμματος

Χρόνος για χτίσιμο των γράφων σε κάθε περίπτωση:

Για το time-expanded χτίσιμο:

Αλεξανδρούπολη	58,6 λεπτά
Κομοτηνή	55,3 λεπτά
Ξάνθη	53,2 λεπτά
Μέσος όρος	55,7 λεπτά

Πίνακας 3: Χρόνος για χτισίματος γράφων σε περίπτωση time-expanded

Για το time-dependent χτίσιμο:

Αλεξανδρούπολη	22.9 λεπτά
Κομοτηνή	21.8 λεπτά
Ξάνθη	20.2 λεπτά
Μέσος όρος	21.6333 λεπτά

Πίνακας 4: Χρόνος για χτισίματος γράφων σε περίπτωση time-dependent

*Πριν προχωρήσουμε υπενθυμίζουμε στον αναγνώστη τη βασική διαφορά μεταξύ τον δύο προσεγγίσεων. Το time-expanded graph δημιουργεί όλες τις συνδέσεις και είναι βασισμένο στα events ενώ το time-dependent κάνει χτίσιμο στην αρχή μόνο των κόμβων και χτίζει τις ακμές on-the-fly πριν από κάθε τρέξιμο του Dijkstra και με βάση τη χρονική στιγμή που το τρέχουμε.

Για αυτό λέγεται και time-dependent, γιατί η δομή του εξαρτάται σε μεγάλο βαθμό από τη χρονική στιγμή στην οποία θα απευθυνθούμε σε αυτό.

Χρόνος για το τρέξιμο του Dijkstra σε time-expanded γράφο με σημεία στην Αλεξανδρούπολη. Για κάθε ζεύγος, ο χρόνος που χρειάστηκε ο Dijkstra. Κάτω από τις στήλες με όνομα Σημείο 1 και Σημείο 2 έχουμε τα ονόματα των στάσεων (IDs) όπως τα περάσαμε από τα GTFS:

Από Σημείο 1	Προς Σημείο 2	Χρόνος
S12	S34	213 χιλιοστά του δευτερολέπτου
S13	S33	240 χιλιοστά του δευτερολέπτου
S35	S42	263 χιλιοστά του δευτερολέπτου
S34	S80	210 χιλιοστά του δευτερολέπτου
S33	S74	198 χιλιοστά του δευτερολέπτου
S128	S144	238 χιλιοστά του δευτερολέπτου
S7	S149	229 χιλιοστά του δευτερολέπτου
S62	S53	174 χιλιοστά του δευτερολέπτου
S28	S12	185 χιλιοστά του δευτερολέπτου
S23	S11	198 χιλιοστά του δευτερολέπτου
		Average = 214.8 χιλιοστά του δευτερολέπτου

Πίνακας 5: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-expanded γράφο στο δίκτυο του Αστικού Αλεξανδρούπολης

Χρόνοι για τρέξιμο του Dijkstra σε time-expanded γράφο μεταξύ δέκα ζευγαριών σημείων στο δίκτυο του Αστικού Κομοτηνής.

Από Σημείο 1	Προς Σημείο 2	Χρόνος
S11	S25	190 χιλιοστά του δευτερολέπτου
S10	S26	222 χιλιοστά του δευτερολέπτου
S33	S44	283 χιλιοστά του δευτερολέπτου
S12	S56	274 χιλιοστά του δευτερολέπτου
S13	S32	250 χιλιοστά του δευτερολέπτου
S38	S44	170 χιλιοστά του δευτερολέπτου

S76	S98	129 χιλιοστά του δευτερολέπτου
S93	S107	199 χιλιοστά του δευτερολέπτου
S31	S9	158 χιλιοστά του δευτερολέπτου
S17	S122	202 χιλιοστά του δευτερολέπτου
		Average = 207.7 χιλιοστά του δευτερολέπτου

Πίνακας 6: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-expanded γράφο στο δίκτυο του Αστικού Κομοτηνής

Παρακάτω παραθέτουμε τους χρόνους για το τρέξιμο του αλγόριθμου Dijkstra σε time-expanded γράφο για τη Ξάνθη:

Από Σημείο 1	Προς Σημείο 2	Χρόνος
S68	S78	174 χιλιοστά του δευτερολέπτου
S0	S100	201 χιλιοστά του δευτερολέπτου
S3	S11	178 χιλιοστά του δευτερολέπτου
S50	S58	215 χιλιοστά του δευτερολέπτου
S78	S71	229 χιλιοστά του δευτερολέπτου
S74	S85	198 χιλιοστά του δευτερολέπτου
S78	S68	199 χιλιοστά του δευτερολέπτου
S36	S32	205 χιλιοστά του δευτερολέπτου
S73	S31	213 χιλιοστά του δευτερολέπτου
S3	S10	235 χιλιοστά του δευτερολέπτου
		Average = 204.7 χιλιοστά του δευτερολέπτου

Πίνακας 7: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-expanded γράφο στο δίκτυο του Αστικού Ξάνθης

Περνάμε τώρα στα χρονικά αποτελέσματα για την time-dependent προσέγγιση και το πόσο γρήγορα τρέχει ένας κλασσικός αλγόριθμος σαν τον Dijkstra πάνω της. Για την Αλεξανδρούπολη:

Από Σημείο 1	Προς Σημείο 2	Χρόνος

S12	S34	22 χιλιοστά του δευτερολέπτου
S13	S33	24 χιλιοστά του δευτερολέπτου
S35	S42	23 χιλιοστά του δευτερολέπτου
S34	S80	24 χιλιοστά του δευτερολέπτου
S33	S74	25 χιλιοστά του δευτερολέπτου
S128	S144	21 χιλιοστά του δευτερολέπτου
S7	S149	21 χιλιοστά του δευτερολέπτου
S62	S53	20 χιλιοστά του δευτερολέπτου
S28	S12	23 χιλιοστά του δευτερολέπτου
S23	S11	22 χιλιοστά του δευτερολέπτου
		Average = 22,5 χιλιοστά του δευτερολέπτου

Πίνακας 8: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-dependent γράφο στο δίκτυο του Αστικού Αλεξανδρούπολης

Για τον time-dependent γράφο και τα αποτελέσματα στη Κομοτηνή:

Από Σημείο 1	Προς Σημείο 2	Χρόνος
S11	S25	24 χιλιοστά του δευτερολέπτου
S10	S26	24 χιλιοστά του δευτερολέπτου
S33	S44	23 χιλιοστά του δευτερολέπτου
S12	S56	24 χιλιοστά του δευτερολέπτου
S13	S32	22 χιλιοστά του δευτερολέπτου
S38	S44	23 χιλιοστά του δευτερολέπτου
S76	S98	24 χιλιοστά του δευτερολέπτου
S93	S107	21 χιλιοστά του δευτερολέπτου
S31	S9	25 χιλιοστά του δευτερολέπτου
S17	S122	24 χιλιοστά του δευτερολέπτου

		Average = 23,4 χιλιοστά του δευτερολέπτου

Πίνακας 9: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-dependent γράφο στο δίκτυο του Αστικού Κομοτηνής

Τελευταία αποτελέσματα που θα παραθέσουμε αφορούν τους χρόνους εκτέλεσης του Dijkstra στη πόλη της Ξάνθης:

Από Σημείο 1	Προς Σημείο 2	Χρόνος
S68	S78	22 χιλιοστά του δευτερολέπτου
S0	S100	21 χιλιοστά του δευτερολέπτου
S3	S11	23 χιλιοστά του δευτερολέπτου
S50	S58	21 χιλιοστά του δευτερολέπτου
S78	S71	21 χιλιοστά του δευτερολέπτου
S74	S85	24 χιλιοστά του δευτερολέπτου
S78	S68	25 χιλιοστά του δευτερολέπτου
S36	S32	24 χιλιοστά του δευτερολέπτου
S73	S31	26 χιλιοστά του δευτερολέπτου
S3	S10	21 χιλιοστά του δευτερολέπτου
		Average = 22,8 χιλιοστά του δευτερολέπτου

Πίνακας 10: Χρόνος που χρειάστηκε ο αλγόριθμος Dijkstra για να τρέξει σε time-dependent γράφο στο δίκτυο του Αστικού Ξάνθης

Στη συνέχεια θα συζητήσουμε αναλυτικότερα τα αποτελέσματα. Αρχικά, με μία γρήγορη ματιά στα αποτελέσματα, και ειδικότερα σε αυτά που αφορούν το Dijkstra και όχι το χτίσιμο του γράφου, μας δημιουργείται η εσφαλμένη εντύπωση ότι οι time dependent γράφοι είναι πολύ πιο γρήγοροι. Αυτό δεν ισχύει γιατί στην εργασία μας δεν χρονομετρήσαμε ακριβώς την όλη διαδικασία της ερώτησης (query) στο σύστημα. Δηλαδή οι timers που χρησιμοποιήσαμε μετρήσαν ακριβώς το χρόνο που κάνει να εκτελεστεί ο Dijkstra πάνω στο γράφο, ενώ αυτός είναι έτοιμος ανεξάρτητα της διαδικασίας που ακολουθήθηκε. Με αυτόν τον τρόπο, εμείς επιδιώξαμε να τεστάρουμε πως ο Dijkstra «πατάει» πάνω στη δομή του κάθε γράφου, δηλαδή στη συνδεσμολογία των κόμβων και των ακμών, που στη μία περίπτωση γίνεται με event - based τρόπο και στην άλλη με περισσότερο ως βάση τα γεωχωρικά δεδομένα.

Σε ένα πραγματικό σύστημα λογισμικού για να δούμε την ολοκληρωτική απόδοση των γράφων θα έπρεπε να μετράμε όχι μόνο το τρέξιμο του Dijkstra αλλά και το χρόνο που χρειάζεται στη περίπτωση μας η μέθοδος addPreQueryEdges(). Ωστόσο, είναι γεγονός ότι με μία τόσο μεγάλη διαφορά στο χρόνο εκτέλεσης, όσο «βαριά» από άποψη υπολογιστικού φόρτου για το σύστημα μας, και να ήταν η addPreQueryEdges(), δύσκολα θα κόντραρε ένας time-expanded γράφος ένα time-dependent.

Συνεχίζοντας την ανάλυση, να αναφερθούμε στους χρόνους χτισίματος του γράφου που εμφανέστατα είναι πολύ μικρότεροι στη περίπτωση του time-dependent. Το γεγονός αυτό τους θέτει ως καταλληλότερους υποψήφιους για χρήση δυναμικών δεδομένων σχεδιασμού ταξιδιού (Realtime-GTFS) μαζί με τα στατικά (GTFS). Ο μικρός χρόνος χτισίματος επιτρέπει στους μηχανικούς μιας διαδικτυακής πλατφόρμας σχεδιασμού ταξιδιού να χτίζουν το γράφο όσο συχνά θέλουν εμπεριέχοντας τα συνεχώς αλλαγμένα δυναμικά δεδομένα.

Τα αποτελέσματα που πήραμε, σε γενικές γραμμές, επιβεβαιώνουν όσο μπορεί να βρει κάποιος στη βιβλιογραφία. Το μέγεθος των time-expanded γράφων αποτελεί μεγάλο εμπόδιο στην αποτελεσματική τους χρήση. Ας πούμε στο δικό μας παράδειγμα, αν η κλάση μας για τον time expanded δημιουργεί τρεις κόμβους για κάθε event (γραμμή) του αρχείου stop_times.txt των GTFS, αυτό σημαίνει ότι από τις 72902 γραμμές-events του stop times θα προκύπταν 218,706 κόμβοι ενώ για την time-dependent μόνο 172 όσες δηλαδή και οι στάσεις του Αστικού ΚΤΕΛ Αλεξανδρούπολης.

Τελικώς θα αναφερθούμε σε μελλοντική δουλειά που πιστεύουμε ότι θα παίξει σημαντικό ρόλο στη ανάπτυξη αυτού του τομέα του σχεδιασμού ταξιδιού. Σημαντικό ρόλο θα έχει η εισαγωγή προυπολογιστικών τεχνικών και στους δύο τύπους γράφων. Επίσης, όπως έχουμε προαναφέρει, οι παραλληλοποίηση της εκτέλεσης του κώδικα, ειδικά με τη χρήση της κάρτας γραφικών θα ενισχύσει τις επιδόσεις και των δύο τύπων γράφων. Τελικά, πιστεύουμε ότι η εισαγωγή πιο μοντέρνων δομών δεδομένων στην κλάση addPreQueryEdges() θα μπορούσε να εξελίξει ακόμη περισσότερο την time-dependent προσέγγιση στους γράφους και να τους καθιερώσει ως το κυριότερο εργαλείο στο σχεδιασμό ταξιδιού.

Κεφάλαιο 6ο: Σύνοψη

Στο κεφάλαιο αυτό γίνεται μία σύνοψη όλων όσων αναφέρθηκαν στα 5 κυρίως κεφάλαια της παρούσας διπλωματικής εργασίας.

Αρχικά στο 1° κατά σειρά κεφάλαιο αναφέρθηκε η ουσία του προβλήματος που μας απασχόλησε και δεν ήταν άλλο από το πρόβλημα σχεδιασμού ταξιδιού. Επίσης έγινε αναφορά στα βήματα που ακολουθούνται κατά την επίλυσή του. Τα βήματα αυτά επιγραμματικά είναι η ανάπτυξη του δικτύου, η μετατροπή αυτού σε μορφή κατάλληλη για περαιτέρω επεξεργασία και η χρησιμοποίηση αυτού από έναν αλγόριθμο που θα υπολογίζει την λύση.

Στο 2° κεφάλαιο λοιπόν έγινε παρουσίαση της έννοιας των δικτύων και η ανάπτυξη των δικτύων που απασχόλησαν την διπλωματική μας εργασία. Η ανάπτυξη των δικτύων έγινε από πρωταρχικό στάδιο που είναι η συλλογή των πρωτογενών δεδομένων μέχρι και την ψηφιοποίησή του.

Στο 3° κεφάλαιο έγινε αναφορά στην έννοια των γράφων. Οι γράφοι είναι μία μορφή παρουσίασης των δικτύων. Επιλέγεται αυτή η μορφή διότι σε αυτή την μορφή είναι πιο εύκολο να χρησιμοποιηθούν τα δεδομένα μας στο επόμενο βήμα. Εμείς επιλέξαμε δύο διαφορετικούς τρόπους μετατροπής σε γράφους, οι οποίοι και κρίνονται για την ταχύτητά τους και την λειτουργικότητά τους.

Στο 4° κεφάλαιο έγινε πλήρης ανάπτυξη του προβλήματος που μας απασχόλησε, καθώς και οι διαδικασίες που χρειάζονται για την εξεύρεση της βέλτιστης λύσης. Στο κεφάλαιο αυτό δηλαδή εκτός από το πρόβλημα, έγινε αναφορά στον αλγόριθμο επίλυσής του, καθώς και στον κώδικα τον οποίο χτίσαμε, σύμφωνα με τον οποίον αναπτύσσεται η κατασκευή των γράφων με δύο προσεγγίσεις.

Τέλος στο κεφάλαιο 5° έγινε μία παρουσίαση αποτελεσμάτων. Στα πλαίσια αυτού του κεφαλαίου, τρέξαμε τον κώδικά μας για συγκεκριμένο αριθμό σημείων σε όλα τα δίκτυα τα οποία αναπτύξαμε. Τα αποτελέσματα σχολιάστηκαν ως προς τρόπος χτισίματος γράφου είναι αποδοτικότερος και ταχύτερος.

ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα Α – 1η κλάση του κώδικα, main.java

```
1  package gr.greenyourmove.gtfsGraph;
2
3  import java.io.*;
4  import java.util.*;
5
6  import org.jgrapht.alg.DijkstraShortestPath;
7  import org.jgrapht.graph.*;
8
9  public class Main {
10
11     public static void main(String[] args) throws IOException, ClassNotFoundException {
12
13         long startTime14 = System.currentTimeMillis();
14
15         //filepaths to GTFS files that we will need
16         String filepathAlexStopTimes = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_alex\\
   stop_times.txt";
17         String filepathKomoStopTimes = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_komo\\
   stop_times.txt";
18         String filepathXanthiStopTimes = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_xanthi
   \\stop_times.txt";
19
20         //filepaths to stops GTFS files
21         String filepathAlexStops = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_alex\\stops.
   txt";
22         String filepathKomoStops = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_komo\\stops.
   txt";
23         String filepathXanthiStops = "C:\\Users\\dimoutsis\\Desktop\\untitled\\src\\gr\\gtfsGraph\\GTFS\\gtfs_xanthi\\
   stops.txt";
24
25         //here we will read,parse and save to ArrayLists the data we choose to run Dijkstra with
26         ArrayList<ArrayList<String>> pointsAlex = TimeExpandedGraph.readGTFS("C:\\Users\\dimoutsis\\Desktop\\untitled\\
   src\\gr\\gtfsGraph\\GTFS\\gtfs_alex\\stop_times.txt");
27         ArrayList<ArrayList<String>> pointsKomo = TimeExpandedGraph.readGTFS("C:\\Users\\dimoutsis\\Desktop\\untitled\\
   src\\gr\\gtfsGraph\\GTFS\\gtfs_alex\\stop_times.txt");
28         ArrayList<ArrayList<String>> pointsXanthi = TimeExpandedGraph.readGTFS("C:\\Users\\dimoutsis\\Desktop\\untitled
   \\src\\gr\\gtfsGraph\\GTFS\\gtfs_alex\\stop_times.txt");
29
30         //actions for time-dependent graph start here
31
32         //now we will create three graph objects using the TimeDependentGraph instance
33         System.out.println("We have started constructing the Time-dependent Graphs!");
34         long startTime0 = System.currentTimeMillis();
35         //initiation of Alexandroupoli's time-dependent graph
36         TimeDependentGraph<ArrayList<ArrayList<String>>, DefaultWeightedEdge> timeDependentGraphInstanceAlex = new
   TimeDependentGraph<>(DefaultWeightedEdge.class);
37         List preQueryListsAlex = timeDependentGraphInstanceAlex.initiateTimeDependentGraph(filepathAlexStopTimes,
   filepathAlexStops);
38         //Oi duo seires kodika pou akolouthoun aforoun tis plirofories pou xreiazomaste gia to time-dependent grafo kai
   to sxhmatismo ton akmwn kathe fora prin to Dijkstra
39         ArrayList<ArrayList<ArrayList<String>>> timeDependentArrayAlex = (ArrayList<ArrayList<ArrayList<String>>>)
   preQueryListsAlex.get(0);
40         ArrayList<String> stopPairsAlex = (ArrayList<String>) preQueryListsAlex.get(1);
41
42         //initiation of Komotini's time dependent graph
43         TimeDependentGraph<ArrayList<ArrayList<String>>, DefaultWeightedEdge> timeDependentGraphInstanceKomo = new
   TimeDependentGraph<>(DefaultWeightedEdge.class);
44         List preQueryListsKomo = timeDependentGraphInstanceKomo.initiateTimeDependentGraph(filepathKomoStopTimes,
   filepathKomoStops);
45         ArrayList<ArrayList<ArrayList<String>>> timeDependentArrayKomo = (ArrayList<ArrayList<ArrayList<String>>>)
   preQueryListsKomo.get(0);
46         ArrayList<String> stopPairsKomo = (ArrayList<String>) preQueryListsKomo.get(1);
47
48         //initiation of Xanthi's time-dependent graph
49         TimeDependentGraph<ArrayList<ArrayList<String>>, DefaultWeightedEdge> timeDependentGraphInstanceXanthi = new
   TimeDependentGraph<>(DefaultWeightedEdge.class);
50         List preQueryListsXanthi = timeDependentGraphInstanceXanthi.initiateTimeDependentGraph(filepathXanthiStopTimes,
   filepathXanthiStops);
51         ArrayList<ArrayList<ArrayList<String>>> timeDependentArrayXanthi = (ArrayList<ArrayList<ArrayList<String>>>)
   preQueryListsXanthi.get(0);
52         ArrayList<String> stopPairsXanthi = (ArrayList<String>) preQueryListsXanthi.get(1);
53
54         long endTime0 = System.currentTimeMillis();
55         long totalTime0 = endTime0 - startTime0;
56
57         //we use the time-dependent constructor to create two new vertices from the information of the files and do the
   routing
58         ArrayList<ArrayList<String>> vertex1 = new ArrayList<>();
59         ArrayList<ArrayList<String>> vertex2 = new ArrayList<>();
60         ArrayList<ArrayList<Object>> toStorePathLengthAndTimeToExecuteAlex = new ArrayList<ArrayList<Object>>();
61         ArrayList<ArrayList<Object>> toStorePathLengthAndTimeToExecuteKomo = new ArrayList<ArrayList<Object>>();
62         ArrayList<ArrayList<Object>> toStorePathLengthAndTimeToExecuteXanthi = new ArrayList<ArrayList<Object>>();
63
64         long startTime1 = System.currentTimeMillis();
65         for (ArrayList<String> setOfPoints:pointsAlex) {
66             toStorePathLengthAndTimeToExecuteAlex.add(new ArrayList<Object>());
67             timeDependentGraphInstanceAlex.addPreQueryEdges(setOfPoints.get(2), setOfPoints.get(0), setOfPoints.get(1),
     timeDependentArrayAlex, stopPairsAlex);
68             vertex1 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(0));
69             vertex2 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(1));
```

```
70              //code for the dijkstra to execute on the time-exapnded graph
71              long startTime2 = System.currentTimeMillis();
72              List shortest_path = DijkstraShortestPath.findPathBetween(timeDependentGraphInstanceAlex, vertex1, vertex2)
    ;
73              if (shortest_path != null) {
74                  System.out.println("Shortest path:");
75                  System.out.println(shortest_path);      //edw kanei print to shortest path pou upologizei
76              }
77              long endTime2 = System.currentTimeMillis();
78              toStorePathLengthAndTimeToExecuteAlex.get(toStorePathLengthAndTimeToExecuteAlex.size()-1).add((endTime2-
    startTime2));
79              toStorePathLengthAndTimeToExecuteAlex.get(toStorePathLengthAndTimeToExecuteAlex.size()-1).add((
    shortest_path.getPathLength));
80          }
81          long endTime1 = System.currentTimeMillis();
82          long totalTime1=endTime1-startTime1;
83
84          //to treksimo ton algorithmwn gia ti Komotini
85          //auto to block kodika einai idio me to apo panw mono pou exei se graph kai listes oti afora ti Komotini
86          long startTime3 = System.currentTimeMillis();
87          for (ArrayList<String> setOfPoints:pointsKomo) {
88              toStorePathLengthAndTimeToExecuteKomo.add(new ArrayList<Object>());
89              timeDependentGraphInstanceKomo.addPreQueryEdges(setOfPoints.get(2), setOfPoints.get(0), setOfPoints.get(1),
      timeDependentArrayKomo, stopPairsKomo);
90              vertex1 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(0));   //parathroume oti edw den
    xreiazetai na kalesoume to composeSimpleVertex
91              vertex2 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(1));   //panw sto instance kapoion apo
    tous grafous alla kateutheian panw sti klasi
92              //code for the dijkstra to execute on the time-exapnded graph
93              long startTime4 = System.currentTimeMillis();
94              List shortest_path = DijkstraShortestPath.findPathBetween(timeDependentGraphInstanceKomo, vertex1, vertex2)
    ;
95              if (shortest_path != null) {
96                  System.out.println("Shortest path:");
97                  System.out.println(shortest_path);      //edw kanei print to shortest path pou upologizei
98              }
99              long endTime4 = System.currentTimeMillis();
100             toStorePathLengthAndTimeToExecuteKomo.get(toStorePathLengthAndTimeToExecuteKomo.size()-1).add((endTime4-
    startTime4));
101             toStorePathLengthAndTimeToExecuteKomo.get(toStorePathLengthAndTimeToExecuteKomo.size()-1).add((
    shortest_path.getPathLength));
102         }
103         long endTime3 = System.currentTimeMillis();
104         long totalTime3=endTime3-startTime3;
105
106         //to treksimo ton algorithmwn gia ti Xanthi
107         //auto to block kodika einai idio me ta duo apo panw mono pou exei se graph kai listes oti afora ti Xanthi
108         long startTime5 = System.currentTimeMillis();
109         for (ArrayList<String> setOfPoints:pointsXanthi) {
110             toStorePathLengthAndTimeToExecuteXanthi.add(new ArrayList<Object>());
111             timeDependentGraphInstanceXanthi.addPreQueryEdges(setOfPoints.get(2), setOfPoints.get(0), setOfPoints.get(1
    ),  timeDependentArrayXanthi, stopPairsXanthi);
112             vertex1 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(0));   //parathroume oti edw den
    xreiazetai na kalesoume to composeSimpleVertex
113             vertex2 = TimeDependentGraph.composeSimpleVertex(setOfPoints.get(1));   //panw sto instance kapoion apo
    tous grafous alla kateutheian panw sti klasi
114             //code for the dijkstra to execute on the time-exapnded graph
115             long startTime6 = System.currentTimeMillis();
116             List shortest_path = DijkstraShortestPath.findPathBetween(timeDependentGraphInstanceXanthi, vertex1,
    vertex2);
117             if (shortest_path != null) {
118                 System.out.println("Shortest path:");
119                 System.out.println(shortest_path);      //edw kanei print to shortest path pou upologizei
120             }
121             long endTime6 = System.currentTimeMillis();
122             toStorePathLengthAndTimeToExecuteXanthi.get(toStorePathLengthAndTimeToExecuteXanthi.size()-1).add((endTime6
    -startTime6));
123             toStorePathLengthAndTimeToExecuteXanthi.get(toStorePathLengthAndTimeToExecuteXanthi.size()-1).add((
    shortest_path.getPathLength));
124         }
125         long endTime5 = System.currentTimeMillis();
126         long totalTime5 = endTime5-startTime5;
127
128         TimeExpandedGraph<ArrayList, DefaultWeightedEdge> timeExpandedGraphInstanceAlex = new TimeExpandedGraph<>(
    DefaultWeightedEdge.class);
129         TimeExpandedGraph<ArrayList, DefaultWeightedEdge> timeExpandedGraphInstanceKomo = new TimeExpandedGraph<>(
    DefaultWeightedEdge.class);
130         TimeExpandedGraph<ArrayList, DefaultWeightedEdge> timeExpandedGraphInstanceXanthi = new TimeExpandedGraph<>(
    DefaultWeightedEdge.class);
131         //The code for initiating the time-expanded graph is much more concetrated than time-dependent graph related
132         //time-expanded graph in order to be built needs much simpler processes than time-dependent, but in great
    extent
133
134         long startTime7 = System.currentTimeMillis();
135         timeExpandedGraphInstanceAlex.initiateTimeExpandedGraph(filepathAlexStopTimes,filepathAlexStops);
136         timeExpandedGraphInstanceAlex.initiateTimeExpandedGraph(filepathKomoStopTimes,filepathKomoStops);
137         timeExpandedGraphInstanceAlex.initiateTimeExpandedGraph(filepathXanthiStopTimes,filepathXanthiStops);
138         long endTime7 = System.currentTimeMillis();
139         long totalTime7 = endTime7-startTime7;
140
```

```
141         ArrayList<Integer> storeTimesAlexTimeExpanded = new ArrayList<>(); //here we keep the time that needed each
    query to run
142         Integer counterAlex = 0;
143         long startTime8 = System.currentTimeMillis();
144         //code for the dijkstra to execute on the time-exapnded graph for Alexandroupoli
145         for (ArrayList<String> setOfPoints:pointsAlex) {
146             //we need to identify the starting and ending vertices in the graph
147             //the starting is in the files and for the ending we need to use the
148             //lists we created in the time-dependent approach
149             Set<ArrayList> vertices =  (Set) timeExpandedGraphInstanceAlex.vertexSet();
150             for (ArrayList<ArrayList<String>> vertex3:vertices) {
151                 Integer timeParsed = TimeExpandedGraph.calcTime(setOfPoints.get(2));
152                 if (setOfPoints.get(0).equals(vertex3.get(1).get(1)) && timeParsed==Integer.parseInt(vertex3.get(2).get
    (1))&&Integer.parseInt(vertex3.get(2).get(0))==0) {
153                     for (ArrayList<ArrayList<String>> vertex4:vertices) {
154                         Integer timeOfTargetVertex = (Integer) toStorePathLengthAndTimeToExecuteAlex.get(counterAlex).
    get(1);
155                         Integer timeOfTargetVertex1 = timeOfTargetVertex + timeParsed;
156                         if (setOfPoints.get(1).equals(vertex4.get(1).get(1)) && (timeOfTargetVertex1 == Integer.
    parseInt(vertex4.get(2).get(1))) && Integer.parseInt(vertex4.get(2).get(0)) == 2) {
157                             long startTime9 = System.currentTimeMillis();
158                             List shortest_path = DijkstraShortestPath.findPathBetween(timeExpandedGraphInstanceAlex,
    vertex3, vertex4);
159                             long endTime9 = System.currentTimeMillis();
160                             long totalTime9 = endTime9 - startTime9;
161                             storeTimesAlexTimeExpanded.add((int) totalTime9);
162                         }
163                     }
164                 }
165             }
166             counterAlex = counterAlex + 1;
167         }
168         long endTime8 = System.currentTimeMillis();
169         long totalTime8 = endTime8-startTime8;
170
171         ArrayList<Integer> storeTimesKomoTimeExpanded = new ArrayList<>(); //here we keep the time that needed each
    query to run
172         Integer counterKomo = 0;
173         long startTime10 = System.currentTimeMillis();
174         //code for the dijkstra to execute on the time-exapnded graph for Komotini
175         for (ArrayList<String> setOfPoints:pointsKomo) {
176             Set<ArrayList> vertices =  (Set) timeExpandedGraphInstanceKomo.vertexSet();
177             for (ArrayList<ArrayList<String>> vertex5:vertices) {
178                 Integer timeParsed = TimeExpandedGraph.calcTime(setOfPoints.get(2));
179                 if (setOfPoints.get(0).equals(vertex5.get(1).get(1)) && timeParsed==Integer.parseInt(vertex5.get(2).get
    (1))&&Integer.parseInt(vertex5.get(2).get(0))==0) {
180                     for (ArrayList<ArrayList<String>> vertex6:vertices) {
181                         Integer timeOfTargetVertex = (Integer) toStorePathLengthAndTimeToExecuteKomo.get(counterKomo).
    get(1);
182                         Integer timeOfTargetVertex1 = timeOfTargetVertex + timeParsed;
183                         if (setOfPoints.get(1).equals(vertex6.get(1).get(1)) && (timeOfTargetVertex1 == Integer.
    parseInt(vertex6.get(2).get(1))) && Integer.parseInt(vertex6.get(2).get(0)) == 2) {
184                             long startTime11 = System.currentTimeMillis();
185                             List shortest_path = DijkstraShortestPath.findPathBetween(timeExpandedGraphInstanceKomo,
    vertex5, vertex6);
186                             System.out.println(shortest_path);
187                             long endTime11 = System.currentTimeMillis();
188                             long totalTime11 = endTime11 - startTime11;
189                             storeTimesKomoTimeExpanded.add((int) totalTime11);
190                         }
191                     }
192                 }
193             }
194             counterKomo = counterKomo + 1;
195         }
196         long endTime10 = System.currentTimeMillis();
197         long totalTime10 = endTime10-startTime10;
198
199         ArrayList<Integer> storeTimesXanthiTimeExpanded = new ArrayList<>(); //here we keep the time that needed each
    query to run
200         Integer counterXanthi = 0;
201         long startTime12 = System.currentTimeMillis();
202         //code for the dijkstra to execute on the time-exapnded graph for Xanthi
203         for (ArrayList<String> setOfPoints:pointsXanthi) {
204             Set<ArrayList> vertices =  (Set) timeExpandedGraphInstanceXanthi.vertexSet();
205             for (ArrayList<ArrayList<String>> vertex7:vertices) {
206                 Integer timeParsed = TimeExpandedGraph.calcTime(setOfPoints.get(2));
207                 if (setOfPoints.get(0).equals(vertex7.get(1).get(1)) && timeParsed==Integer.parseInt(vertex7.get(2).get
    (1))&&Integer.parseInt(vertex7.get(2).get(0))==0) {
208                     for (ArrayList<ArrayList<String>> vertex8:vertices) {
209                         Integer timeOfTargetVertex = (Integer) toStorePathLengthAndTimeToExecuteXanthi.get(
    counterXanthi).get(1);
210                         Integer timeOfTargetVertex1 = timeOfTargetVertex + timeParsed;
211                         if (setOfPoints.get(1).equals(vertex8.get(1).get(1)) && (timeOfTargetVertex1 == Integer.
    parseInt(vertex8.get(2).get(1))) && Integer.parseInt(vertex8.get(2).get(0)) == 2) {
212                             long startTime13 = System.currentTimeMillis();
213                             List shortest_path = DijkstraShortestPath.findPathBetween(timeExpandedGraphInstanceXanthi,
    vertex7, vertex8);
214                             System.out.println(shortest_path);
215                             long endTime13 = System.currentTimeMillis();
```

```
                            long totalTime13 = endTime13 - startTime13;
                            storeTimesXanthiTimeExpanded.add((int) totalTime13);
                        }
                    }
                }
            }
            counterXanthi = counterXanthi + 1;
        }
        long endTime12 = System.currentTimeMillis();
        long totalTime12 = endTime12-startTime12;

        //now our program will print out the time it took every of each components to run
        System.out.println("Printing out results:");
        System.out.println("Time needed to build the 3 time-dependent graphs represented by totalTime0: " + totalTime0)
;
        System.out.println("Time needed to run Dijkstra on the time-Dependent graph for all of our points in
Alexandroupoli represented by totalTime1: " + totalTime1);
        System.out.println("Time needed to run Dijkstra on the time-Dependent graph for all of our points in Komotini
represented by totalTime3: " + totalTime3);
        System.out.println("Time needed to run Dijkstra on the time-Dependent graph for all of our points in Xanthi
represented by totalTime5: " + totalTime5);
        System.out.println("Time needed for Dijkstra to run on the time dependent graph and for each of our single pair
 of points in Alexadroupoli: ");
        for (ArrayList<Object> time:toStorePathLengthAndTimeToExecuteAlex) {
            System.out.println(time.get(0));
        }
        System.out.println("Time needed for Dijkstra to run on the time dependent graph and for each of our single pair
 of points in Komotini: ");
        for (ArrayList<Object> time:toStorePathLengthAndTimeToExecuteKomo) {
            System.out.println(time.get(0));
        }
        System.out.println("Time needed for Dijkstra to run on the time dependent graph and for each of our single pair
 of points in Xanthi: ");
        for (ArrayList<Object> time:toStorePathLengthAndTimeToExecuteXanthi) {
            System.out.println(time.get(0));
        }
        System.out.println("Time needed to build the 3 time-expanded graphs represented by totalTime7: " + totalTime7);
        System.out.println("Time needed to run Dijkstra on the time-Expanded graph for all of our points in
Alexandroupoli represented by totalTime8: " + totalTime8);
        System.out.println("Time needed to run Dijkstra on the time-Expanded graph for all of our points in Komotini
represented by totalTime10: " + totalTime10);
        System.out.println("Time needed to run Dijkstra on the time-Expanded graph for all of our points in Xanthi
represented by totalTime12: " + totalTime12);

        System.out.println("Time needed for Dijkstra to run on the time expanded graph and for each of our single pair
of points in Alexadroupoli: ");
        for (Integer time:storeTimesAlexTimeExpanded) {
            System.out.println(time);
        }
        System.out.println("Time needed for Dijkstra to run on the time expanded graph and for each of our single pair
of points in Komotini: ");
        for (Integer time:storeTimesKomoTimeExpanded) {
            System.out.println(time);
        }
        System.out.println("Time needed for Dijkstra to run on the time expanded graph and for each of our single pair
of points in Ksanthi: ");
        for (Integer time:storeTimesXanthiTimeExpanded) {
            System.out.println(time);
        }

        long endTime14 = System.currentTimeMillis();
        long totalTime14 = endTime14 - startTime14;
        System.out.println("Time needed for our program to run on the whole: " + totalTime14);
    }
}
```

Παράρτημα Β – 2η κλάση του κώδικα, TimeExpandedGraph.java

```java
package gr.greenyourmove.gtfsGraph;

import java.util.*;

import org.jgrapht.*;
import org.jgrapht.graph.*;

import java.io.File;                    //we will use the java.util.Scanner to
import java.io.FileNotFoundException;   //the csv files(GTFS)


//is this the way to go from generic to the type of object I want as parameters
public class TimeExpandedGraph<ArrayList, DefaultWeightedEdge>
        extends DirectedWeightedMultigraph<List, DefaultWeightedEdge>
        implements WeightedGraph<List, DefaultWeightedEdge>
{
    private static final long serialVersionUID = -2209049947111726774L;

    public TimeExpandedGraph(Class<? extends DefaultWeightedEdge> edgeClass) {
        super(edgeClass);
    }
    //in order to add an event, we need to extract from the GTFS file the following data fields from each line
    //for each event
    //trip_id,arrival_time,departure_time,stop_id,stop_sequence,pickup_type,drop_off_type
    //with addEvent we create three edges in our graph, one for the arrival,one for the transfer-change and one for the
    //departure

    //parses the stop_times.txt file with the events and creates the graph
    public void initiateTimeExpandedGraph(String pathToStopTimesFile, String pathToStopsFile) throws
FileNotFoundException {
        ArrayList<ArrayList<String>> stopTimes = new ArrayList<ArrayList<String>>();    //contains the final parsed file
        stopTimes = readGTFS(pathToStopTimesFile);
        ArrayList<ArrayList<String>> stops = new ArrayList<ArrayList<String>>();    //contains the final parsed file
        stops = readGTFS(pathToStopsFile);
        //start of graph initiatiation
        for (ArrayList i:stopTimes)
        {
            String tripIDExtracted = (String) i.get(0);
            String arrivalTimeExtracted = (String) i.get(1);
            String departureTimeExtracted = (String) i.get(2);
            String stopIDExtracted = (String) i.get(3);
            String stopSequenceExtracted = (String) i.get(4);
            this.addEvent(tripIDExtracted, arrivalTimeExtracted, departureTimeExtracted, stopIDExtracted,
stopSequenceExtracted);
        }
        //creation of the edges
        this.addEdges(stopTimes, stops);
    }

    //called upon every line of the GTFS file stopTimes to initiate an event
    public void addEvent(String tripID, String arrivalTime, String departureTime, String stopID, String stopSequence) {
        int arrivalTimeInSeconds = calcTime(arrivalTime);     //calculating arrival time in seconds
        int departureTimeInSeconds = calcTime(departureTime); //calculating departure time in seconds
        String strValueOfArrivalTimeInSeconds = String.valueOf(arrivalTimeInSeconds);
        String strValueOfWaitingTimeInSeconds = String.valueOf(arrivalTimeInSeconds+120);
        String strValueOfDepartureTimeInSeconds = String.valueOf(departureTimeInSeconds);
        for (int i = 0; i < 3; i++)
        {
            ArrayList<ArrayList<String>> listOfLists = new ArrayList<>();       //list that stores the rest of the lists
 and is the vertex
            ArrayList<String> vertexID = new ArrayList<>();        //list that stores the vertexID
            ArrayList<String> strList = new ArrayList<>();         //list that stores tripID and the stopID, the string
values
                                                                   //of an event
            ArrayList<String> intList = new ArrayList<>();       //In this list we store the arrival and departure times
            //adding lists to the listOfLists object that includes the rest of the lists
            listOfLists.add(vertexID);
            listOfLists.add(strList);
            listOfLists.add(intList);
            //inserting appropriate values into the lists
            vertexID.add(tripID + stopSequence + i);        //we insert to the list the vertexID
            strList.add(tripID);
            strList.add(stopID);
            strList.add(stopSequence);
            //adding the type of vertex and departure/arrival times to doubleList
            String vertexType = String.valueOf(i);
            intList.add(vertexType);
            if (i == 0)
            {
                intList.add(strValueOfArrivalTimeInSeconds);        //adding arrivalTime
            } else if (i == 1) {
                intList.add(strValueOfWaitingTimeInSeconds);    //adding arrivalTime + 3 minutes of minimum waiting time
if
                //a person wants to change vehicle
            } else {
                intList.add(strValueOfDepartureTimeInSeconds);      //adding departureTime
            }
            this.addVertex(listOfLists);                  //add the vertex to the current graph
        }
    }
```

```
86
87      //after you have added some vertices to the graph, it connects them with the corresponding edges
88      //it takes no input parameters. It reads the vertices of the graph it is used on.
89      public void addEdges(ArrayList<ArrayList<String>> stopTimes, ArrayList<ArrayList<String>> stops) {
90          //i will create a matrix containing a list for each stop with the events that happen in that stop
91          //in a chronological order
92          Integer stopsCounter =  -1;
93          ArrayList<ArrayList<Integer>> timeMatrixArrival = new ArrayList();
94          ArrayList<ArrayList<Integer>> timeMatrixDeparture = new ArrayList();
95          for (ArrayList<String> stop:stops)
96          {
97              timeMatrixArrival.add(new ArrayList<Integer>());
98              timeMatrixDeparture.add(new ArrayList<Integer>());
99              stopsCounter = stopsCounter + 1;
100             for (ArrayList<String> event:stopTimes) {
101                 if (stop.get(0).equals(event.get(3))) {      //we don't use "==" operator but ".equal()" for strings
102                     timeMatrixArrival.get(stopsCounter).add(calcTime(event.get(1))+120);
103                     timeMatrixDeparture.get(stopsCounter).add(calcTime(event.get(2)));
104                 }
105             }
106         }
107         //sorting of the timeMatrix matrices, timeMatrixArrival has the same length as timeMatrixDeparture
108         for (int i=0;i<timeMatrixArrival.size();i++) {
109             Collections.sort(timeMatrixArrival.get(i));
110             Collections.sort(timeMatrixDeparture.get(i));
111         }
112
113         //with the code above we start adding the actual edges to the graph
114         Set<ArrayList> vertices =  (Set) this.vertexSet();
115         for (ArrayList<ArrayList> vertex : vertices)
116         {
117             //get the three lists of each vertex
118             //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
119
120             ArrayList<String> a = (ArrayList<String>) vertex.get(0);       //get the first array, the vertexID
121             ArrayList<String> b = (ArrayList<String>) vertex.get(1);       //get the second array, the strList
122             ArrayList<String> c = (ArrayList<String>) vertex.get(2);     //get the third array, the doubleList
123             //we parse and extract the properties of each vertex
124             String tripIDParsed = b.get(0);
125             String stopIDParsed = b.get(1);
126             Integer stopSequenceParsed = Integer.parseInt(b.get(2));
127             Integer vertexTypeParsed =  Integer.parseInt(c.get(0));
128             Integer timeParsed =  Integer.parseInt(c.get(1));
129
130             if (vertexTypeParsed==0) {                                      //if the vertex we check is a arrival vertex
131                 //in order to get to the waiting event node and the departure event node, we need clone the node we are
132                 //are currently,parse it and change the appropriate attributes in order to pass it as parameter to the
133                 //edge constructor and correctly correspond the nodes we want
134                 //in order for the waiting copy to correspond to the waiting event we need to change the time of the
135                 //Occurance of the event to departureTime + 180, the type of vertex to 1 and of course the vertexID
136                 //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
137                 ArrayList<ArrayList<String>> listOfLists = new ArrayList<>();
138                 ArrayList<String> vertexID = new ArrayList<>();
139                 ArrayList<String> strList = new ArrayList<>();
140                 ArrayList<String> intList = new ArrayList<>();
141                 listOfLists.add(vertexID);
142                 listOfLists.add(strList);
143                 listOfLists.add(intList);
144                 vertexID.add(b.get(0) + b.get(2) + 1);
145                 strList.add(b.get(0));
146                 strList.add(b.get(1));
147                 strList.add(b.get(2));
148                 intList.add("1");
149                 Integer newWaitingTime = timeParsed + 120;
150                 String stringNewWaitingTime = String.valueOf(newWaitingTime);
151                 intList.add(stringNewWaitingTime);
152                 ArrayList<ArrayList<String>> waitingVertex = listOfLists;
153                 this.setEdgeWeight(this.addEdge( vertex, waitingVertex), 120); //adding edge
154
155                 //and for the connection with the departure node
156                 //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
157                 ArrayList<ArrayList<String>> listOfListsDestination = new ArrayList<>();
158                 ArrayList<String> vertexIDDestination = new ArrayList<>();
159                 ArrayList<String> strListDestination = new ArrayList<>();
160                 ArrayList<String> intListDestination = new ArrayList<>();
161                 listOfListsDestination.add(vertexIDDestination);
162                 listOfListsDestination.add(strListDestination);
163                 listOfListsDestination.add(intListDestination);
164                 vertexIDDestination.add(b.get(0) + c.get(2) + "2");
165                 strListDestination.add(b.get(0));
166                 strListDestination.add(b.get(1));
167                 strListDestination.add(b.get(2));
168                 intListDestination.add("2");
169                 intListDestination.add(c.get(1));     //departure time is always the same with arrival time
170                 ArrayList<ArrayList<String>> departureVertex = listOfListsDestination;
171                 this.setEdgeWeight(this.addEdge( vertex, departureVertex), 0); //adding edge
172             }else if (vertexTypeParsed==1) {                                //if the vertex we check is a arrival
    vertex
173                 //in order to ensure the right connections between events we need to make sure that each node-event
174                 //connects to the next one in the time horizon. We begin by reading the stops.txt file and then sort
```

```
175                 //the time events for each of the stations in chronological order
176                 String[] tokensStop = b.get(1).split("");                   //we parse the stopID and keep the number to
    use as index
177                 int characterCounter = 0;
178                 String stopIDParsed1= "";
179                 for (String character:tokensStop) {
180                     if (characterCounter>0){
181                         stopIDParsed1 = stopIDParsed1 + character;
182                     }
183                     characterCounter += 1;
184                 }
185                 Integer stopIDIndex = Integer.parseInt(stopIDParsed1);  //here's the index
186
187                 ArrayList<Integer> arrivalStopList = timeMatrixArrival.get(stopIDIndex);        //vazontas to index sth
     lista me tis listes
188                 ArrayList <Integer> departureStopList = timeMatrixDeparture.get(stopIDIndex);    //pairnoume tis
    epithimites listes me tous
189
190                 int dayCounter = 0;                                                                                 //
    diadoxikous xronous
191                 for (int i:arrivalStopList) {
192                     dayCounter = dayCounter + 1;
193                     if (timeParsed < i) {
194                         dayCounter = dayCounter - 1;                                                    //make sure that the
    dayCounter does not get equal to the size of the arrivalStopList if we find a next stop
195                         for (ArrayList<ArrayList<String>> waitingVertex : vertices) {
196                             //structure of waitingVertexObject
197                             //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
198                             ArrayList<String> c = (ArrayList<String>) waitingVertex.get(0);      //get the first array
    , the vertexID
199                             ArrayList<String> e = (ArrayList<String>) waitingVertex.get(1);      //get the second array
    , the strList
200                             ArrayList<String> f = (ArrayList<String>) waitingVertex.get(2);    //get the third array,
    the doubleList
201                             String stopIDParsed2 = e.get(1);
202                             String vertexTypeParsed21 = f.get(0);
203                             Integer vertexTypeParsed22 = Integer.parseInt(f.get(0));
204                             String timeParsed21 = f.get(1);
205                             Integer timeParsed22 = Integer.parseInt(f.get(1));
206                             int cost1 = timeParsed22 - timeParsed;
207                             if (stopIDParsed2.equals(stopIDParsed) && vertexTypeParsed22 == vertexTypeParsed &&
    timeParsed22 == i) {
208                                 this.setEdgeWeight(this.addEdge(vertex, waitingVertex), cost1); //adding edge
209                             }
210                         }
211                         break;
212                     } else if (dayCounter == arrivalStopList.size()) {
213                         //this case comes true when in our lists there is no event after the vertex that is being
214                         //checked. The case corresponds to vertices/events that happen around 23:50:00. They still need
215                         //to be connected though to waiting edges later in the night. what we are going to do in order
216                         //to build the right edges between our nodes is that we will add 24 to our stopTimes and search
     again
217                         for (Integer j : arrivalStopList) {
218                             j = j + 86400;
219                         }
220                         for (Integer k : arrivalStopList) {
221                             if (timeParsed < k) {
222                                 for (ArrayList<ArrayList<String>> waitingVertex : vertices) {
223                                     //structure of waitingVertexObject
224                                     //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
225                                     ArrayList<String> d = (ArrayList<String>) waitingVertex.get(0);      //get the
    first array, the vertexID
226                                     ArrayList<String> e = (ArrayList<String>) waitingVertex.get(1);      //get the
    second array, the strList
227                                     ArrayList<String> f = (ArrayList<String>) waitingVertex.get(2);    //get the third
    array, the doubleList
228                                     String stopIDParsed2 = e.get(1);
229                                     String vertexTypeParsed21 = f.get(0);
230                                     Integer vertexTypeParsed22 = Integer.parseInt(f.get(0));
231                                     String timeParsed21 = f.get(1);
232                                     Integer timeParsed22 = Integer.parseInt(f.get(1));
233                                     int cost1 = timeParsed22 - timeParsed;
234                                     if (stopIDParsed2.equals(stopIDParsed) && vertexTypeParsed22 == vertexTypeParsed &&
     timeParsed22 == k) {
235                                         this.setEdgeWeight(this.addEdge(vertex, waitingVertex), cost1); //adding edge
236                                     }
237                                 }
238
239                             }
240                         }
241                     }
242
243                 }
244                 dayCounter = 0;
245                 for (int j:departureStopList) {
246                     dayCounter = dayCounter + 1;
247                     if (timeParsed < j) {
248                         dayCounter = dayCounter - 1;                                                    //make sure that
    the dayCounter does not get equal to the size of the arrivalStopList if we find a next stop
249                         for (ArrayList<ArrayList<String>> departureVertex1 : vertices) {
```

```
250                             //structure of waitingVertexObject
251                             //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
252                             ArrayList<String> g = (ArrayList<String>) departureVertex1.get(0);      //get the first
    array, the vertexID
253                             ArrayList<String> h = (ArrayList<String>) departureVertex1.get(1);      //get the second
    array, the strList
254                             ArrayList<String> k = (ArrayList<String>) departureVertex1.get(2);    //get the third array
    , the doubleList
255                             String stopIDParsed3 = h.get(1);
256                             String vertexTypeParsed31 = k.get(0);
257                             Integer vertexTypeParsed32 = Integer.parseInt(k.get(0));
258                             String timeParsed31 = k.get(1);
259                             Integer timeParsed32 = Integer.parseInt(k.get(1));
260                             int cost2 = timeParsed32 - timeParsed;
261                             if (stopIDParsed3.equals(stopIDParsed) && vertexTypeParsed32 == vertexTypeParsed &&
    timeParsed32 == j) {
262                                 this.setEdgeWeight(this.addEdge(vertex, departureVertex1), cost2); //adding edge
263                             }
264                         }
265                         break;
266                     } else if (dayCounter == departureStopList.size()) {
267                         for (Integer l : departureStopList) {
268                             l = l + 86400;
269                         }
270                         for (Integer k : departureStopList) {
271                             if (timeParsed < k) {
272                                 for (ArrayList<ArrayList<String>> waitingVertex : vertices) {
273                                     //structure of waitingVertexObject
274                                     //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
275                                     ArrayList<String> d = (ArrayList<String>) waitingVertex.get(0);      //get the
    first array, the vertexID
276                                     ArrayList<String> e = (ArrayList<String>) waitingVertex.get(1);      //get the
    second array, the strList
277                                     ArrayList<String> f = (ArrayList<String>) waitingVertex.get(2);    //get the third
    array, the doubleList
278                                     String stopIDParsed2 = e.get(1);
279                                     String vertexTypeParsed21 = f.get(0);
280                                     Integer vertexTypeParsed22 = Integer.parseInt(f.get(0));
281                                     String timeParsed21 = f.get(1);
282                                     Integer timeParsed22 = Integer.parseInt(f.get(1));
283                                     int cost1 = timeParsed22 - timeParsed;
284                                     if (stopIDParsed2.equals(stopIDParsed) && vertexTypeParsed22 == vertexTypeParsed &&
     timeParsed22 == k) {
285                                         this.setEdgeWeight(this.addEdge(vertex, waitingVertex), cost1); //adding edge
286                                     }
287                                 }
288
289                             }
290                         }
291                     }
292                 }
293             }else if (vertexTypeParsed==2) {                                    //if the vertex we check is a arrival vertex
294                 for (int i=0;i<stopTimes.size()-1;i++) {
295                     String tripIDExtracted1 = (String) stopTimes.get(i).get(0);
296                     Integer arrivalTimeExtracted1 = calcTime(stopTimes.get(i).get(1));
297                     Integer departureTimeExtracted1 = calcTime(stopTimes.get(i).get(2));
298                     String stopIDExtracted1 = (String) stopTimes.get(i).get(3);
299                     Integer stopSequenceExtracted1 = Integer.parseInt(stopTimes.get(i).get(4));
300
301                     if (tripIDParsed.equals(tripIDExtracted1)&&stopSequenceParsed.equals(stopSequenceExtracted1)) {
302                         String tripIDExtracted2 = (String) stopTimes.get(i+1).get(0);
303                         String notParsedArrivalTime = (String) stopTimes.get(i+1).get(1);
304                         Integer arrivalTimeExtracted2 = calcTime(stopTimes.get(i+1).get(1));
305                         Integer departureTimeExtracted2 = calcTime(stopTimes.get(i+1).get(2));
306                         String stopIDExtracted2 = (String) stopTimes.get(i+1).get(3);
307                         String stopSequenceExtracted21 = stopTimes.get(i-1).get(4);
308                         Integer stopSequenceExtracted22 = Integer.parseInt(stopTimes.get(i+1).get(4));
309                         if (stopSequenceExtracted22>stopSequenceParsed){
310                             ArrayList<ArrayList<String>> nextArrivalNode = (ArrayList<ArrayList<String>>) this.
    composeSimpleVertex(tripIDExtracted2, notParsedArrivalTime, stopIDExtracted2, stopSequenceExtracted21, "0");
311
312                             this.setEdgeWeight(this.addEdge(vertex, nextArrivalNode), arrivalTimeExtracted2-timeParsed)
    ; //adding edge
313                         }
314                     }
315                 }
316             }
317         }
318     }
319
320     //***
321     //code below is used for the transformation of the hh:mm:ss time format of the GTFS into seconds passed since 00:00
    :00
322     //***
323     public static int calcTime(String time) {
324
325         String[] tokensArrival = time.split(":");                //for the arrival time
326         int arrivalHours = Integer.parseInt(tokensArrival[0]);
327         int arrivalMinutes = Integer.parseInt(tokensArrival[1]);
328         int arrivalSeconds = Integer.parseInt(tokensArrival[2]);
```

```
329
330         int timeInSeconds = arrivalHours * 3600 + arrivalMinutes * 60 + arrivalSeconds;
331
332         return timeInSeconds;
333     }
334
335     public static ArrayList<ArrayList<String>> readGTFS(String pathToFile) throws FileNotFoundException {
336         //code below is for reading and parsing of the csv and initiation of it in memory
337         ArrayList<String> listA = new ArrayList<String>();                  //Arrays A & B are used for parsing the .csv
    files
338         ArrayList<String> listB = new ArrayList<String>();
339         ArrayList<ArrayList<String>> returnedFile = new ArrayList<ArrayList<String>>();    //contains the final parsed
    file
340         Scanner scanner = new Scanner(new File(pathToFile));
341         scanner.useDelimiter(",");
342         while (scanner.hasNext()) {
343             listA.add(scanner.nextLine());
344         }
345         for (String i:listA) {
346             listB.clear();
347             String[] tokensGTFS = i.split(",");
348             for (String j:tokensGTFS)
349             {
350                 listB.add(j);
351             }
352             returnedFile.add((ArrayList<String>) listB.clone());
353         }
354         scanner.close();
355         returnedFile.remove(0);                                          //removing the first line of the stop_times file
356         return returnedFile;
357     }
358
359     //this method takes in
360     public static ArrayList<ArrayList<String>> composeSimpleVertex(String tripID, String Time, String stopID, String
    stopSequence, String vertexType) {
361 //      //[[vertexID],[tripID,stopID,stopSequence],[typeOfVertex,timeOfOccurance]]
362         ArrayList<ArrayList<String>> listOfLists = new ArrayList<>();       //list that stores the rest of the lists
    and is the vertex
363         ArrayList<String> vertexID = new ArrayList<>();                      //list that stores the vertexID
364         ArrayList<String> strList = new ArrayList<>();                       //list that stores tripID and the stopID,
    the string values of an event
365         ArrayList<String> intList = new ArrayList<>();                       //In this list we store the arrival and
    departure times
366         //adding lists to the listOfLists object that includes the rest of the lists
367         listOfLists.add(vertexID);
368         listOfLists.add(strList);
369         listOfLists.add(intList);
370         //inserting appropriate values into the lists
371         vertexID.add(tripID + stopSequence + vertexType);        //we insert to the list the vertexID
372         strList.add(tripID);
373         strList.add(stopID);
374         strList.add(stopSequence);
375         intList.add(vertexType);
376         String timeString = String.valueOf(calcTime(Time));
377         intList.add(timeString);
378         return listOfLists;
379     }
380
381 }
```

Παράρτημα Γ – 3η κλάση του κώδικα, TimeDependentGraph.java

```
1  package gr.greenyourmove.gtfsGraph;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.net.*;
6  import java.util.*;
7
8  import org.jgrapht.*;
9  import org.jgrapht.graph.*;
10
11
12 //with the time dependent graph structure the weights of the graph are calculated(added) on-the-fly, right before we run
   dijkstra
13 public class TimeDependentGraph<ArrayList, DefaultWeightedEdge>
14         extends DirectedWeightedMultigraph<List, DefaultWeightedEdge>
15         implements WeightedGraph<List, DefaultWeightedEdge>
16 {
17
18     public TimeDependentGraph(Class<? extends DefaultWeightedEdge> edgeClass) {
19         super(edgeClass);
20     }
21
22     public List initiateTimeDependentGraph(String pathToStopTimesFile, String pathToStopsFile) throws
   FileNotFoundException
23
24     {
25         //reading the files we need to read
26         ArrayList<ArrayList<String>> stopsGTFSFile = new ArrayList<ArrayList<String>>();
27         stopsGTFSFile = TimeExpandedGraph.readGTFS(pathToStopsFile);            //we borrow the readGTFS method from
   TimeExpandedGraph class
28         ArrayList<ArrayList<String>> stopTimesGTFSFile = new ArrayList<ArrayList<String>>();
29         stopTimesGTFSFile = TimeExpandedGraph.readGTFS(pathToStopTimesFile);        //We borrow the readGTFS method from
    TimeExpandedGraph class
30         this.addVertices(stopsGTFSFile);
31         List listsOfStops = this.createTimeDependentArray(stopTimesGTFSFile, stopsGTFSFile);
32         return listsOfStops;
33     }
34
35     public void addVertices(ArrayList<ArrayList<String>> stopsGTFSFile) {
36         for (ArrayList<String> stop:stopsGTFSFile){
37             ArrayList<ArrayList<String>> eachVertex = new ArrayList<ArrayList<String>>();
38             ArrayList<String> vertexID = new ArrayList<String>();
39             String ID = new String(stop.get(0));
40             eachVertex.add(vertexID);
41             vertexID.add(ID);
42             this.addVertex(eachVertex);
43         }
44     }
45     //ArrayList<ArrayList<ArrayList<String>>>
46     public List createTimeDependentArray(ArrayList<ArrayList<String>> stopTimesGTFSFile, ArrayList<ArrayList<String>>
   stopsGTFSFile) {
47
48         List returnedObject = new ArrayList<>();
49         ArrayList<ArrayList<ArrayList<String>>> timeDependentArray = new ArrayList<>();      //array to return
50
51         Set<ArrayList<ArrayList<String>>> vertices = (Set) this.vertexSet();
52         ArrayList<String> checkedStops = new ArrayList();                          //in this array we store the IDs of the
   stops checked up to now, concatenated
53         Integer indexOfStop = 0;
54         for (ArrayList<ArrayList<String>> stop:vertices) {
55             String stopID = stop.get(0).get(0);
56             for (int i=0;i<stopTimesGTFSFile.size()-1;i++) {
57                 String currentStopID = stopTimesGTFSFile.get(i).get(3);
58                 if ((currentStopID).equals(stopID)) {
59                     Integer currentStopSequence = Integer.parseInt(stopTimesGTFSFile.get(i).get(4));
60                     Integer nextStopSequence = Integer.parseInt(stopTimesGTFSFile.get(i+1).get(4));
61                     if (nextStopSequence == currentStopSequence + 1) {
62                         String nextStopID = stopTimesGTFSFile.get(i+1).get(3);
63                         //check out which stops are involved by parsing the stopID
64                         String concatStopID = currentStopID.concat(nextStopID);
65
66                         if (!(checkedStops.contains(concatStopID))) {                          //Very important part
67                             checkedStops.add(concatStopID);                                    //of our code
68                             timeDependentArray.add(new ArrayList<ArrayList<String>>());        //for the construction
69                             indexOfStop =checkedStops.indexOf(concatStopID);                   //of the timeDependent
70                             timeDependentArray.get(indexOfStop).add(new ArrayList<String>());  //graph. It determines
71                             timeDependentArray.get(indexOfStop).add(new ArrayList<String>());  //he size of the array
72                         }else{                                                                 //and the subArrays
73                             indexOfStop =checkedStops.indexOf(concatStopID);
74                         }
75
76                         //save stats for current stop
77                         String currentTripID = stopTimesGTFSFile.get(i).get(0);
78                         Integer currentArrivalTime = TimeExpandedGraph.calcTime(stopTimesGTFSFile.get(i).get(1));
79                         Integer currentDepartureTime = TimeExpandedGraph.calcTime(stopTimesGTFSFile.get(i).get(2));
80
81                         //save stats for the next stop
82                         String nextTripID = stopTimesGTFSFile.get(i+1).get(0);          //We don't need this assignment
   , obviously they are the same with the currectTripID
83                         Integer nextArrivalTime = TimeExpandedGraph.calcTime(stopTimesGTFSFile.get(i+1).get(1));
```

```
84                          Integer nextDepartureTime = TimeExpandedGraph.calcTime(stopTimesGTFSFile.get(i+1).get(2));
85
86                          String timeOfEventOccurance = Integer.toString(currentArrivalTime);
87                          String costOfEvent = Integer.toString(nextArriva_Time - currentDeparture_Time);
88
89                          timeDependentArray.get(indexOfStop).get(0).add(timeOfEventOccurance);
90                          timeDependentArray.get(indexOfStop).get(1).add(costOfEvent);
91                      }
92                  }
93              }
94          }
95
96          //of course before returning the arrays we need to sort, so that we can use binary search later on
97          for (ArrayList<ArrayList<String>> pairOfStops:TimeDependentArray) {
98              String swap1 = new String();
99              String swap2 = new String();
100             ArrayList<String> toBeSorted = pairOfStops.get(0);
101             ArrayList<String> toBeSortedAccordingly = pairOfStops.get(1);
102             for (int i=0;i<toBeSorted.size();i++) {
103                 for (int j=0;j<toBeSorted.size()-i-1;j++){
104                     //casting string to integers
105                     Integer integerValue1 = Integer.parseInt(toBeSorted.get(j));
106                     Integer integerValue2 = Integer.parseInt(toBeSorted.get(j+1));
107
108                     if (integerValue1>integerValue2) {
109                         //we need to execute two swaps, one for each list, so that we keep them linked up
110                         swap1 = toBeSorted.get(j);
111                         toBeSorted.set(j,toBeSorted.get(j+1));
112                         toBeSorted.set(j+1,swap1);
113
114                         swap2 = toBeSortedAccordingly.get(j);
115                         toBeSortedAccordingly.set(j,toBeSortedAccordingly.get(j+1));
116                         toBeSortedAccordingly.set(j+1,swap2);
117                     }
118                 }
119             }
120         }
121         returnedObject.add(timeDependentArray);
122         returnedObject.add(checkedStops);
123
124         return returnedObject;
125     }
126
127     //we run this method right before the dijkstra queries everytime
128     public void addPreQueryEdges(String time, String startingPoint, String endingPoint, ArrayList<ArrayList<ArrayList<
    String>>> timeDependentArray, ArrayList<String> stopPairs) {
129         removeAllEdges(this);           //removes all edges from the graph and reconstructs them according to user
    input
130         ArrayList<ArrayList<ArrayList<Integer>>> timeDependentArrayInteger = new ArrayList<ArrayList<ArrayList<Integer>
    >>();
131         for (int i=0;i<timeDependentArray.size();i++) {
132             timeDependentArrayInteger.add(new ArrayList<ArrayList<Integer>>());
133             for (int j=0;j<timeDependentArray.get(i).size();j++) {
134                 timeDependentArrayInteger.get(i).add(new ArrayList<Integer>());
135                 for (int k=0;k<timeDependentArray.get(i).get(j).size();k++) {
136                     timeDependentArrayInteger.get(i).get(j).add(Integer.parseInt(timeDependentArray.get(i).get(j).get(k
    )));
137                 }
138             }
139         }
140         Integer index;
141         boolean flagStops = true;
142         Integer timeParsed = TimeExpandedGraph.calcTime(time);
143         Set<ArrayList_<ArrayList<String>>> vertices = (Set) this.vertexSet();
144         ArrayList<String> checkedConnection = new ArrayList();
145         ArrayList<Integer> checkedConnectionFlag = new ArrayList(); //epeidh theloume na vrei ena akrivos sindesmo gia
    kathe zevgari komvwn xrhsimopoioume auto to array
146         for (int i=0;i<vertices.size()-1;i++) {                     //pou exei idio length me to plithos ton komvwn
147             checkedConnectionFlag.add(0);
148         }
149         ArrayList<Integer> timeParsedForPairs = new ArrayList<Integer>();
150         ArrayList<ArrayList<Integer>> edgeBuilderResult = new ArrayList<ArrayList<Integer>>();
151         for (int i=0;i<stopPairs.size()-1;i++) {
152             edgeBuilderResult.add(new ArrayList<Integer>());
153             timeParsedForPairs.add(0);
154             if (stopPairs.get(i).substring(0,startingPoint.length()).equals(startingPoint)) {
155                 timeParsedForPairs.set(i,timeParsed);
156             }
157         }
158         ArrayList<String> currentlyAddedVertices = new ArrayList<String>();
159
160         checkedConnection.add(startingPoint);
161         String concatinatedStops = new String();
162         while (flagStops==true){
163             for (int i=0;i<stopPairs.size();i++) {
164                 for (String check:checkedConnection){
165                     for (ArrayList<ArrayList<String>> vertex:vertices) {
166                         concatinatedStops = check + vertex.get(0).get(0);
167                         System.out.println(stopPairs.get(i) + " " + concatinatedStops);
168                         if (stopPairs.get(i).equals(concatinatedStops)) {
```

```
169                             currentlyAddedVertices.add(vertex.get(0).get(0));
170                             checkedConnectionFlag.set(i,1);
171                             index = getClosestK(timeDependentArrayInteger.get(i).get(0), timeParsedForPairs.get(i));
172                             while (timeDependentArrayInteger.get(i).get(0).get(index)<timeParsed) {
173                                 index = index + 1;
174                             }
175                             edgeBuilderResult.get(i).add(timeDependentArrayInteger.get(i).get(0).get(index));
176                             edgeBuilderResult.get(i).add(timeDependentArrayInteger.get(i).get(1).get(index));
177                             timeParsedForPairs.set(i,timeParsed + timeDependentArrayInteger.get(i).get(1).get(index));
178                         }
179                     }
180                     //allazoume to checkedConnection kai to timeparsed
181                     checkedConnection.clear();
182                     for (String k:currentlyAddedVertices) {
183                         checkedConnection.add(k);
184                     }
185                     currentlyAddedVertices.clear();
186                 }
187             }
188             int sum=0;
189             for (int i=0;i<checkedConnectionFlag.size();i++) {
190                 sum=sum+checkedConnectionFlag.get(i);
191             }
192             if (sum==checkedConnectionFlag.size()){
193                 flagStops=false;
194             }
195
196             String stop1;
197             String stop2;
198             //tha xreiastoume auta ta duo flags gia kathorisoume to onoma apo ekei pou einai concatinated na to
    spasoume se duo
199             boolean flagStop1 = false;
200             boolean flagStop2 = false;
201             //Now that we have gathered all the information we need in edgeBuilderResult we can build the edges
202             for (int i=0;i<edgeBuilderResult.size();i++) {
203                 for (int k=0;k<3;k++) { //epeidh exoume treis periptwseis, an einai 5 grammata ola to concatinated
    string
204                                         //exoume 2 periptwseis kai mia akomh an einai 4 ta grammata tou concatinated
205                     stop1 = stopPairs.get(i).substring(0,k+1);
206                     stop2 = stopPairs.get(i).substring(0,3-k);
207
208                     for (ArrayList<ArrayList<String>> vertex:vertices) {
209                         if (vertex.get(0).get(0).equals(stop1)) {
210                             flagStop1 = true;
211                         }
212                         if (vertex.get(0).get(0).equals(stop2)) {
213                             flagStop2 = true;
214                         }
215                     }
216                     if (flagStop1 && flagStop2) {
217                         break;
218                     }
219                     //we need to cosntruct the vertices in their standard from
220                     ArrayList<ArrayList<ArrayList<String>>> vertex1 = new  ArrayList<ArrayList<ArrayList<String>>>();
221                     ArrayList<ArrayList<ArrayList<String>>> vertex2 = new  ArrayList<ArrayList<ArrayList<String>>>();
222                     ArrayList<ArrayList<String>> subVertex1 = new  ArrayList<ArrayList<String>>();
223                     ArrayList<ArrayList<String>> subVertex2 = new  ArrayList<ArrayList<String>>();
224                     ArrayList<String> subSubVertex1 = new  ArrayList<String>();
225                     ArrayList<String> subSubVertex2 = new  ArrayList<String>();
226                     subSubVertex1.add(stop1);
227                     subSubVertex2.add(stop2);
228                     subVertex1.add(subSubVertex1);
229                     subVertex2.add(subSubVertex2);
230                     vertex1.add(subVertex1);
231                     vertex1.add(subVertex2);
232                     this.setEdgeWeight(this.addEdge(vertex1, vertex2), edgeBuilderResult.get(i).get(0));
233                 }
234             }
235         }
236     }
237
238     public static <V,E> void removeAllEdges(Graph<V, E> graph) {
239         LinkedList<E> copy = new LinkedList<E>();
240         for (E e : graph.edgeSet()) {
241             copy.add(e);
242         }
243         graph.removeAllEdges(copy);
244     }
245
246     public static ArrayList<ArrayList<String>> composeSimpleVertex(String stopID) {
247         ArrayList<ArrayList<String>> listOfLists = new ArrayList<>();       //list that stores the rest of the lists
    and is the vertex
248         ArrayList<String> B = new ArrayList<>();                    //list that stores tripID and the stopID, the
    string values of an event
249         B.add(stopID);
250         listOfLists.add(B);
251
252         return listOfLists;
253     }
254
```

```
255     public static int getClosestK(ArrayList<Integer> a, int x) {
256
257         int low = 0;
258         int high = a.size() - 1;
259
260         if (high < 0)
261             throw new IllegalArgumentException("The array cannot be empty");
262
263         while (low < high) {
264             int mid = (low + high) / 2;
265             assert(mid < high);
266             int d1 = Math.abs(a.get(mid) - x);
267             int d2 = Math.abs(a.get(mid+1) - x);
268             if (d2 <= d1)
269             {
270                 low = mid-1;
271             }
272             else
273             {
274                 high = mid;
275             }
276         }
277         return high;
278     }
279
280 }
281
282
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] www.newsit.gr/

[2] Wikipedia, http://www.wikipedia.org

[3] http://infoman.teikav.edu.gr/e_education/67/files/alg%20eis.pdf

[4] http://www.digitalplan.gov.gr

[5] European Commission. European transport policy for 2010: time to decide. White Paper

[6] A co-modal transport information system in a distributed enviroment, Zhanjun Wang, Khaled Mesghouni, Slim Hammadi

[7] http://tfresource.org/Category:Transportation_networks

[8]www.people.hofstra.edu

[9] Route Planning in Transportation Networks, Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthiaw Muller - Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck, January 8, 2014

[10] www.astikoktel.gr

[11] https://www.jetbrains.com/idea/

[12] www.astikakomotinis.gr/

[13] http://jgrapht.org/

[14] www.astikoxanthis.gr

[15] Nadine Baumann, Martin Skutella, "Solving Evacuation Problems Efficiently", Earliest Arrival Flows with Multiple Sources.

[16] Aldous M. Joan, Wilson J. Robin (2000), Graphs and Applications: An Introductory Approach, Springer Publications, London

[17] Στεφανάκης Εμμανουήλ (2003), Βάσεις Γεωγραφικών Δεδομένων και Συστήματα Γεωγραφικών Πληροφοριών, Εκδόσεις Παπασωτηρίου, Αθήνα

[18] Shekhar Shashi, Chawla Sanjay (2003), Spatial Databases: A Tour, Pearson Education Inc., Upper Saddle River, New Jersey 07458.

[19] Μπέης Σωτήρης, Θεσσαλονίκη, Ιούλιος 2013, Ομάδα Εξελισσόμενων Κοινωνικών Δικτύων με Χρήση Τεχνικών Ανίχνευσης Κοινοτήτων, Διπλωματική Εργασία Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης.

[20] Cormen H. Thomas, Leiserson E. Charles, Rivest L. Ronald and Stein Clifford (2001), Introduction to Algorithms, The MIT Press, Massachussets Institute of Technology.

[21]Google Developers, «GTFS Review & Documentation» [Ηλεκτρονικό]. www.developers.google.com/transit/gtfs.