ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

ΜΕΛΕΤΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΑΛΓΟΡΙΘΜΟΥ ΔΥΝΑΜΙΚΗΣ ΔΡΟΜΟΛΟΓΗΣΗΣ ΜΕ ΠΟΛΛΑΠΛΑ ΜΕΣΑ ΜΕΤΑΦΟΡΑΣ

Χρήστος Μ. Μπότσικας

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Διπλώματος Μηχανολόγου Μηχανικού

2011

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Ζηλιασκόπουλος Θανάσης Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Λυμπερόπουλος Γιώργος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Σαχαρίδης Γιώργος Λέκτορας, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

ΕΥΧΑΡΙΣΤΙΕΣ

Η διπλωματική εργασία έλαβε χώρα στη Σχολή Μηχανολόγων Μηχανικών του Πανεπιστημίου Θεσσαλίας, στο πλαίσιο των δραστηριοτήτων του Εργαστηρίου Συστημάτων Βελτιστοποίησης.

Η εργασία πραγματοποιήθηκε υπό την επίβλεψη, τη συνεχή καθοδήγηση και την αμέριστη συμπαράσταση του Καθηγητή κ. Θ. Ζηλιασκόπουλου, στον οποίο οφείλω ιδιαίτερες ευχαριστίες. Επίσης, θα ήθελα να τονίσω ότι η συνεργασία μου με τον Καθηγητή κ Θ. Ζηλιασκόπουλο και η εμπιστοσύνη που μου έδειξε αυτά τα χρόνια σε ένα ευρύ πλαίσιο ερευνητικών δραστηριοτήτων υπήρξε για εμένα το «μεγαλύτερο σχολείο», που ουσιαστικά διαμόρφωσε τον επαγγελματικό μου χαρακτήρα.

Θα ήθελα επιπλέον να ευχαριστήσω θερμά τον Καθηγητή κ. Λυμπερόπουλο Γιώργο και τον Λέκτορα κ. Σαχαρίδη Γιώργο για την τιμή που μου έκαναν να παραβρεθούν στην εξέταση της διπλωματικής εργασίας μου.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα τον συνάδελφο και φίλο Μάκη Μισδανίτη καθώς και τους συνεργάτες μου στο Εργαστήριο, οι οποίοι συνέβαλαν με τον τρόπο τους στην επίτευξη αυτού του αποτελέσματος.

Κλείνοντας, οφείλω ένα μεγάλο ευχαριστώ στους γονείς μου, τον Μαρίνο και την Αιμιλία Μπότσικα, στον αδερφό μου Ανδρέα Μπότσικα καθώς και στην σύντροφο μου Δώρα Κεραμίδα, για την αμέριστη συμπαράσταση και υπομονή τους. Η συμβολή τους σε κάθε μου βήμα ήταν και θα είναι η σημαντικότερη.

Βόλος, Ιούνιος 2011,

Χρήστος Μ. Μπότσικας

ΜΕΛΕΤΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΑΛΓΟΡΙΘΜΟΥ ΔΥΝΑΜΙΚΗΣ ΔΡΟΜΟΛΟΓΗΣΗΣ ΜΕ ΠΟΛΛΑΠΛΑ ΜΕΣΑ ΜΕΤΑΦΟΡΑΣ

ΧΡΗΣΤΟΣ Μ. ΜΠΟΤΣΙΚΑΣ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών, 2011

Επιβλέπων Καθηγητής: Δρ. Ζηλιασκόπουλος Θανάσης, Καθηγητής Βελτιστοποίηση Συστημάτων Παραγωγής/Μεταφορών

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη του προβλήματος δρομολόγησης με πολλαπλά μεταφορικά μέσα. Ειδικότερα, εξετάζεται η μορφοποίηση ενός συγκοινωνιακού δικτύου σε υπολογιστικές δομές καθώς και η αποτύπωση των δυνατών διαδρομών των υποστηριζόμενων μέσων του δικτύου. Στην συνέχεια εξετάζονται οι πιο διαδεδομένοι αλγόριθμοι εντοπισμού βέλτιστων λύσεων μέσα σε αυτά τα δίκτυα και γίνεται αξιολόγηση αυτών. Έχοντας ολοκληρώσει την επιστημονική επισκόπηση, παρουσιάζεται ο αλγόριθμος time-dependent intermodal least-time path ή εν συντομία TDILTP ο οποίος και υλοποιείται στην παρούσα διπλωματική εργασία σε γλώσσα FORTRAN. Με βάση τον αλγόριθμο TDILTP πραγματοποιούνται πειράματα όπου αξιολογείται η πολυπλοκότητα και η αποδοτικότητα του αλγόριθμου ενώ παράλληλα γίνεται εφαρμογή του αλγόριθμου στο πραγματικό δίκτυο της ΤΡΑΙΝΟΣΕ. Η διπλωματική εργασία ολοκληρώνεται με την παρουσίαση των επιστημονικών πορισμάτων της εργασίας και την παρουσίαση πιθανών μελλοντικών ερευνητικών προοπτικών ανάπτυξης του αλγόριθμου.

Πίνακας περιεχομένων

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

1

Εισαγωγή

Κεφάλαιο 1: Εισαγωγή

1.1 Εισαγωγή

Η έννοια της δρομολόγησης είναι συνυφασμένη με ένα μεγάλο αριθμό τεχνολογικών εφαρμογών. Από τα δίκτυα των ηλεκτρονικών υπολογιστών μέχρι τον προγραμματισμό της βιομηχανικής παραγωγής και την πλοήγηση των οχημάτων στα οδικά δίκτυα, συναντούμε την ανάγκη της εξαγωγής βέλτιστων διαδρομών για μια πληθώρα κριτηρίων βελτιστοποίησης.

Τις τελευταίες δεκαετίες, η επιστημονική κοινότητα στρέφει όλο και περισσότερο το ενδιαφέρον της στην ανάπτυξη σύνθετων αλγόριθμων δρομολόγησης και στην ανάπτυξη ευφυών συστημάτων μεταφορών (Intelligent Transportation Systems, ITS) σε διασυνδεδεμένα εμπορευματικά δίκτυα. Αυτή η προσπάθεια εντάθηκε από το 1991 και ύστερα όταν η Αμερικάνικη κυβέρνηση ψήφισε τον νόμο "Intermodal Surface Transportation Efficiency Act" ή αλλιώς γνωστό ως ISTEA, ο οποίος όρισε κανονισμούς και πολιτικές με στόχο τη βελτίωση της ασφάλειας και τη μείωση της συμφόρησης των συστημάτων μεταφοράς. Αυτό το πλαίσιο νόμου προώθησε τον κατακερματισμό των τεράστιων δικτύων μεταφοράς σε μικρότερα αυτόνομα δίκτυα, τα οποία διασυνδέονται μεταξύ τους με στόχο την καλύτερη και ασφαλέστερη εξυπηρέτηση των ταξιδιωτών και των εμπορευμάτων. Παράλληλα, ο κατακερματισμός των δικτύων μεταφορών έχει σαν στόχο την μείωση του συνολικού κόστους μεταφοράς, επιτρέποντας την χρήση του καταλληλότερου μεταφορικού μέσου σε κάθε τμήμα της ζητούμενης διαδρομής με αποτέλεσμα οι διαδρομές να είναι βέλτιστες, ενώ παράλληλα να μην απαιτείται η κατασκευή ανεξάρτητων υποδομών για κάθε μεταφορικό μέσο για την εξυπηρέτηση όλων των δυνατών κόμβων (National Commission on Intermodal Transportation, 1994).

Στο πλαίσιο αυτό, στόχος της παρούσας διπλωματικής εργασίας είναι η μελέτη του προβλήματος δρομολόγησης σε διασυνδεδεμένα αυτόνομα δίκτυα μεταφορών ή πιο απλά σε δίκτυα με πολλαπλά μεταφορικά μέσα.

Οι ενέργειες που πραγματοποιήθηκαν για την σύνταξη της παρούσας διπλωματικής εργασίας μπορούν να ομαδοποιηθούν όπως απεικονίζονται στο ακόλουθο σχήμα.

Εικόνα 1.1 Ομάδες ενεργειών διπλωματικής εργασίας

Αναλυτικότερα, στην πρώτη ομάδα ενεργειών εξετάστηκαν οι τρόποι μοντελοποίησης ενός συγκοινωνιακού δικτύου σε υπολογιστικές δομές καθώς και οι μεθοδολογίες αποτύπωσης διαδρομών πολλαπλών μέσων ενός δικτύου. Στην συνέχεια, εξετάστηκαν οι πιο διαδεδομένοι αλγόριθμοι εντοπισμού βέλτιστων λύσεων μέσα σε αυτά τα δίκτυα και έγινε η αξιολόγηση αυτών.

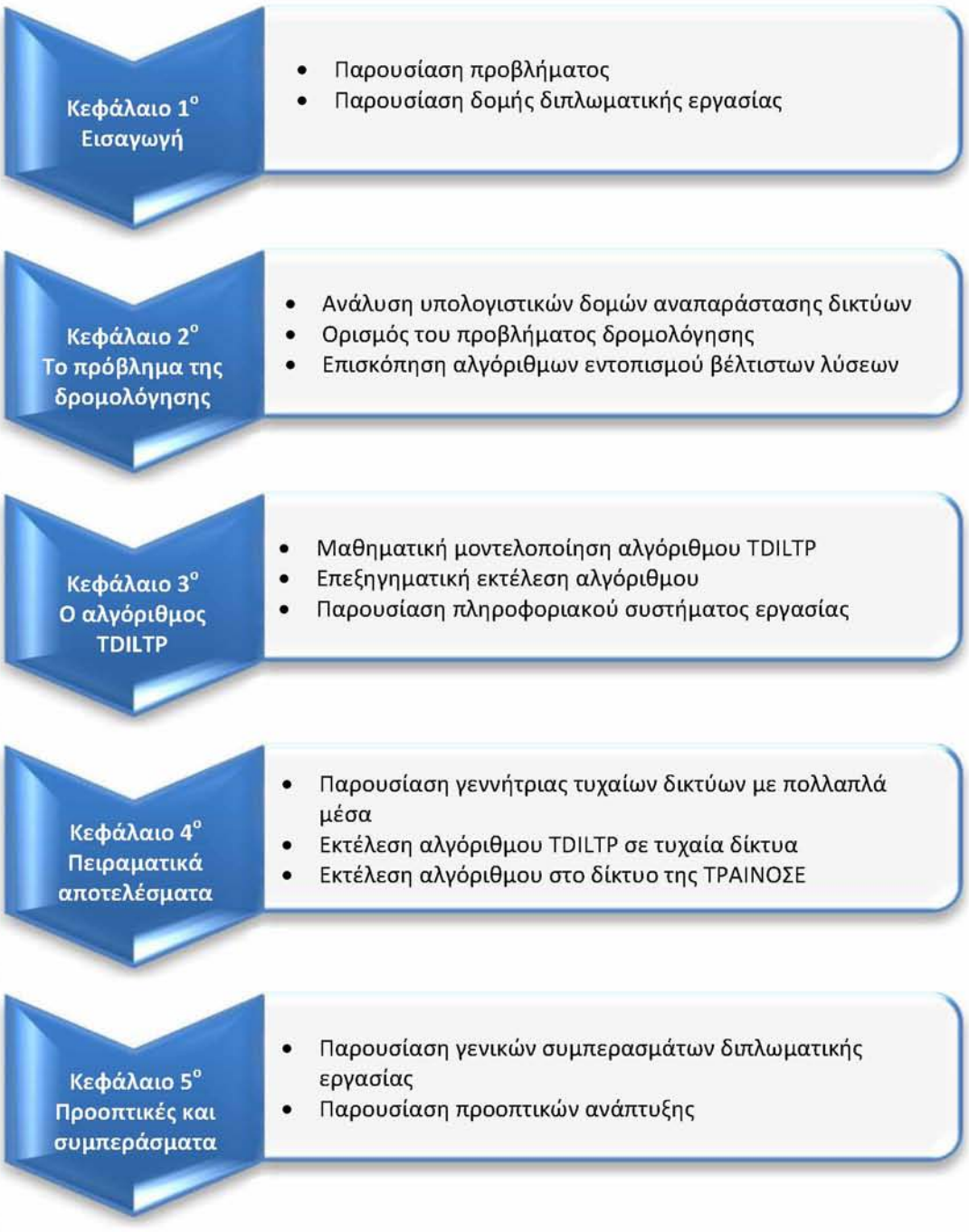
Μετά την επιστημονική ανασκόπηση, μελετήθηκε ο αλγόριθμος time-dependent intermodal least-time path. Ο αλγόριθμος, που εν συντομία αναφέρεται ως TDILTP, αποτελεί έναν πρωτότυπο δυναμικό αλγόριθμο δρομολόγησης, ο οποίος παρουσιάστηκε αρχικά από τους Ziliaskopoulos et al το 2000 (Ziliaskopoulos & Wardell, 2000).

Στην τρίτη ομάδα ενεργειών εντοπίζεται ο σχεδιασμός και η ανάπτυξη ενός ολοκληρωμένου πληροφοριακού συστήματος, το οποίο υποστηρίζει τον αλγόριθμο TDILTP σε όλο του το φάσμα. Αναπτύχθηκε ο τρόπος αναπαράστασης του δικτύου και των δρομολογίων των μέσων που υποστηρίζει, με στόχο την εύκολη χρήση του συστήματος και την ελαχιστοποίηση των δεδομένων εισόδου που απαιτούνται.

Η τελευταία ομάδα ενεργειών αποτελείται από τη δημιουργία τυχαίων δικτύων με πολλαπλά μεταφορικά μέσα και την εκτέλεση του αλγόριθμου πάνω σε αυτά με στόχο την αξιολόγηση της πολυπλοκότητας και της αποτελεσματικότητας του αλγόριθμου. Επιπρόσθετα έγινε εφαρμογή του αλγόριθμου στο πραγματικό δίκτυο της ΤΡΑΙΝΟΣΕ από όπου προέκυψαν χρήσιμα πορίσματα ως προς τον τρόπο με τον οποίο πρέπει να γίνεται η μοντελοποίηση των πραγματικών δικτύων και στο εάν και κατά πόσο είναι δυνατή η εφαρμογή του TDILTP σε μεγάλης κλίμακας πραγματικά προβλήματα.

1.2 Δομή της παρούσας διπλωματικής

Η διπλωματική αποτελείται από πέντε (5) κεφάλαια, όπως απεικονίζονται στο ακόλουθο σχήμα. Η ροή των κεφαλαίων είναι αντίστοιχη με την εξέλιξη της διπλωματικής.



Εικόνα 1.2 Δομή της διπλωματικής

Ακολουθεί συνοπτική περιγραφή του περιεχομένου κάθε κεφαλαίου της διατριβής.

Κεφάλαιο 1: Το παρόν κεφάλαιο αποτελεί την εισαγωγή της διπλωματικής. Γίνεται αναφορά στο πρόβλημα της δυναμικής δρομολόγησης με πολλαπλά μέσα μεταφοράς και παρουσιάζεται η δομή της παρούσας διπλωματικής

εργασίας.

Κεφάλαιο 2: Σε αυτό το κεφάλαιο σκιαγραφείται το πρόβλημα της δρομολόγησης. Εξετάζονται οι διάφορες παραλλαγές του και εντοπίζονται τα ιδιαίτερα χαρακτηριστικά, που το καθιστούν ως ένα από τα πιο δύσκολα συνδυαστικά προβλήματα της εποχής μας. Τέλος, παρουσιάζονται οι σύγχρονες μεθοδολογίες δρομολόγησης και γίνεται επισκόπηση των πρακτικών προσεγγίσεων.

Κεφάλαιο 3: Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά το μαθηματικό μοντέλο του αλγόριθμου Time-Dependent Intermodal List-Time Path (TDILTP) και αναλύεται η φιλοσοφία, τα επιμέρους βήματα και οι συνιστώσες του. Δίνονται παραδείγματα εκτέλεσής του και στο τέλος του κεφαλαίου παρουσιάζεται το πληροφοριακό σύστημα που υλοποιεί τον αλγόριθμο της παρούσας διπλωματικής εργασίας.

Κεφάλαιο 4: Σε αυτό το κεφάλαιο παρουσιάζεται η εφαρμογή του αλγόριθμου σε τυχαία παραγόμενα δίκτυα καθώς και στο δίκτυο του Οργανισμού Σιδηροδρόμων Ελλάδος (ΟΣΕ) και ειδικότερα της θυγατρικής εταιρία ΤΡΑΙΝΟΣΕ, στην οποία ανήκει το σιδηροδρομικό δίκτυο. Ταυτόχρονα, γίνεται αναλυτική παρουσίαση της πιλοτικής εφαρμογής του αλγόριθμου, η αξιολόγηση και τα εξαχθέντα συμπεράσματα.

Κεφάλαιο 5: Στο τελευταίο κεφάλαιο, η διπλωματικής εργασία ολοκληρώνεται παραθέτοντας κάποια γενικά συμπεράσματα καθώς και την παρουσίαση των μελλοντικών προοπτικών και των πεδίων ανάπτυξης και εφαρμογής του εξεταζόμενου αλγόριθμου.

2

Το πρόβλημα της δρομολόγησης

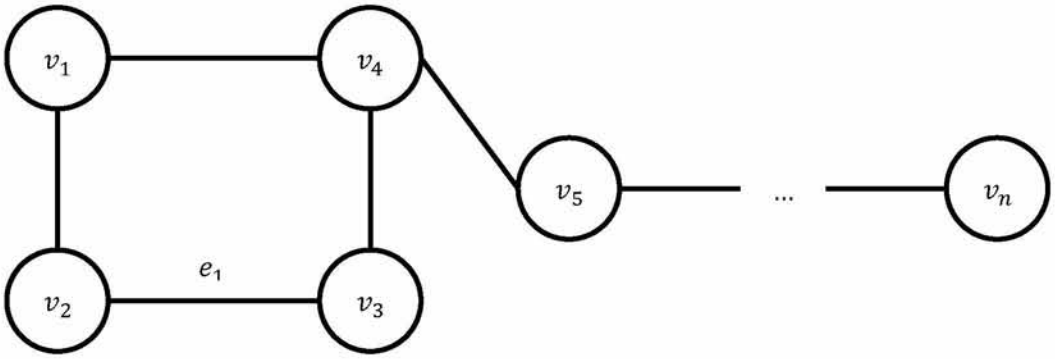
Κεφάλαιο 2: Το πρόβλημα της δρομολόγησης

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα εξετάσουμε την βασική θεωρία των γράφων που χρησιμοποιείται στην μοντελοποίηση των προβλημάτων δρομολόγησης. Στην συνέχεια θα εστιάσουμε την προσοχή μας στα δίκτυα που αποτελούν μια εξελιγμένη μορφή γράφων και χρησιμοποιούνται στους αλγόριθμους επίλυσης των προβλημάτων δρομολόγησης. Ειδικά για τα δίκτυα, θα συγκρίνουμε ορισμένες μορφές υπολογιστικής αποτύπωσης με στόχο την προετοιμασία για το επόμενο κεφάλαιο όπου θα παρουσιαστεί μια υλοποίηση του αλγόριθμου που διαπραγματεύεται η παρούσα διπλωματική εργασία. Αφού παρουσιαστούν οι θεωρητικές βάσεις, θα δοθεί ένας ορισμός για το πρόβλημα της δρομολόγησης και θα παρουσιαστούν τα κυριότερα προβλήματα που εντάσσονται υπό το γενικό πρίσμα της δρομολόγησης. Τέλος, το κεφάλαιο ολοκληρώνεται με την παρουσίαση των πιο δημοφιλών αλγόριθμων εντοπισμών βέλτιστων λύσεων στα προβλήματα δρομολόγησης.

2.2 Γράφοι

Κάθε γράφος G αποτελείται από δύο σύνολα, τα V και E. Το σύνολο $V = \{v_1, v_2, \dots, v_n\}$ είναι ένα πεπερασμένο μη κενό σύνολο τα στοιχεία του οποίου είναι οι κορυφές (vertices) ή αλλιώς κόμβοι (nodes) του γράφου (West, 1996). Το σύνολο $E = \{e_1, e_2, .., e_m\}$ είναι επίσης ένα πεπερασμένο μη κενό σύνολο που έχει ως στοιχεία τις ακμές (edges) ή αλλιώς τόξα (arcs) του γράφου, που συνδέουν μεταξύ τους τους κόμβους. Κάθε ακμή συνδυάζεται με ένα ζεύγος κόμβων, που είναι τα άκρα της ακμής αυτής. Για παράδειγμα: η $e_1 = (v_2 , v_3)$ συμβολίζει την ακμή 1 που συνδέει μεταξύ τους κόμβους 2 και 3.



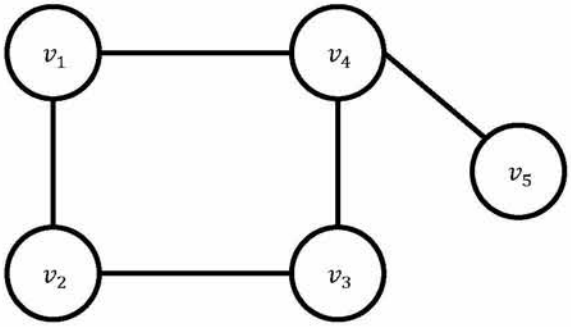
Εικόνα 2.1 Γράφος

Οι γράφοι συμβολίζονται ως G=(V,E), ενώ V(G) και E(G) συμβολίζονται τα σύνολα των κόμβων και των ακμών τους, αντίστοιχα. Αν θεωρήσουμε ότι οι κόμβοι ενός γράφου χρησιμοποιούνται για την παράσταση κάποιων δεδομένων, τότε μπορούμε να πούμε ότι οι ακμές χρησιμοποιούνται για να απεικονίσουν κάποια σχέση που συνδέει μεταξύ τους τα δεδομένα αυτά (Μισυρλής, 2002).

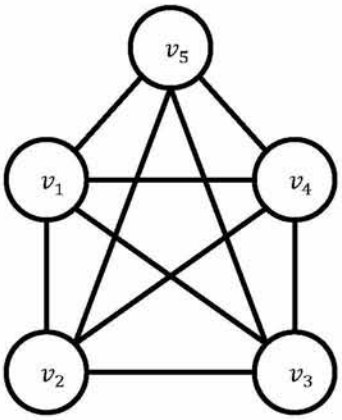
2.2.1 Έννοιες συνυφασμένες με τους γράφους

Οι γράφοι χωρίζονται σε κατευθυνόμενους και μη. Σε ένα ***μη κατευθυνόμενο γράφο***, το ζεύγος των κόμβων που αποτελούν τα άκρα κάθε ακμής του δεν έχει διάταξη. Συνεπώς τα ζεύγη κόμβων (v_i, v_j) και (v_j, v_i) παριστάνουν την ίδια ακμή, η οποία και δεν έχει κατεύθυνση, ή διαφορετικά είναι αμφίδρομη. Από την άλλη, ***κατευθυνόμενος γράφος*** είναι ο γράφος στον οποίο κάθε ακμή αυτού μετατρέπεται σε ένα βέλος με συγκεκριμένη κατεύθυνση, δηλαδή με συγκεκριμένο κόμβο-αφετηρία και συγκεκριμένο κόμβο-προορισμό. Στον κατευθυνόμενο γράφο, η ακμή που ξεκινά από τον κόμβο v_i και καταλήγει στον κόμβο v_j διαφοροποιείται από αυτήν που ξεκινά από τον v_j και καταλήγει στον v_i.

Πλήρης ονομάζεται ο γράφος, στον οποίο κάθε ζεύγος κόμβων του συνδέεται με μία τουλάχιστον ακμή. Αν ο γράφος είναι πλήρης και κατευθυνόμενος, τότε οι κόμβοι μεταξύ τους θα συνδέονται με δύο ακμές. Ουσιαστικά στο γράφο αυτό όλοι οι κόμβοι είναι γειτονικοί μεταξύ τους.



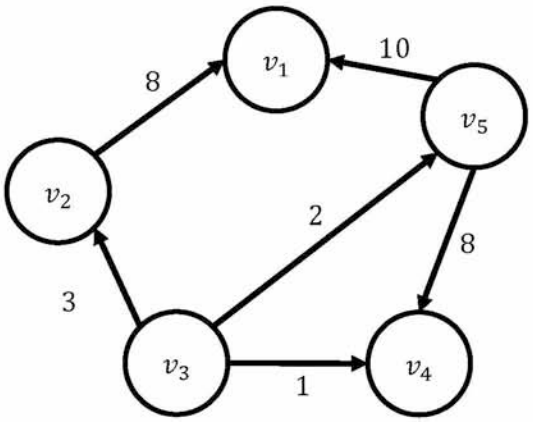
Εικόνα 2.2 Μη πλήρης γράφος



Εικόνα 2.3 Πλήρης γράφος

Γειτονικοί ονομάζονται οι κόμβοι ενός γράφου, οι οποίοι συνδέονται μεταξύ τους με ακμή.

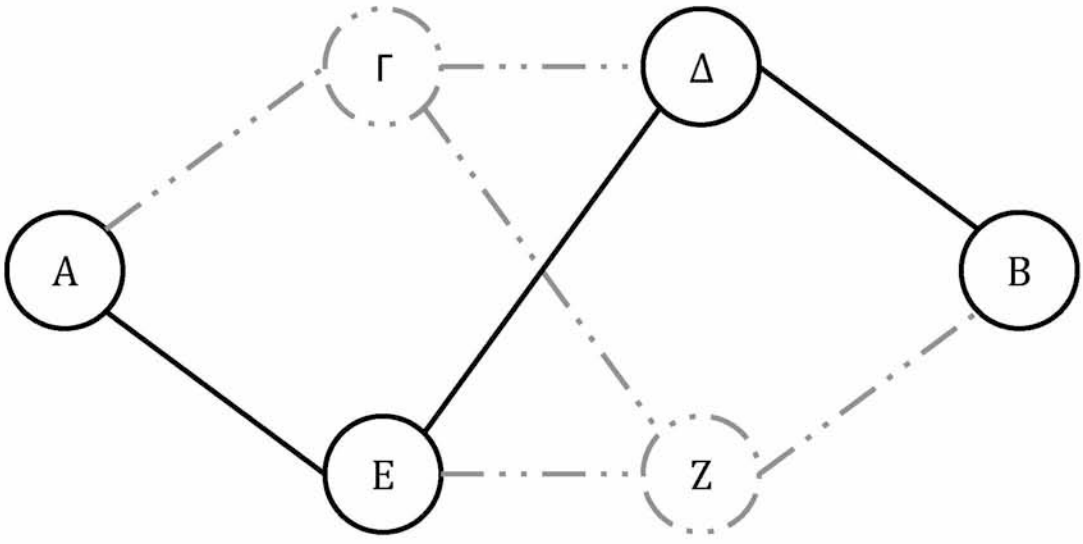
Βάρος της ακμής είναι η τιμή που αντιστοιχίζεται σε αυτήν σε ένα γράφο και παριστάνει διαφορετικό μέγεθος, ανάλογα με το τι αντιπροσωπεύει ο γράφος που εξετάζουμε. Μπορεί να είναι χρόνος για τη μετάβαση από το ένα άκρο της ακμής στο άλλο, απόσταση που συνδέει τις δύο ακμές, φόρτος δικτύου κατά μήκος της ακμής αυτής ή ακόμα και το αποτέλεσμα κάποιας συνάρτησης που να τα συνδυάζει. Με βάση το παρακάτω σχήμα το βάρος της ακμής που συνδέει τους κόμβους v_1 και v_2 είναι 8, τους v_2 και v_3 3, κλπ.



Εικόνα 2.4 Κατευθυνόμενος γράφος με βάρη ακμής

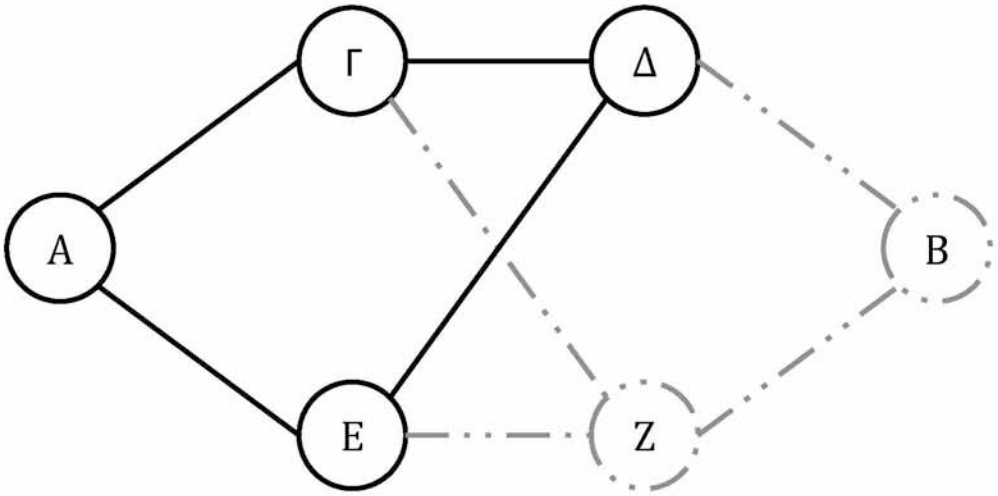
Σε ένα γράφο G, ονομάζουμε ***δρόμο*** μια πεπερασμένη ακολουθία εναλλάξ κόμβων και ακμών του G, που αρχίζει και τελειώνει σε κόμβο και που κάθε ακμή που περιέχεται στην ακολουθία προσπίπτει στον κόμβο που προηγείται και σε αυτόν που έπεται (Ζάχος & Κοζύρης, 2006).

Μονοπάτι ονομάζεται ο δρόμος, στον οποίο κάθε κόμβος και κάθε ακμή του, εμφανίζονται ακριβώς μία φορά.



Εικόνα 2.5 Μονοπάτι ΑΕΔΒ

Κύκλος (ή βρόγχος) είναι ένα κλειστό μονοπάτι, δηλαδή ένα μονοπάτι που έχει αρχή και τέλος την ίδια κορυφή.

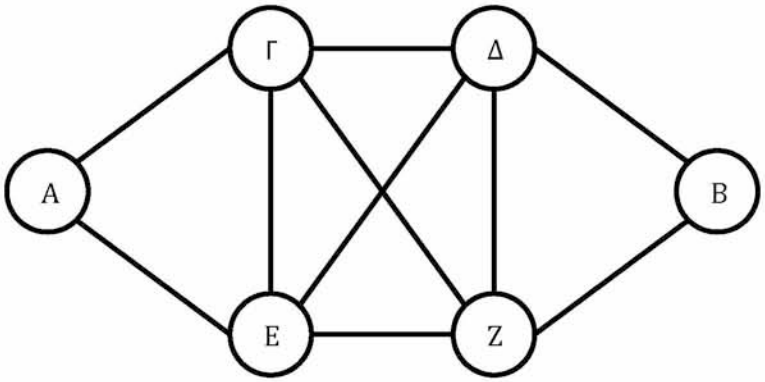


Εικόνα 2.6 Κύκλος ΑΕΔΓΑ στον γράφο

Βαθμός κορυφής ενός γράφου είναι το πλήθος των ακμών που προσπίπτουν ή αναχωρούν από αυτήν (Ζάχος & Κοζύρης, 2006). Ειδικότερα, μιλώντας για κατευθυνόμενους γράφους, ορίζουμε ως έσω-βαθμό (in-degree) ενός κόμβου v_i, το πλήθος των ακμών των οποίων αποτελεί κεφαλή, ενώ ως έξω-βαθμό (out-degree) του κόμβου v_i, ορίζουμε το πλήθος των ακμών των οποίων αποτελεί ουρά (Μισυρλής, 2002).

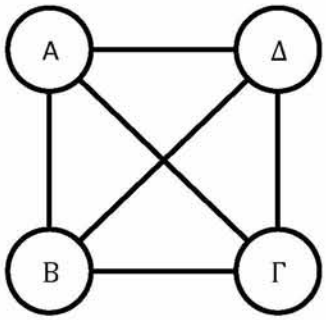
Τάξη ενός γράφου $G(V, E)$ ονομάζεται το πλήθος των κόμβων του και συμβολίζεται με $|V|$. Μέγεθος ενός γράφου G ονομάζεται το πλήθος των ακμών αυτού και συμβολίζεται με $|E|$. Ο αριθμός $d(G) = \frac{1}{|V|}\sum_{v \in V} d(v)$ είναι ο μέσος βαθμός του γράφου G, όπου $|V|$ είναι το πλήθος των ακμών και v κάθε ένας από τους κόμβους του γράφου (Diestel, 2005).

Γράφος Euler ονομάζεται ο γράφος που έχει κύκλο Euler. ***Κύκλος Euler*** είναι ένας κύκλος που περνά ακριβώς μία φορά από κάθε ακμή ενός γράφου G, χωρίς να περνά ακριβώς μια φορά και από κάθε κόμβο. Ένας γράφος έχει κύκλο Euler αν και μόνο αν όλοι οι κόμβοι του έχουν άρτιο βαθμό (Ζάχος & Κοζύρης, 2006).



Εικόνα 2.7 Κύκλος Euler

Γράφος Hamilton ονομάζεται ο γράφος που έχει κύκλο Hamilton. ***Κύκλος Hamilton*** είναι ένας κύκλος που περνά ακριβώς μια φορά από κάθε κόμβο ενός γράφου, χωρίς απαραίτητα να περνά και από όλες τις ακμές (Ζάχος & Κοζύρης, 2006).



Εικόνα 2.8 Γράφος Hamilton

2.3 Δίκτυα

Η έννοια του δικτύου θα μπορούσε να οριστεί ως ένα σύστημα γραμμικών χαρακτηριστικών που συνδέονται σε διασταυρώσεις και κόμβους. Οι κόμβοι μέσα σε ένα δίκτυο συνδέονται μέσω του γραμμικού χαρακτηριστικού που ονομάζεται ακμή. Δηλαδή, κάθε ακμή αντιστοιχεί σε ένα ζεύγος κόμβων (v_i, v_j) και συνδέει τον κόμβο v_i με τον κόμβο v_j.

Όπως γίνεται αντιληπτό, ένα δίκτυο μπορεί να οριστεί ως ένας κατευθυνόμενος γράφος $G = (V, E)$ που αποτελείται από ένα σύνολο κόμβων $V = |n|$ και ένα σύνολο ακμών $E = |m|$, όπου n είναι ο αριθμός των κόμβων και m ο αριθμός των ακμών.

Σε ένα δίκτυο μέσων μεταφοράς, οι κόμβοι αντιστοιχούν σε στάσεις των γραμμών των μέσων μεταφοράς που απαρτίζουν το δίκτυο. Οι ακμές μεταξύ των κόμβων ενώνουν διαδοχικές στάσεις, δηλαδή είναι τμήματα της διαδρομής που ακολουθούν μία ή περισσότερες γραμμές που εξυπηρετούν τις στάσεις-κόμβους.

Ένα μεταφορικό δίκτυο απεικονίζεται συνήθως μέσα στο πλαίσιο ενός γεωγραφικού πληροφοριακού συστήματος (GIS), οπότε κάθε κόμβος αντιστοιχεί σε μία τοποθεσία με γεωγραφικές συντεταγμένες.

Τα μεταφορικά δίκτυα παρίστανται συνήθως με την χρήση ζυγισμένων γράφων. Αυτοί οι γράφοι συνδέουν κάθε ακμή με ένα βάρος, όπως αυτό παρουσιάστηκε στην προηγούμενη ενότητα. Το βάρος αυτό, αποτελεί το κόστος για να διασχίσουμε την ακμή και μπορεί να αναφέρεται σε χρόνο, απόσταση μεταξύ των δύο κόμβων, σε άλλες παραμέτρους ή συνηθέστερα σε συνδυασμό των παραπάνω και άλλων παραμέτρων οπότε και αναφέρεται ως «γενικευμένο κόστος».

2.3.1 Έννοιες συνυφασμένες με τα δίκτυα

Μια από τις βασικότερες έννοιες, που συναντάει κανείς στα δίκτυα, είναι το μονοπάτι. Αντίστοιχα με τον ορισμό που δίνεται στην προηγούμενη ενότητα, σε ένα δίκτυο μεταφοράς, μονοπάτι αποτελεί μία ορισμένη ακολουθία διαδοχικών στάσεων.

Όπως αντιλαμβάνεται κανείς, σε ένα δίκτυο μπορούν να εντοπιστούν αρκετά μονοπάτια ανάλογα μάλιστα με το πλήθος των ακμών και των κόμβων που διαθέτει το εκάστοτε δίκτυο. Από όλα αυτά τα μονοπάτια, ιδιαίτερο ενδιαφέρον παρουσιάζουν εκείνα που έχουν κάποια επιπρόσθετα χαρακτηριστικά όπως για παράδειγμα το συντομότερο μονοπάτι. ***Συντομότερο μονοπάτι*** (shortest path) είναι το μονοπάτι με το ελάχιστο κόστος από έναν κόμβο-αφετηρία προς έναν κόμβο-προορισμό. Πρακτικά λοιπόν, το να βρούμε ένα συντομότερο μονοπάτι σημαίνει να βρούμε την ακολουθία των κόμβων και ακμών που μας οδηγεί από την αφετηρία στον προορισμό με το μικρότερο κόστος.

Αντίστοιχα, ως ***μονοπάτι κύκλος*** ορίζουμε το κλειστό εκείνο μονοπάτι, του οποίου η αφετηρία και ο προορισμός συμπίπτουν. Ιδιαίτερο ενδιαφέρον παρουσιάζουν για κάποια δίκτυα και τα μονοπάτια ***fundamental circuits***. Πρόκειται για κλειστά μονοπάτια τα οποία δεν περιέχουν άλλα κλειστά μονοπάτια.

Όπως γίνεται αντιληπτό, σημαντικό ρόλο στον καθορισμό των μονοπατιών ενός δικτύου, παίζουν οι ακμές του, οι οποίες είναι γνωστές και ως ***σύνδεσμοι*** ενός δικτύου. Αυτές καθορίζουν ποιοι κόμβοι είναι προσβάσιμοι, από ποιους κόμβους είναι προσβάσιμοι καθώς και τα δυνατά μονοπάτια που οδηγούν από έναν κόμβο σε κάποιον άλλον. Κάθε σύνδεσμος μπορεί να χαρακτηρίζεται από μια κατεύθυνση που καθορίζει ποιος κόμβος είναι η κεφαλή και ποιος η ουρά του συνδέσμου. Σε πολλά δίκτυα οι σύνδεσμοι δεν είναι δικατευθυνόμενοι (οι κόμβοι είναι κεφαλές και ουρές ταυτόχρονα για την ίδια ακμή) με αποτέλεσμα τα δίκτυα αυτά να αναπαρίστανται από κατευθυνόμενους γράφους, δηλαδή γράφους που δηλώνουν την κατεύθυνση των συνδέσμων τους.

Ανάλογα με τους συνδέσμους που περιέχει ένα δίκτυο, μπορούμε να ορίσουμε την συνδεσιμότητα και την προσβασιμότητα ενός δικτύου. Με τον όρο ***συνδεσιμότητα*** (connectivity) αναφερόμαστε στο χαρακτηριστικό του δικτύου που μετρά την «προσπάθεια», δηλαδή τον ελάχιστο αριθμό συνδέσμων που απαιτούνται για να φθάσουμε από όλους τους κόμβους σε όλους τους υπόλοιπους κόμβους. Από την άλλη μεριά, ***προσβασιμότητα*** (accessibility) είναι το χαρακτηριστικό του δικτύου που αποτελεί μέτρο της «προσπάθειας» που απαιτείται για να φθάσουμε σε όλους ή σε κάποιους κόμβους από έναν συγκεκριμένο κόμβο.

Πέρα από τα μονοπάτια και τους συνδέσμους, υπάρχουν μερικές ακόμα έννοιες τις οποίες συναντάμε κατά την ανάλυση ενός δικτύου, όπως η έννοια της ***στάσης*** η οποία ορίζεται ως μία τοποθεσία που επισκεπτόμαστε καθώς διατρέχουμε ένα μονοπάτι ή κύκλο και η έννοια του ***κέντρου*** που αποτελεί μία τοποθεσία στην οποία παρέχονται κάποια αγαθά ή υπηρεσίες. Τέλος, ***στροφή*** σε ένα δίκτυο ονομάζουμε την μετάβαση από μία ακμή του δικτύου σε μία άλλη.

2.3.2 Μέτρα δικτύου

Με στόχο την σύγκριση δύο ή περισσοτέρων δικτύων, οι ερευνητές έχουν συντάξει ορισμένους δείκτες/μέτρα τα οποία ποσοτικοποιούν διάφορα χαρακτηριστικά των δικτύων. Τα πιο γνωστά μέτρα είναι τα ακόλουθα (Rodrigue, Comtois, & Slack, 2006):

Δείκτης Άλφα (Alpha Index, α): Αποτελεί μέτρο σύγκρισης της συνδεσιμότητας του δικτύου και συγκρίνει τον αριθμό των fundamental circuits μονοπατιών με τον μέγιστο αριθμό όλων των δυνατών μονοπατιών.

$$\alpha = \frac{u}{2v - 5}$$

Όπου u είναι ο μέγιστος αριθμός των ανεξάρτητων κύκλων σε έναν γράφο, το οποίο υπολογίζεται βάση του αριθμού των κόμβων v, ακμών e και υπο-γράφων(subgraphs) p ως εξής:

$$u = e - v + p$$

Δείκτης Βήτα (Beta Index, β): Ο δείκτης αυτός συγκρίνει τον αριθμό των ακμών με τον αριθμό των κόμβων σε ένα δίκτυο και δίνεται από τον τύπο:

$$\beta = \frac{e}{v}$$

Δείκτης Γάμμα (Gamma Index, γ): Ο δείκτης Γάμμα συγκρίνει τον πραγματικό αριθμό ακμών με τον μέγιστο αριθμό ακμών ($\frac{1}{2}v(v-1)$).

$$\gamma = \frac{e}{\frac{1}{2}v(v-1)}$$

Associated Number (ή Koenig Number): Μετρά την προσβασιμότητα (ή κεντρικότητα) ενός κόμβου από τον αριθμό των ακμών που απαιτούνται για να συνδεθεί αυτός ο κόμβος με τον (τοπολογικά) πιο απομακρυσμένο κόμβο του δικτύου.

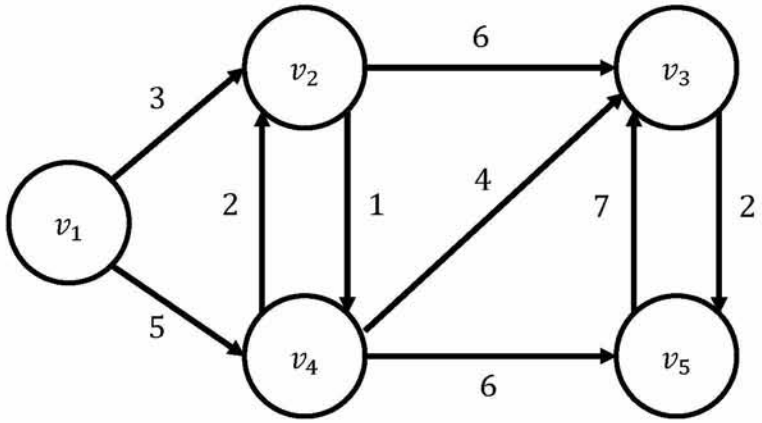
Δείκτης Shimbel: Μετρά τον ελάχιστο αριθμό ακμών που είναι απαραίτητες για να συνδεθεί ένας κόμβος με όλους τους κόμβους στο δίκτυο.

Διάμετρος δικτύου: Είναι ο αριθμός των ακμών σε ένα συντομότερο μονοπάτι μεταξύ του ζεύγους των πιο απομακρυσμένων κόμβων.

Βαθμός κόμβου: Είναι ο αριθμός των (άμεσων) ακμών που συνδέουν έναν κόμβο με τους γειτονικούς του.

2.4 Αναπαράσταση δικτύων

Ένα δίκτυο μπορεί να απεικονιστεί οπτικά σαν έναν γράφο $G = (V, E)$ όπου κάθε ακμή απεικονίζεται ως ένα ευθύγραμμο τμήμα που ενώνει δύο κόμβους. Η κατεύθυνση του βέλους δηλώνει τη διάταξη των κόμβων στην ακμή. Επίσης, δίπλα από κάθε ακμή εμφανίζεται ένας αριθμός που είναι το κόστος για να διασχίσουμε τη συγκεκριμένη ακμή. Τέλος, σε κάθε κόμβο αναγράφουμε ένα μοναδικό σύμβολο που χρησιμεύει για να τον αναγνωρίζουμε από τους υπόλοιπους. Με βάση τα παραπάνω, ένα τυχαίο δίκτυο θα μπορούσε να απεικονιστεί ως εξής:



Εικόνα 2.9 Γραφική αναπαράσταση δικτύου

Εκτός από την προφανή γεωμετρική αναπαράσταση των δικτύων, υπάρχουν διάφοροι τρόποι για να αποθηκευτεί ένα δίκτυο σε ένα υπολογιστικό σύστημα. Αυτό εξαρτάται από τη μορφή του εκάστοτε δικτύου και του αλγόριθμου που χρησιμοποιείται για την επίλυση του προβλήματος. Οι σημαντικότερες αναπαραστάσεις είναι οι ακόλουθες.

2.4.1 Πίνακας γειτνίασης κόμβων (ή πίνακας διπλανών κορυφών)

Θεωρούμε τον γράφο $G = (V, E)$ του παραπάνω σχήματος. Ο πίνακας γειτνίασης (Node-Node Adjacency Matrix) είναι στη γενικότερη μορφή ο $n \times n$ πίνακας $A(G)$, για τον οποίο ισχύει :

$$A(G) = [a_{ij}] \text{ όπου } a_{ij} = \begin{cases} \text{κόστος ακμής}, \text{αν } (v_i, v_j) \in E \\ 0, \text{αλλιώς} \end{cases}$$

Οι γραμμές και οι στήλες του πίνακα γειτνίασης αντιστοιχούν στους κόμβους του δικτύου. Σύμφωνα με τον ορισμό του πίνακα $A(G)$, ένα μηδενικό στοιχείο στην i-οστή γραμμή και την j-οστή στήλη του πίνακα υποδεικνύει ότι δεν υπάρχει ακμή (i, j) από τον κόμβο v_i προς τον κόμβο v_j. Ένα μη μηδενικό στοιχείο στην i-οστή γραμμή και την j-οστή στήλη του πίνακα υποδεικνύει ότι η ακμή (i, j) υπάρχει και δηλώνει την αριθμητική τιμή του κόστους της ακμής αυτής. (Zhan, 1998)

Ο χώρος που απαιτείται για την αποθήκευση ενός πίνακα διπλανών κορυφών είναι $a * n^2$ για ένα δίκτυο n κόμβων, όπου a είναι μια σταθερή ποσότητα.

Η αναπαράσταση με πίνακα διπλανών κορυφών είναι η πιο βασική μορφή αναπαράστασης δικτύου. Είναι η ευκολότερη στην υλοποίηση και η καταλληλότερη για πυκνά δίκτυα. Ο πίνακας γειτνίασης για τον γράφο του σχήματός μας έχει ως εξής :

0	3	0	5	0
0	0	6	2	0
0	0	0	0	2
0	1	4	0	6
0	0	7	0	0

Εικόνα 2.10 Πίνακας γειτνίασης κόμβων

2.4.2 Πίνακας πρόσπτωσης κόμβων – ακμών

Θεωρούμε τον γράφο $G = (V, E)$ του παραπάνω σχήματος. Για να κατασκευάσουμε τον πίνακα πρόσπτωσης πρέπει να αριθμήσουμε τις m ακμές του. Έτσι, θα είναι $E = (e_1, e_2, e_3, \dots, e_m)$. Ο πίνακας πρόσπτωσης κόμβων-ακμών είναι ένας $n \times m$ πίνακας $B(G)$ για τον οποίο ισχύει:

$$B(G) = [b_{ij}], \text{όπου } b_{ij} = \begin{cases} 1, \text{αν η ακμή } e_j \text{προσπίπτει στον } v_i \\ 0, \text{αλλιώς} \end{cases}$$

Οι γραμμές του πίνακα πρόσπτωσης αντιστοιχούν στους κόμβους του δικτύου, ενώ οι στήλες του αντιστοιχούν στις ακμές του δικτύου.

Ένα μηδενικό στοιχείο στην i-οστή γραμμή και j-οστή στήλη του πίνακα υποδεικνύει ότι η ακμή με αρίθμηση j δεν προσπίπτει στον κόμβο v_i, ενώ ένα μη μηδενικό στοιχείο στην ίδια θέση, υποδεικνύει ότι η ακμή με αρίθμηση j προσπίπτει στον κόμβο v_i. (Zhan, 1998)

Ο αποθηκευτικός χώρος που απαιτείται για έναν πίνακα πρόσπτωσης είναι $b * n * m$, όπου n ο αριθμός κόμβων του δικτύου, m ο αριθμός ακμών και b σταθερά. Η αναπαράσταση με πίνακα πρόσπτωσης είναι εύκολη στην υλοποίηση και ενδείκνυται για πυκνά δίκτυα.

Ο γράφος του παραδείγματός μας, είναι ένας ζυγισμένος γράφος. Έτσι, στην περίπτωση που η ακμή e_j προσπίπτει στον v_i, στη θέση (j, i) του πίνακα εμφανίζεται το κόστος της ακμής. Επιπλέον, ο γράφος είναι και κατευθυνόμενος, δηλαδή κάθε ακμή είναι κατευθυνόμενη από την ουρά προς την κεφαλή. Το κόστος για την κορυφή - ουρά του γράφου εμφανίζεται στον πίνακα με θετικό πρόσημο ενώ για την κορυφή - κεφαλή με αρνητικό.

Με βάση την αρίθμηση που θεωρήσαμε και τις παραπάνω παραδοχές κατασκευάζουμε τον πίνακα πρόσπτωσης που αντιστοιχεί στο δίκτυό μας ως εξής :

−3	−5	0	0	0	0	0	0	0
3	0	−6	−2	0	1	0	0	0
0	0	6	0	−2	0	4	0	7
0	5	0	2	0	−1	−4	−6	0
0	0	0	0	2	0	0	6	−7

Εικόνα 2.11 Πίνακας πρόσπτωσης κόμβων – ακμών

2.4.3 Λίστες γειτνίασης κόμβων (ή λίστα διπλανών κορυφών)

Θεωρούμε τον γράφο $G = (V, E)$. Σε κάθε έναν από τους κόμβους του αντιστοιχίζουμε μία λίστα. Σύμφωνα με αυτή την αναπαράσταση, η i-οστή λίστα αντιστοιχεί στον i-οστό κόμβο οπότε περιέχει όλους τους γειτονικούς κόμβους του κόμβου i, δηλαδή όλους τους κόμβους που συνδέονται με τον i μέσω κάποιας ακμής.

Η λίστα μπορεί να υλοποιηθεί ως σειριακή ή ως συνδεδεμένη. Αν ο αριθμός κόμβων της λίστας είναι καθορισμένος εκ των προτέρων, υλοποιείται ως στατική δομή, διαφορετικά αν

θέλουμε να υπάρχει δυνατότητα η λίστα να επεκταθεί ή να συρρικνωθεί κατά την εκτέλεση του προγράμματος, υλοποιείται ως δυναμική.

Η πιο ευέλικτη από τις παραπάνω είναι η συνδεδεμένη, δυναμική λίστα. Σε αυτήν την αναπαράσταση, για κάθε κόμβο κρατάμε τον αριθμό που λειτουργεί ως αναγνωριστικό του (identifier) και σε ένα άλλο πεδίο τον δείκτη προς τον επόμενο κόμβο.

Για την αναπαράσταση των κεφαλών απαιτούνται n λίστες και επιπλέον m στοιχεία για την αποθήκευση των κόμβων προορισμού. Η αναπαράσταση με λίστα διπλανών κορυφών ενδείκνυται αν ο γράφος είναι αραιός. Ακολουθεί η απεικόνιση τις λίστας γειτνίασης που αντιστοιχεί στον γράφο του σχήματός μας:

[1] -> 2, 4
[2] -> 3, 4
[3] -> 5
[4] -> 2, 3, 5
[5] -> 3

Εικόνα 2.12 Λίστα γειτνίασης κόμβων

2.4.4 Αναπαράσταση forward και reverse star

Για την αναπαράσταση αυτή αποθηκεύουμε τις ακμές που ξεκινούν από τους κόμβους (forward star) του γράφου καθώς και τις ακμές που καταλήγουν σε αυτούς (reverse star), χρησιμοποιώντας μονοδιάστατους πίνακες (Zhan, Three Fastest Shortest Path Algorithms on Real Road Networks : Data Structures and Procedures, 2001) (Ahuja, Magnanti, & Orlin, 1993).

Προκειμένου να κατασκευάσουμε την forward star αναπαράσταση ενός γράφου, θα χρειαστεί να αριθμούμε τις ακμές του με κατάλληλο τρόπο. Ο τρόπος με τον οποίο αριθμούνται οι ακμές έχει ως εξής :

Πρώτα, αριθμούνται οι ακμές που ξεκινούν από τον κόμβο 1, στη συνέχεια οι ακμές που ξεκινούν από τον κόμβο 2 κ.ο.κ. Αν από έναν κόμβο ξεκινούν περισσότερες από μία ακμές, η σειρά με την οποία αριθμούνται είναι τυχαία.

Ακολούθως, τα δεδομένα που αφορούν στις ακμές αποθηκεύονται σειριακά στους ακόλουθους τέσσερεις μονοδιάστατους πίνακες:

1. ο πίνακας με τους αύξοντες αριθμούς,
2. ο πίνακας κόμβων από τους οποίους ξεκινούν οι ακμές ($Starting\_Node$),
3. ο πίνακας κόμβων στους οποίους καταλήγουν οι ακμές ($Ending\_Node$) και
4. ο πίνακας που περιέχει τα κόστη των ακμών ($Cost$).

Έτσι, αν σε μία ακμή (i, j) αντιστοιχίσουμε τον αριθμό 3, τότε ο αριθμός του κόμβου από τον οποίο ξεκινά, ο αριθμός του κόμβου που καταλήγει και το κόστος της ακμής θα αποθηκευτούν αντιστοίχως στις $Starting\_Node[3]$, $Ending\_Node[3]$ και $Cost[3]$.

Παράλληλα με την παραπάνω πληροφορία, αποθηκεύουμε για κάθε κόμβο i, τον δείκτη $pointer(i)$, ο οποίος υποδηλώνει τον μικρότερο αύξοντα αριθμό των ακμών που ξεκινούν από τον συγκεκριμένο κόμβο. Στην περίπτωση που δεν υπάρχει ακμή που να ξεκινά από τον κόμβο i, ο $pointer(i)$ τίθεται ίσος με τον $pointer(i+1)$, ενώ για λόγους πληρότητας θέτουμε $pointer(1) = 1$ και $pointer(n+1) = m+1$.

Η αναπαράσταση forward star μπορεί να χρησιμοποιηθεί για να καθορίσει επαρκώς το σύνολο των ακμών που ξεκινούν από οποιονδήποτε κόμβο του γράφου.

Με βάση τις παραπάνω παραδοχές και αφού αριθμήσουμε τις εξερχόμενες ακμές κάθε κόμβου, η αναπαράσταση forward star του γράφου μας έχει ως εξής:

α/α ακμής	Starting_node	Ending_node	Cost
1	1	2	3
2	1	4	5
3	2	3	6
4	2	4	2
5	3	5	2
6	4	2	1
7	4	3	4
8	4	5	6
9	5	3	7

Εικόνα 2.13 Forward star αναπαράσταση δικτύου

Στην συνέχεια παρουσιάζεται ο πίνακας με τους δείκτες των κόμβων που αναφέρθηκε παραπάνω:

Κόμβος	Αριθμός ακμής
1	1
2	3
3	5
4	6
5	9
6	10

Εικόνα 2.14 Πίνακας δεικτών forward star

Σε πλήρη αντιστοιχία με την αναπαράσταση forward star, η αναπαράσταση reverse star μπορεί να χρησιμοποιηθεί για να καθορίσει επαρκώς το σύνολο των εισερχομένων ακμών προς οποιονδήποτε κόμβο. Συνεπώς στην συγκεκριμένη αναπαράσταση αριθμούμε τις ακμές με βάση το ποιες εισέρχονται σε κάθε κόμβο.

Ο πίνακας που ακολουθεί δείχνει την αναπαράσταση reverse star του γράφου μας:

α/α ακμής	Starting_node	Ending_node	Cost
1	1	2	3
2	4	2	1

α/α ακμής	Starting_node	Ending_node	Cost
3	2	3	6
4	4	3	4
5	5	3	7
6	1	4	5
7	2	4	2
8	3	5	2
9	4	5	6

Εικόνα 2.15 Reverse star αναπαράσταση δικτύου

Στην συνέχεια παρουσιάζεται ο πίνακας με τους δείκτες των κόμβων για την reverse star αναπαράσταση:

Κόμβος	Αριθμός ακμής
1	1
2	1
3	3
4	6
5	8
6	10

Εικόνα 2.16 Πίνακας δεικτών reverse star

Είναι φανερό ότι στην περίπτωση που αξιοποιήσουμε τόσο την αναπαράσταση forward star όσο και την αναπαράσταση reverse star για να παραστήσουμε ένα δίκτυο, ένα σημαντικό μέρος των πληροφοριών που αποθηκεύονται είναι περιττές αφού αποτελούν απλές επαναλήψεις των ίδιων δεδομένων.

Για να μειώσουμε τον αριθμό των διπλο-εγγραφών μπορούμε να αποθηκεύσουμε μόνο έναν πίνακα που καλείται $trace$. Ο πίνακας αυτός αποθηκεύει τους αριθμούς των ακμών της forward star αναπαράστασης με την σειρά που εμφανίζονται αυτές στην reverse star αναπαράσταση.

Για παράδειγμα, η δεύτερη ακμή στην αναπαράσταση reverse star είναι η ακμή $(4,2)$ της οποίας ο αριθμός στην αναπαράσταση forward star είναι το 6. Έτσι, το στοιχείο $trace(2) = 6$. Η πέμπτη ακμή στην αναπαράσταση reverse star είναι η $(5,3)$ της οποίας ο αριθμός ακμής στην αναπαράσταση forward star είναι ο 9. Δηλαδή, έχουμε $trace(5) = 9$. Με τον ίδιο τρόπο μπορούμε να υπολογίσουμε όλα τα στοιχεία του πίνακα $trace$ του οποίου το μέγεθος θα είναι m, όσο δηλαδή και το πλήθος των ακμών του δικτύου.

Ο χώρος που απαιτείται για την αποθήκευση ενός δικτύου με n κόμβους και m ακμές με χρήση της αναπαράστασης forward και reverse star είναι $a * n + b * m$, όπου τα a, b είναι κάποιες σταθερές ποσότητες.

Το πλεονέκτημα της αναπαράστασης forward και reverse star σε σύγκριση με τις προαναφερθείσες μεθόδους αναπαράστασης είναι ότι εξοικονομεί χώρο κατά την αποθήκευση των δεδομένων. Η αναπαράσταση αυτή μπορεί να διαβαστεί και να επεξεργαστεί γρήγορα και εύκολα και είναι κατάλληλη τόσο για πυκνά όσο και για αραιά δίκτυα (Dial, Glover, Karney, & Klingman, 1979).

2.4.5 Σύγκριση τρόπων αναπαράστασης

Συνοψίζοντας όλα όσα παρουσιάστηκαν παραπάνω μπορούμε να αποφανθούμε τα εξής για τους τρόπους αναπαράστασης ενός δικτύου:

- Ο πιο βασικός τρόπος αναπαράστασης είναι ο πίνακας γειτνίασης κόμβων. Το πλεονέκτημά του είναι ότι είναι ο ευκολότερος στην υλοποίηση. Παρόλο που «κοστίζει» σε αποθηκευτικό χώρο είναι ο πλέον κατάλληλος για πυκνά δίκτυα.
- Για τον πίνακα κόμβων-ακμών ισχύει ότι και για τον πίνακα γειτνίασης κόμβων. Ωστόσο, ανάμεσα σε αυτούς τους δύο τρόπους, χρησιμοποιείται πιο συχνά ο πίνακας γειτνίασης κόμβων.
- Η λίστα γειτνίασης κόμβων αποθηκεύει για κάθε κόμβο, όλους τους γειτονικούς κόμβους. Ο τρόπος αυτός αναπαράστασης είναι ιδιαίτερα αποδοτικός για αραιά δίκτυα.
- Στη forward και reverse star αναπαράσταση αποθηκεύουμε τις ακμές που ξεκινούν από κάθε κορυφή σε έναν μονοδιάστατο πίνακα και έχουμε πίνακες με μέγεθος ίσο με τον αριθμό των κορυφών του δικτύου. Ο τρόπος αυτός αναπαράστασης είναι ο πλέον επαρκής διότι εξοικονομεί χώρο. Είναι κατάλληλος τόσο για αραιά όσο και για πυκνά δίκτυα.

Με βάση όλα τα παραπάνω, είναι ξεκάθαρο ότι η forward και reverse star αναπαράσταση αποτελεί την βέλτιστη επιλογή όσον αφορά στην αποτύπωση δικτύων (Zhan, Three Fastest Shortest Path Algorithms on Real Road Networks : Data Structures and Procedures, 2001).

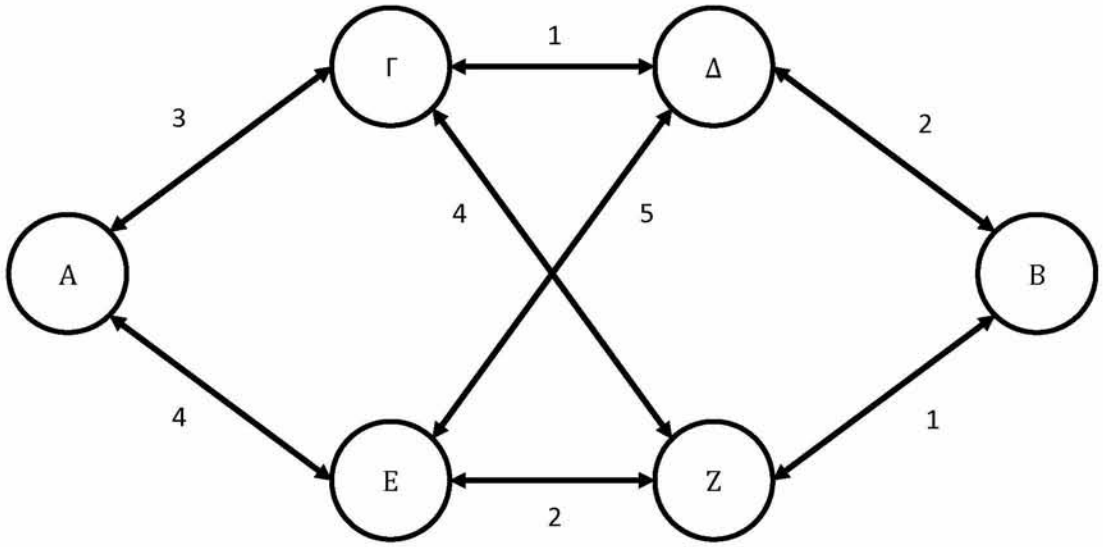
2.5 Δρομολόγηση

Η έννοια της Δρομολόγησης σχετίζεται με την κατάστρωση και σχεδίαση της βέλτιστης διαδρομής πάνω σε ένα δίκτυο, του οποίου οι ακμές (δυνατές μεταβάσεις μεταξύ των κόμβων) χαρακτηρίζονται από κόστη. Όπως έχει αναφερθεί και σε προηγούμενη ενότητα, τα κόστη αυτά μπορεί να είναι ο χρόνος μεταφοράς, η απόσταση, τα έξοδα μεταφοράς ή και συνδυασμός αυτών, ανάλογα με τη φύση του προβλήματος που εξετάζεται. Ο αντικειμενικός σκοπός του προβλήματος είναι ο εντοπισμός της βέλτιστης διαδρομής, δηλαδή της διαδρομής που οδηγεί από έναν κόμβο σε έναν άλλον και το κόστος αυτής είναι το ελάχιστο δυνατόν. Το πρόβλημα της δρομολόγησης αποτελεί ένα από τα πιο πολυσυζητημένα προβλήματα της επιστημονικής περιοχής της βελτιστοποίησης. Οι αλγόριθμοι και οι τεχνικές που χρησιμοποιούνται για την εύρεση ή προσέγγιση της

βέλτιστης διαδρομής, απαντώνται τακτικά σε προβλήματα της Συνδυαστικής Βελτιστοποίησης (Combinatorial Optimization), με προεκτάσεις στις συγγενείς περιοχές της Στοχαστικής Βελτιστοποίησης (Stochastic Optimization) και Βέλτιστου Ελέγχου (Optimal Control).

Προκειμένου να γίνει κατανοητή η έννοια της Δρομολόγησης δίνεται ένα απλό παράδειγμα.

«Στο ακόλουθο σχήμα έχουμε ένα χάρτη που δείχνει όλες τις χιλιομετρικές αποστάσεις μεταξύ κάποιων πόλεων. Αν υποθέσουμε ότι είμαστε στη πόλη Α και θέλουμε να πάμε στην πόλη Β, ποια είναι η συντομότερη διαδρομή που πρέπει να ακολουθήσουμε;»



Εικόνα 2.17 Απλός χάρτης χιλιομετρικών αποστάσεων

Η εύρεση της συντομότερης διαδρομής ή αλλιώς βέλτιστου μονοπατιού ορίζει ένα πρόβλημα δρομολόγησης, το οποίο αποτελεί μία απλοποιημένη εκδοχή της ευρύτερης κλάσης προβλημάτων δρομολόγησης. Το παραπάνω πρόβλημα είναι γνωστό σαν το Πρόβλημα του Συντομότερου Μονοπατιού (Shortest Path Problem). Αποτελεί σημείο αναφοράς για σχεδόν όλα τα προβλήματα δρομολόγησης και απαντάται σήμερα σε πολλές επιστημονικές περιοχές, όπως στη Θεωρητική Πληροφορική και Επιστήμη των Υπολογιστών (Theoretical Informatics and Computer Science), στην Επιχειρησιακή Έρευνα (Operations Research), στα Διακριτά Μαθηματικά (Discrete Mathematics) κ.α.

2.5.1 Δημοφιλή προβλήματα Δρομολόγησης

Ο τρόπος και η δυσκολία εντοπισμού της βέλτιστης διαδρομής εξαρτάται άμεσα από τον τρόπο με τον οποίο είναι ορισμένο το πρόβλημα και την τοπολογία του δικτύου όπως στην περίπτωση κατά την οποία η ζητούμενη διαδρομή είναι κλειστή ή ανοιχτή (μονοπάτι ή κύκλος), αν τα κόστη μεταβάλλονται ή είναι σταθερά, αν η τοπολογία του δικτύου αλλάζει με το χρόνο κ.α. Ανάλογα με τα χαρακτηριστικά τόσο της ζητούμενης διαδρομής όσο και του δικτύου, μπορούμε να κατηγοριοποιήσουμε τα προβλήματα σε κλάσεις με κοινά

χαρακτηριστικά. Παρακάτω δίνονται οι συνοπτικές περιγραφές ορισμένων από τα πιο γνωστά προβλήματα που συναντούμε στην επιστημονική βιβλιογραφία.

Το Πρόβλημα του Πλανόδιου Πωλητή (Travelling Salesman Problem – TSP) αποτελεί το πλέον διαδεδομένο πρόβλημα δρομολόγησης. Απαντάται συχνά στην περιοχή της Θεωρητικής Πληροφορικής και της Μαθηματικής Βελτιστοποίησης (Ahuja, Magnanti, & Orlin, 1993). Ο ορισμός του προβλήματος αυτού περιγράφεται ως εξής:

> *«Δοθείσας μιας λίστας πόλεων και των μεταξύ τους αποστάσεων, να βρεθεί η συντομότερη διαδρομή, που επισκέπτεται κάθε πόλη ακριβώς μία φορά και καταλήγει στην πόλη από την οποία ξεκινά (αφετηρία)»*

Το πρόβλημα διατυπώθηκε για πρώτη φορά το 1930 και αποτελεί βασικό αντικείμενο μελέτης και ανάλυσης για πολλά ακόμη προβλήματα βελτιστοποίησης. Η απλότητα της διατύπωσης του προβλήματος είναι παραπλανητική. Αυτό διαφαίνεται από το γεγονός ότι το TSP είναι ένα από τα πιο συχνά μελετημένα προβλήματα των Υπολογιστικών Μαθηματικών, αφού καμία γενική μέθοδος επίλυσης του δεν έχει βρεθεί μέχρι σήμερα. Λόγω αυτής της μεγάλης δυσκολίας που παρουσιάζει το TSP ως προς την υπολογισιμότητα και την πολυπλοκότητά του, ένας μεγάλος αριθμός ευρετικών αλγορίθμων έχουν προταθεί, παρουσιάζοντας λύσεις ακόμα και στην περίπτωση όπου υπάρχουν πολλές πόλεις στα δεδομένα του προβλήματος. Ορισμένες παραλλαγές του προβλήματος εισάγουν διάφορους περιορισμούς, όπως η ύπαρξη χρονοπαραθύρων (time windows), ή περιορισμένων πόρων (limited resources), οπότε το πρόβλημα γίνεται σαφώς δυσκολότερο. Στη θεωρία της Υπολογιστικής Πολυπλοκότητας, το TSP κατατάσσεται στην NP κλάση προβλημάτων. Αυτό σημαίνει ότι δεν υπάρχει αλγόριθμος που να επιλύει το πρόβλημα σε πολυωνυμικό χρόνο, δηλαδή σε χρόνο που εξαρτάται πολυωνυμικά από το πλήθος των πόλεων που περιέχει στα δεδομένα το πρόβλημα. Επομένως όταν αυξάνονται οι πόλεις, ο απαιτούμενος χρόνος μεγαλώνει εκθετικά, με αποτέλεσμα ένα πρόβλημα μερικών εκατοντάδων πόλεων να απαιτεί μερικά χρόνια για να λυθεί ακριβώς σε έναν υπολογιστή.

Το Πρόβλημα Δρομολόγησης Οχημάτων (Vehicle Routing Problem-VRP) αποτελεί ένα από τα πιο σημαντικά προβλήματα που συναντούμε στην περιοχή της Συνδυαστικής Βελτιστοποίησης (Combinatorial Optimization) και εντάσσεται στην κατηγορία των προβλημάτων Ακέραιου Προγραμματισμού. Τα προβλήματα αυτά αποτελούν συνήθως ειδικές περιπτώσεις προβλημάτων Γραμμικού Προγραμματισμού (Linear Programming), στα οποία όμως οι μεταβλητές λαμβάνουν επιτρεπτές τιμές από ένα σύνολο ακεραίων αριθμών. Τα προβλήματα του Ακέραιου Προγραμματισμού παρουσιάζουν και αυτά πολυπλοκότητα NP-τύπου, δηλαδή δεν έχει βρεθεί αλγόριθμος που να τα επιλύει σε μη πολυωνυμικό χρόνο. Η διατύπωση του προβλήματος είναι αρκετά απλή (Ahuja, Magnanti, & Orlin, 1993):

> *«Ζητούμε τον καθορισμό του συνόλου βελτίστων διαδρομών για έναν στόλο οχημάτων, προκειμένου να εξυπηρετηθούν κάποιοι πελάτες που εντοπίζονται σε καθορισμένους κόμβους ενός δικτύου. Τα οχήματα αναχωρούν από μια ή περισσότερες αποθήκες ανεφοδιασμού (depots), στις οποίες και επιστρέφουν με το πέρας της αποστολής τους»*

Η θεωρητική έρευνα και οι πρακτικές εφαρμογές στο πεδίο δρομολόγησης οχημάτων ξεκίνησαν το 1959 με το «πρόβλημα αποστολής φορτηγών» (truck dispatching problem), το οποίο παρουσιάστηκε από τους Dantzig και Ramser (Dantzig & Ramser, 1959). Ο ορισμός του προβλήματος αυτού περιγράφεται ως εξής:

> *«Να βρεθεί η βέλτιστη δρομολόγηση (σχεδιασμός διαδρομής) ενός στόλου φορτηγών διανομής καυσίμου, μεταξύ ενός τερματικού σταθμού ανεφοδιασμού και ενός μεγάλου αριθμού σταθμών εξυπηρέτησης, οι οποίοι τροφοδοτούνται με καύσιμο από τον σταθμό ανεφοδιασμού»*

Χρησιμοποιώντας μια μέθοδο βασισμένη στις αρχές του Γραμμικού Προγραμματισμού, οι Dantzig και Ramser κατέληξαν σε μια προσεγγιστικά βέλτιστη λύση που περιλάμβανε τέσσερις διαδρομές για ένα πρόβλημα δώδεκα σταθμών εξυπηρέτησης. Μερικά χρόνια αργότερα και συγκεκριμένα το 1964, οι Clarke και Wright πρότειναν έναν ευρετικό αλγόριθμο βασισμένο στην άπληστη μέθοδο (greedy method) (Clarke & Wright, 1964), ο οποίος πέτυχε καλύτερες προσεγγίσεις από αυτές των Dantzig και Ramser. Ακολουθώντας τις παραπάνω μελέτες, εκατοντάδες αλγόριθμοι και μοντέλα προτάθηκαν για την ακριβή και προσεγγιστική επίλυση διαφόρων περιπτώσεων VRP. Σήμερα, υπάρχουν διαθέσιμα στην αγορά αρκετά πακέτα λογισμικού που χρησιμοποιούνται για την επίλυση πραγματικών προβλημάτων διαχείρισης στόλων οχημάτων. Τις τελευταίες τέσσερις δεκαετίες, η επιστημονική κοινότητα έχει στρέψει έντονα την προσοχή της στη μελέτη του VRP. Αυτή η στροφή συμπίπτει με την αύξηση της χρήσης των Κατανεμημένων Συστημάτων (Distributed Systems). Ένα κατανεμημένο σύστημα αποτελείται από γεωγραφικά ανεξάρτητες, αυτόνομες υπολογιστικές συσκευές, που έχουν τη δυνατότητα να επικοινωνούν μεταξύ τους και να λειτουργούν συντονισμένα για την επίτευξη ενός κοινού στόχου. Σε αντίθεση με τα ανεξάρτητα συστήματα, ο σχεδιασμός των κατανεμημένων συστημάτων, η κατανόηση της λειτουργίας τους κάτω από ιδιαίτερες συνθήκες καθώς και η ανάλυση της συμπεριφοράς τους απαιτεί ειδικές γνώσεις και ικανότητες. Το VRP φέρει ως επίκεντρο την ιδέα της διανομής αγαθών ανάμεσα σε ένα σύνολο αποθηκών (depots) και ένα σύνολο γεωγραφικά διασκορπισμένων πελατών (customers). Μπορούμε να θεωρήσουμε τις θέσεις των πελατών σαν κόμβους εξυπηρέτησης πάνω σε ένα (οδικό) δίκτυο, στο οποίο σημειώνουμε όλες τις δυνατές μεταβάσεις (ακμές) που ενώνουν τους κόμβους εξυπηρέτησης μεταξύ τους, αλλά και με τους σταθμούς ανεφοδιασμού. Η συλλογή στερεών αποβλήτων (solid waste collection), η δρομολόγηση σχολικών λεωφορείων (school bus routing), η μεταφορά ατόμων με αναπηρία (transportation of handicapped persons) και η διαχείριση μονάδων συντήρησης (management of maintenance units) αποτελούν μερικά παραδείγματα πραγματικών προβλημάτων στα οποία εφαρμόζονται παρεμφερείς τεχνικές επίλυσης με αυτές που χρησιμοποιούμε στο VRP.

Το Πρόβλημα του Συντομότερου Μονοπατιού (Shortest Path Problem - SPP) αποτελεί ένα κλασικό, συνδυαστικό πρόβλημα βελτιστοποίησης, του οποίου η επίλυση έχει εφαρμογή σε πολλά πραγματικά τεχνικά προβλήματα. Ο ορισμός του προβλήματος δίνεται ως εξής (Ahuja, Magnanti, & Orlin, 1993):

> *«Δίνεται ένας κατευθυνόμενος γράφος $G = (V, A)$ με τους κόμβους του αριθμημένους $1,2, \dots, N$. Κάθε ακμή $(i, j) \in A$ φέρει κόστος ή μήκος ίσο με a_{ij}. Το μήκος του μονοπατιού*

$(i_1, i_2, \dots, i_k)$ είναι το μήκος των ακμών του, δηλαδή η ποσότητα $\sum_{n=1}^{k} a_{i_n i_{n+1}}$. Το μονοπάτι αυτό χαρακτηρίζεται ως το συντομότερο αν έχει το μικρότερο μήκος από όλες τις διαδρομές με τους ίδιους κόμβους πηγής και προορισμού. Το μήκος του συντομότερου μονοπατιού λέγεται αλλιώς και συντομότερη απόσταση. Το πρόβλημα του συντομότερου μονοπατιού αφορά στον υπολογισμό της βέλτιστης διαδρομής μεταξύ επιλεγμένων ζευγών κόμβων.

Παρόλο που η περιγραφή του SPP είναι απλή, το πεδίο εφαρμογής του συγκεκριμένου προβλήματος είναι ιδιαιτέρως ευρύ. Οι περισσότεροι αλγόριθμοι που έχουν σχεδιαστεί για να αντιμετωπίσουν το συγκεκριμένο πρόβλημα ή την ειδική περίπτωση του προβλήματος του ελαχίστου κόστους ροής, μπορούν να θεωρηθούν ως τεχνικές βελτιστοποίησης βασικού ή δυϊκού κόστους. Βέβαια μέσα στα πλαίσια της σχετικής απλοποίησης του SPP, η επίλυση βασίζεται πάνω σε απλές αρχές, χωρίς να δίνεται ιδιαίτερη έμφαση στο κόστος βελτιστοποίησης.

Το SSP, όπως και το TSP, αποτελεί θεμελιώδες πρόβλημα βελτιστοποίησης, για δύο λόγους κυρίως.

- Γιατί οι αλγόριθμοι που το επιλύουν χρησιμοποιούνται σε μια γενικότερη πληθώρα ζητημάτων βελτιστοποίησης.
- Γιατί το πρόβλημα του SSP μοντελοποιείται με την χρήση γράφων και η επίλυση του στηρίζεται στην εξαγωγή συμπερασμάτων και ιδιοτήτων των δοσμένων δομών. Αυτή η έντονη ανάλυση των γράφων κεντρίζει το ενδιαφέρον των ατόμων που ασχολούνται με την επιστημονική περιοχή της Θεωρίας Γράφων.

2.6 Αλγόριθμοι δρομολόγησης

Έχοντας περιγράψει τα βασικά προβλήματα δρομολόγησης, στην συνέχεια του κεφαλαίου ασχολούμαστε με τους πιο διαδεδομένους αλγόριθμους που έχουν σχεδιαστεί για να αντιμετωπίσουν αυτά τα προβλήματα. Οι αλγόριθμοι δρομολόγησης μπορούν να διαχωριστούν σε στατικούς και δυναμικούς ανάλογα με την συμπεριφορά τους σε πιθανές αλλαγές στην δομή του δικτύου. Στα εμβρυακά στάδια της δρομολόγησης, οι επιστήμονες τροποποιούσαν στατικούς αλγόριθμους βελτιστοποίησης, (Psaraftis, 1995). Τα τελευταία χρόνια το σύνολο της επιστημονικής κοινότητας έχει στραφεί προς τους δυναμικούς αλγόριθμους μιας και διαφαίνεται ότι μπορούν να εφαρμοστούν καλύτερα σε πραγματικά προβλήματα. Οι αλγόριθμοι δυναμικής δρομολόγησης αντιδρούν ιδιαίτερα καλά στις ειδικές περιπτώσεις όπου μια συγκεκριμένη διαδρομή του δικτύου γίνεται για κάποιο λόγο μη διαθέσιμη. Οι αλγόριθμοι αυτοί μπορούν να δρομολογήσουν τους επιβάτες μέσα από εναλλακτικές διαδρομές άμεσα και χωρίς να χρειαστεί να ξανατρέξει ο αλγόριθμος.

2.6.1 Μέθοδος απόδοσης ετικέτας

Μια από τις πιο διαδεδομένες μεθόδους που αξιοποιείται από πληθώρα δυναμικών αλγορίθμων δρομολόγησης είναι αυτή της απόδοσης ετικέτας. Κάθε ακμή του δικτύου χαρακτηρίζεται από έναν αριθμό ο οποίος αναφέρεται ως ετικέτα. Η ετικέτα δεν αποτελεί

στοιχείο του γράφου, αλλά αποδίδεται στις ακμές του μέσω μίας μεθόδου, η οποία είναι γνωστή ως labeling method.

Η μέθοδος αυτή κατασκευάζει ένα δένδρο το οποίο αναπαριστά τα συντομότερα μονοπάτια από τον κόμβο - αφετηρία προς κάθε άλλο κόμβο i (Zhan, Three Fastest Shortest Path Algorithms on Real Road Networks : Data Structures and Procedures, 2001). Για κάθε κόμβο i αποθηκεύονται τρεις πληροφορίες:

1. Η τιμή της ετικέτας που αποτελεί το άνω όριο της απόστασης του συντομότερου μονοπατιού από τον κόμβο - αφετηρία έως τον τρέχοντα κόμβο i.
2. Ο κόμβος - πατέρας, που είναι ο κόμβος που προηγείται άμεσα του τρέχοντος κόμβου i στο δένδρο.
3. Η κατάσταση του κόμβου i, η οποία μπορεί να είναι μία από τις παρακάτω:
    I. Δεν έχει εξεταστεί (unreached). Αυτή είναι η αρχική κατάσταση όλων των κόμβων. Συνηθίζεται μάλιστα να δίνουν ετικέτα με τιμή άπειρο (∞) σε όσους κόμβους έχουν αυτή την κατάσταση.
    II. Προσωρινή ετικέτα (temporarily labeled ή απλά labeled). Σε αυτή την κατηγορία ανήκουν οι κόμβοι που αναμένεται περαιτέρω βελτίωση του συντομότερου μονοπατιού προς αυτούς.
    III. Μόνιμη ετικέτα (permanently labeled ή scanned). Ένας κόμβος αποκτά μόνιμη ετικέτα, όταν γνωρίζουμε ότι το τρέχον συντομότερο προς αυτόν μονοπάτι είναι και το απόλυτο συντομότερο μονοπάτι που μπορεί να επιτευχθεί.

Για την απόδοση των τιμών, οι αλγόριθμοι εκτελούν μια αναδρομική συνάρτηση, η οποία εξετάζει όλους τους κόμβους του δικτύου διαδοχικά. Συμβολίζουμε με $\Gamma^{-1}(i)$ το σύνολο των κόμβων που οδηγούν στον κόμβο i, το σύνολο των κόμβων του δικτύου θα το συμβολίσουμε με V και το κόστος μεταφοράς από τον κόμβο i στον j με το σύμβολο C_{ij}. Ορίζουμε τους μονοδιάστατους πίνακες Λ, P, S που έχουν τόσα στοιχεία όσοι και οι κόμβοι του δικτύου και αποθηκεύουν αντίστοιχα την τιμή της ετικέτας, τον πατέρα και την κατάσταση του κάθε κόμβου. Η μέθοδος απόδοσης ετικέτας με κόμβο – αφετηρία τον κόμβο N μπορεί να περιγραφεί με τα ακόλουθα βήματα:

Βήμα 1°: Αρχικοποίηση των πινάκων
Βήμα 1.1: $\Lambda[i] = \infty \quad \forall i \in V$
Βήμα 1.2: $P[i] = \kappa\varepsilon\nu\acute{o} \quad \forall i \in V$
Βήμα 1.3: $S[i] = \text{"unreached"} \quad \forall i \in V$
Βήμα 2°: Εκτελούμε την συνάρτηση Scan για τον κόμβο N.
Βήμα 3°: Τερματισμός μεθόδου.

Εικόνα 2.18 Βήματα μεθόδου απόδοσης ετικετών

Παράμετρος εισόδου: $SelectedNode$, ο τρέχων εξεταζόμενος κόμβος.
Βήμα 1°: Εκκίνηση αναδρομής $\forall i \in \Gamma^{-1}(SelectedNode)$
Αναδρομή 1.1: Εάν $\Lambda[SelectedNode] + C_{iSelectedNode} \geq \Lambda[i]$ πήγαινε στο 1.6
Αναδρομή 1.2: $\Lambda[i] = \Lambda[SelectedNode] + C_{iSelectedNode}$
Αναδρομή 1.3: $P[i] = SelectedNode$
Αναδρομή 1.4: $S[i] = "labeled"$
Αναδρομή 1.5: Εκτελούμε την συνάρτηση Scan για τον κόμβο i.
Αναδρομή 1.6: Επόμενη αναδρομή
Βήμα 2°: $S[SelectedNode] = "scanned"$.
Βήμα 3°: Επιστροφή μεθόδου

Εικόνα 2.19 Βήματα μεθόδου Scan με παράμετρο SelectedNode

2.6.2 Αλγόριθμοι στηριγμένοι στην απόδοση ετικέτας

Οι τρεις πιο διαδεδομένοι αλγόριθμοι επίλυσης του προβλήματος της δρομολόγησης είναι ο αλγόριθμος των Bellman-Ford, ο αλγόριθμος του Dijkstra και ο αλγόριθμος των Floyd-Warshall. Και οι τρεις αλγόριθμοι παρόλο που είναι γνωστοί για την επίλυση του προβλήματος συντομότερου μονοπατιού (Shortest Path Problem, SPP), έχουν χρησιμοποιηθεί με μικρές τροποποιήσεις για την επίλυση και άλλων προβλημάτων χρονοπρογραμματισμού χωρίς να χάνουν τις βασικές τους ιδιότητες (Cormen, Leiserson, Rivest, & Stein, 2001).

Ο αλγόριθμος των Bellman-Ford υπολογίζει τα συντομότερα μονοπάτια μιας πηγής (single source paths) σε γράφους που φέρουν βάρη επί των ακμών. Για γράφους με θετικά μόνο βάρη, ο αλγόριθμος Dijkstra επιλύει το πρόβλημα SPP πολύ πιο σύντομα από ότι ο αντίστοιχος των Bellman-Ford. Για αυτόν τον λόγο ο αλγόριθμος αυτός χρησιμοποιείται κυρίως για γραφήματα με αρνητικά βάρη. Ο αλγόριθμος πήρε το όνομά του από τους δημιουργούς του, τον Richard Bellman και τον Ford Lester. Εάν κάποιος γράφος περιέχει έναν "αρνητικό κύκλο", δηλαδή έναν κύκλο του οποίου το άθροισμα των βαρών είναι αρνητικό, τότε μπορούν να κατασκευαστούν μονοπάτια αυθαίρετα μικρού συνολικού βάρους, οπότε δεν μπορεί να υπάρξει συντομότερη διαδρομή. Ο αλγόριθμος των Bellman-Ford μπορεί να ανιχνεύσει κύκλους αρνητικού βάρους και δεν πρόκειται να κατασκευάσει μονοπάτι στο οποίο επαναλαμβάνεται κάποιος κόμβος (Bellman R. , 1958).

Ο αλγόριθμος του Dijkstra επινοήθηκε από τον Ολλανδό επιστήμονα Edsger Dijkstra το 1959 (Dijkstra, 1959), και είναι ένας αλγόριθμος αναζήτησης που επιλύει το πρόβλημα του συντομότερου μονοπατιού πάνω σε ένα γράφο με μη αρνητικά βάρη. Ο αλγόριθμος παράγει το συντομότερο δέντρο (Shortest Path Tree) όλων των μονοπατιών που οδηγούν από τον κόμβο πηγής στον κόμβο προορισμού. Ο αλγόριθμος αυτός χρησιμοποιείται πολύ συχνά στα προβλήματα δρομολόγησης. Ο Edward F.Moore είχε παρουσιάσει το 1957 έναν αλγόριθμο παρόμοιο με αυτόν του Dijkstra (Moore, 1959). Για έναν δοσμένο κόμβο πηγής πάνω στο γράφο, ο αλγόριθμος του Dijkstra υπολογίζει το μονοπάτι με το χαμηλότερο κόστος (δηλ. το συντομότερο μονοπάτι) ανάμεσα στον κόμβο πηγής και οποιονδήποτε άλλο κόμβο του γράφου (single-source algorithm). Ακόμα ο αλγόριθμος μπορεί να χρησιμοποιηθεί σε προβλήματα συντομότερου μονοπατιού ενός προορισμού, σταματώντας τα βήματά του μόλις ο επιθυμητός κόμβος προορισμού περιληφθεί στο βέλτιστο μονοπάτι

που σταδιακά υπολογίζουμε. Στην περίπτωση που οι κόμβοι ενός γράφου αναπαριστούν πόλεις και τα κόστη των δυνατών μεταβάσεων (ακμών) αναπαριστούν τις χιλιομετρικές αποστάσεις, είναι προφανές πως ο αλγόριθμος του Dijkstra μπορεί να χρησιμοποιηθεί για να βρεθούν οι συντομότερες διαδρομές μεταξύ μιας δεδομένης πόλης και όλων των άλλων πόλεων. Για το λόγο αυτό, ο συγκεκριμένος αλγόριθμος χρησιμοποιείται πολύ συχνά στην επίλυση προβλημάτων δρομολόγησης και συγκεκριμένα σε ό,τι αφορά την μεταφορά πακέτων στα δίκτυα υπολογιστών μιας και αποτελεί τμήμα γνωστών πρωτοκόλλων δρομολόγησης όπως το IS-IS και το OSFP (Open Shortest Path First).

Ο αλγόριθμος των Floyd-Warshall (αλλιώς WFI αλγόριθμος ή Roy-Floyd) είναι μια αναλυτική μέθοδος εύρεσης των βέλτιστων μονοπατιών πάνω σε γράφους που φέρουν κόστη επί των ακμών τους (θετικά και/ή αρνητικά). Μια εκτέλεση του αλγορίθμου θα υπολογίσει τα συνολικά μήκη (total lengths) των βέλτιστων μονοπατιών για όλα τα δυνατά ζεύγη διακεκριμένων κόμβων του γράφου, χωρίς ωστόσο να μας επιστρέφει την ακολουθία των κόμβων που απαρτίζουν τα μονοπάτια αυτά. Ο αλγόριθμος, που βασίζεται στον Δυναμικό Προγραμματισμό, δημοσιεύτηκε με την τρέχουσα μορφή του από τον Robert Floyd το 1962 (Floyd, 1962). Ωστόσο, είναι παρόμοιος με αλγορίθμους που είχαν παρουσιαστεί από τον Bernard Roy το 1959 (Roy, 1959) και από τον Stephen Warshall το 1962 (Warshall, 1962).

Ο αλγόριθμος A* είναι ένας αλγόριθμος που χρησιμοποιείται ευρέως για τον εντοπισμό διαδρομών και για την διάσχιση γράφων. Περιγράφηκε για πρώτη φορά από τους Peter Hart, Nils Nilsson και Bertram Raphael το 1968 (Hart, Nilsson, & Raphael, 1968) και αποτελεί μια παραλλαγή του Dijkstra. Ο A* πετυχαίνει καλύτερες επιδόσεις σε ό,τι αφορά το χρόνο εκτέλεσης εφαρμόζοντας ευρετικές τεχνικές για την αποφυγή διάσχισης κόμβων που δείχνουν ότι δεν μπορούν να προσφέρουν καλές λύσεις.

2.7 Συμπεράσματα

Τις τελευταίες τρεις δεκαετίες η προσέγγιση του γενικότερου προβλήματος της δρομολόγησης έχει αποτελέσει ένα ευρύ και ανεξάντλητο πεδίο συνεχούς έρευνας από τη διεθνή επιστημονική κοινότητα. Η άμεση επίπτωση των ωφελειών της δρομολόγησης στην καθημερινότητα οδήγησε την επιστημονική κοινότητα στη δημιουργία προηγμένων τεχνικών και μαθηματικών προσεγγίσεων, οι οποίες είναι ικανές να μοντελοποιήσουν τα πολυσύνθετα πραγματικά προβλήματα.

Αξιοποιώντας τα δίκτυα σαν εργαλείο αποτύπωσης τοπολογιών, οι επιστημονικές προσπάθειες επικεντρώθηκαν στην εύρεση των βέλτιστων μονοπατιών μέσα στις ακμές και τους κόμβους του εξεταζόμενου δικτύου. Αυτές οι προσπάθειες εφαρμόστηκαν πρακτικά σε τοπολογίες δικτύων υπολογιστών και σε δίκτυα διανομής αγαθών, στα οποία το μέσο μεταφοράς ήταν ένα και το συντομότερο μονοπάτι αποτελούσε πάντα την βέλτιστη λύση. Στην καθημερινότητα όμως έχουμε να αντιμετωπίσουμε το πρόβλημα της μετακίνησης με πολλαπλά μεταφορικά μέσα. Σε αυτό το πρόβλημα, ο κάθε κόμβος του δικτύου μπορεί να είναι προσβάσιμος με διαφορετικό μέσο, ενώ παράλληλα οι δυνατές επιλογές του κάθε επιβάτη είναι πρακτικά άπειρες.

Στα πλαίσια αυτά, η παρούσα διπλωματική εργασία εστιάζει την προσοχή της στο καθημερινό πρόβλημα των μεταφορών και εξετάζει έναν αλγόριθμο που υπόσχεται εφαρμόσιμες λύσεις στο παραπάνω πρόβλημα. Στο επόμενο κεφάλαιο θα αναλύσουμε τον αλγόριθμο time-dependent intermodal least-time path (TDILTP) (Ziliaskopoulos & Wardell, 2000) παραθέτοντας παραδείγματα εκτέλεσης του αλγόριθμου και παρουσιάζοντας το πληροφοριακό σύστημα που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας.

3

Ο αλγόριθμος δυναμικής δρομολόγησης TDILTP

Κεφάλαιο 3: Ο αλγόριθμος δυναμικής δρομολόγησης TDILTP

3.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο εξετάσθηκαν τα μαθηματικά εργαλεία των γράφων και των δικτύων, με βάση τα οποία η επιστημονική κοινότητα έχει προτείνει πληθώρα αλγορίθμων εύρεσης συντομότερων διαδρομών. Έχοντας διαμορφώσει εικόνα για τις ερευνητικές αυτές προσπάθειες και την φιλοσοφία που τις χαρακτηρίζει, σε αυτό το κεφάλαιο θα εστιάσουμε σε έναν σύγχρονο δυναμικό αλγόριθμο δρομολόγησης, ο οποίος φέρει την ονομασία time-dependent intermodal least-time path ή εν συντομία TDILTP. Ο συγκεκριμένος αλγόριθμος αναζητάει βέλτιστες διαδρομές σε ένα δίκτυο στο οποίο κινούνται πολλαπλά μεταφορικά μέσα. Όπως γίνεται αντιληπτό, η εφαρμογή του συγκεκριμένου αλγόριθμου βρίσκει άμεση εφαρμογή στην καθημερινότητα του απλού πολίτη, ο οποίος καλείται να ταξιδέψει στο αστικό δίκτυο επιλέγοντας τον συνδυασμό των μεταφορικών μέσων που θα τον οδηγήσουν στον προορισμό του στον ελάχιστο χρόνο.

Ειδικότερα, το κεφάλαιο ξεκινάει με την μαθηματική μοντελοποίηση του αλγόριθμου TDILTP όπου επεξηγούνται οι μαθηματικοί τύποι που αξιοποιούνται για την εύρεση της βέλτιστης λύσης. Στην συνέχεια, παρουσιάζεται η επίλυση ενός απλού προβλήματος, με την βοήθεια του οποίου ο αναγνώστης θα έχει την ευκαιρία να κατανοήσει σε βάθος τον τρόπο λειτουργίας του αλγόριθμου. Όπως θα αποδειχθεί από το παράδειγμα, η επίλυση πολύπλοκων προβλημάτων χωρίς την χρήση υπολογιστικών συστημάτων είναι πρακτικά αδύνατη. Βάση αυτού, το κεφάλαιο ολοκληρώνεται με την παρουσίαση της προγραμματιστικής υλοποίησης του αλγόριθμου σε γλώσσα Fortran, η οποία μπορεί να εντοπίσει βέλτιστες λύσεις με μεγάλα δίκτυα, των οποίων η πολυπλοκότητα είναι αυξημένη.

3.2 Περιγραφή του αλγόριθμου

Ο αλγόριθμος που χρησιμοποιήθηκε στην παρούσα διπλωματική ονομάζεται time-dependent intermodal least-time path (TDILTP) (Ziliaskopoulos & Wardell, 2000).

Ο συγκεκριμένος αλγόριθμος αποτυπώνει το δίκτυο με τον κατευθυνόμενο γράφο $G = (V, A, T, M)$ όπου V είναι το σύνολο των διαθέσιμων κόμβων, A είναι το σύνολο των ακμών του δικτύου, $T = \{t_0, t_0 + \Delta t, t_0 + 2\Delta t, t_0 + (|T| - 1)\Delta t\}$ είναι το σύνολο των διακριτών χρόνων που ορίζουν την περίοδο αναφοράς (όπως η ώρα αιχμής) όπου το Δt είναι η ελάχιστη χρονική διάρκεια που μπορεί να οριστεί στα δεδομένα μας (πχ 1 λεπτό αν όλα τα δεδομένα μας είναι με ακρίβεια λεπτού) και M είναι το σύνολο των διαθέσιμων μέσων μεταφοράς. Σε αυτόν το γράφο ορίζουμε το σύνολο $\Im = \{\tau_{ij}^x(t)\}$ που αποτελείται από το σύνολο των μη αρνητικών χρόνων μεταφοράς $\tau_{ij}^x(t)$ που ορίζεται για κάθε ακμή $(i, j) \in A$ και για κάθε $x \in M$ και $t \in T$. Στην πράξη το $\tau_{ij}^x(t)$ ορίζει τον χρόνο που απαιτείται για την μεταφορά από τον κόμβο i στον κόμβο j με το μεταφορικό μέσο x όταν η ώρα αναχώρησης από τον κόμβο i είναι t. Συνάμα, ο αλγόριθμος ορίζει το διάνυσμα $\Xi = \{\xi_{ikj}^{xy}(t)\}$ το οποίο αποτελείται από τα στοιχεία $\xi_{ikj}^{xy}(t)$, τα οποία αναπαριστούν τον χρόνο που απαιτείται για την μετεπιβίβαση από το μέσο x στο μέσο y όταν ταξιδεύουμε από τον κόμβο k στον κόμβο j μέσω του κόμβου i την χρονική στιγμή t. Το $\xi_{ikj}^{xy}(t)$ λαμβάνει την τιμή άπειρο όταν δεν είναι δυνατή η μετεπιβίβαση από το μέσο x στο μέσο y στον κόμβο i. Όταν τα μέσα x και y συμπίπτουν ($x \equiv y$), το $\xi_{ikj}^{xy}(t)$ αναπαριστά τον χρόνο στροφής στον κόμβο i από την ακμή (k, i) στην ακμή (i, j) με το μεταφορικό μέσο x. Εάν αυτή η στροφή απαγορεύεται για κάποιες χρονικές στιγμές, τότε η τιμή αυτή μπορεί να είναι το άπειρο.

Σύμβολο	***Περιγραφή συμβολισμού***
V:	Οι διαθέσιμοι κόμβοι
A:	Οι ακμές του δικτύου
T:	Διακριτοί χρόνοι αναχώρησης
M:	Τα μεταφορικά μέσα
$\Im$:	Οι χρόνοι μεταφοράς σε κάθε ακμή
Ξ:	Οι χρόνοι μετεπιβίβασης από μέσο σε μέσο
Δt:	Η διαφορά χρόνου
$\tau_{ij}^x(t)$:	Ο χρόνος μεταφοράς από την ακμή (i, j) με το μέσο x αναχωρώντας από το i την χρονική στιγμή t
$\xi_{ikj}^{xy}(t)$:	Η καθυστέρηση μετεπιβίβασης στον κόμβο i, όταν φτάνουμε στον κόμβο i από την ακμή (k, i) με το μέσο x την χρονική στιγμή t και πάμε στην ακμή (i, j) με το μέσο y
Λ_i:	Το διάνυσμα ετικετών του κόμβου i
$\lambda_{ik}^x(t)$:	Ο χρόνος διαδρομής από τον κόμβο i, όταν φθάνουμε στον i από την ακμή (k, i) με το μέσο x την χρονική στιγμή t, προς τον προορισμό N με τελικό μέσο μεταφοράς το x_f
$\pi_{ik}^x(t)$:	Το μονοπάτι από τον κόμβο i, όταν φθάνουμε στον i από την ακμή (k, i) με το μέσο x την χρονική στιγμή t, προς τον προορισμό N με τελικό μέσο μεταφοράς το x_f
$\Gamma(I)$:	Οι δυνατοί κόμβοι προορισμού από τον κόμβο i
$\Gamma^{-1}(I)$:	Οι κόμβοι που οδηγούν στον κόμβο i
N:	Ο κόμβος προορισμού για τον οποίο εντοπίζονται τα συντομότερα μονοπάτια από όλους τους υπόλοιπους κόμβους του δικτύου
x_f:	Το μέσο μεταφοράς με το οποίο φτάνει στο προορισμό N ο επιβάτης

Πίνακας 3.1 Μαθηματικοί συμβολισμοί και περιγραφή τους

Για να επιτρέψει ο αλγόριθμος να εισερχόμαστε στο σύστημα σε έναν κόμβο (πέρα από την κλασική μεταφορά στον κόμβο μέσω μιας ακμής του A), ορίζεται ο κόμβος εισαγωγής στο σύστημα i' για κάθε κόμβο του δικτύου. Οι θεωρητικές ακμές (i', i) έχουν μηδενικό χρόνο μεταφοράς για όλους τους κόμβους, αλλά έχουν μη αρνητικές τιμές μετεπιβίβασης $\xi_{ii'j}^{xy}(t)$ για να μπορέσει ο αλγόριθμος να υπολογίσει πιθανές καθυστερήσεις κατά την εισαγωγή στο κόμβο. Αυτές οι καθυστερήσεις μπορούν να ταυτίζονται με τον χρόνο εισαγωγής στο δίκτυο από το παρκινγκ όπου έχει παρκάρει ο επιβάτης το όχημά του.

Για κάθε κόμβο i ορίζεται το διάνυσμα Λ_i των ετικετών του κόμβου ως εξής:

$$\Lambda_i = \begin{bmatrix} \lambda_{ik_1}^{x_1}(t_0) & \lambda_{ik_2}^{x_1}(t_0) & \cdots & \lambda_{ik_1}^{x_m}(t_0) & \cdots & \lambda_{ik_n}^{x_m}(t_0) \\ \lambda_{ik_1}^{x_1}(t_0+\Delta t) & \lambda_{ik_2}^{x_1}(t_0+\Delta t) & \cdots & \lambda_{ik_1}^{x_m}(t_0+\Delta t) & \cdots & \lambda_{ik_n}^{x_m}(t_0+\Delta t) \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ \lambda_{ik_1}^{x_1}(t_0+(|T|-1)\Delta t) & \lambda_{ik_2}^{x_1}(t_0+(|T|-1)\Delta t) & \cdots & \lambda_{ik_1}^{x_m}(t_0+(|T|-1)\Delta t) & \cdots & \lambda_{ik_n}^{x_m}(t_0+(|T|-1)\Delta t) \end{bmatrix}$$

Όπου x_m είναι το τελευταίο μέσο μεταφοράς του M και k_n είναι ο τελευταίος κόμβος του συνόλου $\{\Gamma^{-1}(i), i'\}$. Για να μπορέσουμε να αποθηκεύσουμε σωστά όλη την πληροφορία, θα πρέπει να αποθηκεύουμε ξεχωριστή ετικέτα για κάθε κόμβο, κόμβο προέλευσης και χρόνο αναχώρησης αφού το βέλτιστο μονοπάτι μπορεί να περιέχει διαφορετικούς συνδυασμούς για τον ίδιο κόμβο.

Ουσιαστικά, η ετικέτα $\lambda_{ik}^{x}(t)$ υποδηλώνει τον συνολικό χρόνο του βέλτιστου μονοπατιού από τον κόμβο i, προερχόμενοι από τον κόμβο k μέσω της ακμής (k, i). Αντίστοιχα, η ετικέτα $\lambda_{ii'}^{x}(t)$ προσδιορίζει τον συνολικό χρόνο του βέλτιστου μονοπατιού όταν ο κόμβος i είναι ο κόμβος εκκίνησης.

Παράλληλα με τους κόμβους εισαγωγής i', ορίζεται και ο κόμβος εξόδου από το δίκτυο N'' ο οποίος προστίθεται στον κόμβο προορισμού N για να επιτραπεί η έξοδος από το δίκτυο. Η ακμή (N, N'') έχει μηδενικό χρόνο μεταφοράς για κάθε χρονική στιγμή και μη αρνητικό χρόνο μετεπιβίβασης $\xi_{NkN''}^{xx_f}(t)$ ο οποίος υποδηλώνει τις πιθανές καθυστερήσεις μετεπιβίβασης από το μέσο x στο μέσο εξόδου x_f καθώς φτάνει ο επιβάτης στον κόμβο N με το μέσο x από τον κόμβο k. Η καθυστέρηση αυτή μπορεί να αντιστοιχεί στον χρόνο εύρεσης πάρκινγκ κοντά στον τερματικό κόμβο όταν το x_f είναι τα πόδια και το x είναι το αυτοκίνητο.

Το μονοπάτι από έναν κόμβο i με το μεταφορικό μέσο x την χρονική στιγμή t, με προορισμό τον κόμβο προορισμού N με μέσο μεταφοράς εξόδου το x_f αναπαρίσταται από μια αλληλουχία κόμβων και χρονικών στιγμών όπως ακολούθως:

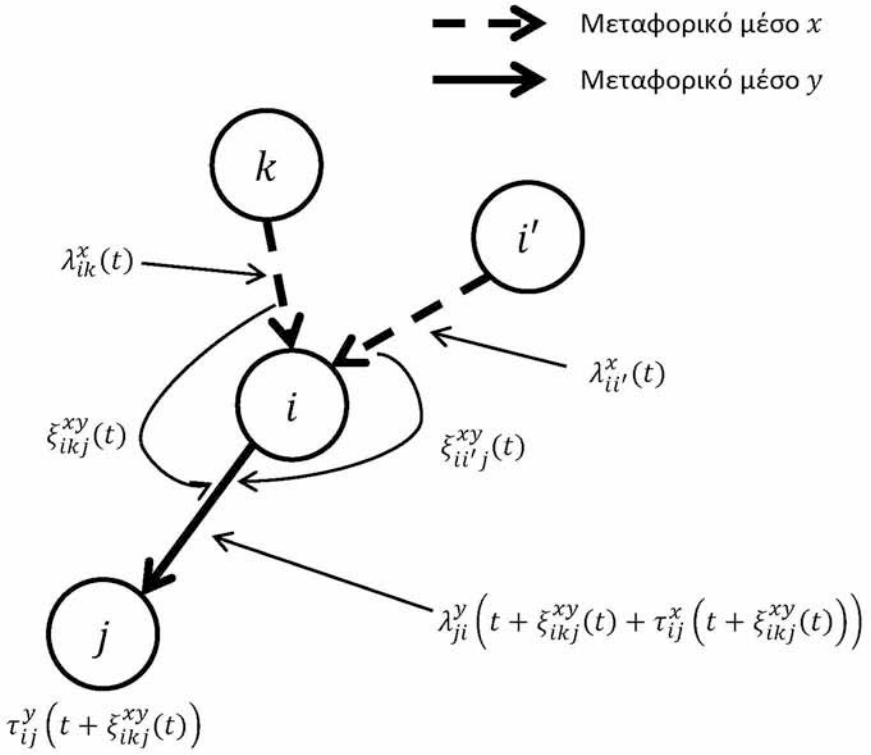
$$\pi_{ii'}^{x}(t) = \{(n_1 = i', x_1 = x, t_1^0 = t), (i, x_2, t_1^1), (n_2, x_2, t_2^0), \dots, (n_k = N, x_k = x_f, t_k^1), (N'', x_f, t_{k+1}^1)\}$$

Όπως μπορεί κανείς να παρατηρήσει, στο πρώτο σημείο ξεκινάμε από τον κόμβο i με το μέσο x την χρονική στιγμή t και στην συνέχεια οδηγούμαστε πάλι στον κόμβο i με νέο μεταφορικό μέσο, το οποίο σημαίνει ότι ενώ πρακτικά ξεκινάμε από τον εικονικό κόμβο

εισαγωγής i' με το μέσο x, μπορούμε να αλλάξουμε μέσο και από εκεί και ύστερα να πλοηγηθούμε στο δίκτυο.

Μια παραδοχή που γίνεται με την ανωτέρω σημειογραφία είναι ότι οι αλλαγές των μέσων γίνονται μόνο στους κόμβους του δικτύου. Η συγκεκριμένη παραδοχή μπορεί να μην είναι πάντα αληθείς σε πραγματικά δίκτυα. Στην περίπτωση αυτή, εισάγουμε εικονικούς κόμβους στο δίκτυο, στις ακμές όπου γίνεται η αλλαγή των μεταφορικών μέσων, χωρίς να επηρεάζεται η γενικότητα του προβλήματος.

Όλοι οι παραπάνω συμβολισμοί αναπαρίστανται στο ακόλουθο σχήμα:



Εικόνα 3.1 Απεικόνιση χρόνων μεταφοράς, χρόνων μετεπιβίβασης και ετικετών

Με βάση τους παραπάνω ορισμούς, μπορούμε να ορίσουμε το πρόβλημα βελτιστοποίησης της συγκεκριμένης διπλωματικής εργασίας, ως εξής:

Σε ένα δίκτυο G, υπολόγισε τα χρονικά συντομότερα μονοπάτια $\pi^x_{ii'}(t)$ από κάθε κόμβο εισόδου, μέσο μεταφοράς και χρόνο αναχώρησης προς το κόμβο προορισμού N, λαμβάνοντας υπόψη όλα τα διαθέσιμα μέσα μεταφοράς και τους χρόνους μετεπιβίβασης από μέσο σε μέσο.

Για να είναι μια ετικέτα βέλτιστη και συνεπώς να ανήκει στο συντομότερο μονοπάτι, θα πρέπει σύμφωνα με την αρχή βελτιστοποίησης του Bellman (Bellman R. , 1958) να ισχύει ότι:

$$\lambda_{ik}^{x}(t) = \min\left\{\xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)\ \forall j \in \Gamma(i), \forall\, y \in M\right\}, \forall k \in \{\Gamma^{-1}(i), i'\}\ \forall\, x \in M, \forall t \in T\ \kappa\alpha\iota\ \forall i \in V\backslash N$$

Με συνθήκη τερματισμού την:

$$\lambda_{Nk}^{x}(t) = \xi_{NkN''}^{xx_f}(t) \quad \forall\, x \in M, \forall t \in T\ \kappa\alpha\iota\ \forall k \in \{\Gamma^{-1}(N), N'\}$$

Στους παραπάνω τύπους, $\Gamma(i)$ είναι το σύνολο των κόμβων προορισμών από τον κόμβο i ενώ $\Gamma^{-1}(i)$ είναι το σύνολο των κόμβων από τους οποίους μπορεί ο επιβάτης να μετακινηθεί στον κόμβο i. Επιπλέον, με τον συμβολισμό $V\backslash N$ αναγράφεται το σύνολο V (οι διαθέσιμοι κόμβοι) εξαιρουμένου του κόμβου N.

Για να μπορέσουμε να συνυπολογίσουμε τους χρόνους αναμονής μέχρι την άφιξη ενός μέσου σε έναν κόμβο (όπως τα τρένα ή τα λεωφορεία) μπορούμε να τους ενσωματώσουμε στον χρόνο μετεπιβίβασης. Στην περίπτωση όπου ένα μέσο μεταφοράς y εξυπηρετεί μια ακμή (i, j) κάθε r χρονικές μονάδες, ο χρόνος μετεπιβίβασης από το μέσο μεταφοράς x, ενώ προέρχεται ο επιβάτης από την ακμή (k, i) στο μέσο μεταφοράς y την χρονική στιγμή $t \in T$, δίνεται από τον ακόλουθο τύπο:

$$\xi_{ikj}^{xy}(t) = \xi_{ikj}^{xy}\left(t + r\Delta t - \underset{r}{\mathrm{MOD}}(t)\right) + r\Delta t - \underset{r}{\mathrm{MOD}}(t)$$

Όπου $\mathrm{MOD}_r(t)$ είναι το υπόλοιπο της διαίρεσης του χρόνου t προς r ή αλλιώς ο χρόνος αναμονής του μεταφορικού μέσου y από την στιγμή που θα φτάσει ο επιβάτης στον κόμβο i με το μεταφορικό μέσο x (Dial, Rutherford, & Quillian, 1979) (Florian, 1977) (Spiess & Florian, 1989).

Ο εξεταζόμενος αλγόριθμος TDILTP στηρίζεται στην αναζήτηση με ετικέτες και ξεκινάει την αναζήτηση του από τον κόμβο προορισμού N, τον οποίο τοποθετεί στην λίστα κόμβων προς έλεγχο.

Η ***λίστα κόμβων προς έλεγχο (Scan Eligible list, SE list)*** αποτελεί μια παραλλαγή της αντίστοιχης λίστας που χρησιμοποιείται στους αλγόριθμους διόρθωσης στατικής ετικέτας (Aho, Hopcroft, & Ullman, 1983). Η υλοποίηση αυτής της λίστας στηρίζεται στην δομής της ουράς με δύο σημεία εισόδου/εξόδου (double ended queue) και οι ενέργειες εισαγωγής και εξαγωγής υποβοηθούνται από ένα μονοδιάστατο πίνακα, ο οποίος αποκαλείται Deque (Pallotino, 1984). Ο πίνακας αυτός έχει τόσα στοιχεία όσα τα στοιχεία του συνόλου V και η τιμή του κάθε κόμβου i δίνεται από τον ακόλουθο τύπο:

$$Deque(i) = \begin{cases} 0 & \text{αν ο κόμβος i δεν έχει μπει ποτέ στην λίστα} \\ -1 & \text{αν ο κόμβος i έχει ξαναμπεί στην λίστα στο παρελθόν} \\ j & \text{εάν ο κόμβος i είναι στην λίστα και ο κόμβος j είναι ο επόμενος του i} \\ \infty & \text{εάν ο κόμβος i είναι ο τελευταίος στην λίστα} \end{cases}$$

Για να γίνονται οι έλεγχοι πιο γρήγορα, ο αλγόριθμος διατηρεί δύο δείκτες (pointers), ένα στο πρώτο στοιχείο ($FirstNode$) της λίστας και ένα στο τελευταίο ($LastNode$). Οι κόμβοι της λίστας βγαίνουν μόνο από την μπροστινή μεριά, ενώ η εισαγωγή ενός κόμβου i στην λίστα γίνεται είτε από μπροστά εάν η αντίστοιχη τιμή στον βοηθητικό πίνακα $Deque(i) =$

-1 ή από την πίσω πλευρά εάν $Deque(i) = 0$. Σε οποιαδήποτε άλλη τιμή του $Deque(i)$ ο κόμβος δεν εισάγεται στην λίστα SE.

Στην αρχή, η λίστα κόμβων προς έλεγχο περιλαμβάνει μόνο το κόμβο προορισμού N. Οι ετικέτες λ_{Ni}^{x} του κόμβου N παίρνουν την τιμή $\xi_{NkN''}^{xx_f}$, το οποίο ισοδυναμεί με τον χρόνο μετεπιβίβασης από το μέσο x στο τελικό μέσο x_f στον εικονικό τερματικό κόμβο N''. Όλες οι άλλες ετικέτες λαμβάνουν την τιμή άπειρο (∞).

Στην πρώτη αναδρομή του αλγόριθμου, αφαιρείται ο κόμβος N από την λίστα κόμβων προς έλεγχο και ενημερώνονται οι ετικέτες όλων των κόμβων που οδηγούν στον κόμβο προορισμού N με βάση την ακόλουθη τιμή:

$$\lambda_{ik}^{x}(t) = \xi_{ikN}^{xy}(t) + \tau_{iN}^{y}\left(t + \xi_{ikN}^{xy}(t)\right) + \lambda_{Ni}^{y}\left(t + \xi_{ikN}^{xy}(t) + \tau_{iN}^{y}\left(t + \xi_{ikN}^{xy}(t)\right)\right)$$

$$\forall i \in \Gamma^{-1}(N), \forall t \in T, \forall k \in \{\Gamma^{-1}(i), i'\} \kappa\alpha\iota\ \forall x, y \in M$$

Όλοι αυτοί οι κόμβοι εισάγονται στην ***λίστα κόμβων προς έλεγχο (λίστα SE).***

Στην συνέχεια, αφαιρείται ο πρώτος κόμβος j από την λίστα SE για τον οποίο ελέγχουμε όλους τους κόμβους i που οδηγούν σε αυτόν (δηλαδή $i \in \Gamma^{-1}(j)$) και ενημερώνουμε τις ετικέτες τους σύμφωνα με τον ακόλουθο τύπο:

$$\lambda_{ik}^{x}(t) = \min_{\forall y \in M}\left\{\lambda_{ik}^{x}(t), \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)\right\}$$

$$\forall i \in \Gamma^{-1}(j), \forall t \in T, \forall k \in \{\Gamma^{-1}(i), i'\} \kappa\alpha\iota\ \forall x \in M$$

Εάν τουλάχιστον μια ετικέτα του διανύσματος Λ_i του κόμβου i έχει τροποποιηθεί, τότε ο κόμβος εισάγεται στην λίστα SE. Η διαδικασία, που περιγράφηκε στην τελευταία παράγραφο, επαναλαμβάνεται μέχρι η λίστα κόμβων προς έλεγχο να αδειάσει, οπότε και τερματίζει ο αλγόριθμος.

3.3 Εφαρμογή του αλγόριθμου σε μικρής κλίμακας πρόβλημα

Τα βήματα του αλγόριθμου TDILTP μπορούν να συνοψιστούν στο ακόλουθο σχήμα:

Βήμα 1°: Αρχικοποίηση των ετικετών Λ_i
  Βήμα 1.1: $\Lambda_i = \infty \quad \forall i \in V \backslash N$
  Βήμα 1.2: $\lambda_{Nk}^{x}(t) = \xi_{NkN''}^{xx_f}(t) \quad \forall t \in T, \forall x \neq x_f \in M, \forall k \in \{\Gamma^{-1}(N), N'\}$
  Βήμα 1.3: $\lambda_{Nk}^{x_f}(t) = 0 \quad \forall t \in T, \forall k \in \{\Gamma^{-1}(N), N'\}$
Βήμα 2°: Εισαγωγή του κόμβου προορισμού N στην λίστα κόμβων προς έλεγχο (λίστα SE).
Βήμα 3°: Εάν η λίστα SE είναι άδεια, πήγαινε στο βήμα 6
Βήμα 4°: Αφαίρεσε τον πρώτον κόμβο j από την λίστα SE και κάνε τα ακόλουθα:
  Αναδρομή 4.1: Για κάθε κόμβο $i \in \Gamma^{-1}(j)$
  Αναδρομή 4.2: Για κάθε μεταφορικό μέσο $x \in M$
  Αναδρομή 4.3: Για κάθε μεταφορικό μέσο $y \in M$
  Αναδρομή 4.4: Για κάθε κόμβο $k \in \{\Gamma^{-1}(i), i'\}$
  Αναδρομή 4.5: Για κάθε χρονική στιγμή $t \in T$
    Βήμα 4.1: Εάν

$$\lambda_{ik}^{x}(t) > \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)$$

συνέχισε στο βήμα 4.2, αλλιώς πήγαινε στο βήμα 4.3,

Βήμα 4.2: Βάλε τον κόμβο i στην λίστα SE και θέσε την ετικέτα του

$$\lambda_{ik}^{x}(t) = \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)$$

Βήμα 4.3: Συνέχισε την αναδρομή.

Βήμα 5°: Πήγαινε στο βήμα 3.

Βήμα 6°: Τερματισμός αλγόριθμου.

Εικόνα 3.2 Βήματα αλγόριθμου TDILTP

Στα βήματα που περιγράφονται στην παραπάνω εικόνα, μπορεί κανείς να παρατηρήσει ότι ο τύπος που περιγράφηκε στην μαθηματική μοντελοποίηση του αλγόριθμου της προηγούμενης ενότητας:

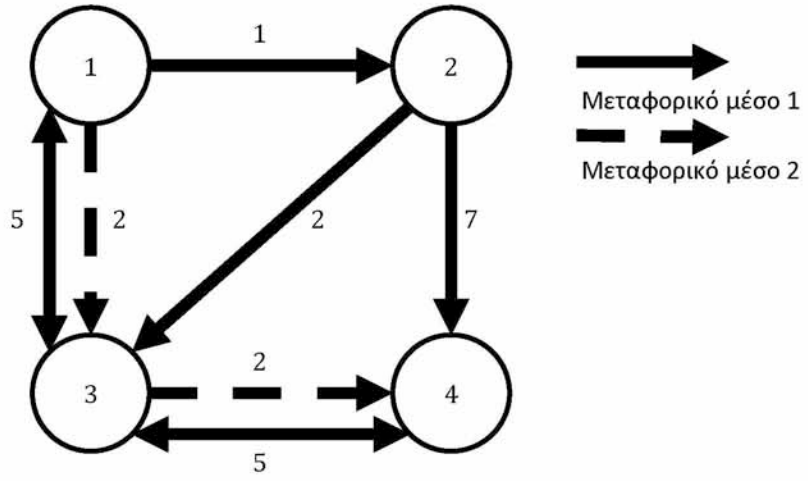
$$\lambda_{ik}^{x}(t) = \min_{\forall y \in M}\left\{\lambda_{ik}^{x}(t), \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)\right\}$$

υλοποιείται ελαφρώς διαφορετικά. Ειδικότερα, αντί του εντοπισμού του ολικού ελάχιστου για κάθε μεταφορικό μέσο y, ο προσδιορισμός της τιμής του $\lambda_{ik}^{x}(t)$ γίνεται σταδιακά για όλα τα μεταφορικά μέσα y καθώς γίνεται η αναδρομή 4.3. Συνεπώς στο βήμα 4.1 γίνεται έλεγχος εάν το νέο μεταφορικό μέσο y επιφέρει βελτίωση στο ήδη υπολογισμένο $\lambda_{ik}^{x}(t)$ και εάν όντως επιφέρει βελτίωση αντικαθίσταται η ετικέτα του κόμβου στο βήμα 4.2, αλλιώς συνεχίζει αμέσως η αναδρομή, από το βήμα 4.3. Τα βήματα 4.1 και 4.2 μπορούν να γραφούν και διαφορετικά αν στο βήμα 4.1 εφαρμόσουμε τον τύπο:

$$\lambda_{ik}^{x}(t) = \min\left\{\lambda_{ik}^{x}(t), \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)\right\}$$

Στη συνέχεια, στο βήμα 4.2 γίνεται ο έλεγχος εάν η τιμή του $\lambda_{ik}^{x}(t)$ μεταβλήθηκε για να προσθέσουμε τον κόμβο i στην λίστα SE.

Το πρόβλημα που θα επιλυθεί, αποτελείται από τέσσερις (4) κόμβους και δύο (2) μεταφορικά μέσα. Η αναπαράσταση του δικτύου απεικονίζεται στο ακόλουθο σχήμα.



Εικόνα 3.3 Αναπαράσταση δικτύου προβλήματος

Το πρόβλημα θα επιλυθεί για δέκα (10) χρονικές μονάδες ($T = [1,10]$) με κόμβο προορισμού τον κόμβο 4 και αφετηρία τον κόμβο 1. Τα μέσα μεταφοράς εισόδου και εξόδου από το σύστημα ταυτίζονται και είναι το 1°.

Οι χρόνοι μεταφοράς για το 1° μεταφορικό μέσο στις ακμές του προβλήματος παρουσιάζονται στον ακόλουθο πίνακα.

κόμβος j / κόμβος i	1	2	3	4
1	–	1	5	∞
2	∞	–	2	7
3	5	∞	–	5
4	∞	∞	5	–

Πίνακας 3.2 Χρόνοι μεταφοράς για το μέσο 1 ($\tau_{ij}^1(t), \forall t \in T$)

Από τον παραπάνω πίνακα φαίνεται ότι ο χρόνος μεταφοράς για την ακμή (1,2) είναι σταθερός για κάθε χρονική στιγμή και ισούται με μία χρονική μονάδα. Αυτό συμβολίζεται ως $\tau_{12}^1(t) = 1, \forall t \in T$. Στην περίπτωση όπου η ακμή δεν υποστηρίζει το μεταφορικό μέσο 1 όπως στην ακμή (2,1) τότε ο χρόνος μεταφοράς ισούται με το άπειρο, δηλαδή $\tau_{21}^1(t) = \infty, \forall t \in T$.

Το 2° μεταφορικό μέσο δεν είναι διαθέσιμο όλες τις χρονικές στιγμές στις ακμές που το υποστηρίζουν. Για να γίνει αντιληπτή η λειτουργία του μέσου θα μπορούσε κανείς να το παρομοιάσει με ένα λεωφορείο ή ένα τρένο. Πιο συγκεκριμένα, για την μετακίνηση στην ακμή (1,3) απαιτούνται 2 χρονικές μονάδες, ενώ το μεταφορικό μέσο ξεκινάει από τον κόμβο 1 την χρονική στιγμή 2. Στην περίπτωση που κάποιος φτάσει πριν την αναχώρηση του μέσου σε κάποιο κόμβο, στο χρόνο μεταφοράς προστίθεται ο χρόνος αναμονής. Στο παρακάτω πίνακα φαίνονται οι χρόνοι μεταφοράς για τις υποστηριζόμενες ακμές. Σημειώνονται με μπλε οι συνολικοί χρόνοι μεταφοράς και αναμονής όταν υπάρχει μελλοντική αναχώρηση του μέσου. Για τις ακμές που δεν αναφέρονται, ο χρόνος μεταφοράς ισούται με άπειρο.

t	$\tau_{13}^2(t)$	$\tau_{31}^2(t)$	$\tau_{34}^2(t)$	$\tau_{43}^2(t)$
1	3	∞	6	∞
2	2	∞	5	∞
3	∞	∞	4	∞
4	∞	∞	3	∞
5	∞	∞	2	∞
6	∞	∞	∞	∞
7	∞	∞	∞	∞
8	∞	∞	∞	∞
9	∞	∞	∞	∞
10	∞	∞	∞	∞

Πίνακας 3.3 Χρόνοι μεταφοράς για το μέσο 2 ($\tau_{ij}^2(t), \forall t \in T$)

Στην συνέχεια σημειώνονται όλες οι δυνατές μετεπιβιβάσεις του παραδείγματος που εξετάζουμε, καθώς και οι χρονικές καθυστερήσεις που επιφέρουν. Οποιαδήποτε

μετεπιβίβαση δεν εμφανίζεται στον ακόλουθο πίνακα θεωρείται αδύνατη και η τιμή της ισούται με το άπειρο. Εξαίρεση σε αυτό το κανόνα αποτελούν οι χρόνοι εισόδου στο δίκτυο $k = i' \equiv i$ που θεωρούνται ίσοι με το 0 για το ίδιο μεταφορικό μέσο ($x = y$).

k	x	i	y	j	$\xi_{ikj}^{xy}(t), \forall t \in T$
1	1	2	1	3	0
				4	0
		3	1	1	0
				4	0
			2	4	1
	2	3	1	1	1
				4	1
			2	4	0
2	1	3	1	1	0
				4	0
			2	4	1
		4	1	3	0
				4″	0
3	1	1	1	2	0
				3	0
			2	3	1
		4	1	3	0
				4″	0
	2	4	1	3	1
				4″	1
4	1	3	1	1	0
				4	0
			2	4	1

Πίνακας 3.4 Χρόνοι μετεπιβίβασης $\xi_{ikj}^{xy}(t), \forall t \in T$

Ο χρόνος μετεπιβίβασης από το μεταφορικό μέσο x στο μεταφορικό μέσο y, στο κόμβο i όταν ο κόμβος προέλευσης είναι ο k και ο κόμβος προορισμού είναι ο j, μπορεί να βρεθεί εντοπίζοντας τη σωστή γραμμή στον παραπάνω πίνακα, αντιστοιχίζοντας τις τιμές των μεταβλητών: k, x, i, y, j. Παραδείγματος χάρη, η μετεπιβίβαση στο μέσο $y = 2$ στον κόμβο $i = 3$ προερχόμενος από το κόμβο $k = 2$ με το μέσο $x = 1$ και με προορισμό το κόμβο $j = 4$ απαιτεί $\xi_{324}^{12}(t) = 1$ χρονική μονάδα.

Στη μεταβλητή k οι κόμβοι που συμβολίζονται με ένα τόνο (′) υποδηλώνουν κόμβο εισόδου, ενώ στην μεταβλητή j οι κόμβοι με δύο τόνους (″) υποδηλώνουν κόμβο εξόδου.

Στον συγκεκριμένο παράδειγμα, μπορούμε να παρατηρήσουμε ότι όταν κινούμαστε από ένα κόμβο k σε ένα κόμβο i με προορισμό τον κόμβο j χωρίς να αλλάξουμε μεταφορικό μέσο τότε το χρονικό κόστος της μετεπιβίβασης είναι μηδενικό. Αυτή η παρατήρηση μπορεί να γενικευτεί για όλες τις δυνατές μετεπιβιβάσεις στις οποίες δεν έχουμε αλλαγή μεταφορικού μέσου, ως εξής: $\xi_{ikj}^{xx}(t) = 0, \forall i, k, j \in V, \forall x \in M \; και \; \forall t \in T$.

Αντίστοιχα, από τον ορισμό του συγκεκριμένου παραδείγματος ο χρόνος αλλαγής από το μεταφορικό μέσο 1 στο μεταφορικό μέσο 2 (σε όσους κόμβους είναι εφικτή) και αντίστροφα, είναι πάντα ίσος με μία χρονική μονάδα. Αυτή η παρατήρηση μπορεί να γραφεί ως: $\xi_{ikj}^{xy}(t) = 1, \forall i,k,j \in V, \forall x,y \in M, x \neq y \; και \; \forall t \in T$.

Ξεκινώντας την εκτέλεση του αλγορίθμου και όπως ορίζει το βήμα 1.1, απειρίζονται όλες οι ετικέτες $\Lambda_i = \infty$ για όλους του κόμβους εκτός του 4ου, ο οποίος είναι ο κόμβος προορισμού.

Στην συνέχεια, στο βήμα 1.2 αρχικοποιούνται οι ετικέτες του 4ου κόμβου για κάθε x μεταφορικό μέσο που δεν είναι το μέσο εξόδου ($\lambda_{Nk}^{x}(t) = \xi_{NkN''}^{xx_f}(t) \quad \forall t \in T, \forall x \neq x_f \in M, \forall k \in \{\Gamma^{-1}(N), N'\}$). Συγκεκριμένα ο τύπος της αρχικοποίησης μπορεί να γραφεί $\lambda_{ik}^{x}(t) = \xi_{ikj}^{xy}(t)$ για $j = 4'', y = 1, i = 4, x = 2$ και $k \in \{2,3,4'\}$.

$Μεταβλητές \; αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
$*$	4	1	$*$	2										
$*$	4	1	$*$	3										
$*$	4	1	$*$	$4'$										
$4''$	4	2	1	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$4''$	4	2	1	3	1	1	1	1	1	1	1	1	1	1
$4''$	4	2	1	$4'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.5 Το διάνυσμα Λ_4 μετά το βήμα 1.2

Να σημειωθεί ότι στον πίνακα διανυσμάτων, όπως στον παραπάνω, οι μεταβλητές y και j δεν είναι άμεσα εξαρτημένες μεταβλητές του $\lambda_{ik}^{x}(t)$ μιας και αυτές υπολογίζονται με την συνάρτηση min επί των τιμών που προκύπτουν από τον τύπο που περικλείει αυτές τις μεταβλητές. Έτσι για τον εντοπισμό μιας τιμής του $\lambda_{ik}^{x}(t)$, αυτές οι μεταβλητές δεν λαμβάνονται υπόψη. Ακόμα, στους πίνακες τονίζονται οι μεταβολές των τιμών με γκρι χρώμα για να είναι εύκολος ο εντοπισμός τους.

Στον βήμα 1.3, οι ετικέτες για τον κόμβο εξόδου 4 και το μεταφορικό μέσο εξόδου 1, μηδενίζονται ($\lambda_{Nk}^{x_f}(t) = 0 \quad \forall t \in T, \forall k \in \{\Gamma^{-1}(N), N'\}$) και ολοκληρώνοντας την αρχικοποίηση στο βήμα 1 προκύπτει το ακόλουθο διάνυσμα Λ_4.

$Μεταβλητές \; αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
$*$	4	1	$*$	2	0	0	0	0	0	0	0	0	0	0
$*$	4	1	$*$	3	0	0	0	0	0	0	0	0	0	0
$*$	4	1	$*$	$4'$	0	0	0	0	0	0	0	0	0	0
$*$	4	2	$*$	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$*$	4	2	$*$	3	1	1	1	1	1	1	1	1	1	1
$*$	4	2	$*$	$4'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.6 Το διάνυσμα Λ_4 μετά το βήμα 1.3

Στο 2° βήμα της εκτέλεσης του αλγορίθμου, εισάγεται ο κόμβος προορισμού 4 στην λίστα SE και στον παρακάτω πίνακα φαίνεται η κατάσταση όλων των μεταβλητών που σχετίζονται με αυτή ($Deque(i)$, $FirstNode$ και $LastNode$).

$Deque(i)$				$SE\ λίστα$	$FirstNode$	$LastNode$
1	2	3	4			
0	0	0	∞	4	4	4

Πίνακας 3.7 Η λίστα SE μετά το βήμα 2

Σύμφωνα με το 4° βήμα, εξάγεται ο πρώτος κόμβος της λίστας SE δηλαδή ο κόμβος $j = 4$ και ξεκινάει η αναδρομική διαδικασία. Με βάση τις αναδρομές 4.1 έως 4.5, το βήμα 4.1 εκτελείται για πρώτη φορά για τις τιμές $j = 4, i = 2, x = 1, y = 1, k = 1, t = 1$. Να σημειωθεί ότι το σύνολο $\Gamma^{-1}(2)$ είναι $\{1\}$ αφού μόνο ο κόμβος 1 οδηγεί στον 2. Ο υπολογισμός του $\lambda_{21}^{1}(1)$ γίνεται με αντικατάσταση στο τύπο που περιγράφηκε στην παραπάνω ενότητα, όπως φαίνεται στην συνέχεια.

$$\lambda_{ik}^{x}(t) = min\left\{\lambda_{ik}^{x}(t), \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right) + \lambda_{ji}^{y}\left(t + \xi_{ikj}^{xy}(t) + \tau_{ij}^{y}\left(t + \xi_{ikj}^{xy}(t)\right)\right)\right\} \xrightarrow[j=4,i=2,x=1,y=1,k=1,t=1]{}$$

$$\lambda_{21}^{1}(1) = min\left\{\lambda_{21}^{1}(1), \xi_{214}^{11}(1) + \tau_{24}^{1}\left(1 + \xi_{214}^{11}(1)\right) + \lambda_{42}^{1}\left(1 + \xi_{214}^{11}(1) + \tau_{24}^{1}\left(1 + \xi_{214}^{11}(1)\right)\right)\right\} = min\left\{\infty, 0 + \tau_{24}^{1}(1+0) + \lambda_{42}^{1}\left(1 + 0 + \tau_{24}^{1}(1+0)\right)\right\} = min\{\infty, 7 + \lambda_{42}^{1}(1+7)\} = min\{\infty, 7+0\} = min\{\infty, 7\} = 7$$

Επειδή η τιμή του $\lambda_{21}^{1}(1)$ άλλαξε, θα πρέπει να βάλουμε τον κόμβο $i = 2$ στην λίστα SE. Για να μπει ο κόμβος στην λίστα, θα πρέπει να ελέγξουμε πρώτα τον δείκτη $Deque(2)$. Αφού η τιμή του δείκτη είναι 0, ο κόμβος θα μπει στο τέλος της λίστας και οι μεταβλητές που σχετίζονται με την λίστα SE θα διαμορφωθούν ως εξής.

$Deque(i)$				$SE\ λίστα$	$FirstNode$	$LastNode$
1	2	3	4			
0	∞	0	−1	2	2	2

Πίνακας 3.8 Η λίστα SE μετά το βήμα 4.2 για $i = 2$

Η τιμή του $Deque$ για τον κόμβο 4 μεταβλήθηκε σε -1 εξαιτίας του ότι έγινε η εξαγωγή του κόμβου από την λίστα SE στην αρχή της εκτέλεσης του 4ου βήματος.

Ομοίως γίνεται ο υπολογισμός όλων των ετικετών για κάθε χρονική στιγμή t σύμφωνα με την αναδρομή 4.5. Μετά το πέρας των επαναλήψεων του t έχουμε το ακόλουθο διάνυσμα ετικετών.

$Μεταβλητές\ αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	2	1	1	1	7	7	7	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.9 Το διάνυσμα $\lambda_{21}^{1}(t)$ μετά την αναδρομή 4.5

Στο παραπάνω πίνακα βλέπουμε ότι οι τιμές του διανύσματος για τις πρώτες τρείς χρονικές στιγμές μεταβλήθηκαν και έχουν την ίδια τιμή. Αυτό δεν πρέπει να μας προβληματίζει γιατί:

- Τα $\xi_{214}^{11}(t) = 0, \forall t \in T$ αφού δεν υπάρχει αλλαγή μεταφορικού μέσου.
- Τα $\tau_{24}^{1}(t) = 7, \forall t \in T$ αφού δεν μεταβάλλεται χρονικά ο χρόνος μεταφοράς στην ακμή $(2,4)$ για το μεταφορικό μέσο 1.
- Το αρχικό $\lambda_{42}^{1}\left(t + \xi_{214}^{11}(t) + \tau_{24}^{1}\left(t + \xi_{214}^{11}(t)\right)\right) = \lambda_{42}^{1}(t+7) = 0, \forall t+7 \in T$ όπως υπολογίστηκε στην αρχικοποίηση του αλγόριθμου.

Για $t + 7 > 10 \Rightarrow t > 3$ το διάνυσμα λ_{21}^{1} δεν έχει τιμές, γιατί το πρόβλημα δεν είναι ορισμένο για τις χρονικές στιγμές $t > 10$ και συνεπώς στους υπολογισμούς το $\lambda_{42}^{1}\left(t + \xi_{214}^{11}(t) + \tau_{24}^{1}\left(t + \xi_{214}^{11}(t)\right)\right) = \lambda_{42}^{1}(t+7) = \infty, \forall t > 3$.

Έχοντας ολοκληρώσει την αναδρομή 4.5 ($\forall t \in T$), συνεχίζει με τον επόμενο κόμβο k που είναι ο κόμβος εισόδου στο δίκτυο από τον κόμβο 2 που συμβολίζεται με $2'$, όπως ορίζει η αναδρομή 4.4. Θα πρέπει να υπολογίσουμε τα αντίστοιχα $\xi_{22'4}^{11}(t) + \tau_{24}^{1}\left(t + \xi_{22'4}^{11}(t)\right) + \lambda_{42}^{1}\left(t + \xi_{22'4}^{11}(t) + \tau_{24}^{1}\left(t + \xi_{22'4}^{11}(t)\right)\right)$, όπου $\xi_{22'4}^{11}(t) = 0$ σύμφωνα με την εξαίρεση του κανόνα που ορίστηκε παραπάνω στα δεδομένα και τα υπόλοιπα υπολογίζονται όπως παραπάνω. Οι τιμές φαίνονται στο παρακάτω πίνακα.

$\boldsymbol{Μεταβλητές\ αναδρομής \setminus t}$					$\boldsymbol{\lambda_{ik}^{x}(t)}$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	2	1	1	$2'$	7	7	7	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.10 Το διάνυσμα $\lambda_{22'}^{1}(t)$ μετά την αναδρομή 4.5

Παρά το γεγονός ότι κάποιες τιμές του $\lambda_{ik}^{x}(t)$ άλλαξαν, το i δεν θα μπει στην λίστα SE καθώς η τιμή του δείκτη $Deque(2)$ ισούται με άπειρο.

Να σημειωθεί ότι το συγκεκριμένο διάνυσμα αναπαριστά μέχρι στιγμής τη βέλτιστη διαδρομή μέχρι τον προορισμό μας αν μπούμε στο δίκτυο από τον κόμβο 2. Συνεπώς αποθηκεύεται το μονοπάτι:

$$\pi_{22'}^{1}(t) = \{(2', 1, t), (2,1,t), (4,1,t+7), (4'', 1, t+7)\}, \forall t < 3$$

Στην συνέχεια, υπολογίζονται οι τιμές του $\lambda_{ik}^{x}(t)$ για τους υπόλοιπους συνδυασμούς μεταφορικών μέσων όπως ορίζεται από τις αναδρομές 4.3 και 4.2, οι τιμές των οποίων φαίνονται στον ακόλουθο πίνακα.

$\boldsymbol{Μεταβλητές\ αναδρομής \setminus t}$					$\boldsymbol{\lambda_{ik}^{x}(t)}$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	2	1	2	1	7	7	7	∞	∞	∞	∞	∞	∞	∞
4	2	1	2	2'	7	7	7	∞	∞	∞	∞	∞	∞	∞
4	2	2	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Μεταβλητές αναδρομής \ t					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	2	2	1	$2'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	2	2	2	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	2	2	2	$2'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.11 Το διάνυσμα Λ_2 μετά την αναδρομή 4.2

Εξαιτίας του ότι το μεταφορικό μέσο 2 δεν υποστηρίζεται στον κόμβο 2, οι τιμές του $\lambda_{ik}^{x}(t)$ δεν αλλάζουν.

Για να ολοκληρωθεί το 4° βήμα, εξετάζεται ο εναπομείναντας κόμβος $i = 3$ από την λίστα $\Gamma^{-1}(4) = \{2,3\}$.

Μεταβλητές αναδρομής \ t					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	3	1	1	1	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	2	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	4	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	$3'$	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	2	1	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	1	2	2	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	1	2	4	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	1	2	$3'$	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	2	1	1	6	6	6	6	∞	∞	∞	∞	∞	∞
4	3	2	1	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	3	2	1	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	3	2	1	$3'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	3	2	2	1	6	6	5	4	3	∞	∞	∞	∞	∞
4	3	2	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	3	2	2	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	3	2	2	$3'$	7	6	5	4	3	∞	∞	∞	∞	∞

Πίνακας 3.12 Το διάνυσμα Λ_3

$$\lambda_{31}^{1}(t) = \min\left\{\lambda_{31}^{1}(t), \xi_{314}^{11}(t) + \tau_{34}^{1}\left(t + \xi_{314}^{11}(t)\right) + \lambda_{43}^{1}\left(t + \xi_{314}^{11}(t) + \tau_{34}^{1}\left(t + \xi_{314}^{11}(t)\right)\right)\right\}$$
$$= \min\left\{\infty, 0 + \tau_{34}^{1}(t+0) + \lambda_{43}^{1}\left(t + 0 + \tau_{34}^{1}(t+0)\right)\right\}$$
$$= \min\left\{\infty, \tau_{34}^{1}(t) + \lambda_{43}^{1}\left(t + \tau_{34}^{1}(t)\right)\right\} = \min\{\infty, 5 + \lambda_{43}^{1}(t+5)\}$$

$Deque(i)$				SE λίστα	$FirstNode$	$LastNode$
1	2	3	4			
0	3	∞	−1	2,3	2	3

Πίνακας 3.13 Η λίστα SE μετά το βήμα 4.2 για $i = 3$

$$\lambda_{32}^{1}(t) = min\left\{\lambda_{32}^{1}(t), \xi_{324}^{11}(t) + \tau_{34}^{1}\left(t + \xi_{324}^{11}(t)\right) + \lambda_{43}^{1}\left(t + \xi_{324}^{11}(t) + \tau_{34}^{1}\left(t + \xi_{324}^{11}(t)\right)\right)\right\}$$
$$= min\left\{\infty, 0 + \tau_{34}^{1}(t+0) + \lambda_{43}^{1}\left(t + 0 + \tau_{34}^{1}(t+0)\right)\right\}$$
$$= min\left\{\infty, \tau_{34}^{1}(t) + \lambda_{43}^{1}\left(t + \tau_{34}^{1}(t)\right)\right\} = min\{\infty, 5 + \lambda_{43}^{1}(t+5)\}$$

$$\lambda_{31}^{1}(t) = min\left\{\lambda_{31}^{1}(t), \xi_{314}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{314}^{12}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{314}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{314}^{12}(t)\right)\right)\right\}$$
$$= min\left\{5{,}1 + \tau_{34}^{2}(t+1) + \lambda_{43}^{2}\left(t + 1 + \tau_{34}^{2}(t+1)\right)\right\}$$

$$\lambda_{32}^{1}(t) = min\left\{\lambda_{32}^{1}(t), \xi_{324}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{324}^{12}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{324}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{324}^{12}(t)\right)\right)\right\}$$
$$= min\left\{5{,}1 + \tau_{34}^{2}(t+1) + \lambda_{43}^{2}\left(t + 1 + \tau_{34}^{2}(t+1)\right)\right\}$$

$$\lambda_{34}^{1}(t) = min\left\{\lambda_{34}^{1}(t), \xi_{344}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{344}^{12}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{344}^{12}(t) + \tau_{34}^{2}\left(t + \xi_{344}^{12}(t)\right)\right)\right\}$$
$$= min\left\{5{,}1 + \tau_{34}^{2}(t+1) + \lambda_{43}^{2}\left(t + 1 + \tau_{34}^{2}(t+1)\right)\right\}$$

$$\lambda_{31}^{2}(t) = min\left\{\lambda_{31}^{2}(t), \xi_{314}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{314}^{21}(t)\right) + \lambda_{43}^{1}\left(t + \xi_{314}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{314}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, 1 + \tau_{34}^{1}(t+1) + \lambda_{43}^{1}\left(t + 1 + \tau_{34}^{1}(t+1)\right)\right\}$$

$$\lambda_{32}^{2}(t) = min\left\{\lambda_{32}^{2}(t), \xi_{324}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{324}^{21}(t)\right) + \lambda_{43}^{1}\left(t + \xi_{324}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{324}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{34}^{1}(t+\infty) + \lambda_{43}^{1}\left(t + \infty + \tau_{34}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{34}^{2}(t) = min\left\{\lambda_{34}^{2}(t), \xi_{344}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{344}^{21}(t)\right) + \lambda_{43}^{1}\left(t + \xi_{344}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{344}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{34}^{1}(t+\infty) + \lambda_{43}^{1}\left(t + \infty + \tau_{34}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{33'}^{2}(t) = min\left\{\lambda_{33'}^{2}(t), \xi_{33'4}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{33'4}^{21}(t)\right)\right.$$
$$\left. + \lambda_{43}^{1}\left(t + \xi_{33'4}^{21}(t) + \tau_{34}^{1}\left(t + \xi_{33'4}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{34}^{1}(t+\infty) + \lambda_{43}^{1}\left(t + \infty + \tau_{34}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{31}^{2}(t) = min\left\{\lambda_{31}^{2}(t), \xi_{314}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{314}^{22}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{314}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{314}^{22}(t)\right)\right)\right\}$$
$$= min\left\{6{,}0 + \tau_{34}^{2}(t+0) + \lambda_{43}^{2}\left(t + 0 + \tau_{34}^{2}(t+0)\right)\right\}$$

$$\lambda_{32}^{2}(t) = min\left\{\lambda_{32}^{2}(t), \xi_{324}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{324}^{22}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{324}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{324}^{22}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{34}^{2}(t+\infty) + \lambda_{43}^{2}\left(t + \infty + \tau_{34}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{34}^{2}(t) = min\left\{\lambda_{34}^{2}(t), \xi_{344}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{344}^{22}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{344}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{344}^{22}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{34}^{2}(t+\infty) + \lambda_{43}^{2}\left(t + \infty + \tau_{34}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{33'}^{2}(t) = min\left\{\lambda_{33'}^{2}(t), \xi_{33'4}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{33'4}^{22}(t)\right) + \lambda_{43}^{2}\left(t + \xi_{33'4}^{22}(t) + \tau_{34}^{2}\left(t + \xi_{33'4}^{22}(t)\right)\right)\right\} = min\left\{\infty, 0 + \tau_{34}^{2}(t+0) + \lambda_{43}^{2}\left(t + 0 + \tau_{34}^{2}(t+0)\right)\right\}$$

Ολοκληρώνοντας το 4° βήμα για τον κόμβο $j = 4$ ελέγχεται η λίστα SE και εξάγεται ο κόμβος $j = 2$ για να συνεχιστεί η επαναληπτική διαδικασία με $i \in \{\Gamma^{-1}(2)\} = \{1\}$ και $k \in \{\Gamma^{-1}(1), 1'\} = \{3,1'\}$.

Μεταβλητές αναδρομής \ t					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
2	1	1	1	3	8	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	1	1′	8	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	2	3	8	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	2	1′	8	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	1	1′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	2	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	2	1′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.14 Το διάνυσμα Λ_1

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{132}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{11}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{132}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{11}(t)\right)\right)\right\} = min\left\{\infty, 0 + \tau_{12}^{1}(t+0) + \lambda_{21}^{1}\left(t + 0 + \tau_{12}^{1}(t+0)\right)\right\}$$

$Deque(i)$				SE λίστα	FirstNode	LastNode
1	2	3	4			
∞	−1	1	−1	3,1	3	1

Πίνακας 3.15 Η λίστα SE μετά το βήμα 4.2 για $i = 1$

$$\lambda_{11'}^{1}(t) = min\left\{\lambda_{11'}^{1}(t), \xi_{11'2}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{11}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{11'2}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{11}(t)\right)\right)\right\} = min\left\{\infty, 0 + \tau_{12}^{1}(t+0) + \lambda_{21}^{1}\left(t + 0 + \tau_{12}^{1}(t+0)\right)\right\}$$

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{132}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{12}(t)\right) + \lambda_{21}^{2}\left(t + \xi_{132}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{12}(t)\right)\right)\right\} = min\left\{8, \infty + \tau_{12}^{2}(t+\infty) + \lambda_{21}^{2}\left(t + \infty + \tau_{12}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{11'}^{1}(t) = min\left\{\lambda_{11'}^{1}(t), \xi_{11'2}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{11'2}^{12}(t)\right) + \lambda_{21}^{2}\left(t + \xi_{11'2}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{11'2}^{12}(t)\right)\right)\right\} = min\left\{8, \infty + \tau_{12}^{2}(t+\infty) + \lambda_{21}^{2}\left(t + \infty + \tau_{12}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{13}^{2}(t) = min\left\{\lambda_{13}^{2}(t), \xi_{132}^{21}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{21}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{132}^{21}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{12}^{1}(t+\infty) + \lambda_{21}^{1}\left(t + \infty + \tau_{12}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{11'}^{2}(t) = min\left\{\lambda_{11'}^{2}(t), \xi_{11'2}^{21}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{21}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{11'2}^{21}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{12}^{1}(t+\infty) + \lambda_{21}^{1}\left(t + \infty + \tau_{12}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{13}^{2}(t) = min\left\{\lambda_{13}^{2}(t), \xi_{132}^{22}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{22}(t)\right) + \lambda_{21}^{2}\left(t + \xi_{132}^{22}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{22}(t)\right)\right)\right\} = min\left\{\infty, \infty + \tau_{12}^{2}(t+\infty) + \lambda_{21}^{2}\left(t + \infty + \tau_{12}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{11'}^{2}(t) = min\left\{\lambda_{11'}^{2}(t), \xi_{11'2}^{22}(t) + \tau_{12}^{2}\left(t + \xi_{11'2}^{22}(t)\right) + \lambda_{21}^{2}\left(t + \xi_{11'2}^{22}(t) + \tau_{12}^{2}\left(t + \xi_{11'2}^{22}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{12}^{2}(t+\infty) + \lambda_{21}^{2}\left(t + \infty + \tau_{12}^{2}(t+\infty)\right)\right\}$$

Ακολούθως, εξάγεται ο κόμβος $j = 3$ από την λίστα SE και συνεχίζει με $i \in \{1,2,4\}$ και $k \in \{3,1'\},\{1,2'\},\{2,3,4'\}$ αντίστοιχα για τα τρία i.

$Deque(i)$				SE λίστα	$FirstNode$	$LastNode$
1	2	3	4			
∞	-1	-1	-1	1	1	1

Πίνακας 3.16 Η λίστα SE στην αρχή του βήματος 4 για $j = 3$

$Μεταβλητές\ αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
3	1	1	1	3	8	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	1	1	$1'$	8	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	1	2	3	7	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	1	2	$1'$	8	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	1	2	1	$1'$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	1	2	2	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	1	2	2	$1'$	7	6	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.17 Το διάνυσμα Λ_1

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{133}^{11}(t) + \tau_{13}^{1}\left(t + \xi_{133}^{11}(t)\right) + \lambda_{31}^{1}\left(t + \xi_{133}^{11}(t) + \tau_{13}^{1}\left(t + \xi_{133}^{11}(t)\right)\right)\right\}$$
$$= min\left\{8{,}0 + \tau_{13}^{1}(t+0) + \lambda_{31}^{1}\left(t + 0 + \tau_{13}^{1}(t+0)\right)\right\}$$
$$= min\{8{,}5 + \lambda_{31}^{1}(t+5)\} = min\{8, \infty\}$$

$$\lambda_{11'}^{1}(t) = min\left\{\lambda_{11'}^{1}(t), \xi_{11'3}^{11}(t) + \tau_{13}^{1}\left(t + \xi_{11'3}^{11}(t)\right) + \lambda_{31}^{1}\left(t + \xi_{11'3}^{11}(t) + \tau_{13}^{1}\left(t + \xi_{11'3}^{11}(t)\right)\right)\right\} = min\left\{8{,}0 + \tau_{13}^{1}(t+0) + \lambda_{31}^{1}\left(t + 0 + \tau_{13}^{1}(t+0)\right)\right\}$$

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{133}^{12}(t) + \tau_{13}^{2}\left(t + \xi_{133}^{12}(t)\right) + \lambda_{31}^{2}\left(t + \xi_{133}^{12}(t) + \tau_{13}^{2}\left(t + \xi_{133}^{12}(t)\right)\right)\right\} = min\left\{8{,}1 + \tau_{13}^{2}(t+1) + \lambda_{31}^{2}\left(t + 1 + \tau_{13}^{2}(t+1)\right)\right\}$$

$$\lambda_{11'}^{1}(t) = min\left\{\lambda_{11'}^{1}(t), \xi_{11'3}^{12}(t) + \tau_{13}^{2}\left(t + \xi_{11'3}^{12}(t)\right) + \lambda_{31}^{2}\left(t + \xi_{11'3}^{12}(t) + \tau_{13}^{2}\left(t + \xi_{11'3}^{12}(t)\right)\right)\right\} = min\left\{8, \infty + \tau_{13}^{2}(t+\infty) + \lambda_{31}^{2}\left(t + \infty + \tau_{13}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{13}^{2}(t) = min\left\{\lambda_{13}^{2}(t), \xi_{133}^{21}(t) + \tau_{13}^{1}\left(t + \xi_{133}^{21}(t)\right) + \lambda_{31}^{1}\left(t + \xi_{133}^{21}(t) + \tau_{13}^{1}\left(t + \xi_{133}^{21}(t)\right)\right)\right\} = min\left\{\infty, \infty + \tau_{13}^{1}(t+\infty) + \lambda_{31}^{1}\left(t + \infty + \tau_{13}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{11'}^{2}(t) = min\left\{\lambda_{11'}^{2}(t), \xi_{11'3}^{21}(t) + \tau_{13}^{1}\left(t + \xi_{11'3}^{21}(t)\right) + \lambda_{31}^{1}\left(t + \xi_{11'3}^{21}(t) + \tau_{13}^{1}\left(t + \xi_{11'3}^{21}(t)\right)\right)\right\} = min\left\{\infty, \infty + \tau_{13}^{1}(t+\infty) + \lambda_{31}^{1}\left(t + \infty + \tau_{13}^{1}(t+\infty)\right)\right\}$$

$$\lambda_{13}^{2}(t) = min\left\{\lambda_{13}^{2}(t), \xi_{133}^{22}(t) + \tau_{13}^{2}\left(t + \xi_{133}^{22}(t)\right) + \lambda_{31}^{2}\left(t + \xi_{133}^{22}(t) + \tau_{13}^{2}\left(t + \xi_{133}^{22}(t)\right)\right)\right\} = min\left\{\infty, \infty + \tau_{13}^{2}(t+\infty) + \lambda_{31}^{2}\left(t + \infty + \tau_{13}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{11'}^{2}(t) = min\left\{\lambda_{11'}^{2}(t), \xi_{11'3}^{22}(t) + \tau_{13}^{2}\left(t + \xi_{11'3}^{22}(t)\right) + \lambda_{31}^{2}\left(t + \xi_{11'3}^{22}(t) + \tau_{13}^{2}\left(t + \xi_{11'3}^{22}(t)\right)\right)\right\} = min\left\{\infty, 0 + \tau_{13}^{2}(t+0) + \lambda_{31}^{2}\left(t + 0 + \tau_{13}^{2}(t+0)\right)\right\} = min\left\{\infty, \tau_{13}^{2}(t) + \lambda_{31}^{2}\left(t + \tau_{13}^{2}(t)\right)\right\}$$

$\boldsymbol{Μεταβλητές\ αναδρομής \setminus t}$					$\boldsymbol{\lambda_{ik}^{x}(t)}$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
3	2	1	1	1	7	6	7	∞	∞	∞	∞	∞	∞	∞
3	2	1	1	2′	7	6	7	∞	∞	∞	∞	∞	∞	∞
3	2	1	2	1	7	6	7	∞	∞	∞	∞	∞	∞	∞
3	2	1	2	2′	7	6	7	∞	∞	∞	∞	∞	∞	∞
3	2	2	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	2	2	1	2′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	2	2	2	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	2	2	2	2′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.18 Το διάνυσμα Λ_2

$$\lambda_{21}^{1}(t) = min\left\{\lambda_{21}^{1}(t), \xi_{213}^{11}(t) + \tau_{23}^{1}\left(t + \xi_{213}^{11}(t)\right) + \lambda_{32}^{1}\left(t + \xi_{213}^{11}(t) + \tau_{23}^{1}\left(t + \xi_{213}^{11}(t)\right)\right)\right\}$$
$$= min\left\{7{,}0 + \tau_{23}^{1}(t+0) + \lambda_{32}^{1}\left(t + 0 + \tau_{23}^{1}(t+0)\right)\right\}$$
$$= min\{7{,}2 + \lambda_{32}^{1}(t+2)\}$$

Από τη στιγμή που το $\lambda_{21}^{1}(2)$ άλλαξε, ελέγχεται ο κόμβος $i = 2$ για το αν θα εισαχθεί στην λίστα SE. Από τη στιγμή που το $Deque(2) = -1$ ο κόμβος θα μπει στην αρχή της λίστας και οι μεταβλητές που σχετίζονται με την λίστα SE θα διαμορφωθούν ως εξής.

$Deque(i)$				SE λίστα	$FirstNode$	$LastNode$
1	2	3	4			
∞	1	−1	−1	2,1	2	1

Πίνακας 3.19 Η λίστα SE μετά το βήμα 4.2 για $i = 2$

$$\lambda_{22'}^{1}(t) = min\left\{\lambda_{22'}^{1}(t), \xi_{22'3}^{11}(t) + \tau_{23}^{1}\left(t + \xi_{22'3}^{11}(t)\right) + \lambda_{32}^{1}\left(t + \xi_{22'3}^{11}(t) + \tau_{23}^{1}\left(t + \xi_{22'3}^{11}(t)\right)\right)\right\}$$
$$= min\{7{,}0 + \tau_{23}^{1}(t+0) + \lambda_{32}^{1}(t + 0 + \tau_{23}^{1}(t+0))\}$$

$$\lambda_{21}^{1}(t) = min\left\{\lambda_{21}^{1}(t), \xi_{213}^{12}(t) + \tau_{23}^{2}\left(t + \xi_{213}^{12}(t)\right) + \lambda_{32}^{2}\left(t + \xi_{213}^{12}(t) + \tau_{23}^{2}\left(t + \xi_{213}^{12}(t)\right)\right)\right\}$$
$$= min\left\{7, \infty + \tau_{23}^{2}(t+\infty) + \lambda_{32}^{2}\left(t + \infty + \tau_{23}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{22'}^{1}(t) = min\left\{\lambda_{22'}^{1}(t), \xi_{22'3}^{12}(t) + \tau_{23}^{2}\left(t + \xi_{22'3}^{12}(t)\right) + \lambda_{32}^{2}\left(t + \xi_{22'3}^{12}(t) + \tau_{23}^{2}\left(t + \xi_{22'3}^{12}(t)\right)\right)\right\}$$
$$= min\left\{7, \infty + \tau_{23}^{2}(t+\infty) + \lambda_{32}^{2}\left(t + \infty + \tau_{23}^{2}(t+\infty)\right)\right\}$$

$$\lambda_{21}^{2}(t) = min\left\{\lambda_{21}^{2}(t), \xi_{213}^{21}(t) + \tau_{23}^{1}\left(t + \xi_{213}^{21}(t)\right) + \lambda_{32}^{1}\left(t + \xi_{213}^{21}(t) + \tau_{23}^{1}\left(t + \xi_{213}^{21}(t)\right)\right)\right\}$$
$$= min\left\{\infty, \infty + \tau_{23}^{1}(t+\infty) + \lambda_{32}^{1}\left(t + \infty + \tau_{23}^{1}(t+\infty)\right)\right\}$$

$Μεταβλητές\ αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
3	4	1	1	2	0	0	0	0	0	0	0	0	0	0
3	4	1	1	3	0	0	0	0	0	0	0	0	0	0
3	4	1	1	4′	0	0	0	0	0	0	0	0	0	0
3	4	1	2	2	0	0	0	0	0	0	0	0	0	0
3	4	1	2	3	0	0	0	0	0	0	0	0	0	0
3	4	1	2	4′	0	0	0	0	0	0	0	0	0	0
3	4	2	1	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	4	2	1	3	1	1	1	1	1	1	1	1	1	1
3	4	2	1	4′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	4	2	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	4	2	2	3	1	1	1	1	1	1	1	1	1	1
3	4	2	2	4′	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.20 Το διάνυσμα Λ_4

Ολοκληρώνοντας το 4° βήμα για τον κόμβο $j = 3$, εξάγεται ο κόμβος $j = 2$ με $i \in \{1\}$ και $k \in \{3,1'\}$.

$Μεταβλητές\ αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
2	1	1	1	3	7	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	1	1′	7	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	2	3	7	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	2	1′	7	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	1	1′	7	6	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	2	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	2	2	1′	7	6	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.21 Το διάνυσμα Λ_1

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{132}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{11}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{132}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{132}^{11}(t)\right)\right)\right\}$$

$$\lambda_{11'}^{1}(t) = min\left\{\lambda_{11'}^{1}(t), \xi_{11'2}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{11}(t)\right) + \lambda_{21}^{1}\left(t + \xi_{11'2}^{11}(t) + \tau_{12}^{1}\left(t + \xi_{11'2}^{11}(t)\right)\right)\right\}$$

$$\lambda_{13}^{1}(t) = min\left\{\lambda_{13}^{1}(t), \xi_{132}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{12}(t)\right) + \lambda_{21}^{2}\left(t + \xi_{132}^{12}(t) + \tau_{12}^{2}\left(t + \xi_{132}^{12}(t)\right)\right)\right\}$$

$$\lambda_{11'}^{1}(t) = min\{\lambda_{11'}^{1}(t), \xi_{11'2}^{12}(t) + \tau_{12}^{2}(t + \xi_{11'2}^{12}(t)) + \lambda_{21}^{2}(t + \xi_{11'2}^{12}(t) + \tau_{12}^{2}(t + \xi_{11'2}^{12}(t)))\}$$

$$\lambda_{13}^{2}(t) = min\{\lambda_{13}^{2}(t), \xi_{132}^{21}(t) + \tau_{12}^{1}(t + \xi_{132}^{21}(t)) + \lambda_{21}^{1}(t + \xi_{132}^{21}(t) + \tau_{12}^{1}(t + \xi_{132}^{21}(t)))\}$$

$$\lambda_{11'}^{2}(t) = min\{\lambda_{11'}^{2}(t), \xi_{11'2}^{21}(t) + \tau_{12}^{1}(t + \xi_{11'2}^{21}(t)) + \lambda_{21}^{1}(t + \xi_{11'2}^{21}(t) + \tau_{12}^{1}(t + \xi_{11'2}^{21}(t)))\}$$

$$\lambda_{13}^{2}(t) = min\{\lambda_{13}^{2}(t), \xi_{132}^{22}(t) + \tau_{12}^{2}(t + \xi_{132}^{22}(t)) + \lambda_{21}^{2}(t + \xi_{132}^{22}(t) + \tau_{12}^{2}(t + \xi_{132}^{22}(t)))\}$$

$$\lambda_{11'}^{2}(t) = \min\{\lambda_{11'}^{2}(t), \xi_{11'2}^{22}(t) + \tau_{12}^{2}(t + \xi_{11'2}^{22}(t)) + \lambda_{21}^{2}(t + \xi_{11'2}^{22}(t) + \tau_{12}^{2}(t + \xi_{11'2}^{22}(t)))\}$$

Στην συνέχεια ελέγχεται η λίστα SE και εξάγετε ο κόμβος $j = 1$ και επαναλαμβάνεται το 4° βήμα με $i \in \{3\}$ και $k \in \{1,2,4,3'\}$.

$Μεταβλητές\ αναδρομής \setminus t$					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
1	3	1	1	1	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	1	2	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	1	4	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	1	3′	5	5	5	5	5	∞	∞	∞	∞	∞
1	3	1	2	1	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	2	2	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	2	4	5	5	5	4	5	∞	∞	∞	∞	∞
1	3	1	2	3′	5	5	5	5	5	∞	∞	∞	∞	∞
1	3	2	1	1	6	6	5	4	3	∞	∞	∞	∞	∞
1	3	2	1	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	3	2	1	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Μεταβλητές αναδρομής \ t					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
1	3	2	1	3′	7	6	5	4	3	∞	∞	∞	∞	∞
1	3	2	2	1	6	6	5	4	3	∞	∞	∞	∞	∞
1	3	2	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	3	2	2	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	3	2	2	3′	7	6	5	4	3	∞	∞	∞	∞	∞

Πίνακας 3.22 Το διάνυσμα Λ_3

$$\lambda_{31}^{1}(t) = min\left\{\lambda_{31}^{1}(t), \xi_{311}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{311}^{11}(t)\right) + \lambda_{13}^{1}\left(t + \xi_{311}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{311}^{11}(t)\right)\right)\right\}$$
$$= min\left\{\lambda_{31}^{1}(t), \tau_{31}^{1}(t) + \lambda_{13}^{1}\left(t + \tau_{31}^{1}(t)\right)\right\}$$

$$\lambda_{32}^{1}(t) = min\left\{\lambda_{32}^{1}(t), \xi_{321}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{321}^{11}(t)\right) + \lambda_{13}^{1}\left(t + \xi_{321}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{321}^{11}(t)\right)\right)\right\}$$
$$= min\left\{\lambda_{32}^{1}(t), \tau_{31}^{1}(t) + \lambda_{13}^{1}\left(t + \tau_{31}^{1}(t)\right)\right\}$$

$$\lambda_{34}^{1}(t) = min\left\{\lambda_{34}^{1}(t), \xi_{341}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{341}^{11}(t)\right) + \lambda_{13}^{1}\left(t + \xi_{341}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{341}^{11}(t)\right)\right)\right\}$$
$$= min\left\{\lambda_{34}^{1}(t), \tau_{31}^{1}(t) + \lambda_{13}^{1}\left(t + \tau_{31}^{1}(t)\right)\right\}$$

$$\lambda_{33'}^{1}(t) = \min\left\{\lambda_{33'}^{1}(t), \xi_{33'1}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{33'1}^{11}(t)\right) + \lambda_{13}^{1}\left(t + \xi_{33'1}^{11}(t) + \tau_{31}^{1}\left(t + \xi_{33'1}^{11}(t)\right)\right)\right\}$$
$$= \min\left\{\lambda_{33'}^{1}(t), \tau_{31}^{1}(t) + \lambda_{13}^{1}\left(t + \tau_{31}^{1}(t)\right)\right\}$$

$$\lambda_{31}^{1}(t) = min\left\{\lambda_{31}^{1}(t), \xi_{311}^{12}(t) + \tau_{31}^{2}\left(t + \xi_{311}^{12}(t)\right) + \lambda_{13}^{2}\left(t + \xi_{311}^{12}(t) + \tau_{31}^{2}\left(t + \xi_{311}^{12}(t)\right)\right)\right\}$$
$$= min\left\{\lambda_{31}^{1}(t), \infty + \tau_{31}^{2}(t + \infty) + \lambda_{13}^{2}\left(t + \infty + \tau_{31}^{2}(t + \infty)\right)\right\}$$

Ολοκληρώνοντας το 4° βήμα, ελέγχεται η λίστα SE και καθώς αυτή είναι άδεια, τερματίζεται η εκτέλεση του αλγορίθμου.

Στον ακόλουθο πίνακα συνοψίζονται όλες οι ετικέτες για όλα τα πιθανά μονοπάτια βέλτιστων διαδρομών που οδηγούν στον κόμβο προορισμού 4. Με την βοήθειά του είναι δυνατό να βρεθεί για κάθε χρονική στιγμή t, ο κόμβος j στον οποίο πρέπει να μεταβούμε με το μεταφορικό μέσο y, από τον κόμβο i στον οποίο μεταβήκαμε με το μεταφορικό μέσο x από τον κόμβο k. Αυτό είναι εφικτό εντοπίζοντας την τελευταία φορά που μεταβλήθηκε το $\lambda_{ik}^{x}(t)$ για τις παραπάνω μεταβλητές. Οι τιμές παρουσιάζονται με την σειρά που υπολογίστηκαν. Μία τιμή του $\lambda_{ik}^{x}(t)$ είναι σημειωμένη είτε όταν για τις μεταβλητές αναδρομής άλλαξε και είναι ελάχιστη, είτε όταν άλλαξε και δεν είναι ελάχιστη, ενώ δεν είναι σημειωμένη όταν η τιμή του $\lambda_{ik}^{x}(t)$ παρέμεινε σταθερή. Σε κάθε περίπτωση, το βέλτιστο μονοπάτι εντοπίζεται με βάση τις συνολικά ελάχιστες τιμές του $\lambda_{ik}^{x}(t)$, ανεξαρτήτως j και y.

Μεταβλητές αναδρομής \ t					$\lambda_{ik}^{x}(t)$									
j	i	x	y	k	1	2	3	4	5	6	7	8	9	10
4	4	2	1	3	1	1	1	1	1	1	1	1	1	1
4	4	1	1	2	0	0	0	0	0	0	0	0	0	0
4	4	1	1	3	0	0	0	0	0	0	0	0	0	0
4	4	1	1	4′	0	0	0	0	0	0	0	0	0	0
4	2	1	1	1	7	7	7	∞	∞	∞	∞	∞	∞	∞
4	2	1	1	2′	7	7	7	∞	∞	∞	∞	∞	∞	∞
4	3	1	1	1	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	2	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	4	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	1	3′	5	5	5	5	5	∞	∞	∞	∞	∞
4	3	1	2	1	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	1	2	2	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	1	2	4	5	5	5	4	5	∞	∞	∞	∞	∞
4	3	2	1	1	6	6	6	6	∞	∞	∞	∞	∞	∞
4	3	2	2	1	6	6	5	4	3	∞	∞	∞	∞	∞
4	3	2	2	3′	7	6	5	4	3	∞	∞	∞	∞	∞
2	1	1	1	3	8	8	∞	∞	∞	∞	∞	∞	∞	∞
2	1	1	1	1′	8	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	1	2	3	7	8	∞	∞	∞	∞	∞	∞	∞	∞
3	1	2	2	1′	7	6	∞	∞	∞	∞	∞	∞	∞	∞
3	2	1	1	1	7	6	7	∞	∞	∞	∞	∞	∞	∞
3	2	1	1	2′	7	6	7	∞	∞	∞	∞	∞	∞	∞
2	1	1	1	1′	7	8	∞	∞	∞	∞	∞	∞	∞	∞

Πίνακας 3.23 Το διάνυσμα Λ_i

Το βέλτιστο μονοπάτι, όταν ο κόμβος έναρξης είναι ο $k = 1'$ και το μεταφορικό μέσο είναι $x = 1$, είναι το ακόλουθο:

1. Ξεκινώντας την χρονική στιγμή $t = 1$. (Οι τιμές σημειώνονται στον πίνακα)
    a. Στο χρόνο $t = 1$ θα είναι στον κόμβο $i = 1$ προερχόμενος από τον κόμβο $k = 1'$ με το μεταφορικό μέσο $x = 1$. Θα πρέπει να μεταβεί στον κόμβο $j = 2$ με το μεταφορικό μέσο $y = 1$. Η τιμή $\lambda_{11'}^{1}(1) = 7$ υποδηλώνει ότι ο συνολικός βέλτιστος χρόνος μέχρι τον προορισμό είναι 7 χρονικές μονάδες. Ο χρόνος μετεπιβίβασης δεν υπάρχει και ο συνολικός χρόνος για την συγκεκριμένη μετάβαση είναι $\tau_{12}^{1}(1) = 1$ χρονική μονάδα.
    b. Στο χρόνο $t = 2$ θα είναι στον κόμβο $i = 2$ προερχόμενος από τον κόμβο $k = 1$ με το μεταφορικό μέσο $x = 1$. Θα πρέπει να μεταβεί στον κόμβο $j = 3$ με το μεταφορικό μέσο $y = 1$. Ο χρόνος μετεπιβίβασης δεν υπάρχει και ο συνολικός χρόνος για την συγκεκριμένη μετακίνηση είναι $\tau_{23}^{1}(2) = 2$ χρονικές μονάδες.

c. Στο χρόνο $t = 4$ θα είναι στον κόμβο $i = 3$ προερχόμενος από τον κόμβο $k = 2$ με το μεταφορικό μέσο $x = 1$. Θα πρέπει να μεταβεί στον κόμβο $j = 4$ με το μεταφορικό μέσο $y = 2$. Ο χρόνος μετεπιβίβασης είναι 1 χρονική μονάδα και ο συνολικός χρόνος για την συγκεκριμένη μετάβαση είναι $\tau_{34}^{1}(4) = 3$ χρονικές μονάδες.

d. Στο χρόνο $t = 7$ θα είναι στον κόμβο $i = 4$ προερχόμενος από τον κόμβο $k = 3$ με το μεταφορικό μέσο $x = 2$. Ο κόμβος είναι τερματικός αλλά θα πρέπει να γίνει αλλαγή μεταφορικού μέσου σε $y = 1$. Ο χρόνος μετεπιβίβασης είναι 1 χρονική.

e. Στο χρόνο $t = 8$ έχει ολοκληρωθεί η μετάβαση από τον αρχικό κόμβο 1 στο κόμβο προορισμού 4.

2. Ξεκινώντας την χρονική στιγμή $t = 2$. (Οι τιμές σημειώνονται στον πίνακα)

a. Στο χρόνο $t = 2$ θα είναι στον κόμβο $i = 1$ προερχόμενος από τον κόμβο $k = 1'$ με το μεταφορικό μέσο $x = 1$. Θα πρέπει να μεταβεί στον κόμβο $j = 2$ με το μεταφορικό μέσο $y = 1$. Ο χρόνος μετεπιβίβασης δεν υπάρχει και ο συνολικός χρόνος για την συγκεκριμένη μετάβαση είναι $\tau_{12}^{1}(1) = 1$ χρονική μονάδα.

b. Στο χρόνο $t = 3$ θα είναι στον κόμβο $i = 2$ προερχόμενος από τον κόμβο $k = 1$ με το μεταφορικό μέσο $x = 1$. Θα πρέπει να μεταβεί στον κόμβο $j = 4$ με το μεταφορικό μέσο $y = 1$. Ο χρόνος μετεπιβίβασης δεν υπάρχει και ο συνολικός χρόνος για την συγκεκριμένη μετακίνηση είναι $\tau_{24}^{1}(3) = 7$ χρονικές μονάδες.

c. Στο χρόνο $t = 10$ θα είναι στον κόμβο $i = 4$ προερχόμενος από τον κόμβο $k = 2$ με το μεταφορικό μέσο $x = 1$. Ο κόμβος είναι τερματικός και έτσι θα έχει ολοκληρωθεί η μετάβαση από τον αρχικό κόμβο 1 στο κόμβο προορισμού 4.

3. Ξεκινώντας οποιαδήποτε χρονική στιγμή $t \geq 3$, δεν μπορεί να υπολογιστεί βέλτιστο μονοπάτι καθώς δεν υπάρχουν επαρκή δεδομένα στον ορισμό του συγκεκριμένου παραδείγματος. Πρακτικά αυτό σημαίνει ότι οποιαδήποτε διαδρομή και να ελεγχτεί δεν θα μπορέσει να φτάσει στον κόμβο προορισμού 4 πριν την $10^{η}$ χρονική στιγμή, στην οποία και τελειώνουν τα δεδομένα.

3.4 Υλοποίηση του αλγόριθμου με Fortran

Όπως φάνηκε από το προηγούμενο παράδειγμα το πλήθος των πράξεων που απαιτούνται για την εύρεση των βέλτιστων διαδρομών σε ένα μικρής κλίμακας πρόβλημα είναι αρκετά μεγάλο. Γνωρίζοντας δε ότι η πολυπλοκότητα του προβλήματος αυξάνει εκθετικά με την προσθήκη νέων κόμβων και μέσων μεταφοράς, η ανάγκη για χρήση υπολογιστικών συστημάτων γίνεται επιτακτική. Σε αυτό το πλαίσιο, στη συνέχεια γίνεται παρουσίαση της υλοποίησης του υπολογιστικού συστήματος που πραγματοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας με χρήση της γλώσσας προγραμματισμού Fortran.

Η Fortran επιλέχτηκε ως γλώσσα προγραμματισμού καθώς θεωρείται ταχύτατη σε ό,τι αφορά τους υπολογισμούς πράξεων και γενικότερα συνίσταται για την ανάπτυξη επιστημονικών αλγόριθμων (Prentice, 1998). Μάλιστα, με στόχο την δοκιμή της ταχύτητας της Fortran ως προς άλλες πιο εξελιγμένες αντικειμενοστραφείς γλώσσες προγραμματισμού, υλοποιήθηκε ο ίδιος αλγόριθμος και σε C# όπου γίνεται σχολιασμός της απόδοσής τους στο παράρτημα.

Ο αλγόριθμος και τα δεδομένα εισόδου έχουν σχεδιαστεί έτσι ώστε να ελαχιστοποιηθεί η απαίτηση σε μνήμη και παράλληλα να μειωθεί ο όγκος εισαγωγής δεδομένων στην φάση της μοντελοποίησης του δικτύου.

Στις επόμενες ενότητες του κεφαλαίου περιγράφονται τόσο τα δεδομένα εισόδου του υλοποιημένου αλγορίθμου όσο και ο ίδιος ο αλγόριθμος.

3.4.1 Δεδομένα εισόδου

Όπως έχει αναφερθεί στις παραπάνω ενότητες, για να λειτουργήσει ο αλγόριθμος απαιτούνται οι ακόλουθες οντότητες:

1. Το δίκτυο το οποίο περιγράφεται με παραλλαγή της reverse start αναπαράστασης.
2. Οι χρόνοι ταξιδιού από κόμβο σε κόμβο με όλα τα δυνατά μεταφορικά μέσα, το οποίο στην ουσία είναι το κόστος κάθε ακμής ανά μέσο μεταφοράς.
3. Ο χρόνος μετεπιβίβασης από μέσο σε μέσο σε κάθε κόμβο και για κάθε χρονική στιγμή. Αυτός ο χρόνος περικλείει τον χρόνο αλλαγής μεταφορικού μέσου (πχ ο χρόνος επιβίβασης σε ένα λεωφορείο ενώ ο χρήστης είναι πεζός) και τον χρόνο αναμονής του μέσου επιβίβασης (πχ ο χρόνος μέχρι να έρθει το λεωφορείο). Στην συγκεκριμένη μεταβλητή, δίνοντας μια πολύ μεγάλη τιμή, μπορεί να μοντελοποιηθεί ο περιορισμός ότι δεν μπορούμε να μετακινηθούμε από αυτόν τον κόμβο με ένα όχημα εάν δεν έχουμε φτάσει στον συγκεκριμένο κόμβο με το συγκεκριμένο όχημα.
4. Ο χρόνος αναμονής του κάθε μέσου σε κάθε κόμβο καθώς περνάει από εκεί (πχ ο χρόνος αναμονής του λεωφορείου σε κάθε στάση).

5. Οι περιορισμοί που υπάρχουν σε κάθε κόμβο σε ό,τι αφορά τη κίνηση των μέσων μεταφοράς. Σε μερικές περιπτώσεις, δεν είναι δυνατόν ο χρήστης να «στρίψει» από την μια ακμή σε μια άλλη καθώς διέρχεται από έναν κόμβο (όπως όταν υπάρχει διάζωμα στον δρόμο). Για να μπορέσουμε να ενσωματώσουμε αυτή την πληροφορία στον αλγόριθμο, ορίζουμε για αυτή την «στροφή» ένα πολύ μεγάλο χρόνο αναμονής με στόχο να αποτρέψουμε τον αλγόριθμο από το να επιλέξει την συγκεκριμένη κίνηση.

Οι τελευταίες τρεις μεταβλητές, ενσωματώνονται στην μεταβλητή $\xi_{ikj}^{xy}(t)$ που έχει περιγραφεί στην παραπάνω ενότητα.

Πέρα από τις παραπάνω οντότητες που περιγράφουν πλήρως το δίκτυο μεταφορών, ο αλγόριθμος απαιτεί και τα στοιχεία έναρξης και τα στοιχεία τερματισμού. Τα στοιχεία έναρξης αποτελούνται από τον κόμβο έναρξης και το χρόνο έναρξης της διαδρομής ενώ τα στοιχεία τερματισμού ισοδυναμούν με τον κόμβο τερματισμού και το χρόνο τερματισμού της διαδρομής.

Η περιγραφή του δικτύου γίνεται με βάση ένα αρχείο εισόδου, παράδειγμα του οποίου περιέχεται στο παράρτημα Αρχείο Εισόδου το οποίο και περιγράφει το πρόβλημα της ενότητας 3.3. Για να είναι ευανάγνωστο το αρχείο, κάθε δεδομένο βρίσκεται σε νέα γραμμή, ενώ η κάθε ομάδα δεδομένων διαχωρίζεται με μια κενή γραμμή (που στο παράδειγμα του παραρτήματος σημειώνεται με παύλες). Το αρχείο εισόδου αποτελείται από τρείς βασικές ενότητες.

Εικόνα 3.4 Δομή αρχείου εισόδου

Στην πρώτη ενότητα περιγράφονται οι κόμβοι του δικτύου με βάση την reverse start αναπαράσταση. Ειδικότερα, στην πρώτη γραμμή αναγράφεται ο αριθμός των κόμβων του

δικτύου. Στην συνέχεια, για κάθε κόμβο αναγράφεται ο αύξων αριθμός (α/α) του κόμβου, ο αριθμός των κόμβων που φτάνουν σε αυτόν τον κόμβο (predecessor nodes) και ακολούθως οι α/α αυτών των κόμβων.

Αφού ολοκληρωθεί η περιγραφή των κόμβων σύμφωνα με την reverse start αναπαράσταση, στην επόμενη ενότητα, περιγράφονται συνεπτυγμένοι οι χρόνοι ταξιδιού από κόμβο σε κόμβο με όλα τα δυνατά μεταφορικά μέσα για κάθε χρονική στιγμή. Ειδικότερα, έχει αναπτυχθεί ένας τρόπος περιγραφής της ζητούμενης πληροφορίας με βάση ιεραρχικούς κανόνες. Στην πρώτη γραμμή αναγράφεται ο αριθμός των κανόνων που ακολουθούν, καθώς αυτό είναι υποχρεωτικό από την Fortran για να μπορέσει να διαβάσει σωστά το πλήθος των κανόνων. Στην συνέχεια ακολουθούν οι κανόνες που αναγράφονται, ένας σε κάθε γραμμή.

Ο κανόνας περιέχει έξι νούμερα. Το πρώτο νούμερο αναγράφει τον α/α του κόμβου από τον οποίο ξεκινάει η ακμή. Το δεύτερο νούμερο είναι ο α/α του κόμβου προορισμού. Το τρίτο νούμερο είναι ο αύξων αριθμός του μεταφορικού μέσου[1]. Το τέταρτο νούμερο αποτελεί τον χρόνο έναρξης του κανόνα (σε λεπτά από την αρχή της ημέρας). Το πέμπτο νούμερο αποτελεί τον χρόνο λήξης του κανόνα και το τελευταίο νούμερο αποτελεί τον χρόνο σε λεπτά για να φτάσει το συγκεκριμένο μέσο από τον κόμβο έναρξης στον κόμβο προορισμού. Συγκεντρωτικά, η δομή του κανόνα φαίνεται στην ακόλουθη εικόνα:

Περιγραφή ακμής		Μεταφορικό μέσο	Χρονική ισχύς κανόνα		Χρόνος διαδρομής
Κόμβος Έναρξης	Κόμβος προορισμού		Αρχή	Λήξη	

Εικόνα 3.5 Μορφή κανόνα χρόνου διαδρομής

Όπως αναφέρθηκε προηγουμένως, οι κανόνες αυτοί είναι ιεραρχικοί. Ειδικότερα, ο κανόνας που είναι ορισμένος στο κάτω μέρος της λίστας, είναι ισχυρότερος από όλους τους παραπάνω. Αυτό γίνεται κατανοητό στην περίπτωση που για το ίδιο μέσο και για την ίδια ακμή έχουμε δύο διαφορετικούς κανόνες των οποίων η χρονική ισχύς επικαλύπτεται. Σε αυτή την περίπτωση, ο κανόνας που βρίσκεται πιο κάτω στην λίστα των κανόνων, υπερισχύει. Έστω ότι για την ακμή από τον κόμβο 1 στον κόμβο 2 με το μεταφορικό μέσο 1 ισχύει ότι ο χρόνος διαδρομής είναι 5 λεπτά για όλη την ημέρα εκτός από το μεσημέρι (840 λεπτά από την έναρξη της ημέρας ή 14:00 έως 960 λεπτά από την έναρξη της ημέρας ή 16:00) όπου ο χρόνος είναι 10 λεπτά. Αυτή η συνθήκη θα μπορούσε να γραφεί με τους ακόλουθους ισοδύναμους τρόπους:

Περιγραφή ακμής		Μεταφορικό μέσο	Χρονική ισχύς κανόνα		Χρόνος διαδρομής
Κόμβος Έναρξης	Κόμβος προορισμού		Αρχή	Λήξη	
1ος τρόπος αναπαράστασης συνθήκης					
1	2	1	0	839	5
1	2	1	840	960	10

[1] Από αυτούς τους αύξοντες αριθμούς, ο κώδικας αναγνωρίζει το πλήθος των μεταφορικών μέσων που χρησιμοποιούνται στο δίκτυο.

Περιγραφή ακμής		Μεταφορικό μέσο	Χρονική ισχύς κανόνα		Χρόνος διαδρομής
Κόμβος Έναρξης	Κόμβος προορισμού		Αρχή	Λήξη	
1	2	1	961	1440	5
$2^{ος}$ τρόπος αναπαράστασης συνθήκης					
1	2	1	0	1440	5
1	2	1	840	960	10

Εικόνα 3.6 Παράδειγμα ιεραρχίας κανόνων χρόνου διαδρομής

Συνεπώς, οι κανόνες θα πρέπει να ορίζονται από τον γενικότερο στον ειδικότερο κανόνα με στόχο ο ειδικότερος κανόνας να υπερισχύει σε περίπτωση σύγκρουσης χρονικής ισχύος.

Οποιαδήποτε μεταφορά δεν περιγράφεται σε αυτόν τον πίνακα κανόνων θεωρείται αδύνατη για το συγκεκριμένο δίκτυο.

Με βάση τους παραπάνω κανόνες, για να μπορέσουμε να περιγράψουμε τις αναχωρήσεις ενός μέσου μαζικής μεταφοράς (πχ τις στάσεις ενός λεωφορείου) θα πρέπει σε κάθε στάση/κόμβο να γράψουμε σαν αρχή και λήξη του κανόνα την ίδια χρονική στιγμή, δηλαδή την ώρα αναχώρησης. Για παράδειγμα ο ακόλουθος πίνακας περιγράφει τις αναχωρήσεις ενός λεωφορείου (μεταφορικό μέσο με αύξων αριθμό 2) από τον κόμβο 1 στον κόμβο 2 που περνάει από τον κόμβο 1 κάθε μια ώρα από τις 10 το πρωί μέχρι και τις 12 το μεσημέρι και θέλει 20 λεπτά για να φτάσει στον κόμβο 2.

Περιγραφή ακμής		Μεταφορικό μέσο	Χρονική ισχύς κανόνα		Χρόνος διαδρομής
Κόμβος Έναρξης	Κόμβος προορισμού		Αρχή	Λήξη	
1	2	2	600	600	20
1	2	2	660	660	20
1	2	2	720	720	20

Εικόνα 3.7 Παράδειγμα περιγραφής δρομολογίων λεωφορείου με κανόνες

Τέλος, στην τρίτη και τελευταία ενότητα του αρχείου εισόδου ορίζεται η μεταβλητή $\xi_{ikj}^{xy}(t)$ του αλγόριθμου πάλι με την μορφή κανόνων. Στην ουσία, σε αυτή την ενότητα περιγράφουμε όλες τις δυνατές μετεπιβιβάσεις και όλες τις δυνατές στροφές από μια ακμή σε μια άλλη. Στην πρώτη γραμμή αναγράφεται ο αριθμός των κανόνων που ακολουθούν, μιας και αυτό είναι υποχρεωτικό από την Fortran για να μπορέσει να διαβάσει σωστά το πλήθος των κανόνων. Στην συνέχεια ακολουθούν οι κανόνες που αναγράφονται ένας σε κάθε γραμμή.

Ειδικότερα, ο κάθε κανόνας περιέχει οκτώ νούμερα. Το πρώτο νούμερο αναγράφει τον α/α του κόμβου από τον οποίο προέρχεται ο χρήστης. Το δεύτερο νούμερο είναι το μεταφορικό μέσο με το οποίο έρχεται ο χρήστης στον κόμβο. Το τρίτο νούμερο είναι ο αύξων αριθμός του κόμβου όπου πραγματοποιείται η μετεπιβίβαση. Το τέταρτο νούμερο είναι ο α/α του μέσου στο οποίο θα γίνει η μετεπιβίβαση. Το πέμπτο νούμερο είναι ο αύξων αριθμός του κόμβου προορισμού μετά την μετεπιβίβαση. Το έκτο νούμερο αποτελεί τον χρόνο έναρξης του κανόνα (σε λεπτά από την αρχή της ημέρας). Το έβδομο νούμερο αποτελεί τον χρόνο

λήξης του κανόνα και το τελευταίο νούμερο αποτελεί τον χρόνο σε λεπτά για να γίνει η μετεπιβίβαση από το ένα μέσο στο άλλο. Συγκεντρωτικά, η δομή του κανόνα φαίνεται στην ακόλουθη εικόνα:

Προερχόμενος από κόμβο	Από μεταφορικό Μέσο	Όταν είναι στον κόμβο	Μετεπιβίβαση σε μεταφορικό μέσο	Με προορισμό τον κόμβο	Χρονική ισχύς κανόνα		Χρόνος μετεπιβίβασης
					Αρχή	Λήξη	

Εικόνα 3.8 Μορφή κανόνα μετεπιβίβασης

Αντίστοιχα με τους παραπάνω κανόνες χρόνου διαδρομής και αυτοί οι κανόνες ορίζονται ιεραρχικά με αποτέλεσμα ο χρήστης να πρέπει να τους έχει εισάγει από τον γενικότερο στον ειδικότερο κανόνα.

Ένα παράδειγμα χρήσης των κανόνων μετεπιβίβασης δίνεται στον ακόλουθο πίνακα:

Προερχόμενος από κόμβο	Από μεταφορικό Μέσο	Όταν είναι στον κόμβο	Μετεπιβίβαση σε μεταφορικό μέσο	Με προορισμό τον κόμβο	Χρονική ισχύς κανόνα		Χρόνος μετεπιβίβασης
					Αρχή	Λήξη	
2	1	4	1	3	0	1440	0
2	1	4	1	4	0	1440	0
3	1	4	1	3	0	1440	0
3	2	4	1	3	0	1440	1
3	1	4	1	4	0	1440	0
3	2	4	1	4	0	1440	1
4	1	4	1	3	0	1440	0

Εικόνα 3.9 Παράδειγμα κανόνων μετεπιβίβασης

Στο παραπάνω παράδειγμα περιγράφονται όλες οι δυνατές μετεπιβιβάσεις στον κόμβο 4 (του παραδείγματος που χρησιμοποιήθηκε για να επεξηγηθεί ο αλγόριθμος στην ενότητα 3.3).

Σημαντικό είναι να αναφερθεί ότι σε κάθε κόμβο θα πρέπει να αναγράφεται μια εγγραφή εξόδου του χρήστη από το σύστημα (ή αλλιώς εγγραφή άφιξης του χρήστη στον τερματικό κόμβο) στην οποία ο κόμβος στον οποίο βρίσκεται ο χρήστης (τρίτη στήλη) ισοδυναμεί με τον προορισμό του χρήστη (πέμπτη στήλη) για κάθε δυνατό μέσο με το οποίο μπορεί να βρεθεί σε αυτόν τον κόμβο ο χρήστης (δεύτερη στήλη) και για κάθε κόμβο προέλευσης, όπου το μεταφορικό μέσο στο οποίο μετεπιβιβάζεται είναι αυτό με αύξοντα αριθμό 1. Αυτές οι εγγραφές εξασφαλίζουν ότι ο αλγόριθμος θα μπορέσει να κάνει σωστά την μετεπιβίβαση στο τελικό κόμβο προορισμού υπολογίζοντας τον σωστό χρόνο εξόδου από

το δίκτυο. Στο παραπάνω παράδειγμα, οι εγγραφές 2, 5 και 6 περιγράφουν αυτούς τους κανόνες εξόδου.

Όπως γίνεται αντιληπτό, με την παραπάνω εγγραφή μπορούμε να περιγράψουμε τον χρόνο που χρειάζεται για να βρει ο χρήστης χώρο στάθμευσης και να φτάσει πραγματικά στον τελικό του προορισμό εάν μετακινείται με αυτοκίνητο.

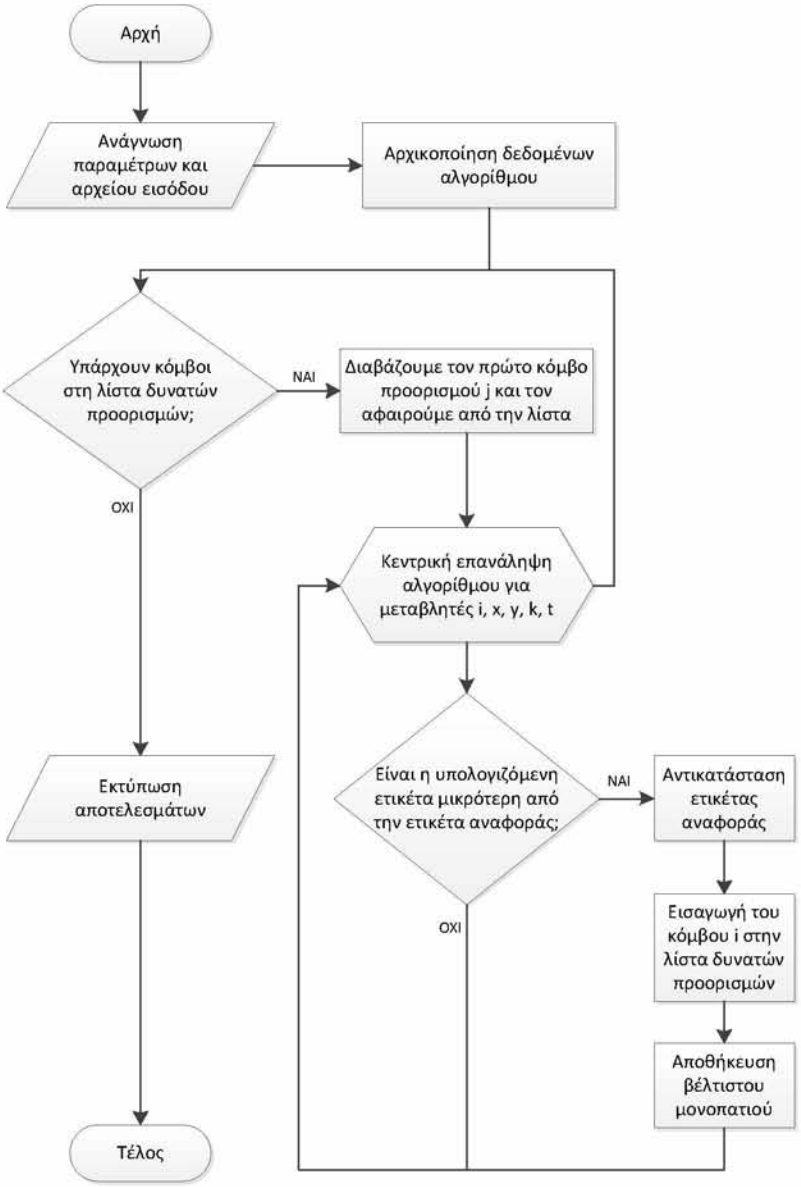
Αντίστοιχα, θα πρέπει να οριστούν εγγραφές εισόδου στο σύστημα στις οποίες ο κόμβος στον οποίο βρίσκεται ο χρήστης (τρίτη στήλη) ισοδυναμεί με τον κόμβο προέλευσης του χρήστη (πρώτη στήλη) για κάθε κόμβο προορισμού (πέμπτη στήλη) και για κάθε δυνατό μέσο με το οποίο μπορεί να βρεθεί σε αυτόν τον κόμβο ο χρήστης (τέταρτη στήλη), όπου το μεταφορικό μέσο με το οποίο εισέρχεται (δεύτερη στήλη) είναι αυτό με αύξοντα αριθμό 1. Εξαίρεση σε αυτήν την απαίτηση αποτελεί η περίπτωση που στο κόμβο εισόδου, δεν υπάρχει αλλαγή μέσου και ο χρόνος μετεπιβίβασης θεωρείται ίσως με μηδέν. Παράδειγμα αυτής της εξαίρεσης αποτελεί η τελευταία γραμμή η οποία μπορεί να παραληφθεί.

Με χρήση των παραπάνω κανόνων, μπορούμε ταυτόχρονα να περιγράψουμε τις διαδρομές των express λεωφορείων, στα οποία ενώ ο χρήστης που είναι μέσα στο λεωφορείο μπορεί να κατέβει σε όλους τους κόμβους, ένας χρήστης ο οποίος δεν είναι στην αφετηρία του δρομολογίου δεν μπορεί να ανέβει στο λεωφορείο σε κάποιον ενδιάμεσο κόμβο. Αυτό γίνεται απλά παραλείποντας τους κανόνες επιβίβασης στο συγκεκριμένο μέσο από όλους τους κόμβους, εκτός της αφετηρίας.

Τέλος, ο αλγόριθμος απαιτεί και τα στοιχεία έναρξης και τα στοιχεία τερματισμού για να μπορέσει να εκτελέσει την αναζήτηση στο περιγραφόμενο δίκτυο. Όταν ο τελικός χρήστης εκτελέσει το πρόγραμμα, μπορεί να δώσει τις παραμέτρους From Node και To Node όπου δίνει τον αύξων αριθμό του κόμβου έναρξης και τερματισμού ενώ μπορεί να προσδιορίσει το From Time και το To Time που είναι οι χρόνοι έναρξης και λήξης της αναζήτησης των δρομολογίων και καθορίζουν τις χρονικές μονάδες αναζήτησης.

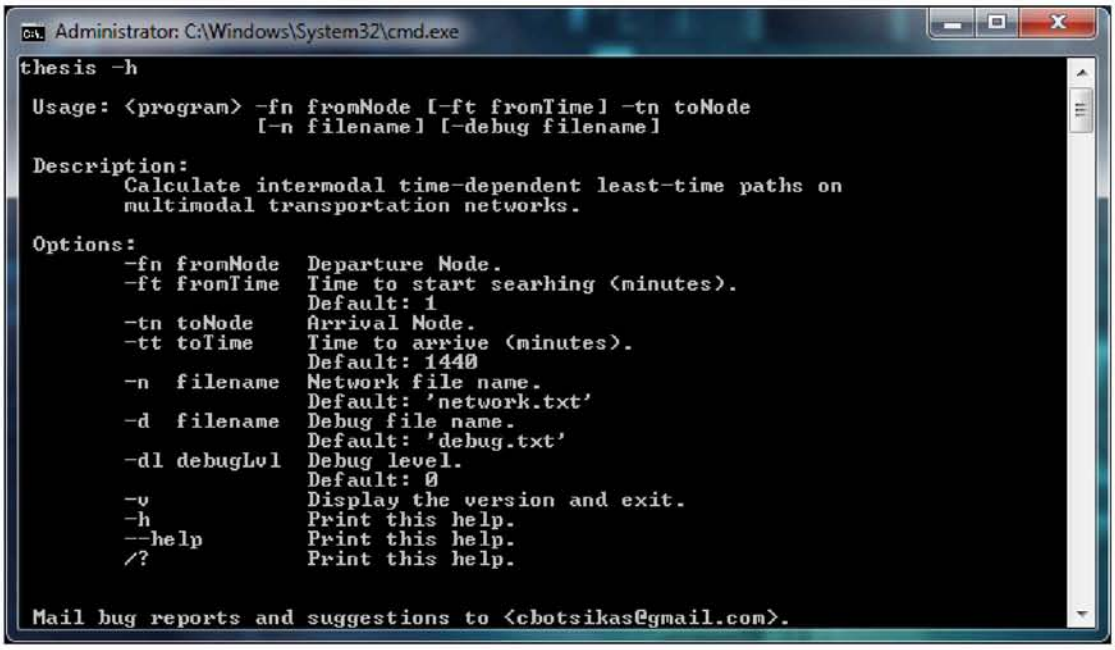
3.4.2 Η υλοποιημένη εφαρμογή

Ο αλγόριθμος που υλοποιήθηκε στα πλαίσια της διπλωματικής δίνεται στο παράρτημα Πηγαίος Κώδικας. Το διάγραμμα ροής (work flow diagram) παρουσιάζεται στην ακόλουθη εικόνα.



Εικόνα 3.10 Διάγραμμα ροής αλγορίθμου TDILTP

Αρχικά έχουμε την κλήση της υπορουτίνας $commandLine()$ που είναι υπεύθυνη για την ανάγνωση των ορισμάτων (command line arguments) με τα οποία κλήθηκε η εφαρμογή και στα οποία πρέπει να περιλαμβάνονται ο κόμβος αφετηρίας και κόμβος προορισμού. Μέσω των ορισμάτων αυτών μπορούμε να εισάγουμε στο σύστημα και άλλες επιλογές όπως τη χρονική στιγμή αφετηρίας ή την τοποθεσία του αρχείου εισόδου, στο οποίο περιγράφεται το δίκτυο (προκαθορισμένο είναι το αρχείο network.txt που βρίσκεται μαζί με την εφαρμογή). Μία λίστα ορισμάτων που δέχεται η εφαρμογή φαίνεται στην παρακάτω εικόνα, η οποία προκύπτει ζητώντας βοήθεια (/?, -h, --help).



Εικόνα 3.11 Στιγμιότυπο οθόνης βοήθειας

Στη συνέχεια, γίνεται η ανάγνωση των κόμβων i μαζί με όλους τους κόμβους j που οδηγούν σε αυτόν (predecessors) από το αρχείο εισόδου. Για τη διαχείριση αυτής της πληροφορίας έχει οριστεί ένας πίνακας nodes στον οποίο σε κάθε στοιχείο του υπάρχει ένα αντικείμενο (Type) Node, το οποίο περιγράφει έναν κόμβο. Το πλήθος των στοιχείων αυτού του πίνακα είναι ίσο με το πλήθος των κόμβων του δικτύου ($maxNodes$). Κατά την αποθήκευση των predecessors κάποιου κόμβου (i) προσθέτουμε πάντα και τον κόμβο εισαγωγής στο σύστημα (i'), ο οποίος έχει ως αύξοντα αριθμό ίδιο με αυτό του i. Για παράδειγμα όταν έχουμε 4 κόμβους ο εικονικός κόμβος εισόδου στο σύστημα για τον κόμβο 1, είναι ο κόμβος με α/α 1.

Για την καλύτερη αποθήκευση της πληροφορίας που σχετίζεται με κάθε κόμβο έχει δημιουργηθεί ένα νέο αντικείμενο (Type) Node στο οποίο μπορούμε να αποθηκεύσουμε όλη την άμεσα σχετιζόμενη με αυτόν πληροφορία. Για παράδειγμα για τον κόμβο i μπορούμε να πάρουμε όλους τους κόμβους j που οδηγούν σε αυτόν ($nodes(i)\%predecessors(:)$) ή όλους τους σχετιζόμενους κανόνες χρόνων διαδρομής ($nodes(i)\%timeTables(:,:)$) κλπ.

Για να ολοκληρωθεί η ανάγνωση του αρχείου εισόδου, είναι απαραίτητη η ανάγνωση των κανόνων χρόνων διαδρομής και των κανόνων μετεπιβίβασης που υλοποιούνται με την βοήθεια των υπορουτίνων $addTimeTable()$ και $addSwitchDelay()$ αντίστοιχα των οποίων τα ορίσματα υπάρχουν στο παράρτημα Πηγαίος Κώδικας.

Έχοντας πλέον όλες τις απαραίτητες πληροφορίες για το σύστημα που θέλουμε να λύσουμε, περνάμε στην αρχικοποίηση των δεδομένων. Πρέπει για κάθε κόμβο του δικτύου, να αρχικοποιήσουμε τις ετικέτες του ($nodes(i)\%labels(:,:,:)$) με μία πολύ μεγάλη τιμή εκτός από τις ετικέτες του κόμβου προορισμού, του οποίου τις ετικέτες τις αρχικοποιούμε με την τιμή μηδέν (0) στην περίπτωση που το μέσο μεταφοράς είναι διάφορο του μέσου

προορισμού, ενώ σε αντίθετη περίπτωση παίρνει την τιμή του χρόνου μετεπιβίβασης από το διαφορετικό μέσο μεταφοράς στο μέσο προορισμού.

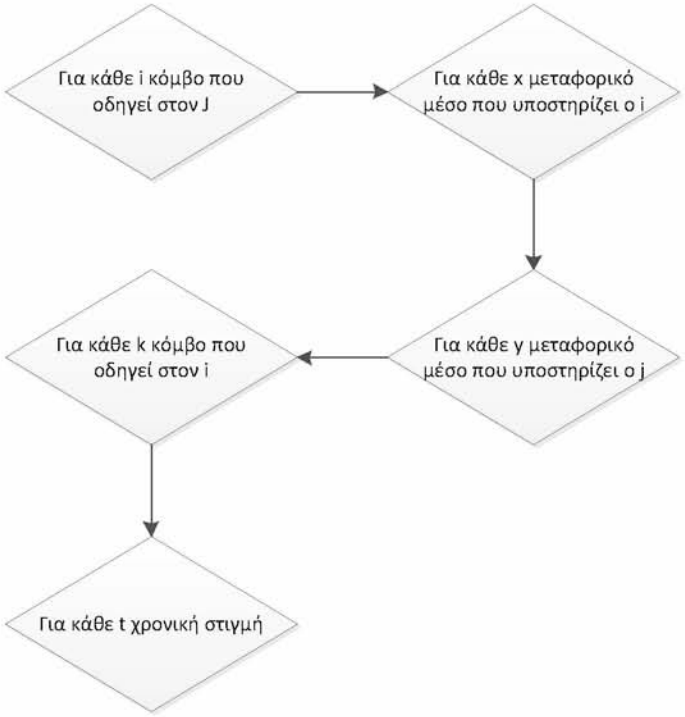
Ο χρόνος μετεπιβίβασης σε ένα κόμβο i μία χρονική στιγμή t, όταν μεταφερόμαστε με το μεταφορικό μέσο x από τον κόμβο k και θέλουμε να πάμε στον κόμβο j με το μεταφορικό μέσο y, υπολογίζεται με βάση τους κανόνες που έχουμε εισάγει στο αρχείο εισόδου χρησιμοποιώντας την συνάρτηση $getSwitchDelay()$, τα ορίσματα της οποίας υπάρχουν στο παράρτημα Πηγαίος Κώδικας.

Ολοκληρώνοντας την αρχικοποίηση, εισάγουμε τον κόμβο προορισμού στη λίστα κόμβων προς έλεγχο (Scan Eligible list). Η διαδικασία εισαγωγής κόμβων στην λίστα αυτή γίνεται με την βοήθεια της υπορουτίνας $addScanEligible()$ και είναι υπεύθυνη για να ενημερώσει τις απαραίτητες μεταβλητές.

Ξεκινώντας την αναζήτηση βέλτιστων λύσεων επαναλαμβάνουμε τις ακόλουθες διαδικασίες όσο υπάρχουν κόμβοι στην λίστα κόμβων προς έλεγχο.

Θέτουμε κόμβο προορισμού j τον πρώτο κόμβο από την παραπάνω λίστα και τον αφαιρούμε από την λίστα. Για την διαδικασία αυτή χρησιμοποιούμε την συνάρτηση $getScanEligible()$.

Για κάθε κόμβο i που οδηγεί στον κόμβο προορισμού j (χωρίς να συμπεριλάβουμε τον εικονικό κόμβο εισόδου στο j για το δίκτυο) με κάθε υποστηριζόμενο από τον κόμβο i μεταφορικό μέσο x, αλλά και για κάθε y μεταφορικό μέσο που υποστηρίζει ο κόμβος j καθώς και για κάθε κόμβο k που οδηγεί στο κόμβο i, υπολογίζουμε σε κάθε χρονική στιγμή t την ετικέτα κόστους (label). Η παραπάνω αναδρομή φαίνεται διαγραμματικά στην ακόλουθη εικόνα.



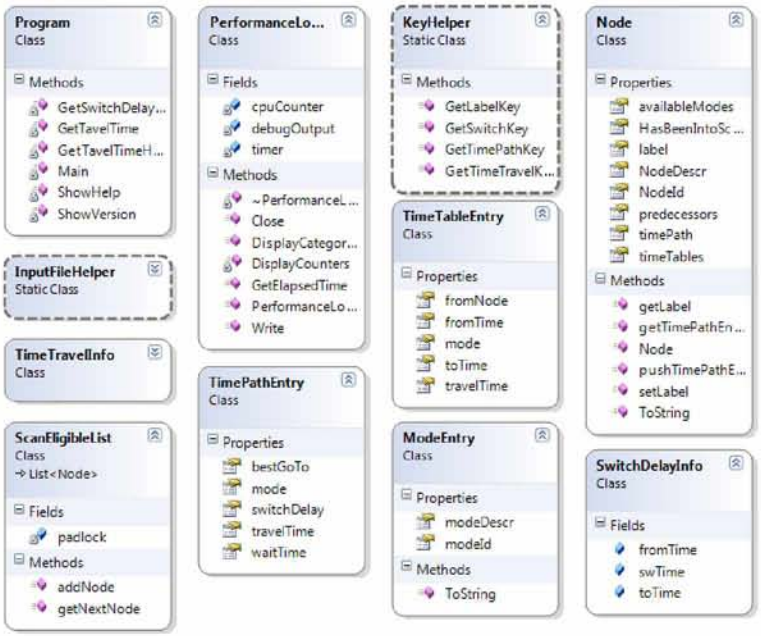
Εικόνα 3.12 Ανάλυση επανάληψης αλγόριθμου

Στη συνέχεια, τη συγκρίνουμε με την υπάρχουσα ετικέτα αναφοράς στον κόμβο i όταν προερχόμαστε από τον κόμβο k με το μεταφορικό μέσο x. Εάν είναι μικρότερη την αντικαθιστούμε. Για αυτόν τον έλεγχο αλλά και την πιθανή αντικατάσταση χρησιμοποιούμε την συνάρτηση $setLabel()$. Εάν γίνει η αντικατάσταση, εισάγουμε αν χρειάζεται, τον κόμβο i στην λίστα κόμβων προς έλεγχο με την προαναφερθείσα υπορουτίνα $addScanEligible()$ και καταγράφουμε τις πληροφορίες βέλτιστης μετακίνησης από το κόμβο i στο κόμβο j, όταν έχουμε φτάσει στον i με το x μεταφορικό μέσο την t χρονική στιγμή $(nodes(i)\%timepath)$.

Μόλις ολοκληρωθούν οι παραπάνω επαναλήψεις και πλέον δεν υπάρχουν κόμβοι στην λίστα κόμβων προς έλεγχο, έχουμε βρει τις βέλτιστες διαδρομές για κάθε χρονική στιγμή από τον κόμβο αφετηρίας στον κόμβο τερματισμού και μπορούμε να εξάγουμε τα αποτελέσματα.

Ο ίδιος ακριβώς αλγόριθμος υλοποιήθηκε και σε C# (η υλοποίηση βρίσκεται στο παράρτημα) με στόχο την δοκιμή της ταχύτητας της γλώσσας προγραμματισμού Fortran σε σχέση με την πιο εξελιγμένη αντικειμενοστραφή C#. Οι γραμμές κώδικα (μαζί με τα σχόλια) που απαιτήθηκαν σε Fortran είναι 839 ενώ στην C# γράφηκαν 802 γραμμές και χρησιμοποιήθηκαν έτοιμες βιβλιοθήκες τόσο στην FORTRAN (511 γραμμών) όσο και στην C# (1102 γραμμές). Η διαφορά στον αριθμό των γραμμών είναι αμελητέα ειδικά αν λάβουμε υπόψη ότι η συγγραφή του κώδικα σε C# έγινε αφού είχε ολοκληρωθεί ο κώδικας σε Fortran με αποτέλεσμα να υπάρχουν βελτιστοποιήσεις.

Οι κλάσεις που σχεδιάστηκαν στην αντικειμενοστραφή γλώσσα C# απεικονίζονται στο ακόλουθο σχήμα:



Εικόνα 3.13 Διάγραμμα κλάσεων της C#

3.5 Συμπεράσματα

Στο παρόν κεφάλαιο παρουσιάστηκε ο δυναμικός αλγόριθμος δρομολόγησης TDILTP (Time-Dependent Intermodal Least-Time Path) καθώς και το πληροφοριακό σύστημα που υποστηρίζει τις λειτουργίες αυτού. Ο αλγόριθμος αυτός προσεγγίζει με έναν καινοτόμο και ευφυή τρόπο το πρόβλημα της δυναμικής δρομολόγησης με πολλαπλά μεταφορικά μέσα αξιολογώντας τις πιθανές διαδρομές με βάση τον χρόνο που απαιτείται.

Στο επόμενο κεφάλαιο παρουσιάζεται ένα ακόμα λογισμικό που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής, το οποίο έχει σαν στόχο τον σχεδιασμό δικτύων μεταφοράς με πολλαπλά μέσα. Στην συνέχεια θα παρουσιαστεί η εφαρμογή του υλοποιημένου αλγορίθμου πάνω σε τυχαία δίκτυα που δημιουργήθηκαν με το παραπάνω λογισμικό καθώς και η πιλοτική εφαρμογή που πραγματοποιήθηκε στο δίκτυο της ΤΡΑΙΝΟΣΕ Α.Ε. Οι εφαρμογές αυτές θα δώσουν τη δυνατότητα αξιολόγησης της λειτουργίας του αλγορίθμου καθώς και της αποτελεσματικότητάς του, τα οποία σχολιάζονται στα συμπεράσματα της παρούσας διπλωματικής.

4

Πειραματικά Αποτελέσματα

Κεφάλαιο 4: Πειραματικά αποτελέσματα

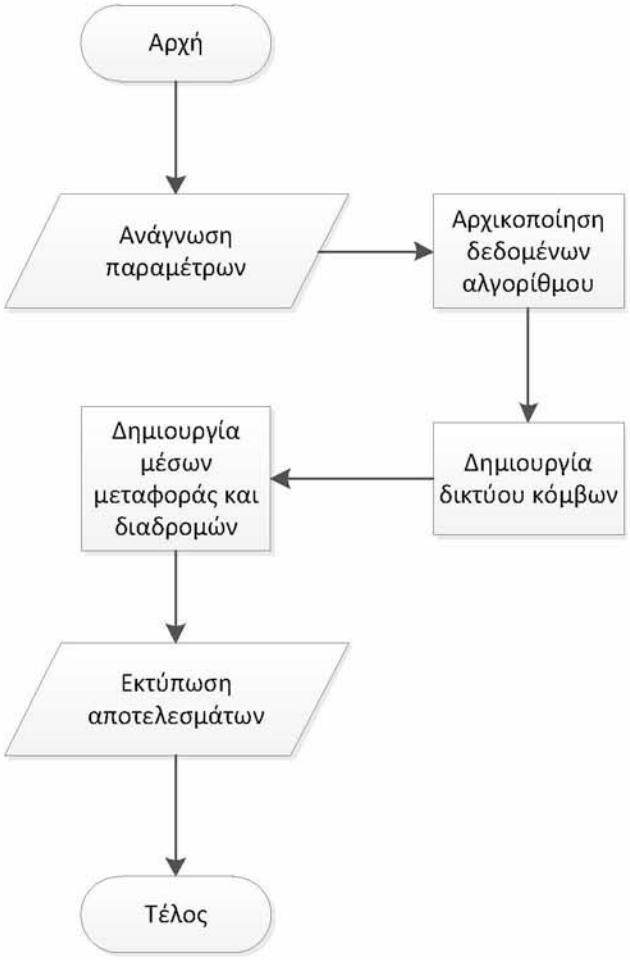
4.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο παρουσιάστηκε ο δυναμικός αλγόριθμος δρομολόγησης TDILTP (Time-Dependent Intermodal Least-Time Path) καθώς και το πληροφοριακό σύστημα που υποστηρίζει τις λειτουργίες αυτού. Σκοπός αυτού του κεφαλαίου είναι η παρουσίαση της πιλοτικής εφαρμογής του αλγόριθμου τόσο σε τυχαία παραγόμενα δίκτυα όσο και στο πραγματικό δίκτυο της ΤΡΑΙΝΟΣΕ, καθώς και η ανάλυση των αποτελεσμάτων.

Στις παραγράφους που ακολουθούν παρουσιάζονται αρχικά τα χαρακτηριστικά των πεδίων εφαρμογής, η εφαρμογή του αλγόριθμου και τα αποτελέσματα που απορρέουν. Σε πρώτο στάδιο γίνεται μια σύντομη παρουσίαση της εφαρμογής που αναπτύχθηκε για να παράγει τυχαία δίκτυα. Στην συνέχεια παρουσιάζονται τα δίκτυα που χρησιμοποιήθηκαν για να δοκιμαστεί η απόδοση του αλγόριθμου και παρουσιάζονται οι διάφορες μετρήσεις που έγιναν. Τέλος, γίνεται μια σύντομη παρουσίαση της παρούσας κατάστασης της εταιρίας ΤΡΑΙΝΟΣΕ, στην οποία εφαρμόστηκε το πιλοτικό σύστημα για τον εντοπισμό βέλτιστων διαδρομών μέσα στο δίκτυό της και στη συνέχεια παρατίθενται τα συμπεράσματα που προέκυψαν από αυτήν την πιλοτική εφαρμογή.

4.2 Τυχαία δίκτυα

Στα πλαίσια της σωστής ανάπτυξης και βελτιστοποίησης του αλγορίθμου TDILP ήταν αναγκαίο να κατασκευαστεί μία γεννήτρια δικτύων πολλαπλών μεταφορικών μέσων. Το διάγραμμα ροής του αλγόριθμου που χρησιμοποιείται από την γεννήτρια παρουσιάζεται ακολούθως.



Εικόνα 4.1 Διάγραμμα ροής αλγορίθμου γεννήτριας

Αφού πρώτα γίνει η ανάγνωση των παραμέτρων για την κατασκευή του τυχαίου δικτύου, επιλέγεται τυχαία η διασπορά των γειτονικών κόμβων. Ο καθορισμός της διασποράς των γειτονικών κόμβων είναι αρκετά σημαντικός καθώς επιθυμείται η κατασκευή αληθοφανών δικτύων, στα οποία δεν θα μπορούσε να υπάρχει απευθείας σύνδεση δύο κόμβων που τυπολογικά δεν θα ήταν γειτονικοί. Στην συνέχεια, για κάθε κόμβο επιλέγεται τυχαία το πλήθος των γειτονικών κόμβων του.

Η δημιουργία του δικτύου γίνεται υπολογίζοντας για κάθε κόμβο τους γειτονικούς του, επιλέγοντας τυχαία κάποιους αριθμούς μέσα στο εύρος της διασποράς. Κάθε φορά που επιλέγεται κάποιος γειτονικός, ενημερώνεται και αυτός για την γειτνίαση. Ταυτόχρονα υπολογίζεται και η απόσταση των κόμβων αυτών που είναι συνάρτηση της διαφοράς των αυξόντων αριθμών του καθενός. Αυτή η πληροφορία χρησιμοποιείται στην πορεία για να υπολογιστούν οι χρόνοι διαδρομών του κάθε μεταφορικού μέσου σε κάθε ακμή.

Στην συνέχεια, για κάθε μεταφορικό μέσο, υπολογίζεται τυχαία η ταχύτητα του μέσου και οι κόμβοι εκκίνησης και τερματισμού. Ο κόμβος εκκίνησης επιλέγεται τυχαία από το σύνολο των κόμβων με τα λιγότερα μεταφορικά μέσα ενώ ο τερματικός κόμβος επιλέγεται τυχαία από το σύνολο των κόμβων. Για να υπολογιστεί το μονοπάτι που θα ακολουθήσει το μεταφορικό μέσο, υλοποιήθηκε μία παραλλαγή του Depth-First Search (DFS) (Depth-first

search, 2011) (Knuth, 1997) (Tarjan, 1972) αλγορίθμου, στην οποία αναζητείται το μονοπάτι ελέγχοντας πρώτα τις ακμές με τα λιγότερα μεταφορικά μέσα. Αφού βρεθεί το μονοπάτι υπολογίζονται και αποθηκεύονται οι χρόνοι διαδρομής καθώς επίσης και οι predecessors του κάθε κόμβου.

Τέλος, γίνεται η αποθήκευση του δικτύου σε αρχείο με την μορφή που έχει ήδη περιγραφεί σε προηγούμενο κεφάλαιο, έτσι ώστε να αποτελέσει το αρχείο εισόδου του αλγόριθμου που εξετάζεται στην παρούσα διπλωματική εργασία.

4.3 Πειραματικά αποτελέσματα

Η υλοποίηση του αλγορίθμου δοκιμάστηκε σε τυχαία δίκτυα αποτελούμενα από 50, 100, 500, 1000, 1500 και 5000 κόμβους. Αυτά τα δίκτυα κατασκευάστηκαν με γεννήτρια που παρουσιάστηκε στην προηγούμενη ενότητα, με κάθε κόμβο να έχει από 2 έως 4 γειτονικούς κόμβους.

Για κάθε ομάδα πειραμάτων που πραγματοποιήθηκαν μεταβαλλόταν κάθε φορά ένα από τα χαρακτηριστικά του δικτύου που είναι το πλήθος των κόμβων, ο αριθμός των μέσων και το χρονικό παράθυρο αναζήτησης εφικτής διαδρομής. Όλα τα πειράματα εκτελέστηκαν σε φορητό Η/Υ με επεξεργαστή Intel(R) Core(TM)2 Duo CPU T9400 @ 2.53GHz, μνήμη DDR2 4GB, δίσκο SATA @ 5400RPM και λειτουργικό Windows 7 x64.

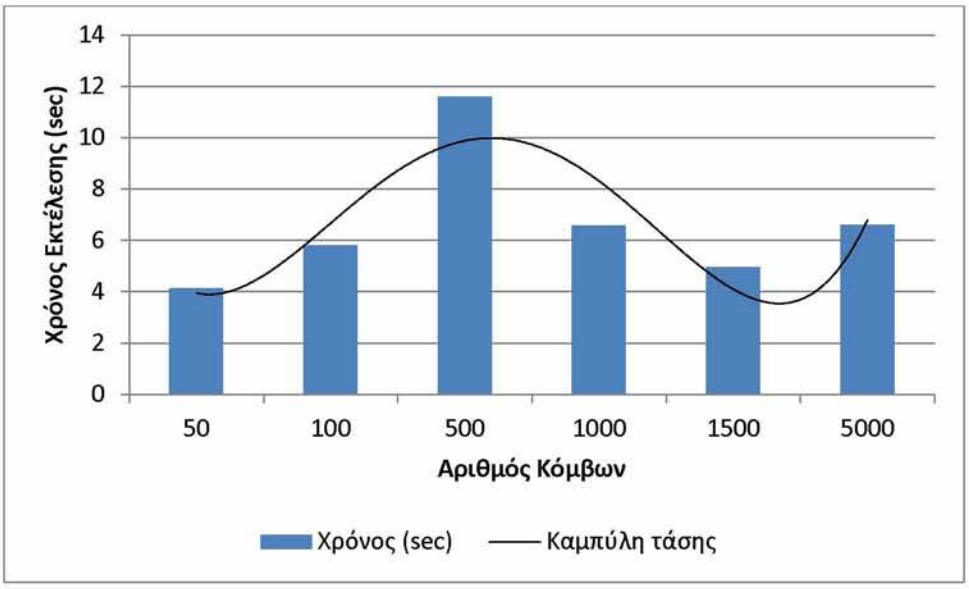
Ο υπολογιζόμενος χρόνος εκτέλεσης περιλαμβάνει το χρόνο ανάγνωσης των δεδομένων, το χρόνο αρχικοποίησης, το χρόνο δέσμευσης της απαιτούμενης μνήμης και το χρόνο εκτέλεσης. Για τον υπολογισμό των μέσων τιμών χρόνου και μνήμης της εκτέλεσης πραγματοποιήθηκαν τόσες επαναλήψεις όσοι και οι κόμβοι του εκάστοτε δικτύου μείον ένα αλλάζοντας τον κόμβο προορισμού.

Στον ακόλουθο πίνακα φαίνεται η πρώτη ομάδα δοκιμών όπου μεταβάλλεται μόνο ο αριθμός των κόμβων του δικτύου. Σε κάθε δίκτυο υπάρχουν τριάντα (30) μέσα μεταφορά και αναζητείται λύση για εκατό (100) χρονικές μονάδες.

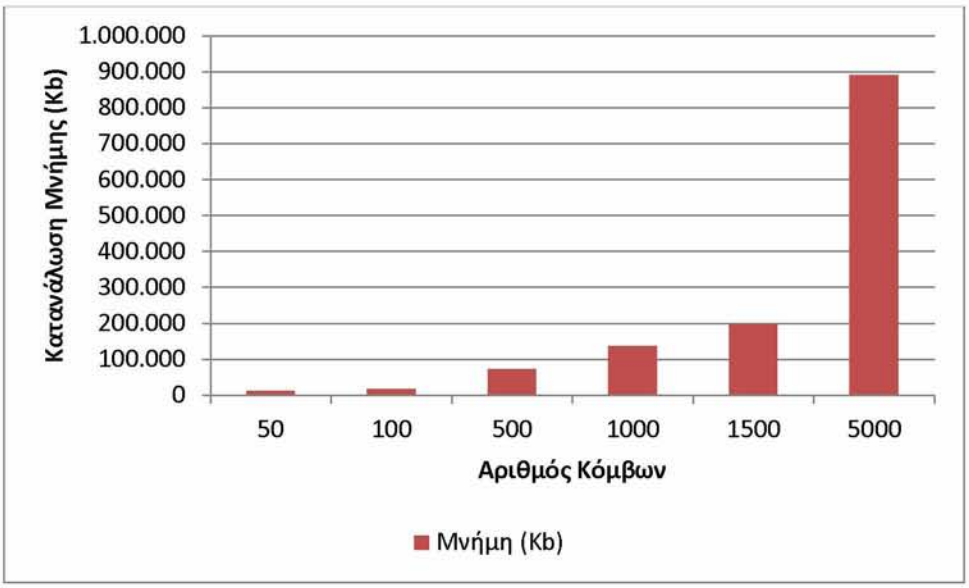
Κόμβοι	50	100	500	1.000	1.500	5.000
Μέσα Μεταφοράς	30					
Χρόνοι Διαδρομής	408	650	2.968	5.871	8.210	45.920
Κανόνες Μετεπιβίβασης	3.922	5.256	23.265	44.342	60.516	509.140
Χρονικές Μονάδες Αναζήτησης	100					
Μέσος Χρόνος Εκτέλεσης (sec)	4,118	5,804	11,607	6,584	4,961	6,604
Κατανάλωση Μνήμης (Kb)	11.224	18.084	72.156	137.292	197.236	891.308

Πίνακας 4.1 Απόδοση σε δίκτυα με αυξανόμενο πλήθος κόμβων

Η γραφική απεικόνιση των παραπάνω δεδομένων φαίνεται στα ακόλουθα δύο διαγράμματα. Στο πρώτο αποτυπώνεται ο απαιτούμενος χρόνος εκτέλεσης, ενώ στο δεύτερο η μέγιστη απαιτούμενη μνήμη σε σχέση με το πλήθος των κόμβων του δικτύου.



Εικόνα 4.2 Απαιτούμενος χρόνος εκτέλεσης σε σχέση με το πλήθος των κόμβων του δικτύου



Εικόνα 4.3 Απαιτούμενη μνήμη σε σχέση με το πλήθος των κόμβων του δικτύου

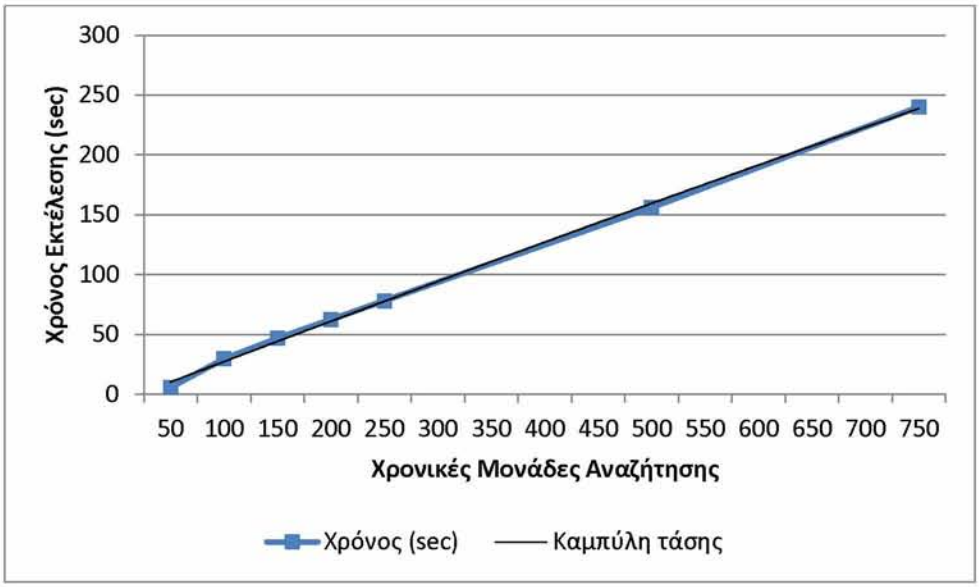
Όσον αφορά στο πρώτο διάγραμμα, παρατηρούμε ότι μέχρι τους 500 κόμβους, ο χρόνος εκτέλεσης αυξάνει εκθετικά, ενώ στην συνέχεια μειώνεται και αρχίζει πάλι να αυξάνει από τους 1500 και πάνω. Αυτή η συμπεριφορά μπορεί να εξηγηθεί λαμβάνοντας υπόψη ότι από τους 1000 κόμβους και πάνω, με δεδομένο ότι έχουμε μόνο 30 μέσα μεταφοράς, τα δίκτυα είναι πολύ αραιά με αποτέλεσμα να ελέγχονται λιγότερα μεταφορικά μέσα ανά κόμβο και να μπαίνουν λιγότεροι κόμβοι στην SE λίστα. Ταυτόχρονα, η αύξηση του χρόνου εκτέλεσης που παρατηρείται στους 1500 κόμβους και πάνω, οφείλεται στην αύξηση των δεδομένων εισόδου και συνεπώς στο χρόνο ανάγνωσης και δέσμευσης της μνήμης. Για αυτό το λόγο, σε αυτό το μέγεθος δικτύων, ο χρόνος δέσμευσης της απαιτούμενης μνήμης αποτελεί μεγάλο ποσοστό του χρόνου εκτέλεσης και συνεπώς τον επηρεάζει εμφανώς.

Για την δεύτερη ομάδα δοκιμών, η αναζήτηση έγινε σε δίκτυο με 500 κόμβους και 40 μεταφορικά μέσα, μεταβάλλοντας τις χρονικές μονάδες αναζήτησης. Τα αποτελέσματα αυτών των δοκιμών συνοψίζονται στον ακόλουθο πίνακα.

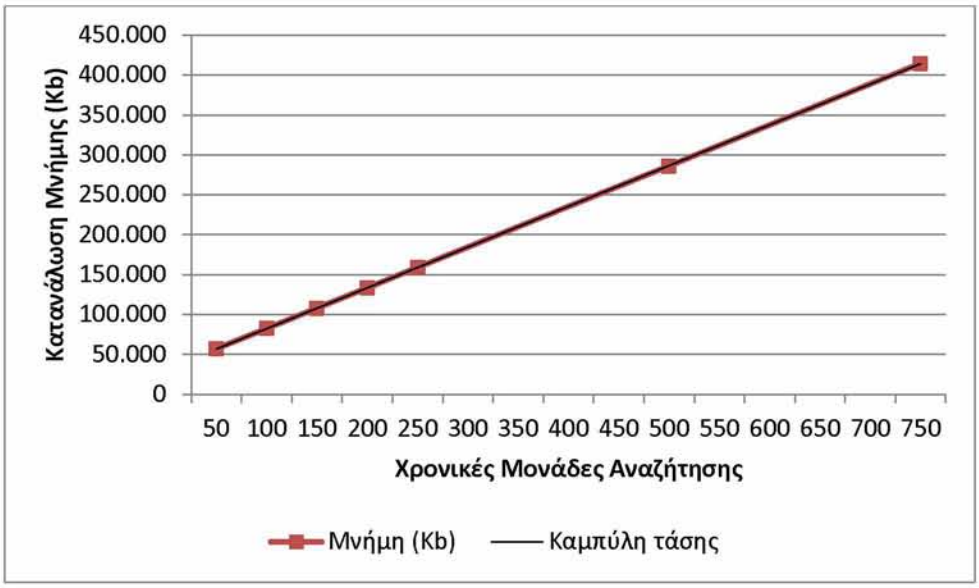
Κόμβοι	500						
Μέσα Μεταφοράς	40						
Χρόνοι Διαδρομής	3.056						
Κανόνες Μετεπιβίβασης	23.733						
Χρονικές Μονάδες Αναζήτησης	50	100	150	200	250	500	750
Μέσος Χρόνος Εκτέλεσης (sec)	5,569	29,396	46,795	62,229	78,062	155,859	293,897
Κατανάλωση Μνήμης (Kb)	56.992	82.244	107.696	133.184	158.716	286.048	414.472

Πίνακας 4.2 Απόδοση σε δίκτυα με αυξανόμενες χρονικές μονάδες αναζήτησης

Η γραφική απεικόνιση των παραπάνω δεδομένων φαίνεται στα ακόλουθα δύο διαγράμματα. Στο πρώτο αποτυπώνεται ο απαιτούμενος χρόνος εκτέλεσης, ενώ στο δεύτερο η μέγιστη απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης.



Εικόνα 4.4 Απαιτούμενος χρόνος εκτέλεσης σε σχέση με τις χρονικές μονάδες αναζήτησης



Εικόνα 4.5 Απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης

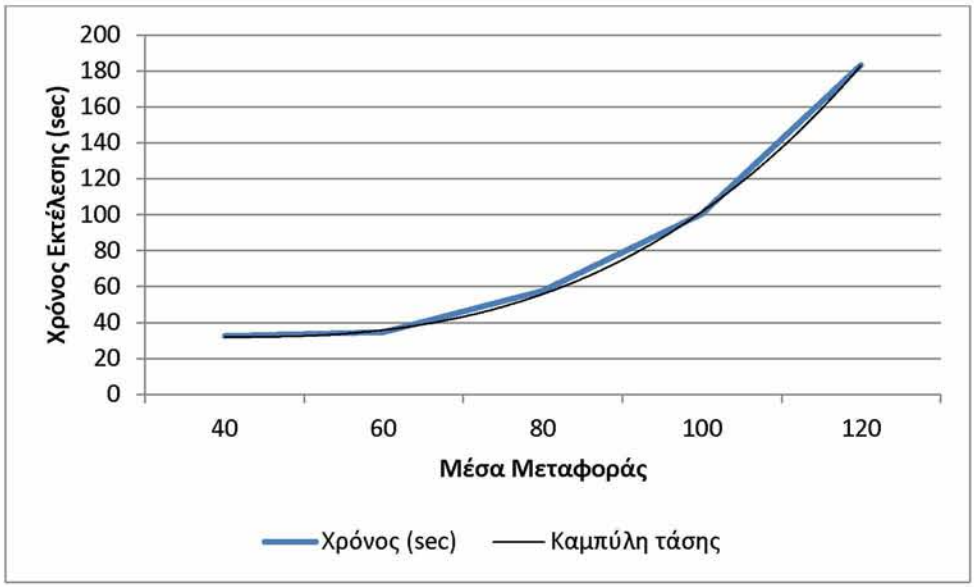
Παρατηρούμε ότι ο χρόνος εκτέλεσης παρουσιάζει αύξηση που προσεγγίζεται από πολυώνυμο $2^{ου}$ βαθμού (Ziliaskopoulos & Wardell, 2000), ενώ η μνήμη παρουσιάζει γραμμική αύξηση.

Τέλος, στην τρίτη ομάδα δοκιμών μελετάται η απόδοση της υλοποίησης όταν μεταβάλλεται ο αριθμός των μεταφορικών μέσων σε ένα δίκτυο αποτελούμενο από 500 κόμβους και γίνεται αναζήτηση για 100 χρονικές μονάδες.

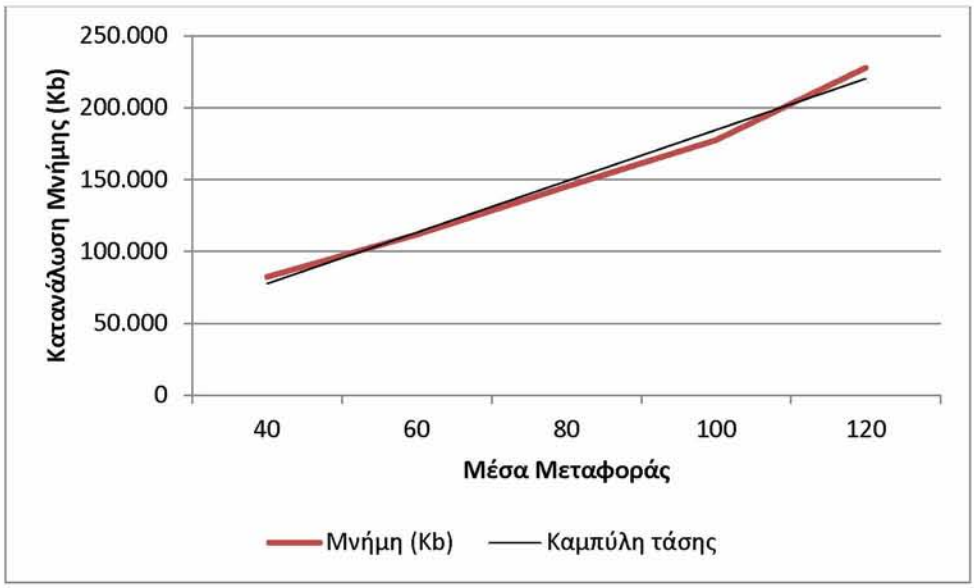
Κόμβοι	500				
Μέσα Μεταφοράς	40	60	80	100	120
Χρόνοι Διαδρομής	3.056	3.198	3.444	3.894	4.485
Κανόνες Μετεπιβίβασης	23.733	24.848	28.564	35.729	46.786
Χρονικές Μονάδες Αναζήτησης	100				
Μέσος Χρόνος Εκτέλεσης (sec)	32,396	34,559	57,627	100,308	183,254
Κατανάλωση Μνήμης (Kb)	82.244	111.592	145.200	177.424	227.716

Πίνακας 4.3 Απόδοση σε δίκτυα με αυξανόμενο πλήθος μεταφορικών μέσων

Η γραφική απεικόνιση των παραπάνω δεδομένων φαίνεται στα ακόλουθα δύο διαγράμματα. Στο πρώτο αποτυπώνεται ο απαιτούμενος χρόνος εκτέλεσης, ενώ στο δεύτερο η μέγιστη απαιτούμενη μνήμη σε σχέση με το πλήθος των μεταφορικών μέσων.



Εικόνα 4.6 Απαιτούμενος χρόνος εκτέλεσης σε σχέση με το πλήθος των μεταφορικών μέσων



Εικόνα 4.7 Απαιτούμενη μνήμη σε σχέση με το πλήθος των μεταφορικών μέσων

Παρατηρούμε ότι ο χρόνος εκτέλεσης παρουσιάζει αύξηση που προσεγγίζεται από πολυώνυμο 3^{ου} βαθμού (Ziliaskopoulos & Wardell, 2000), ενώ η μνήμη παρουσιάζει γραμμική αύξηση.

4.4 Το δίκτυο της ΤΡΑΙΝΟΣΕ

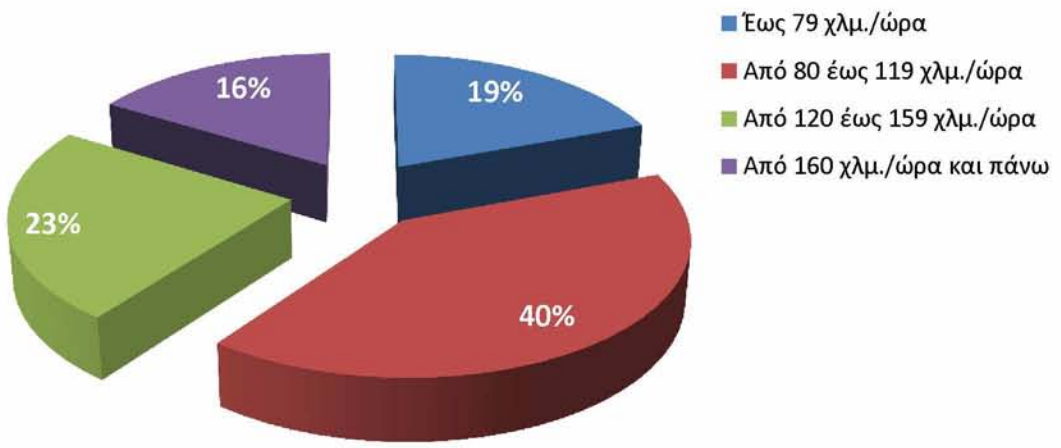
Ο ΟΣΕ ιδρύθηκε το 1970 και λειτουργεί ως όμιλος εταιρειών παροχής υπηρεσιών διαχείρισης και εκμετάλλευσης της Εθνικής Σιδηροδρομικής Υποδομής, εκτέλεσης των αναπτυξιακών έργων υποδομής και αξιοποίησης της ακίνητης περιουσίας.

Η ΤΡΑΙΝΟΣΕ Α.Ε. ιδρύθηκε το 2005, αρχικά ως θυγατρική της εταιρίας ΟΣΕ Α.Ε., με σκοπό την παροχή υπηρεσιών μεταφοράς επιβατών και εμπορευμάτων.

Σήμερα, η εταιρία λειτουργεί ως ανεξάρτητη από τον Όμιλο ΟΣΕ εταιρία του ελληνικού δημοσίου, ενώ προς το παρόν αποτελεί τη μοναδική εταιρία παροχής σιδηροδρομικών μεταφορών στη χώρα μας, λειτουργώντας προαστιακά, εθνικά και περιφερειακά δρομολόγια.

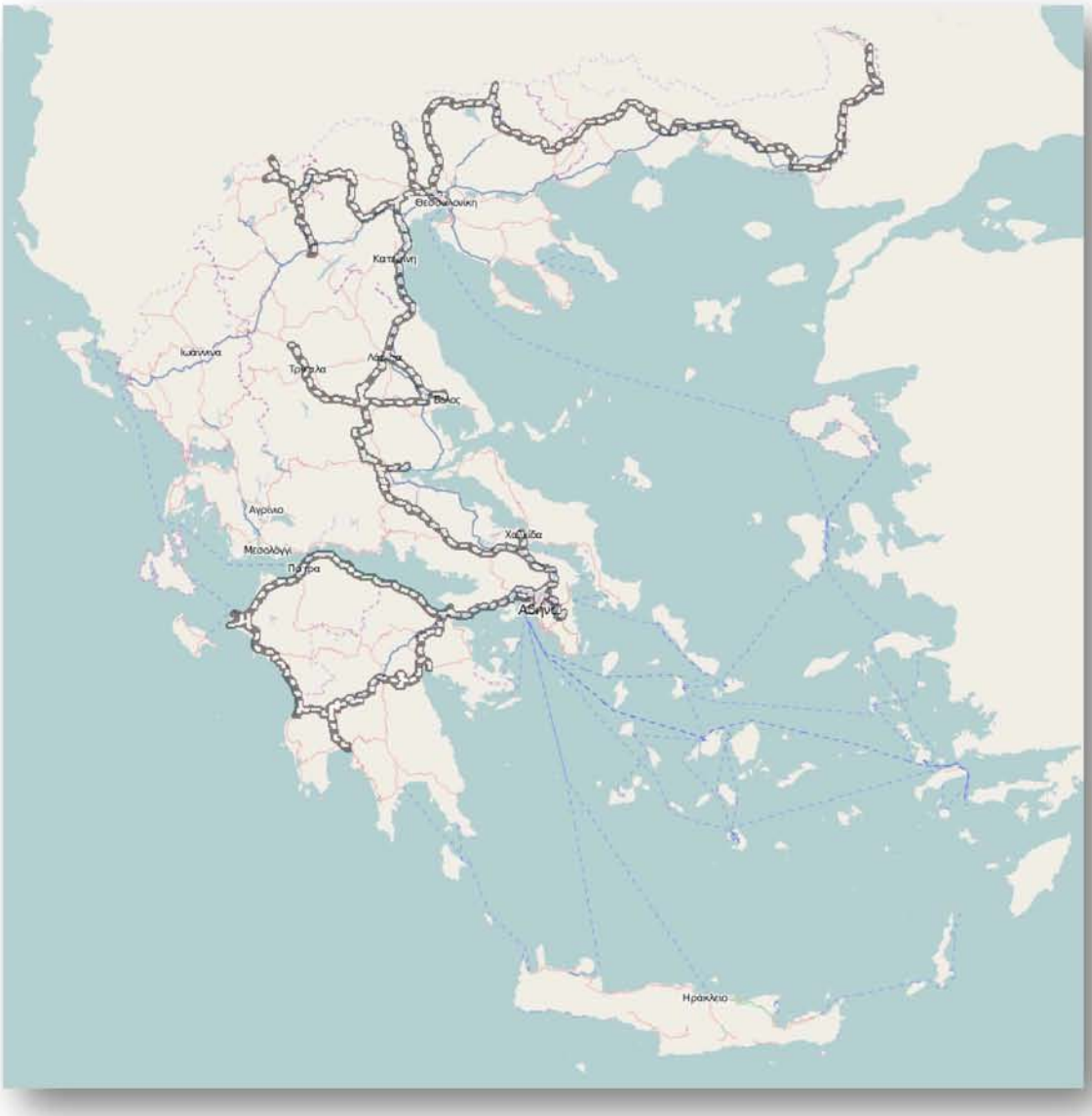
Πραγματοποιώντας πάνω από 500 δρομολόγια την ημέρα, τα τρένα της εταιρίας καλύπτουν ένα σιδηροδρομικό δίκτυο που ξεπερνά τα 2.500 χλμ. Ειδικότερα, το σημερινό δίκτυο ανέρχεται σε 2552 km (γραμμές σε εκμετάλλευση), εκ των οποίων το 70% αφορά γραμμή κανονικού εύρους (1435 mm), όπως είναι καθιερωμένο στην Ευρώπη, αλλά και διεθνώς.

Η ανώτατη ταχύτητα είναι σήμερα 160 km/h, η οποία εφαρμόζεται στο 18% του σιδηροδρομικού δικτύου.



Εικόνα 4.8 Ταξινόμηση γραμμών με βάση την ανωτάτη επιτρεπόμενη ταχύτητα

Μέσα από αυτό το δίκτυο, η ΤΡΑΙΝΟΣΕ Α.Ε. μεταφέρει 15 εκατομμύρια επιβάτες και 4,5 εκατομμύρια τόνους εμπορευμάτων σε ετήσια βάση.



Εικόνα 4.9 Σιδηροδρομικό δίκτυο ΤΡΑΙΝΟΣΕ

Στο δίκτυο της ΤΡΑΙΝΟΣΕ, το οποίο απεικονίζεται στην παραπάνω εικόνα, υπάρχουν πάνω από 300 κόμβοι – στάσεις και κινούνται πάνω του πάνω από 300 δρομολόγια σε καθημερινή βάση.

4.5 Εφαρμογής στο δίκτυο της ΤΡΑΙΝΟΣΕ

Για την μοντελοποίηση του δικτύου της ΤΡΑΙΝΟΣΕ συλλέχθηκαν όλα τα δρομολόγια που ήταν ενεργά τον Ιούνιο του 2010. Τα δεδομένα ήταν στην μορφή δρομολογίων αναφέροντας μονάχα τις στάσεις που κάνει κάθε αμαξοστοιχία καθώς και την ώρα της αναχώρησης από τον εκάστοτε σταθμό. Να σημειωθεί ότι το κάθε δρομολόγιο αποτελεί διαφορετική αμαξοστοιχία ακόμα και στην περίπτωση που το δρομολόγιο εκτελείται από τον ίδιο συρμό. Για παράδειγμα για την διαδρομή Βόλος – Λάρισα οι πρώτες δύο αμαξοστοιχίες είναι οι 1571 και 1573. Αυτές εκτελούνται από τον ίδιο συρμό, ο οποίος

ξεκινάει από τον Βόλο σαν 1571, επιστρέφει ως 1572 και επαναλαμβάνει το δρομολόγιο ως 1573.

Η μοντελοποίηση του δικτύου έγινε ορίζοντας κάθε σταθμό σαν κόμβο του δικτύου και κάθε αμαξοστοιχία ως μεταφορικό μέσο του δικτύου. Επομένως, οι κόμβοι του δικτύου συνδέθηκαν με βάση τις στάσεις των αμαξοστοιχιών και όχι με την τοπολογική σύνδεση τους. Για παράδειγμα η Λάρισα για την σιδηροδρομική γραμμή μεταξύ Λάρισας και Θεσσαλονίκης συνδέεται με την Ραψάνη, την Κατερίνη και την Θεσσαλονίκη καθώς υπάρχουν αμαξοστοιχίες, οι οποίες αφού φύγουν από την Λάρισα, κάνουν την απευθείας επόμενη στάση σε έναν από τους τρείς προαναφερθέντες σταθμούς. Τοπολογικά η Κατερίνη δεν είναι ο αμέσως επόμενος σταθμός από την Λάρισα αλλά υπάρχουν αμαξοστοιχίες που για πρώτη φορά, μετά την Λάρισα, κάνουν στάση εκεί. Αυτή η ιδιομορφία της μοντελοποίησης του δικτύου επιφέρει αύξηση στην πολυπλοκότητα του αλγόριθμου αναζήτησης μιας και αυτού του είδους οι απευθείας συνδέσεις, αναγκάζουν τον αλγόριθμο να εξετάσει τους συγκεκριμένους κόμβους περισσότερες φορές.

Με βάση την παραπάνω μοντελοποίηση του δικτύου προέκυψαν 345 κόμβοι, 5.001 συνδέσεις και 943.058 δυνατές μετεπιβιβάσεις, όπου κινούνται 373 μεταφορικά μέσα σε χρονικό διάστημα 2.192 λεπτών. Ειδικά για τα μεταφορικά μέσα, το 1° είναι τα πόδια και χρειάζεται να οριστεί για να υπολογιστεί ως μέσο εισόδου και εξόδου από το σύστημα. Για το χρονικό διάστημα, ενώ η μέρα έχει 1.440 λεπτά, παρατηρούμε ότι στην μοντελοποίηση υπάρχουν αμαξοστοιχίες για τις οποίες γίνεται αλλαγή ημέρας κατά την διάρκεια του ταξιδιού. Για παράδειγμα, η αμαξοστοιχία 360 που εκτελεί το δρομολόγιο Αθήνα – Σόφια, αναχωρεί από την Αθήνα στις 22:55 και φτάνει στην Σόφια στις 12:32 την επόμενη μέρα. Συνεπώς ο χρόνος άφιξης στην Σόφια είναι στο $24 * 60 + 12 * 60 + 32 = 2.192$ λεπτό από την έναρξη των δεδομένων εισόδου. Για λόγους πληρότητας, αναφέρεται ότι στο δίκτυο που κατασκευάστηκε οι κόμβοι έχουν τον ακόλουθο αριθμό κόμβων προέλευσης:

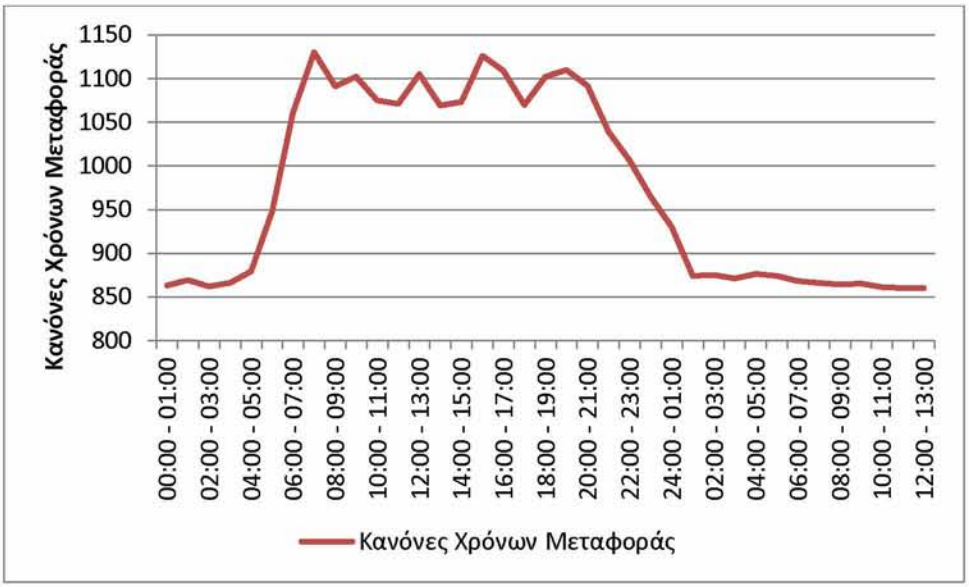
Αριθμός κόμβων προέλευσης	Αριθμός κόμβων με το συγκεκριμένο χαρακτηριστικό
1	27
2	224
3	45
4	24
5	13
6	3
7	6
8	2
9	0
10	0
11	0
12	0
13	1

Πίνακας 4.4 Αριθμός κόμβων προέλευσης στο δίκτυο της ΤΡΑΙΝΟΣΕ

Όπου τους περισσότερους σταθμούς τους έχει το Λιανοκλάδι, το οποίο έχει πιθανές επόμενες στάσεις στους σταθμούς Γοργοπόταμο, ΧΣ 2+850 Λαμία, Θήβα Βοιωτίας, Λάρισα, Τιθορέα, Αγγεία, Αθήνα, Μπράλο, Παλαιοφάρσαλο, Λειβαδιά, Αμφίκλεια, Λαμία και στάση Θαυμάκου.

Από τον πίνακα παρατηρούμε ότι ο μέσος αριθμός κόμβων προέλευσης ανά κόμβο είναι κοντά στο 2.49. Παρατηρούμε επίσης ότι 27 κόμβοι έχουν έναν μόνο κόμβο προέλευσης. Αυτοί οι κόμβοι είναι οι τερματικοί σταθμοί του δικτύου όπου μπορούμε να φτάσουμε σε αυτούς από ένα και μόνο σημείο και δεν μπορούμε να συνεχίσουμε την πορεία μας σε επόμενο κόμβο. Τέλος, παρατηρούμε ότι η πληθώρα των κόμβων έχει 2 κόμβους προέλευσης, που είναι το αναμενόμενο σε ένα δίκτυο σταθερής τροχιάς όπου τα μεταφορικά μέσα έχουν δύο κατευθύνσεις και συνεπώς φτάνουν σε κάθε κόμβο από τον προηγούμενο και τον επόμενο σταθμό, με αντίθετη κατεύθυνση.

Τέλος, θα πρέπει να σημειωθεί ότι το δίκτυο της ΤΡΑΙΝΟΣΕ παρουσιάζει αυξημένη κίνηση των αμαξοστοιχιών από τις 420 χρονικές μονάδες μέχρι τις 1320, δηλαδή από τις 7:00 το πρωί μέχρι τις 22:00 το βράδυ. Πέρα από αυτές τις ώρες, υπάρχουν μόνο ελάχιστες αμαξοστοιχίες και τα πόδια ως διαθέσιμα μεταφορικά μέσα. Στο ακόλουθο διάγραμμα απεικονίζεται το πλήθος των κανόνων χρόνων διαδρομής που είναι εφικτοί σε κάθε χρονικό διάστημα.



Εικόνα 4.10 Κανόνες χρόνων μεταφοράς στο δίκτυο της ΤΡΑΙΝΟΣΕ στο σύνολο των δεδομένων εισόδου

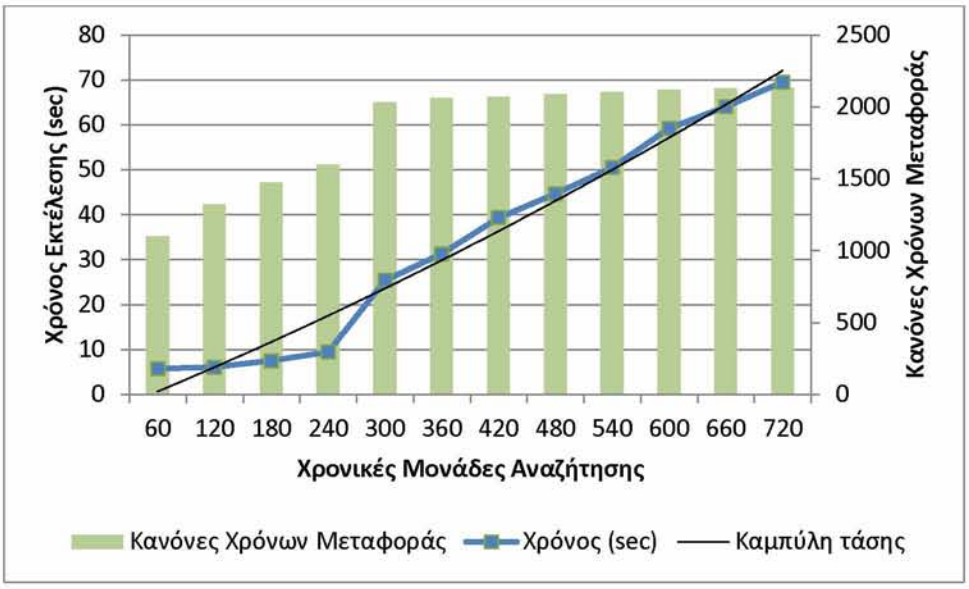
Για την αποτίμηση της απόδοσης του αλγόριθμου στο πραγματικό δίκτυο της ΤΡΑΙΝΟΣΕ εκτελέστηκαν πολλαπλές επαναλήψεις για πολλαπλούς χρόνους αναζήτησης. Τα πειράματα αυτά εκτελέστηκαν σε φορητό Η/Υ με επεξεργαστή Intel(R) Core(TM) i7 CPU Q820 @ 1.73GHz, μνήμη DDR3 8GB, δίσκο SSD και λειτουργικό Windows 7 x64. Η αλλαγή του Η/Υ ήταν αναγκαστική καθώς η απαίτηση σε μνήμη ορισμένων μετρήσεων ξεπερνούσε τα 4GB του συστήματος της ενότητας 4.3.

Στον ακόλουθο πίνακα φαίνεται η πρώτη ομάδα δοκιμών, όπου μεταβάλλονται οι χρονικές μονάδες αναζήτησης κατά 60 λεπτά. Να σημειωθεί ότι οι χρόνοι διαδρομής αυξάνονται όσο αυξάνεται ο χρόνος αναζήτησης. Αυτό συμβαίνει επειδή όσο αυξάνεται ο χρόνος αναζήτησης, τόσοι περισσότεροι διακριτοί κανόνες είναι εφικτοί και πρέπει να εξεταστούν. Επίσης, σημειώνεται ότι η αναζήτηση γίνεται μέσα στην ημέρα με στόχο σε κάθε αναζήτηση να περιέχεται η χρονική στιγμή 1140 (19:00 το απόγευμα) ως μεσαία τιμή.

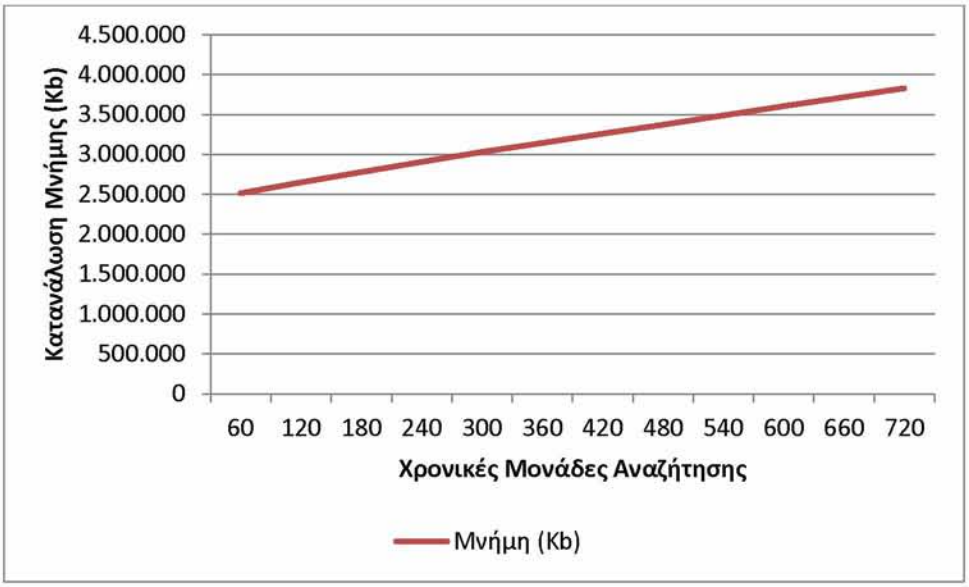
Κόμβοι	Μέσα Μεταφοράς	Χρόνοι Διαδρομής	Κανόνες Μετεπιβίβασης	Χρονικές Μονάδες Αναζήτησης	Μέσος Χρόνος Εκτέλεσης (sec)	Κατανάλωση Μνήμης (Kb)
345	373	1.097	943.058	60	5,663	2.511.544
345	373	1.320	943.058	120	6,022	2.649.896
345	373	1.473	943.058	180	7,473	2.777.352
345	373	1.599	943.058	240	9,376	2.903.172
345	373	2.032	943.058	300	25,319	3.027.736
345	373	2.064	943.058	360	31,284	3.143.380
345	373	2.074	943.058	420	39,312	3.257.232
345	373	2.091	943.058	480	44,679	3.371.440
345	373	2.102	943.058	540	50,497	3.485.080
345	373	2.121	943.058	600	59,233	3.601.320
345	373	2.129	943.058	660	64,038	3.714.676
345	373	2.135	943.058	720	69,544	3.827.696

Πίνακας 4.5 Απόδοση στο δίκτυο ΤΡΑΙΝΟΣΕ με αύξηση στις χρονικές μονάδες αναζήτησης κατά 60 λεπτά

Η γραφική απεικόνιση των παραπάνω δεδομένων φαίνεται στα ακόλουθα δύο διαγράμματα. Στο πρώτο αποτυπώνεται ο απαιτούμενος χρόνος εκτέλεσης σε συνδυασμό με το πλήθος των κανόνων των χρόνων διαδρομής που είναι ενεργοί, ενώ στο δεύτερο η μέγιστη απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης.



Εικόνα 4.11 Απαιτούμενος χρόνος εκτέλεσης και πλήθος ενεργών κανόνων σε σχέση με τις χρονικές μονάδες αναζήτησης (μεταβολή 60 χρονικές μονάδες)



Εικόνα 4.12 Απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης (μεταβολή 60 χρονικές μονάδες)

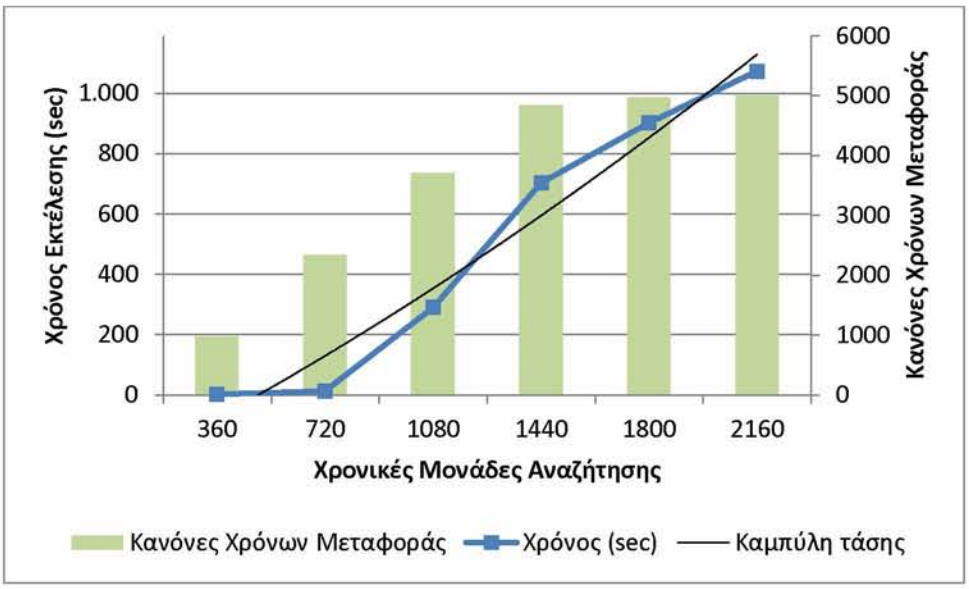
Όσον αφορά στο πρώτο διάγραμμα, παρατηρούμε ότι στις 300 χρονικές μονάδες αναζήτησης ενεργοποιούνται αισθητά περισσότεροι κανόνες από ό,τι στις προηγούμενες μετρήσεις με αποτέλεσμα να επηρεάζεται αντίστοιχα ο χρόνος εκτέλεσης. Αν εξαιρέσουμε αυτό το σημείο, η συμπεριφορά της υλοποίησης στο δίκτυο της ΤΡΑΙΝΟΣΕ είναι αντίστοιχη με αυτή των τυχαίων δικτύων που εξετάστηκαν στην ενότητα 4.3. Η μνήμη παρουσιάζει γραμμική αύξηση ως προς την αύξηση των χρονικών μονάδων αναζήτησης, όπως ήταν αναμενόμενο από τα προηγούμενα πειράματα.

Στην συνέχεια, πραγματοποιήθηκε μια δεύτερη ομάδα δοκιμών στο ίδιο δίκτυο, μεταβάλλοντας τις χρονικές μονάδες αναζήτησης κατά 360 λεπτά. Χρόνος έναρξης όλων των αναζητήσεων είναι τα μεσάνυχτα που ορίζουν την αρχή της ημέρας. Στόχος αυτών των πειραμάτων ήταν η μελέτη της συμπεριφοράς του αλγόριθμου σε μεγάλες τιμές του χρόνου αναζήτησης. Τα αποτελέσματα αυτών των δοκιμών συνοψίζονται στον ακόλουθο πίνακα.

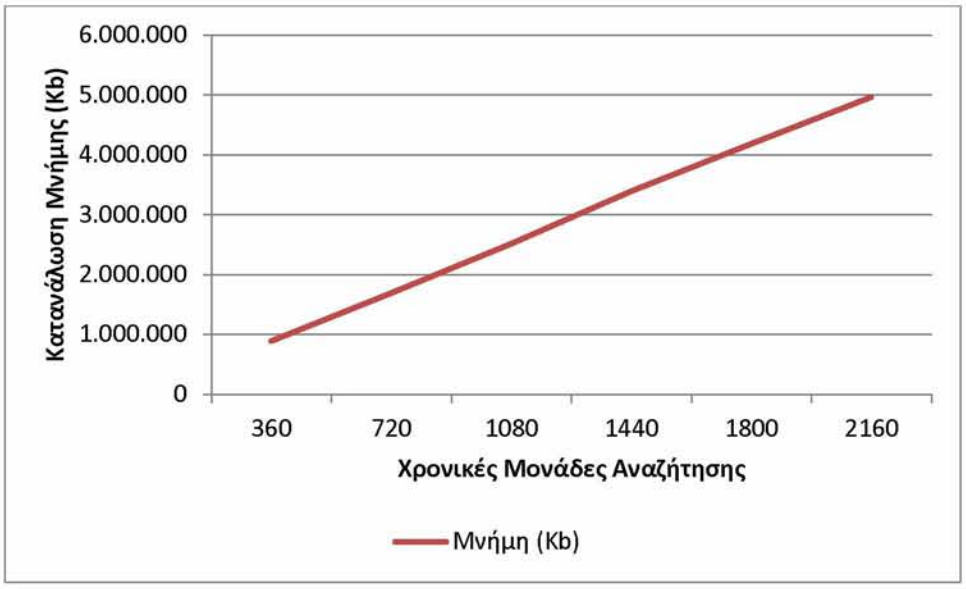
Κόμβοι	Μέσα Μεταφοράς	Χρόνοι Διαδρομής	Κανόνες Μετεπιβίβασης	Χρονικές Μονάδες Αναζήτησης	Μέσος Χρόνος Εκτέλεσης (sec)	Κατανάλωση Μνήμης (Kb)
345	373	986	943.058	360	1,480	889.520
345	373	2.335	943.058	720	12,262	1.692.552
345	373	3.705	943.058	1.080	289,614	2.514.608
345	373	4.841	943.058	1.440	702,432	3.390.180
345	373	4.976	943.058	1.800	901,962	4.184.100
345	373	5.001	943.058	2.160	1.072,392	4.962.928

Πίνακας 4.6 Απόδοση στο δίκτυο ΤΡΑΙΝΟΣΕ με αύξηση στις χρονικές μονάδες αναζήτησης κατά 360 λεπτά

Η γραφική απεικόνιση των παραπάνω δεδομένων φαίνεται στα ακόλουθα δύο διαγράμματα. Στο πρώτο αποτυπώνεται ο απαιτούμενος χρόνος εκτέλεσης σε συνδυασμό με το πλήθος των κανόνων των χρόνων διαδρομής που είναι ενεργοί, ενώ στο δεύτερο η μέγιστη απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης.



Εικόνα 4.13 Απαιτούμενος χρόνος εκτέλεσης και πλήθος ενεργών κανόνων σε σχέση με τις χρονικές μονάδες αναζήτησης (μεταβολή 360 χρονικές μονάδες)



Εικόνα 4.14 Απαιτούμενη μνήμη σε σχέση με τις χρονικές μονάδες αναζήτησης (μεταβολή 360 χρονικές μονάδες)

Από τις απαιτήσεις σε υπολογιστικό χρόνο, μπορούμε να παρατηρήσουμε ότι μέχρι τις 720 χρονικές στιγμές ο χρόνος είναι αρκετά μικρός (κάτω από 12 δευτερόλεπτα) σε σχέση με τις υπόλοιπες μετρήσεις. Αυτό συμβαίνει επειδή οι μετρήσεις ξεκινάνε από την αρχή της ημέρας με αποτέλεσμα να μην έχουν ξεκινήσει τα δρομολόγιά τους οι αμαξοστοιχίες της ΤΡΑΙΝΟΣΕ. Από την χρονική στιγμή 720, δηλαδή από τις 12:00 το μεσημέρι και μετά φαίνεται η αύξηση του πλήθους των κανόνων με αποτέλεσμα την ραγδαία αύξηση του απαιτούμενου χρόνου εκτέλεσης. Αντίστοιχα μετά τις 24:00, υπάρχουν ελάχιστα δρομολόγια ενεργά στο δίκτυο, με αποτέλεσμα ο ρυθμός αύξησης των κανόνων να ελαττώνεται και ταυτόχρονα να ελαττώνεται ο ρυθμός αύξησης της υπολογιστικής ισχύος.

Τέλος, οι απαιτήσεις σε μνήμη που απαιτούνται από την υλοποίηση φαίνεται να ακολουθούν τις κατανομές που παρουσιάστηκαν στην ενότητα 4.3, γεγονός το οποίο υποδηλώνει ότι ο αλγόριθμος έχει σταθερή συμπεριφορά και σε πραγματικά δίκτυα, όσον αφορά στην κατανάλωση μνήμης.

4.6 Συμπεράσματα

Με βάση τις παραπάνω πειραματικές διαδικασίες, ο υλοποιημένος αλγόριθμος φαίνεται να παρουσιάζει χαρακτηριστικά αντίστοιχα με αυτά που αναφέρονται από τους Ziliaskopoulos et al (Ziliaskopoulos & Wardell, 2000). Ειδικότερα, ο απαιτούμενος χρόνος εύρεσης της βέλτιστης λύσης παρουσιάζει πολυωνυμική αύξηση σε σχέση με:

- την αύξηση των χρονικών μονάδων αναζήτησης. Ο συγκεκριμένος ρυθμός αύξησης προσεγγίζεται από πολυώνυμο $2^{ου}$ βαθμού.
- την αύξηση των κόμβων του εξεταζόμενου δικτύου. Ο ρυθμός αύξησης που παρουσιάστηκε στα τυχαία δίκτυα, προσεγγίζεται από πολυώνυμο $5^{ου}$ βαθμού.

- την αύξηση των μέσων του εξεταζόμενου δικτύου. Το πολυώνυμο που προσεγγίζει αυτόν τον ρυθμό αύξησης είναι $3^{ου}$ βαθμού.

Ο αλγόριθμος παρουσιάζει γραμμική αύξηση σε απαιτήσεις μνήμης τόσο με την αύξηση των χρονικών μονάδων αναζήτησης, όσο και με την αύξηση των κόμβων και μέσων του εξεταζόμενου δικτύου.

Σχετικά με την γεννήτρια τυχαίων δικτύων, τα παραγόμενα δίκτυα παρουσιάζουν ομοιόμορφη κατανομή των μέσων μεταφοράς και των συνδέσεων των κόμβων, γεγονός που δεν συμβαίνει στα πραγματικά δίκτυα. Με βάση αυτήν την ομοιόμορφη κατανομή των δικτύων, ο αλγόριθμος παρουσιάζει σταθερή συμπεριφορά ανεξάρτητα από την χρονική περίοδο στην οποία αναζητείται λύση. Από την άλλη μεριά, το δίκτυο της ΤΡΑΙΝΟΣΕ παρουσιάζει αυξημένη κινητικότητα σε συγκεκριμένη χρονική περίοδο (από τις 7:00 το πρωί μέχρι τις 22:00 το βράδυ) με αποτέλεσμα ο αλγόριθμος να έχει διαφορετικές απαιτήσεις σε υπολογιστικό χρόνο όταν αναζητούμε λύση για τις ίδιες χρονικές μονάδες αναζήτησης με διαφορετικά χρονικά σημεία έναρξης.

Ειδικά για τις πειραματικές δοκιμές που εκτελέστηκαν στο δίκτυο της ΤΡΑΙΝΟΣΕ, η μέση αναζήτηση μιας διαδρομής στην διάρκεια μιας ημέρας (1440 χρονικές μονάδες αναζήτησης) εκτελείται σε 11,5 λεπτά. Ο χρόνος αυτός κρίνεται ικανοποιητικός, ειδικότερα λαμβάνοντας υπόψη ότι το δίκτυο στο οποίο εκτελείται η αναζήτηση δεν είναι βελτιστοποιημένο για αναζήτηση. Το δίκτυο θα μπορούσε να ομαδοποιεί τις αμαξοστοιχίες που εκτελούν τις ίδιες διαδρομές με στόχο την μείωση των μέσων μεταφοράς και των κανόνων χρόνων μεταφοράς, η οποία θα οδηγούσε σε επιτάχυνση της αναζήτησης. Παράλληλα, οι κόμβοι του δικτύου θα μπορούσαν να είχαν λιγότερους κόμβους προέλευσης, περιγράφοντας το δίκτυο με βάση την φυσική τους τοπολογία και ορίζοντας άπειρο κόστος μετεπιβίβασης στους σταθμούς όπου μια αμαξοστοιχία δεν κάνει στάση.

5

Προοπτικές και Συμπεράσματα

Κεφάλαιο 5: Προοπτικές και συμπεράσματα

5.1 Εισαγωγή

Η παρούσα διπλωματική έχει ως στόχο την μελέτη και εφαρμογή αλγορίθμου δυναμικής δρομολόγησης με πολλαπλά μέσα μεταφοράς. Για την επίτευξη του παραπάνω στόχου, η εργασία επικεντρώνεται σε ένα σύνολο διαδοχικών ενεργειών, όπως αναλυτικά παρουσιάζονται στα προηγούμενα κεφάλαια και αφορούν:

- Στην ανάλυση και μελέτη της διεθνούς επιστημονικής βιβλιογραφίας σε επιστημονικά πεδία, που αφορούν στη δυναμική δρομολόγηση. Με βάση αυτήν την έρευνα εντοπίστηκε και ερευνήθηκε με ιδιαίτερη ανάλυση ο αλγόριθμος TDILTP, ο οποίος αναγνωρίσθηκε ως κατάλληλος για τη μοντελοποίηση και επίλυση του προβλήματος της δρομολόγησης με πολλαπλά μέσα μεταφοράς.
- Στην ανάπτυξη υποστηρικτικού πληροφοριακού συστήματος για την αξιολόγηση του αλγόριθμου TDILTP.
- Στην αξιολόγηση του αλγόριθμου, μέσα από την εφαρμογή του σε ένα πλήθος από τυχαία παραγόμενα δίκτυα καθώς και την πιλοτική εφαρμογή του στο δίκτυο τρένων της ΤΡΑΙΝΟΣΕ.

Το παρόν κεφάλαιο παρουσιάζει τα γενικά συμπεράσματα που απορρέουν από τη διπλωματική καθώς και τις προοπτικές που διαφαίνονται για περαιτέρω ερευνητικές δραστηριότητες.

5.2 Συμπεράσματα

Μέσα από την βιβλιογραφική ανασκόπηση που πραγματοποιήθηκε στα πλαίσια της διπλωματικής εργασίας, η επιστημονική κοινότητα φαίνεται να στρέφει όλο και περισσότερο το ενδιαφέρον της στην ανάπτυξη σύνθετων αλγόριθμων δρομολόγησης και στην ανάπτυξη ευφυών συστημάτων μεταφορών. Ειδικά τις τελευταίες δύο δεκαετίες, μεγάλο πλήθος ερευνητών έχουν προτείνει τόσο μεθοδολογίες αποτύπωσης δικτύων όσο και αλγόριθμους εντοπισμού βέλτιστων διαδρομών μέσα σε αυτά. Ένας από τους αλγόριθμούς που ξεχωρίζει είναι ο time-dependent intermodal least-time path (Ziliaskopoulos & Wardell, 2000).

Ο αλγόριθμος TDILTP είναι απλός στην κατανόηση και ο κώδικας που τον υλοποιεί είναι αρκετά σύντομος. Ο χρόνος εκτέλεσης για τον εντοπισμό βέλτιστης διαδρομής αυξάνει πολυωνιμικά σε σχέση με:

- την αύξηση των χρονικών μονάδων αναζήτησης. Ο συγκεκριμένος ρυθμός αύξησης προσεγγίζεται από πολυώνυμο $2^{ου}$ βαθμού.
- την αύξηση των κόμβων του εξεταζόμενου δικτύου. Ο ρυθμός αύξησης που παρουσιάστηκε στα τυχαία δίκτυα, προσεγγίζεται από πολυώνυμο $5^{ου}$ βαθμού.
- την αύξηση των μέσων του εξεταζόμενου δικτύου. Το πολυώνυμο που προσεγγίζει αυτόν τον ρυθμό αύξησης είναι $3^{ου}$ βαθμού.

Ο αλγόριθμος παρουσιάζει γραμμική αύξηση σε απαιτήσεις μνήμης σε σχέση με:

- την αύξηση των χρονικών μονάδων αναζήτησης
- την αύξηση των κόμβων και
- την αύξηση μέσων του εξεταζόμενου δικτύου.

Οι χρόνοι εντοπισμού βέλτιστης λύσης κυμαίνονται από 100 millisecond σε μικρά προβλήματα ή πολύ αραιά δίκτυα έως μερικά λεπτά σε μεγάλα πραγματικά δίκτυα όπως αυτό της ΤΡΑΙΝΟΣΕ.

Στην πειραματική διαδικασία που έγινε στο δίκτυο της ΤΡΑΙΝΟΣΕ, ο χρόνος έναρξης και λήξης των κανόνων ορίστηκε σε λεπτά από την αρχής της ημέρας. Αυτό δεν είναι περιοριστικό και θα μπορούσαν να είχαν οριστεί όλα τα δεδομένα σε δευτερόλεπτα από την αρχή της ημέρας για μεγαλύτερη ακρίβεια. Βέβαια, η συλλογή των δεδομένων με ακρίβεια δευτερολέπτου είναι μια πολύ επίπονη διαδικασία και κατ' επέκταση είναι αδύνατη όταν εμπλέκονται μέσα, τα οποία δεν τηρούν αυτή την ακρίβεια. Σε γενικές γραμμές, η μονάδα που πρέπει να επιλεγεί είναι η μικρότερη δυνατή για την οποία υπάρχουν ακριβή στοιχεία. Παραδείγματος χάρη, αν συμπεριληφθούν δρομολόγια λεωφορείων ή τρένων τα οποία φτάνουν στους προορισμούς τους με απόκλιση δεκαλέπτου, η μονάδα που θα πρέπει να επιλεχθεί είναι τα δεκάλεπτα από την αρχή της ημέρας.

Παράλληλα, ο αλγόριθμος υπολογίζοντας την δομή $timepath$, αποθηκεύει τις βέλτιστες διαδρομές από όλους τους κόμβους προς τον κόμβο προορισμού. Συνεπώς, εάν αποθηκευτεί αυτή η δομή σε ένα σύστημα βάσης δεδομένων και επαναληφθεί η διαδικασία για κάθε κόμβο του δικτύου, η βάση δεδομένων θα περιέχει όλες τις βέλτιστες διαδρομές για οποιοδήποτε ζεύγος κόμβων εισόδου/εξόδου. Με αυτόν τον τρόπο, όταν ένα μεγάλο δίκτυο παραμένει σταθερό ως προς την δομή του, η αναζήτηση βέλτιστων διαδρομών μπορεί να γίνει ταχύτατα κάνοντας μια απλή αναζήτηση στην βάση δεδομένων.

5.3 Προοπτικές

Μια πιθανή εξέλιξη του αλγόριθμου θα ήταν η εισαγωγή περαιτέρω έξυπνων ελέγχων, οι οποίοι θα γλιτώνουν μερικές επαναλήψεις με στόχο την μείωση του υπολογιστικού χρόνου που απαιτείται για τον εντοπισμό των βέλτιστων λύσεων. Για παράδειγμα, στον αλγόριθμο που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής αντί να εξετάζονται όλα τα διαθέσιμα μέσα σε όλους τους κόμβους, εξετάζονται μόνο τα μέσα εκείνα που βρίσκονται τόσο στον κόμβο που εξετάζεται όσο και στον επιλεγμένο κόμβο προορισμού.

Στο ίδιο πλαίσιο, θα μπορούσε να σχεδιαστεί μια διαδικασία προετοιμασίας των δεδομένων πριν την εκτέλεση του αλγόριθμου με στόχο την ελάττωση του χρόνου αναζήτησης. Σε αυτή την διαδικασία θα επιλεγόντουσαν μόνο τα δρομολόγια που είναι εφικτά μέσα στα χρονικά όρια της αναζήτησης καθώς και μόνο τα μέσα που πιθανόν να συμβάλουν στην εύρεση της βέλτιστης λύσης. Συνεπώς θα αποκλείονταν περιττοί κόμβοι και μεταφορικά μέσα και θα ελαττωνόταν το πεδίο αναζήτησης.

Επιπρόσθετα, ο αλγόριθμος θα μπορούσε να εμπλουτιστεί αξιοποιώντας πορίσματα από το επιστημονικό πεδίο της Πολυκριτηριακής Ανάλυσης. Στην τρέχουσα μορφή του, ο αλγόριθμος υπολογίζει την βέλτιστη διαδρομή με βάση ένα και μόνο κόστος το οποίο είναι ο χρόνος που απαιτείται για την διάσχιση της εκάστοτε εξεταζόμενης διαδρομής. Στην πραγματικότητα, ο ταξιδιώτης προσπαθεί να ελαχιστοποιήσει τόσο τον χρόνο όσο και τα έξοδα που απαιτούνται για να φτάσει στον προορισμό του, ενώ παράλληλα μπορεί να προσπαθεί να μειώσει και την εκπομπή ρύπων (carbon footprint) που απαιτεί η κίνηση του. Προς αυτή την κατεύθυνση, θα πρέπει οι ετικέτες των κόμβων να υπολογίζονται με μια πολυκριτηριακή συνάρτηση η οποία να λαμβάνει υπόψη της όλες τις παραπάνω παραμέτρους και να τις συνυπολογίζει με κάποια βάρη.

Τέλος, όσον αφορά στον αλγόριθμο, θα ήταν χρήσιμη η επέκτασή του για να υποστηρίζει ασαφείς χρόνους μεταφοράς και αναμονής. Στην παρούσα μορφή του, ο αλγόριθμος υπολογίζει τις ετικέτες με βάση μια σαφώς ορισμένη τιμή. Στην πράξη όμως, όταν το πρόγραμμα των λεωφορείων αναφέρει μια συγκεκριμένη ώρα αναχώρησης, αυτό σπάνια τηρείται, ειδικά στην περίπτωση των ενδιάμεσων κόμβων. Παράλληλα, ο χρόνος που απαιτείται ανάμεσα σε δύο κόμβους μπορεί να τροποποιηθεί από μια πληθώρα εξωτερικών παραγόντων που είναι πρακτικά αδύνατο να μοντελοποιηθούν. Με την χρήση ασαφών χρόνων, δηλαδή ορίζοντας τον ελάχιστο και τον μέγιστο χρόνο που μπορεί να απαιτηθεί για να γίνει η μεταφορά από έναν κόμβο σε έναν άλλο και προσδιορίζοντας την πιθανότητα που έχει κάθε τιμή (συνήθως με την επιλογή κάποιας κατανομής), ο

αλγόριθμος θα μπορούσε να εντοπίζει την πιθανή βέλτιστη διαδρομή μέσα σε ένα περιβάλλον όπου υπάρχει μεγάλη αβεβαιότητα.

Όσον αφορά στην υλοποιημένη εφαρμογή, θα μπορούσε να τροποποιηθεί κατάλληλα για να προσφέρει ακόμα περισσότερες επιλογές και διευκολύνσεις. Ειδικά για το αρχείο εισόδου της εφαρμογής, οι κανόνες θα μπορούσαν να υποστηρίζουν κάποιο σύμβολο (πχ *) για να μπορεί ο χρήστης να περιγράφει δυνατές μετεπιβιβάσεις ανεξαρτήτως των κόμβων. Για παράδειγμα θα μπορούσε ο χρήστης να ορίσει ότι για όλους τους κόμβους ο χρόνος αποβίβασης (δηλαδή στην επιβίβαση στο μεταφορικό μέσο με τα πόδια με αύξοντα αριθμό 1) από το τρένο (μεταφορικό μέσο με αύξων αριθμό 2) είναι 1 λεπτό απλά δίνοντας τον ακόλουθο κανόνα:

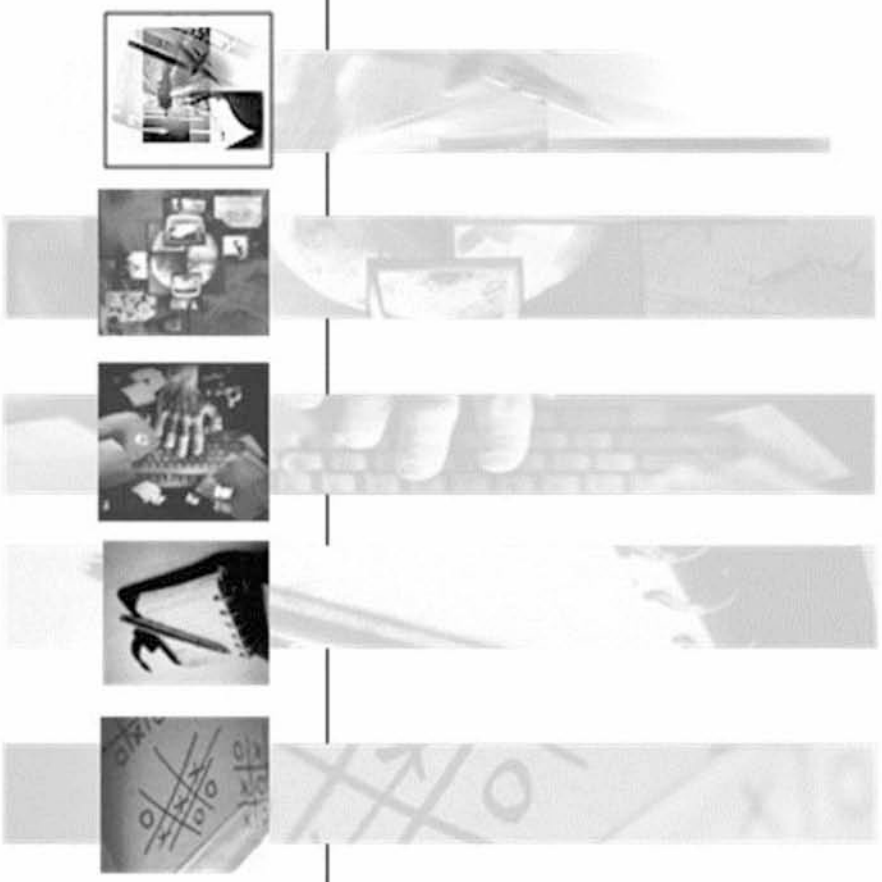
Προερχόμενος από κόμβο	Από μεταφορικό μέσο	Όταν είναι στον κόμβο	Μετεπιβίβαση σε μεταφορικό μέσο	Με προορισμό τον κόμβο	Χρονική ισχύς κανόνα		Χρόνος μετεπιβίβασης
					Αρχή	Λήξη	
*	2	*	1	*	0	1440	1

Εικόνα 5.1 Παράδειγμα χρήσης γενικών κανόνων μετεπιβίβασης

Εάν οι συγκεκριμένοι κανόνες ήταν εφικτοί, τότε ο χρήστης θα μπορούσε να ορίσει και εξαιρέσεις στον κανόνα, δίνοντας μια τεράστια τιμή στον χρόνο μετεπιβίβασης (πχ 99999).

Θα μπορούσε επίσης να τροποποιηθεί η εφαρμογή για να επιτρέπει στον χρήστη να επιλέξει το μέσο με το οποίο θα φτάσει στον προορισμό του ή και το μέσο με το οποίο ξεκινάει. Η συγκεκριμένη υλοποίηση που παρουσιάστηκε στην παρούσα εργασία, θεωρούσε ότι ο χρήστης θα φτάσει και θα ξεκινήσει με τα πόδια (άσχετα αν τελικά γίνει μετεπιβίβαση από κάποιο άλλο μέσο) μιας και αυτό απλοποιεί τις παραμέτρους που πρέπει να δώσει ο χρήστης για να πάρει τα αποτελέσματα που ζητάει. Μάλιστα, επειδή είναι υποχρεωτικό ο χρήστης να ξεκινήσει και να φτάσει με τα πόδια, ο αλγόριθμος συνυπολογίζει τον χρόνο που χρειάζεται για να μπει για παράδειγμα στο αυτοκίνητο και να βρει θέση παρκινγκ στον τελικό προορισμό του, αν η βέλτιστη λύση είναι να πάει με το αυτοκίνητο. Από την άλλη μεριά, αν ο αλγόριθμος τροποποιηθεί κατάλληλα για να επιτρέπει στον χρήστη να επιλέξει το μέσο με το οποίο ξεκινάει, θα μπορούσε να παραβλέψει τον χρόνο επιβίβασης στο αυτοκίνητο αν ο χρήστης είναι ήδη μέσα στο αυτοκίνητο την στιγμή που ζητάει την βέλτιστη διαδρομή.

Τέλος, μια δυνατή επέκταση της εφαρμογής είναι και η δημιουργία ενός γραφικού περιβάλλοντος στο οποίο ο χρήστης θα μπορούσε να μοντελοποιήσει το δίκτυό του καθώς και να εκτελέσει τον αλγόριθμο. Αυτό θα βελτίωνε την εμπειρία του χρήστη, ειδικά στο στάδιο της σύνταξη των κανόνων μετεπιβίβασης και χρόνου διαδρομής και θα ήταν δυνατό να αποφευχθούν πιθανές παραλήψεις του χρήστη στο στάδιο της μοντελοποίησης.



Βιβλιογραφία

Βιβλιογραφία

National Commission on Intermodal Transportation. (1994). *Toward a National Intermodal Transportation System, Final Report to Congress.* Washington, DC: US DOT.

Depth-first search. (2011, May 14). Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Depth-first_search

Aho, A., Hopcroft, J., & Ullman, J. (1983). *Data Structures and Algorithms.* Massachusetts: Addison-Wesley, Reading.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows : theory, algorithms, and applications / Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin.* Prentice Hall, Englewood Cliffs, N.J. :.

Bellman, R. (1958). On a Routing Problem. *Quart. Appl. Math. 16*, 87-90.

Clarke, G., & Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *OPERATIONS RESEARCH, 12*, 568-581.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms.* The MIT Press.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *MANAGEMENT SCIENCE, 6*, 80-91.

Dial, R. B., Glover, F., Karney, D., & Klingman, D. (1979). A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*.

Dial, R. B., Rutherford, G., & Quillian, L. (1979). *Transit Network Analysis: INET.* Springfield: University of Michigan Library.

Diestel, R. (2005). *Graph Theory, Electronic version of the 3rd edition.* Springer- Verlag Heidelberg book.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271.

Florian, M. (1977). A Traffic Equilibrium Model of Travel by Car and Public Transit Modes. *Transportation Science, 11*, 166-179.

Floyd, R. W. (1962, June). Algorithm 97: Shortest path. *Commun. ACM, 5*(6), 345--.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 100–107.

Knuth, D. E. (1997). *The Art of Computer Programming* (3rd εκδ., Τόμ. 1). Boston: Addison-Wesley.

Moore, E. F. (1959). The shortest path through a maze. *International Symposium on the Theory of Switching* (σσ. 285–292). Cambridge, Massachusetts: Cambridge: Harvard University Press.

Pallotino, S. (1984). Shortest-Path Methods: Complexity, Interrelations and New Propositions. *Networks 14*, 257-267.

Prentice, D. J. (1998). *COMPARISON OF FORTRAN AND C.* Ανάκτηση από USER NOTES ON FORTRAN PROGRAMMING (UNFP): http://www.ibiblio.org/pub/languages/fortran/ch1-2.html

Psaraftis, H. N. (1995). Dynamic vehicle routing: Status and prospects. *Annals of Operations Research Volume 61, Number 1*, 143-164.

Rodrigue, J.-P., Comtois, C., & Slack, B. (2006). *The Geography of Transport Systems.* London: Routledge, 296 pages.

Roy, B. (1959). Transitivité et connexité. *C. R. Acad. Sci. Paris 249*, 216-218.

Spiess, H., & Florian, M. (1989). Optimal Strategies: A New Assignment Model for Transit Networks. *Transportation Research Part B, 23B*, 83-102.

Tarjan, R. E. (1972). Depth-first search and linear graph algorithms. *sicomp, 1*, 146--160.

Warshall, S. (1962, January). A Theorem on Boolean Matrices. *J. ACM, 9*(1), 11--12.

West, D. B. (1996). *Introduction to Graph Theory - Second edition.* London: Prentice Hall.

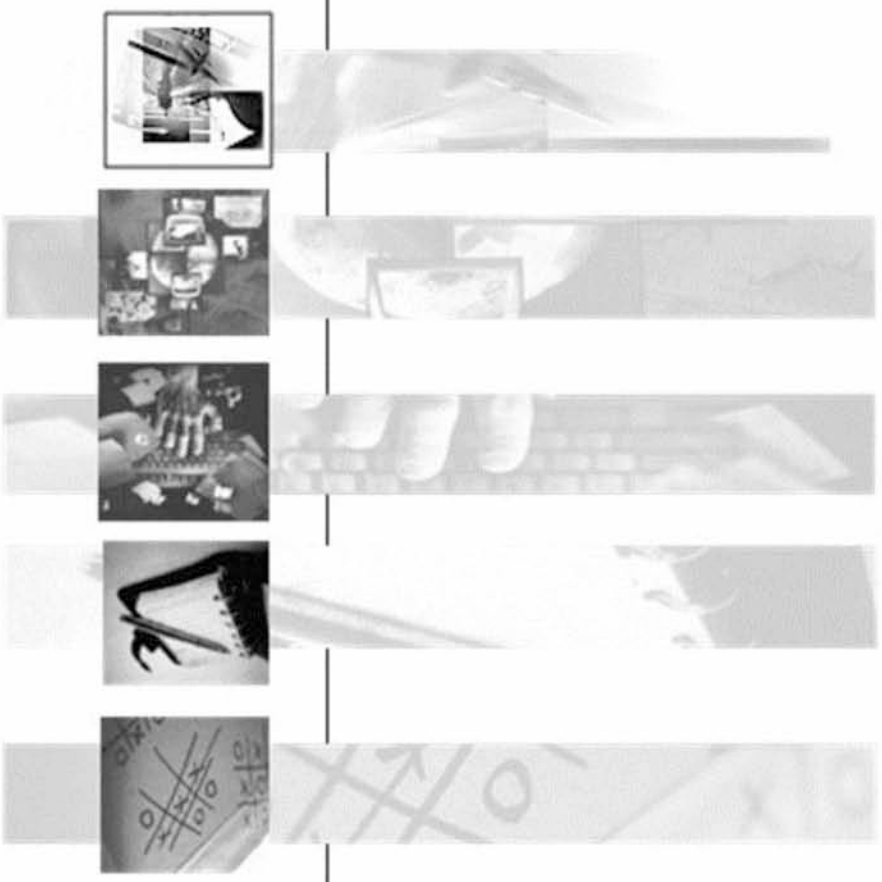
Zhan, F. B. (1998, 11 5). *Representing Networks*. Ανάκτηση από NCGIA Core Curriculum in Geographic Information Science: http://www.ncgia.ucsb.edu/giscc/units/u064/u064.html

Zhan, F. B. (2001). Three Fastest Shortest Path Algorithms on Real Road Networks : Data Structures and Procedures. *Journal of Geographic Information and Decision Analysis, vol.1, no.1*, 69-82.

Ziliaskopoulos, A. K., & Wardell, W. (2000). An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research, 125*, 486-502.

Ζάχος, Σ., & Κοζύρης, Ν. (2006). *Εισαγωγή στην επιστήμη των υπολογιστών: Θεωρητική Πληροφορική, Γλώσσες προγραμματισμού, Οργάνωση Υπολογιστών.* Αθήνα: ΕΜΠ.

Μισυρλής, Ν. (2002). Αφηρημένοι τύποι δεδομένων. Στο *Δομές δεδομένων με C.*



Παράρτημα

Παράρτημα

Αρχείο Εισόδου

Στο ακόλουθο αρχείο εισόδου έχουν μπει σχόλια (σημειωμένα με !) για την καλύτερη επεξήγηση του αρχείου. Το δίκτυο που περιγράφεται είναι αυτό που επιλύθηκε αναλυτικά στην ενότητα 3.3.

```
   4 !number of nodes
------
   1 !node
   1 !number of predecessors
   3 !predecessor #1
------
   2
   1
   1
------
   3 !node
   3 !number of predecessors
   1 !predecessor #1
   2 !predecessor #2
   4 !predecessor #3
------
   4
   2
   2
   3
------------
   9 !number of travel time rules
   1   2   1   0 1440   1
   1   3   1   0 1440   5
   1   3   2   2      2   2
   2   3   1   0 1440   2
   2   4   1   0 1440   7
   3   1   1   0 1440   5
   3   4   1   0 1440   5
   3   4   2   5      5   2
   4   3   1   0 1440   5
```

```
------------
 23 !number of switch delay rules
   1   1   2   1   3   0 1440   0
   1   1   2   1   4   0 1440   0
   1   1   3   1   1   0 1440   0
   1   1   3   1   4   0 1440   0
   1   1   3   2   4   0 1440   1
   1   2   3   1   1   0 1440   1
   1   2   3   1   4   0 1440   1
   1   2   3   2   4   0 1440   0
   2   1   3   1   1   0 1440   0
   2   1   3   1   4   0 1440   0
   2   1   3   2   4   0 1440   1
   2   1   4   1   3   0 1440   0
   2   1   4   1   4   0 1440   0
   3   1   1   1   2   0 1440   0
   3   1   1   1   3   0 1440   0
   3   1   1   2   3   0 1440   1
   3   1   4   1   3   0 1440   0
   3   1   4   1   4   0 1440   0
   3   2   4   1   3   0 1440   1
   3   2   4   1   4   0 1440   1
   4   1   3   1   1   0 1440   0
   4   1   3   1   4   0 1440   0
   4   1   3   2   4   0 1440   1
```

Πηγαίος Κώδικας

Κώδικας FORTRAN

Αρχείο TDILTP.f90

```
Program TDILTP
	Use ArrayMethods
	Use HashTable
	Implicit none

	Type Node
		Integer, Allocatable :: predecessors(:)
		Integer, Allocatable :: availableModes(:)
		Integer, Allocatable :: timeTables(:,:)
		Integer :: timeTablesCount = 0
		Integer, Allocatable :: labels(:,:,:)
		type(DICT_STRUCT), pointer :: hashData
		Integer, Allocatable :: timepath(:,:,:,:)
		Integer :: deque = 0
	EndType

	Integer :: i, j, x, y, t, k, label, iIndex, kIndex, xIndex, yIndex
	Integer :: maxNodes, fromNode, toNode, predecessors, predecessor
	Integer :: travelTimes, fromTime, toTime, travelTime, maxTime
	Integer :: realTravelTimes, realSwitchDelays
	Integer :: switchDelays, switchDelay, fromMode, atNode, toMode, delayTime
	Integer :: mode, maxModes=0
	Integer :: searchFromNode, searchFromMode, searchFromTime, searchToNode, searchToMode
	Integer :: next_i, next_t, next_x, start_t
	Real :: timeS, timeF
	Type(Node), Allocatable :: nodes(:)
	Integer, Allocatable :: scanEligibles(:)
	Integer :: firstEligible = -1, lastEligible = -1
	Logical :: isDebugLoaded, isNetworkLoaded
	Integer :: debugLvl = 0
	Type(DICT_DATA) :: hashDataTemp
	Integer :: hashKey(4)
```

```
        ! preset variables
        searchFromNode = 1
        searchFromMode = 1
        searchFromTime = 1
        maxTime = 1440
        searchToNode = 4
        searchToMode = 1
        isDebugLoaded = .False.
        isNetworkLoaded = .False.

        ! Read command line arguments and overide presets
        Call commandLine()

        !open default files if not set on cmd line
        If (.not.isNetworkLoaded) Open(10, file='network.txt', action='read')
        If (.not.isDebugLoaded) Open(11, file='debug.txt', action='write')

        ! Start timing the process
        Call cpu_time(timeS)

        ! Start reading
        Read(10,901) maxNodes
        Allocate(nodes(maxNodes))

        ! Set a dummy hashkey for hash table init
        hashKey(1) = 0
        hashKey(4) = 0
        hashKey(3) = 0
        hashKey(2) = 0

        ! Read nodes and predecessors
        Do i=1, maxNodes
                !initialize hash table for node
                call dict_create( nodes(i)%hashData, hashKey, hashDataTemp )

                Read(10,*)
                Read(10,901) toNode
                Read(10,901) predecessors
                Do j=1, predecessors
                        Read(10,901) fromNode
                        If (.not.inArray(nodes(toNode)%predecessors, fromNode)) Call ↙
push(nodes(toNode)%predecessors, fromNode)
                EndDo
                Call push(nodes(toNode)%predecessors, toNode)

        EndDo
        Write(*,*) 'Nodes:',maxNodes

        ! Skip section line
        Read(10,*)

        ! Read travel time rules
        Read(10,901) travelTimes
        Do i=1, travelTimes
                Read(10,906) fromNode, toNode, mode, fromTime, toTime, travelTime
                If ((toTime<searchFromTime .or. fromTime>maxTime)) Cycle
                realTravelTimes = realTravelTimes + 1
                If (mode > maxModes) maxModes = mode
                Call addTimeTable(fromNode, toNode, mode, fromTime, toTime, travelTime)
                If (.not.inArray(nodes(fromNode)%availableModes, mode)) Call ↙
push(nodes(fromNode)%availableModes, mode)
                If (.not.inArray(nodes(toNode)%availableModes, mode)) Call ↙
push(nodes(toNode)%availableModes, mode)
        EndDo

        Write(*,*) 'Modes:',maxModes
        Write(*,*) 'Time Intervals:',maxTime - searchFromTime
        Write(*,*) 'Travel Times:',travelTimes," needed:", realTravelTimes

        ! Skip section line
        Read(10,*)
```

```
        !Read switch delay rules
        Read(10,*) switchDelays
        Do i=1, switchDelays
                Read(10,908) fromNode, fromMode, atNode, toMode, toNode, fromTime, toTime, delayTime
                If ((toTime<searchFromTime .or. fromTime>maxTime)) Cycle
                realSwitchDelays = realSwitchDelays + 1
                Call addSwitchDelay(atNode, fromNode, toNode, fromMode, toMode, fromTime, toTime, delayTime)
        EndDo
        Write(*,*) 'Switch Delays:',switchDelays," needed:", realSwitchDelays

        !debug start
        If (debugLvl >= 1) Then
                Write(11,'(A4, A4, A4, 5000I6)') 't','i','j',(t,t=1,maxModes)
                Do j = 1, maxNodes
                        Do iIndex = 1, Size(nodes(j)%predecessors)
                                i = nodes(j)%predecessors(iIndex)
                                Do t = 1, maxTime
                                        Write(11,'(I4, I4, I4, 5000I6)') t,i,j,(getTravelTime(i, j, y, ↙
t),y=1,maxModes)
                                EndDo
                        EndDo
                EndDo
        EndIf
        !debug end

        ! Allocate memory and value init
        Write(*,*) 'Initializing'
        Do i=1,maxNodes
                Allocate(nodes(i)%labels(Size(nodes(i)%predecessors),maxModes,maxTime))
                Allocate(nodes(i)%timepath(maxTime,Size(nodes(i)%availableModes), ↙
Size(nodes(i)%predecessors),5))
                nodes(i)%labels(:,:,:) = 99999
                nodes(i)%timepath(:,:,:,:) = 99999
        EndDo
        Write(*,*) '   allocated'

        ! Initialize labels for last node
        Do kIndex = 1, Size(nodes(searchToNode)%predecessors)
                k = nodes(searchToNode)%predecessors(kIndex)
                Do xIndex = 1, Size(nodes(searchToNode)%availableModes)
                        x = nodes(searchToNode)%availableModes(xIndex)
                        If (x == searchToMode) Then
                                nodes(searchToNode)%labels(kIndex,x,:) = 0
                        Else
                                Do t = searchFromTime, maxTime
                                        nodes(searchToNode)%labels(kIndex,x,t) = ↙
getSwitchDelay(searchToNode, k, searchToNode, x, searchToMode, t)
                                EndDo
                        EndIf
                EndDo
        EndDo
        Write(*,*) '          Done.'

        If (debugLvl >= 1) Then
                Write(11,'(A2, A2, A2, A2, A2, A2, 5000I6)') 'l','j','i','x','y','k',(t,t=1,maxTime)
        EndIf

        ! Start main search process
        Write(*,*) 'Calculating'

        Call addScanEligible(searchToNode)
        !Read switch delay rules
        Read(10,*) switchDelays
        Do i=1, switchDelays
                Read(10,908) fromNode, fromMode, atNode, toMode, toNode, fromTime, toTime, delayTime
                If ((toTime<searchFromTime .or. fromTime>maxTime)) Cycle
                realSwitchDelays = realSwitchDelays + 1
                Call addSwitchDelay(atNode, fromNode, toNode, fromMode, toMode, fromTime, toTime, delayTime)
        EndDo
        Write(*,*) 'Switch Delays:',switchDelays," needed:", realSwitchDelays
```

```
        !debug start
        If (debugLvl >= 1) Then
                Write(11,'(A4, A4, A4, 5000I6)') 't','i','j',(t,t=1,maxModes)
                Do j = 1, maxNodes
                        Do iIndex = 1, Size(nodes(j)%predecessors)
                                i = nodes(j)%predecessors(iIndex)
                                Do t = 1, maxTime
                                        Write(11,'(I4, I4, I4, 5000I6)') t,i,j,(getTravelTime(i, j, y, ↙
t),y=1,maxModes)
                                EndDo
                        EndDo
                EndDo
        EndIf
        !debug end

        ! Allocate memory and value init
        Write(*,*) 'Initializing'
        Do i=1,maxNodes
                Allocate(nodes(i)%labels(Size(nodes(i)%predecessors),maxModes,maxTime))
                Allocate(nodes(i)%timepath(maxTime,Size(nodes(i)%availableModes), ↙
Size(nodes(i)%predecessors),5))
                nodes(i)%labels(:,:,:) = 99999
                nodes(i)%timepath(:,:,:,:) = 99999
        EndDo
        Write(*,*) '   allocated'

        ! Initialize labels for last node
        Do kIndex = 1, Size(nodes(searchToNode)%predecessors)
                k = nodes(searchToNode)%predecessors(kIndex)
                Do xIndex = 1, Size(nodes(searchToNode)%availableModes)
                        x = nodes(searchToNode)%availableModes(xIndex)
                        If (x == searchToMode) Then
                                nodes(searchToNode)%labels(kIndex,x,:) = 0
                        Else
                                Do t = searchFromTime, maxTime
                                        nodes(searchToNode)%labels(kIndex,x,t) = ↙
getSwitchDelay(searchToNode, k, searchToNode, x, searchToMode, t)
                                EndDo
                        EndIf
                EndDo
        EndDo
        Write(*,*) '          Done.'

        If (debugLvl >= 1) Then
                Write(11,'(A2, A2, A2, A2, A2, A2, 5000I6)') 'l','j','i','x','y','k',(t,t=1,maxTime)
        EndIf

        ! Start main search process
        Write(*,*) 'Calculating'

        Call addScanEligible(searchToNode)

        ! Start main recursion
        Do While (Size(scanEligibles)>0)
                If (debugLvl >= 1) Write(11,*) 'SE:',scanEligibles
                j = getScanEligible()
                If (debugLvl >= 1) Write(*,*) '   j', j
                Do iIndex = 1, Size(nodes(j)%predecessors) - 1 ! do not include entrance node
                        i = nodes(j)%predecessors(iIndex)
                        If (debugLvl >= 3) Write(11,*) 'i:[',nodes(j)%predecessors,'] (-last) ',iIndex
                        Do xIndex = 1, Size(nodes(i)%availableModes)
                                x = nodes(i)%availableModes(xIndex)
                                Do yIndex = 1, Size(nodes(j)%availableModes)
                                        y = nodes(j)%availableModes(yIndex)
                                        Do kIndex = 1, Size(nodes(i)%predecessors)
                                                k = nodes(i)%predecessors(kIndex)
                                                If (debugLvl >= 3) Write(11,*) 'k:[',nodes(i)%predecessors,'] ↙
',kIndex
```

```
                                                Do t = searchFromTime, maxTime
                                                        switchDelay = getSwitchDelay(i, k, j, x, y, t)
                                                        if (switchDelay>=99999) Cycle
                                                        travelTime = switchDelay + getTravelTime(i, j, y, t + ↙
switchDelay)
                                                        if (travelTime>=99999) Cycle
                                                        label = travelTime + getLabel(j,i,y,t+travelTime)
                                                        if (label>=99999) Cycle
                                                        If (setLabel(i, k, x, t, label)) Then
                                                                If (debugLvl >= 1) ↙
Write(11,'("T:",I0,"|l_",I0,".",I0,".",I0,":",I0,"|",I4,"-",I2,"->",I4,"-",I2,"->",I4,"|X:",I5,"|t:",I4,"|l",I2,I2,I2":",I5)') &

                t,i,k,x,label,k,x,i,y,j,switchDelay, getTravelTime(i, j, y, t + switchDelay),j,i,y,getLabel(j,i,y,t+travelTime)
                                                        Call addScanEligible(i)
                                                        !if you are on i node by x mode at time t
                                                        nodes(i)%timepath(t,xIndex,kIndex,1) = j ↙
!best go to node
                                                        nodes(i)%timepath(t,xIndex,kIndex,2) = y ↙
!by mode
                                                        nodes(i)%timepath(t,xIndex,kIndex,3) = ↙
switchDelay !mode change wait
                                                        nodes(i)%timepath(t,xIndex,kIndex,4) = ↙
travelTime - switchDelay - getTravelTime(i, j, y, t + switchDelay, .True.) !wait for
                                                        nodes(i)%timepath(t,xIndex,kIndex,5) = ↙
travelTime !total travel for
                                                                EndIf
                                                EndDo
                                                If (debugLvl >= 1) Write(11,'(A2, I2, I2, I2, I2, I2, 144016)') ↙
'l',j,i,x,y,k,(getLabel(i,k,x,t),t=1,maxTime)
                                        EndDo
                                EndDo
                        EndDo
                EndDo
        EndDo
        Write(*,*) '        Done'

        If (debugLvl >= 1) Then
                Write(11,*) '---'
                Write(11,'(A2, A2, A2, 5A6)') "i", "t", "x", "TNode", "BMode", "CWait", "WaitF", "Total"
                Do t = 1, maxTime
                        Do i = 1, maxNodes
                                Do kIndex = 1, Size(nodes(i)%predecessors)
                                        k = nodes(i)%predecessors(kIndex)
                                        Do xIndex = 1, Size(nodes(i)%availableModes)
                                                x = nodes(i)%availableModes(xIndex)
                                                If (nodes(i)%timepath(t,xIndex,kIndex,1) /= 99999) ↙
Write(11,'(I2, I2, I2, I2, 5I6)') i, t, x, k, nodes(i)%timepath(t,xIndex,kIndex,1), & ↙
                                                nodes(i)%timepath(t,xIndex,kIndex,2), ↙
nodes(i)%timepath(t,xIndex,kIndex,3), &
                                                nodes(i)%timepath(t,xIndex,kIndex,4), ↙
nodes(i)%timepath(t,xIndex,kIndex,5)
                                        EndDo
                                EndDo
                        EndDo
                EndDo
        EndIf

        ! Output found solutions
        Write(*,*) 'Writing Data'
        Do start_t = searchFromTime, maxTime
                If (debugLvl >= 3) Write(*,*) '    t:',start_t
                i = searchFromNode
                x = searchFromMode
                t = start_t
                k = i
                travelTime = 0
                Write(11,*) '---'
                Do While (i /= searchToNode)
                        xIndex = arraySearch(nodes(i)%availableModes, x)
                        kIndex = arraySearch(nodes(i)%predecessors, k)
```

```
                    If (nodes(i)%timepath(t,xIndex,kIndex,1) /= 99999) Then !fix
                        Write(11,'(A,I4,A,I4,A,I4,A,I2,A,I4,A,I3,A,I4,A,I3,A,I4,A)') &
                                                    'At time ', t, ' at node',i,' from node ',k,' wait ↙
',nodes(i)%timepath(t,xIndex,kIndex,4),'min to go to node ', &
                                                    nodes(i)%timepath(t,xIndex,kIndex,1), ' by mode ', &
                                                    nodes(i)%timepath(t,xIndex,kIndex,2),'. You need ', ↙
nodes(i)%timepath(t,xIndex,kIndex,3), 'min to swap from mode', x,&
                                                    '. That will take total ', ↙
nodes(i)%timepath(t,xIndex,kIndex,5),'min.'
                        travelTime = travelTime + nodes(i)%timepath(t,xIndex,kIndex,5)
                        next_i = nodes(i)%timepath(t,xIndex,kIndex,1)
                        next_t = t + nodes(i)%timepath(t,xIndex,kIndex,5)
                        next_x = nodes(i)%timepath(t,xIndex,kIndex,2)
                        k = i
                        i = next_i
                        t = next_t
                        x = next_x
                    Else
                        exit
                    EndIf
                EndDo
                If (i == searchToNode) Then
                    If (x /= searchToMode) Then
                        switchDelay = getSwitchDelay(i, k, i, x, searchToMode, t)
                        Write(11,'("At time ",I0," you reach the destination but you need ",I0," to swap ↙
from ",I0,".")') t, switchDelay, x
                        t = t + switchDelay
                        travelTime = travelTime + switchDelay
                    EndIf
                    Write(11,'("You will reach the destination at ",I0,".")') t
                EndIf
                Write(11,'(A,I0,A)') "Total: ",travelTime,"min."
            EndDo
            Write(*,*) '         Done'

            ! Stop timing the process
            Call cpu_time(timeF)
            Write(*,*) "Time = ", timeF - timeS," seconds."

            ! Get System process info to log memory usage. ONLY WINDOWS
            call processInfo()

            ! Close opened files
            Close(10)
            Close(11)

            ! Safelly deallocate used memory
            Do i=1,maxNodes
                If (allocated(nodes(i)%predecessors)) Deallocate(nodes(i)%predecessors)
                If (allocated(nodes(i)%availableModes)) Deallocate(nodes(i)%availableModes)
                If (allocated(nodes(i)%timeTables)) Deallocate(nodes(i)%timeTables)
                If (allocated(nodes(i)%labels)) Deallocate(nodes(i)%labels)
                If (allocated(nodes(i)%timepath)) Deallocate(nodes(i)%timepath)
                call dict_destroy( nodes(i)%hashData )
            EndDo
            Deallocate(nodes)
            Deallocate(scanEligibles)

            ! Used formats
            901 Format(1I6)
            906 Format(6I6)
            908 Format(8I6)

            Contains

                Subroutine addTimeTable(fromNode, toNode, mode, fromTime, toTime, travelTime)
                    Integer :: fromNode, toNode, mode, fromTime, toTime, travelTime, count
                    Integer, Allocatable :: temp(:,:)
                    count = nodes(toNode)%timeTablesCount
                    If (count>0) Then
                        Allocate(temp(count, 5))
                        temp(1:count,1:5) = nodes(toNode)%timeTables(1:count,1:5)
                        Deallocate(nodes(toNode)%timeTables)
                    EndIf
```

```
                Allocate(nodes(toNode)%timeTables(count+1, 5))
                If (count>0) nodes(toNode)%timeTables(1:count, :) = temp(1:count, :)
                nodes(toNode)%timeTables(count+1, :) = (/fromNode, mode, fromTime, toTime, ↙
travelTime/)
                nodes(toNode)%timeTablesCount = nodes(toNode)%timeTablesCount + 1
                If (count>0) Deallocate(temp)
        EndSubroutine addTimeTable

        Integer Function getTravelTime(fromNode, toNode, mode, time, withoutWait)
                Integer :: fromNode, toNode, mode, time, i, travelTime, tempTravelTime
                Logical, Optional :: withoutWait
                Logical :: logic_withoutWait
                If (Present(withoutWait)) Then
                        If (withoutWait) Then
                                logic_withoutWait = .True.
                        Else
                                logic_withoutWait = .False.
                        EndIf
                Else
                        logic_withoutWait = .False.
                EndIf
                If (fromNode==toNode) Then
                        getTravelTime = 0
                        return
                EndIf
                travelTime = 99999
                Do i=1, nodes(toNode)%timeTablesCount
                        If (nodes(toNode)%timeTables(i,1) == fromNode .AND. &
                                nodes(toNode)%timeTables(i,2) == mode                   ) Then
                                If (nodes(toNode)%timeTables(i,3) <= time .AND. &
                                        nodes(toNode)%timeTables(i,4) >= time           ) Then
                                        travelTime = nodes(toNode)%timeTables(i,5)
                                Else If (nodes(toNode)%timeTables(i,3) > time) Then
                                        If (logic_withoutWait) Then
                                                tempTravelTime = nodes(toNode)%timeTables(i,5)
                                        Else
                                                tempTravelTime = ↙
nodes(toNode)%timeTables(i,5)+(nodes(toNode)%timeTables(i,3)-time)
                                        EndIf
                                        travelTime = tempTravelTime
                                EndIf
                        EndIf
                EndDo
                getTravelTime = travelTime
        EndFunction getTravelTime

        Subroutine addScanEligible(node)
                Integer :: node
                If (debugLvl >= 2) Write(*,*)"addSE:",node," deque=",nodes(node)%deque
                If (firstEligible == -1) Then
                        firstEligible = node
                        lastEligible = node
                        Call push(scanEligibles, node)
                        nodes(lastEligible)%deque = 99999
                Else
                        If (nodes(node)%deque == 0) Then
                                Call push(scanEligibles, node)
                                nodes(node)%deque = 99999
                                If (lastEligible > -1) nodes(lastEligible)%deque = node
                                lastEligible = node
                        Else If (nodes(node)%deque == -1) Then
                                Call push(scanEligibles, node, .True.)
                                If (firstEligible > -1) nodes(node)%deque = firstEligible
                                firstEligible = node
                        EndIf
                EndIf
        EndSubroutine addScanEligible
```

```
		Integer Function getScanEligible()
			Integer :: node = -1, count
			Integer, Allocatable :: temp(:)
			count = Size(scanEligibles)
			If (count > 0) Then
				Allocate(temp(count))
				temp(:) = scanEligibles(:)
				If (Allocated(scanEligibles)) Deallocate(scanEligibles)
				Allocate(scanEligibles(count-1))
				scanEligibles(:) = temp(2:count)
				node = firstEligible
				If (firstEligible == lastEligible) Then
					firstEligible = -1
					lastEligible = -1
				Else
					firstEligible = nodes(node)%deque
				EndIf
				nodes(node)%deque = -1
				If (debugLvl >= 2) Write(*,*)"getSE:",node," deque=",nodes(node)%deque
			EndIf
			getScanEligible = node
		EndFunction getScanEligible

		Logical Function setLabel(toNode, fromNode, mode, time, label)
			Integer :: fromNode, toNode, mode, time, label, fromNodeIdx
			fromNodeIdx = arraySearch(nodes(toNode)%predecessors, fromNode)
			If (label.GT.0 .AND. fromNodeIdx.GT.0 .AND. label.LT.nodes(toNode)%labels(fromNodeIdx, ↙
mode, time)) Then
				If (debugLvl >= 2) Write(11,'(A, I4, A, I5, A, I5)') 'At ', time,' was ', ↙
nodes(toNode)%labels(fromNodeIdx, mode, time), ' is ', label
				nodes(toNode)%labels(fromNodeIdx, mode, time) = label
				setLabel = .True.
			Else
				setLabel = .False.
			EndIf
		EndFunction setLabel

		Integer Function getLabel(toNode, fromNode, mode, time)
			Integer :: fromNode, toNode, mode, time, label, fromNodeIdx
			label = 0
			fromNodeIdx = arraySearch(nodes(toNode)%predecessors, fromNode)
			If (toNode>Size(nodes)) label = 99999
			If (fromNodeIdx == 0) label = 99999
			If (mode>Size(nodes(toNode)%labels,2)) label = 99999
			If (time>Size(nodes(toNode)%labels,3)) label = 99999
			If (label==0) label = nodes(toNode)%labels(fromNodeIdx, mode, time)
			If (debugLvl >= 2) Write(11,'("l_",i0,".",i0,"^",i0," (",i0,")=",i0)') toNode, fromNode, mode, ↙
time,label
			getLabel = label
		EndFunction getLabel

		Subroutine addSwitchDelay(atNode, fromNode, toNode, fromMode, toMode, fromTime, toTime, ↙
delayTime)
			Integer:: fromNode, fromMode, atNode, toMode, toNode, fromTime, toTime, delayTime, count
			Integer, Allocatable :: temp(:,:)
			type(DICT_DATA) :: hashData, hashDataTemp
			If (debugLvl >= 2) Write(11,'("+X_",i0,".",i0,".",i0,"^",i0,".",i0," (",I0,"-",I0,")=",I0)') atNode, ↙
fromNode, toNode, fromMode, toMode, fromTime, toTime, delayTime

			hashKey(1)  = fromNode
			hashKey(2)  = toNode
			hashKey(3)  = fromMode
			hashKey(4)  = toMode
			hashData = dict_get_key(nodes(atNode)%hashData, hashKey)

			count = hashData%count

			If (count+1>hashData%size) Then
				If (count>0) Then
					hashDataTemp = hashData
					Deallocate(hashData%rules)
				EndIf
```

```
                    hashData%size = (count+1)*2
                    Allocate(hashData%rules(hashData%size, 3))
                    If (count>0) hashData%rules(1:count, :) = hashDataTemp%rules(1:count, :)
                    If (count>0) Deallocate(hashDataTemp%rules)
                EndIf
                hashData%rules(count+1, :) = (/fromTime, toTime, delayTime/)
                hashData%count = count+1
                call dict_add_key( nodes(atNode)%hashData, hashKey, hashData )
            EndSubroutine addSwitchDelay

            Integer Function getSwitchDelay(atNode, fromNode, toNode, fromMode, toMode, time)
                Integer:: atNode, fromNode, toNode, fromMode, toMode, time, i, delayTime, tempTravelTime
                type(DICT_DATA) :: hashData
                If (fromNode==atNode .And. fromMode == toMode) Then
                    If (debugLvl >= 2) Write(11,'("-X_",i0,".",i0,".",i0,"^",i0,".",i0," (",I0,")=",I0)') ↙
atNode, fromNode, toNode, fromMode, toMode, time, 0
                    getSwitchDelay = 0
                    return
                EndIf

                hashKey(1)  = fromNode
                hashKey(2)  = toNode
                hashKey(3)  = fromMode
                hashKey(4)  = toMode
                hashData = dict_get_key(nodes(atNode)%hashData, hashKey)

                delayTime = 99999
                Do i=1, hashData%count
                    If (hashData%rules(i,1) <= time .AND. &
                        hashData%rules(i,2) >= time                    ) Then
                        delayTime = hashData%rules(i,3)
                    EndIf
                EndDo
                If (debugLvl >= 2) Write(11,'("X_",i0,".",i0,".",i0,"^",i0,".",i0," (",I0,")=",I0)') atNode, ↙
fromNode, toNode, fromMode, toMode, time, delayTime
                getSwitchDelay = delayTime
            EndFunction getSwitchDelay

            Subroutine commandLine()
                Integer :: i
                Character :: arg*2048
                If (Command_argument_count() > 0) Then
                    i = 1
                    Do While (i <= Command_argument_count())
                        call Get_command_argument(i,arg)
                        SelectCase (Trim(arg))
                            Case ("-h","--help","/?")
                                Write(*,*)
                                Write(*,*) "Usage: <program> -fn fromNode [-ft ↙
fromTime] -tn toNode"
                                Write(*,*) "            [-n filename] [-debug filename]"
                                Write(*,*)
                                Write(*,*) "Description:"
                                Write(*,*) Char(9), "Calculate intermodal time- ↙
dependent least-time paths on"
                                Write(*,*) Char(9), "multimodal transportation ↙
networks."
                                Write(*,*)
                                Write(*,*) "Options:"
                                Write(*,*) Char(9), "-fn fromNode  Departure Node."
                                Write(*,*) Char(9), "-ft fromTime  Time to start ↙
searhing (minutes)."
                                Write(*,*) Char(9), "          Default: 1"
                                Write(*,*) Char(9), "-tn toNode    Arrival Node."
                                Write(*,*) Char(9), "-tt toTime    Time to arrive ↙
(minutes)."
                                Write(*,*) Char(9), "          Default: 1440"
```

```
                                        Write(*,*) Char(9), "-n  filename  Network file name."
                                        Write(*,*) Char(9), "            Default: 'network.txt'"
                                        Write(*,*) Char(9), "-d  filename  Debug file name."
                                        Write(*,*) Char(9), "            Default: 'debug.txt'"
                                        Write(*,*) Char(9), "-dl debugLvl  Debug level."
                                        Write(*,*) Char(9), "            Default: 0"
                                        Write(*,*) Char(9), "-v          Display the version and ↙
exit."
                                        Write(*,*) Char(9), "-h          Print this help."
                                        Write(*,*) Char(9), "--help      Print this help."
                                        Write(*,*) Char(9), "/?          Print this help."
                                        Write(*,*)
                                        Write(*,*)
                                        Write(*,*) "Mail bug reports and suggestions to ↙
<cbotsikas@gmail.com>."
                                        call exit()
                                    Case ("-v")
                                        Write(*,*)
                                        Write(*,*) "Version: 1.110620"
                                        Write(*,*)
                                        Write(*,*) "Copyright (C) 2010 Chirstos Botsikas"
                                        Write(*,*)
                                        Write(*,*) "Writen by Christos Botsikas ↙
<cbotsikas@gmail.com>."
                                        call exit()
                                    Case ("-fn")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Read(arg,*) searchFromNode
                                    Case ("-tn")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Read(arg,*) searchToNode
                                    Case ("-ft")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Read(arg,*) searchFromTime
                                    Case ("-tt")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Read(arg,*) maxTime
                                    Case ("-d")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Open(11, file=Trim(arg), action='write')
                                        isDebugLoaded = .True.
                                    Case ("-dl")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Read(arg,*) debugLvl
                                    Case ("-n")
                                        i = i + 1
                                        call Get_command_argument(i,arg)
                                        Open(10, file=Trim(arg), action='read')
                                        isNetworkLoaded = .True.
                                    Case Default
                                        Write(*,*) Char(13)//Char(10)//"The syntax of the ↙
command is incorrect."//Char(13)//Char(10)//"Try -h for help."
                                        call exit()
                                EndSelect
                                i = i + 1
                            EndDo
                        Else
                            Write(*,*) Char(13)//Char(10)//"The syntax of the command is ↙
incorrect."//Char(13)//Char(10)//"Try -h for help."
                            call exit()
                        EndIf
                    EndSubroutine commandLine

                    Subroutine processInfo()
                        Integer :: getpid, pid, eof
```

```
                Character :: cmd*512, file*512, line*50
                Logical :: fileExists = .False.
                pid = getpid()

                Write(file,'("_taskinfo_",i0,".txt")') pid
                Write(cmd,'("tasklist /V /FO list /fi ""PID eq ",i0,""" > ",A)') pid, Trim(file)
                Call System(Trim(cmd))
                Open(15, file=Trim(file), action='read')
                Read(15,*)
                Do
                        Read(15,'(A50)',IOSTAT=eof) line
                        If (eof==-1) Exit
                        Write(*,*) line
                End do
                Close(15)
                Call unlink(Trim(file))
        EndSubroutine processInfo
EndProgram
```

Αρχείο ArrayMethods.f90

```
! Usefull array methods writen by Christos Botsikas

Module ArrayMethods
        Interface push
                Module Procedure pushInt, pushReal
        EndInterface

        !Sample Usage Write(*,*) pop(demo, tempInt), tempInt
        Interface pop
                Module Procedure popInt, popReal
        EndInterface

        Interface inArray
                Module Procedure inArrayInt, inArrayReal
        EndInterface

        Interface arraySearch
                Module Procedure arraySearchInt, arraySearchReal
        EndInterface

        Contains

                Subroutine pushInt(this, val, front)
                        Integer, Allocatable, Intent(inout) :: this(:)
                        Integer :: val
                        Integer, Allocatable :: temp(:)
                        Logical, Optional :: front
                        Logical :: logic_front
                        Integer :: count
                        count = 0
                        If (Allocated(this)) Then
                                count = Size(this)
                                Allocate(temp(count))
                                temp(:) = this(:)
                                Deallocate(this)
                        EndIf
                        Allocate(this(count+1))
                        If (count == 0) Then
                                this(1) = val
                                return
                        EndIf
                        If (Present(front)) Then
                                If (front) Then
                                        logic_front = .True.
                                Else
                                        logic_front = .False.
                                EndIf
                        Else
                                logic_front = .False.
                        EndIf
```

```
        If (logic_front) Then
                this(1) = val
                this(2:count+1) = temp(1:count)
        Else
                this(1:count) = temp(1:count)
                this(count+1) = val
        EndIf
EndSubroutine pushInt

Subroutine pushReal(this, val, front)
        Real, Allocatable, Intent(inout) :: this(:)
        Real :: val
        Integer, Allocatable :: temp(:)
        Logical, Optional :: front
        Logical :: logic_front
        Integer :: count
        count = 0
        If (Allocated(this)) Then
                count = Size(this)
                Allocate(temp(count))
                temp(:) = this(:)
                Deallocate(this)
        EndIf
        Allocate(this(count+1))
        If (count == 0) Then
                this(1) = val
                return
        EndIf
        If (Present(front)) Then
                If (front) Then
                        logic_front = .True.
                Else
                        logic_front = .False.
                EndIf
        Else
                logic_front = .False.
        EndIf
        If (logic_front) Then
                this(1) = val
                this(2:count+1) = temp(1:count)
        Else
                this(1:count) = temp(1:count)
                this(count+1) = val
        EndIf
EndSubroutine pushReal


Logical Function popInt(this, val, front)
        Integer, Allocatable, Intent(inout) :: this(:)
        Integer, Allocatable :: temp(:)
        Integer :: val
        Logical, Optional :: front
        Logical :: logic_front
        Integer :: count

        If (Present(front)) Then
                If (front) Then
                        logic_front = .True.
                Else
                        logic_front = .False.
                EndIf
        Else
                logic_front = .False.
        EndIf

        count = 0
        If (Allocated(this)) Then
                count = Size(this)
                Allocate(temp(count))
                temp(:) = this(:)
                Deallocate(this)

                If (count > 1) Then
                        Allocate(this(count-1))
                        If (logic_front) Then
                                val = temp(1)
                                this(1:count-1) = temp(2:count)
```

```
                        Else
                            val = temp(count)
                            this(1:count-1) = temp(1:count-1)
                        EndIf
                    Else
                        val = temp(1)
                    EndIf
                    popInt = .True.
                Else
                    val = 0
                    popInt = .False.
                EndIf
EndFunction popInt

Logical Function popReal(this, val, front)
                Real, Allocatable, Intent(inout) :: this(:)
                Real, Allocatable :: temp(:)
                Real :: val
                Logical, Optional :: front
                Logical :: logic_front
                Integer :: count

                If (Present(front)) Then
                    If (front) Then
                        logic_front = .True.
                    Else
                        logic_front = .False.
                    EndIf
                Else
                    logic_front = .False.
                EndIf

                count = 0
                If (Allocated(this)) Then
                    count = Size(this)
                    Allocate(temp(count))
                    temp(:) = this(:)
                    Deallocate(this)

                    If (count > 1) Then
                        Allocate(this(count-1))
                        If (logic_front) Then
                            val = temp(1)
                            this(1:count-1) = temp(2:count)
                        Else
                            val = temp(count)
                            this(1:count-1) = temp(1:count-1)
                        EndIf
                    Else
                        val = temp(1)
                    EndIf
                    popReal = .True.
                Else
                    val = 0
                    popReal = .False.
                EndIf
EndFunction popReal

Logical Function inArrayInt(this, val)
                Integer, Allocatable :: this(:)
                Integer :: val, pos
                If (Allocated(this)) Then
                    Do pos = 1, size(this)
                        If (this(pos) == val) Then
                            inArrayInt = .True.
                            return
                        EndIF
                    EndDo
                EndIf
                inArrayReal = .False.
EndFunction inArrayReal

Integer Function arraySearchInt(this, val)
                Integer, Allocatable :: this(:)
                Integer :: val, pos
                If (Allocated(this)) Then
```

```
                Do pos = 1, size(this)
                    If (this(pos) == val) Then
                        arraySearchInt = pos
                        return
                    EndIF
                EndDo
            EndIf
            arraySearchInt = 0
        EndFunction arraySearchInt

        Integer Function arraySearchReal(this, val)
            Integer, Allocatable :: this(:)
            Real :: val
            Integer :: pos
            If (Allocated(this)) Then
                Do pos = 1, size(this)
                    If (this(pos) == val) Then
                        arraySearchReal = pos
                        return
                    EndIF
                EndDo
            EndIf
            arraySearchReal = 0
        EndFunction arraySearchReal
EndModule
```

Αρχείο HashTables.f90

```
! Hash Table needed modules

module HashTableMod

!
! The length of the keys
!
integer, parameter :: DICT_KEY_LENGTH = 14

!
! The data that will be stored with each key
type HashData
        Integer :: count = 0
        Integer :: size = 0
        Integer,allocatable :: rules(:,:)
end type
end module


module HashTable
  use HashTableMod, DICT_DATA => HashData
  implicit none
        type(DICT_DATA) :: DICT_NULL
  include "specialDictionary.fi"
end module
```

Αρχείο specialDictionary.fi

```
! Provided by FLIBS http://flibs.sourceforge.net/
! Modified by Christos Botsikas to incorporate array hash key instead of string.

! dictionary.f90 --
!    Include file for defining dictionaries:
!    a mapping of strings to some data
!
!    See the example/test program for the way to use this
!
!    Note:
!    Use is made of a hash table. This should speed up most
!    operations. The algorithm for determining the hashkey
!    is taken from Kernighan and Pike: The Practice of Programming
!
!    Note:
!    - Define the length of the strings as
!      parameter "DICT_KEY_LENGTH"
!    - Define a derived type for the data
!      to be stored
!    - Also define a "null" value - DICT_NULL
!      of type DICT_DATA, for use when the
!      key is not found.
!    - Put both in a separate module, that
!      will be used.
!
!    $Id: dictionary.f90,v 1.4 2009/05/09 14:34:50 arjenmarkus Exp $
!
type LIST_DATA
   Integer :: key(4)
   type(DICT_DATA)            :: value
end type LIST_DATA

type HASH_LIST
   type(LINKED_LIST), pointer :: list
end type HASH_LIST

type DICT_STRUCT
   private
   type(HASH_LIST), pointer, dimension(:) :: table
end type DICT_STRUCT

!
! We do not want everything to be public
!
private :: LIST_DATA
private :: HASH_LIST
private :: LINKED_LIST
private :: list_create
private :: list_destroy
private :: list_count
private :: list_next
private :: list_insert
private :: list_insert_head
private :: list_delete_element
private :: list_get_data
private :: list_put_data
private :: dict_get_elem
private :: dict_hashkey

integer, parameter, private :: hash_size  = 4993
integer, parameter, private :: multiplier = 31

include 'linkedlist.fi'
! Download from FLIBS http://flibs.sourceforge.net/ included in datastructures package as linkedlist.f90

!
! Routines and functions specific to dictionaries
!
```

```
! dict_create --
!    Create and initialise a dictionary
! Arguments:
!    dict      Pointer to new dictionary
!    key       Key for the first element
!    value     Value for the first element
! Note:
!    This version assumes a shallow copy is enough
!    (that is, there are no pointers within the data
!    to be stored)
!    It also assumes the argument list does not already
!    refer to a list. Use dict_destroy first to
!    destroy up an old list.
!
subroutine dict_create( dict, key, value )
   type(DICT_STRUCT), pointer   :: dict
   integer, intent(in) ::  key(4)
   type(DICT_DATA), intent(in)  :: value

   type(LIST_DATA)          :: data
   integer                  :: i
   integer                  :: hash

   allocate( dict )
   allocate( dict%table(hash_size) )

   do i = 1,hash_size
      dict%table(i)%list => null()
   enddo

   data%key   = key
   data%value = value

   hash = dict_hashkey( key)
   call list_create( dict%table(hash)%list, data )

end subroutine dict_create

! dict_destroy --
!    Destroy an entire dictionary
! Arguments:
!    dict      Pointer to the dictionary to be destroyed
! Note:
!    This version assumes that there are no
!    pointers within the data that need deallocation
!
subroutine dict_destroy( dict )
   type(DICT_STRUCT), pointer  :: dict

   integer                  :: i

   do i = 1,size(dict%table)
      if ( associated( dict%table(i)%list ) ) then
         call list_destroy( dict%table(i)%list )
      endif
   enddo
   deallocate( dict%table )
   deallocate( dict )

end subroutine dict_destroy

! dict_add_key
!    Add a new key
! Arguments:
!    dict      Pointer to the dictionary
!    key       Key for the new element
!    value     Value for the new element
! Note:
```

```
!    If the key already exists, the
!    key's value is simply replaced
!
subroutine dict_add_key( dict, key, value )
   type(DICT_STRUCT), pointer   :: dict
   integer, intent(in) :: key(4)
   type(DICT_DATA), intent(in)  :: value

   type(LIST_DATA)          :: data
   type(LINKED_LIST), pointer   :: elem
   integer                  :: hash

   elem => dict_get_elem( dict, key )

   if ( associated(elem) ) then
      elem%data%value = value
   else
      data%key   = key
      data%value = value
      hash       = dict_hashkey( key )
      if ( associated( dict%table(hash)%list ) ) then
         call list_insert( dict%table(hash)%list, data )
      else
         call list_create( dict%table(hash)%list, data )
      endif
   endif

end subroutine dict_add_key

! dict_delete_key
!    Delete a key-value pair from the dictionary
! Arguments:
!    dict      Dictionary in question
!    key       Key to be removed
!
subroutine dict_delete_key( dict, key )
   type(DICT_STRUCT), pointer   :: dict
   integer, intent(in) :: key(4)

   type(LINKED_LIST), pointer   :: elem
   integer                  :: hash

   elem => dict_get_elem( dict, key )

   if ( associated(elem) ) then
      hash = dict_hashkey( key)
      call list_delete_element( dict%table(hash)%list, elem )
   endif
end subroutine dict_delete_key

! dict_get_key
!    Get the value belonging to a key
! Arguments:
!    dict      Pointer to the dictionary
!    key       Key for which the values are sought
!
function dict_get_key( dict, key ) result(value)
   type(DICT_STRUCT), pointer   :: dict
   integer, intent(in) :: key(4)
   type(DICT_DATA)          :: value
```

```
   type(DICT_STRUCT), pointer  :: dict
   integer, intent(in) ::  key(4)
   type(DICT_DATA), intent(in)  :: value

   type(LIST_DATA)          :: data
   integer                  :: i
   integer                  :: hash

   allocate( dict )
   allocate( dict%table(hash_size) )

   do i = 1,hash_size
      dict%table(i)%list => null()
   enddo

   data%key   = key
   data%value = value

   hash = dict_hashkey( key)
   call list_create( dict%table(hash)%list, data )

end subroutine dict_create

! dict_destroy --
!    Destroy an entire dictionary
! Arguments:
!    dict      Pointer to the dictionary to be destroyed
! Note:
!    This version assumes that there are no
!    pointers within the data that need deallocation
!
subroutine dict_destroy( dict )
   type(DICT_STRUCT), pointer  :: dict

   integer                  :: i
   type(LIST_DATA)          :: data
   type(LINKED_LIST), pointer  :: elem

   elem => dict_get_elem( dict, key )

   if ( associated(elem) ) then
      value = elem%data%value
   else
      value = DICT_NULL
   endif
end function dict_get_key

! dict_has_key
!    Check if the dictionary has a particular key
! Arguments:
!    dict      Pointer to the dictionary
!    key       Key to be sought
!
function dict_has_key( dict, key ) result(has)
   type(DICT_STRUCT), pointer  :: dict
   integer, intent(in) :: key(4)
   logical                  :: has

   type(LINKED_LIST), pointer  :: elem

   elem => dict_get_elem( dict, key )

   has = associated(elem)
end function dict_has_key

! dict_get_elem
!    Find the element with a particular key
! Arguments:
```

```
!   dict     Pointer to the dictionary
!   key      Key to be sought
!
function dict_get_elem( dict, key ) result(elem)
  type(DICT_STRUCT), pointer  :: dict
  integer, intent(in) :: key(4)

  type(LINKED_LIST), pointer  :: elem
  integer                     :: hash

  hash = dict_hashkey( key)

  elem => dict%table(hash)%list
  do while ( associated(elem) )
    if ( elem%data%key(1) .eq. key(1) .and. elem%data%key(2) .eq. key(2)  .and. elem%data%key(3) .eq. key(3)  .and. ↙ el-
em%data%key(4) .eq. key(4)) then
      exit
    else
      elem => list_next( elem )
    endif
  enddo
end function dict_get_elem

! dict_hashkey
!   Determine the hash value from the string
! Arguments:
!   key      String to be examined
!
integer function dict_hashkey( key )
  integer, intent(in) :: key(4)

  integer                     :: hash
  integer                     :: i

  dict_hashkey = 0
  dict_hashkey = 1 + mod( (((key(1)*multiplier) + key(2))*multiplier + key(3))*multiplier + key(4), hash_size )
end function dict_hashkey
```

Κώδικας C#

Αρχείο Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using NDesk.Options; //download file from http://www.ndesk.org/Options

namespace Thesis.Net
{
  class Program
  {
    static void Main(string[] args)
    {
      var searchFromNodeId = -1;
      var searchFromModeId = 0;
      var searchFromTime = 0;
      var searchToTime = 1440;
      var searchToNodeId = -1;
      var searchToModeId = 0;
      var inputFileName = "network.txt";
      var outputFileName = "solution.txt";

      bool doExit = false;//Check if we need to exit (eg help or version)
      var argsParser = new OptionSet()
                {
                  {"h|?|help", v => { ShowHelp();doExit = true;}},
                  {"v", v =>{ShowVersion();doExit = true;}},
                  {"fn=", v => searchFromNodeId = int.Parse(v) - 1 },
                  {"tn=", v => searchToNodeId = int.Parse(v) - 1 },
                  {"ft=", v => searchFromTime = int.Parse(v) },
                  {"tt=", v => searchToTime = int.Parse(v) },
                  {"o=", v => outputFileName=v },
                  {"n=", v => inputFileName=v }
                };
      argsParser.Parse(args);//Parse the cmd args
      if (doExit) return;
      if (searchFromNodeId<0 || searchToNodeId<0)//If wrong input
      {
        Console.WriteLine("The syntax of the command is incorrect.");
        ShowHelp();
        return;
      }

      var debugFile = outputFileName.Replace("txt", "debug.txt");
      var perfLog = new PerformanceLogger(debugFile);
      perfLog.Write("Start");

      string inputLine; //A buffer to keep each read line from the input file

      List<Node> nodes = new List<Node>();//The network nodes
      Dictionary<int,ModeEntry> modes = new Dictionary<int, ModeEntry>();//and the modes

      //Hash table to keep the switch delays
      Dictionary<string,List<SwitchDelayInfo>> switchDelays = new Dictionary<string, List<SwitchDelayInfo>>();

      Console.WriteLine("Reading input file {0}.",inputFileName);
      using (var fileReader = new System.IO.StreamReader(System.IO.File.OpenRead(inputFileName)))
      {
        //FirstLine contains the number of nodes
        inputLine = fileReader.ReadLine();
        var maxNodes = InputFileHelper.ParseInt(inputLine);//The number of nodes
        Console.WriteLine("\tProcessing {0} Nodes", maxNodes);
        //Create each node):
        for (var cnodeIdx = 0; cnodeIdx < maxNodes; ++cnodeIdx)
        {
          nodes.Add(new Node(){NodeId = cnodeIdx,NodeDescr = String.Format("Node {0}",cnodeIdx+1)});
        }
        //read each node's predicessors
        for (var count = 0; count < maxNodes;++count )
        {
          fileReader.ReadLine();//Skip a line
          inputLine = fileReader.ReadLine();
          var cnodeIdx = InputFileHelper.ParseInt(inputLine)-1;//Read the refering node
          var cn = nodes[cnodeIdx];//retrieve node
         //Read the number of predicessors
          var predicessorsNumber = InputFileHelper.ParseInt(fileReader.ReadLine());
```

```
    for (var cpred = 0; cpred<predicessorsNumber;++cpred)//For each predicessor
    {
      var cpredIdx = InputFileHelper.ParseInt(fileReader.ReadLine()) - 1;//The current predicessors idx
      cn.predecessors.Add(nodes[cpredIdx]);//add it to the list
    }
  }
  fileReader.ReadLine();//Skip a line
  //The number of travelTime entries
  var travelTimesNo = InputFileHelper.ParseInt(fileReader.ReadLine());
  Console.WriteLine("\tProcessing {0} travel time entries", travelTimesNo);
  //We will be discovering the available modes and creating the timetable for the toNode
  for (int ctt = 0; ctt < travelTimesNo; ctt++)
  {
    inputLine = fileReader.ReadLine();
    //We will be reading the following entries
    // 0     1    2    3     4      5
    //fromNode, toNode, mode, fromTime, toTime, travelTime
    var data = InputFileHelper.ParseInts(inputLine, 6);
    ModeEntry cmode = null;//The current mode
    //Check if mode exists
    if (!modes.ContainsKey(data[2]-1))
    {
      cmode = new ModeEntry() {modeId = data[2] - 1, modeDescr = String.Format("Mode {0}", data[2])};
      modes.Add(data[2] - 1, cmode);
    }else
    {
      cmode = modes[data[2] - 1];
    }
    var fromNode = nodes[data[0] - 1];
    var toNode = nodes[data[1] - 1];
    var te = new TimeTableEntry();
    te.fromNode = fromNode;
    te.mode = cmode;
    te.fromTime = data[3];//Times are read as is
    te.toTime = data[4];
    te.travelTime = data[5];
    if (!((te.toTime + te.travelTime <= searchFromTime)||(te.fromTime>=searchToTime)))
    {
      //Add the mode as an available mode
      //  to the fromNode
      if (!fromNode.availableModes.Contains(cmode))
      {
        fromNode.availableModes.Add(cmode);
      }
      //  and the toNode
      if (!toNode.availableModes.Contains(cmode))
      {
        toNode.availableModes.Add(cmode);
      }
      //Add the timetable entry to the toNode
      toNode.timeTables.Add(te);
    }
  }
  Console.WriteLine("\tDiscovered {0} modes", modes.Count);
  fileReader.ReadLine();//Skip a line
  //The number of switch delay entries
  var switchDelayNo = InputFileHelper.ParseInt(fileReader.ReadLine());

  Console.WriteLine("\tProcessing {0} switch delay entries", switchDelayNo);
  for (int csd = 0; csd < switchDelayNo; csd++)
  {
    inputLine = fileReader.ReadLine();
    //We will be reading the following entries
    // 0     1      2    3    4     5     6      7
    //fromNode, fromMode, atNode, toMode, toNode, fromTime, toTime, delayTime
    var data = InputFileHelper.ParseInts(inputLine, 8);
    var fromNode = nodes[data[0] - 1];
    var fromMode = modes[data[1] - 1];
    var atNode = nodes[data[2] - 1];
    var toMode = modes[data[3] - 1];
    var toNode = nodes[data[4] - 1];
    var fromTime = data[5];
    var toTime = data[6];
    var delayTime = data[7];

    //Keep track of only the switch delays that affect the search
    if (!((toTime <= searchFromTime) || (fromTime >= searchToTime)))
    {
      //Create a hash table to store the SwitchDelays
      if (fromTime < searchFromTime)
          fromTime = searchFromTime;
        if (toTime > searchToTime)
          toTime = searchToTime;
```

```
                var key = KeyHelper.GetSwitchKey(atNode.NodeId, fromNode.NodeId, toNode.NodeId,
                                fromMode.modeId, toMode.modeId);

                if (!switchDelays.ContainsKey(key))
                  {
                    switchDelays.Add( key, new List<SwitchDelayInfo>());
                  }
                  switchDelays[key].Add(new SwitchDelayInfo() { fromTime = fromTime, toTime = toTime, swTime = delayTime });

              }
            }
            fileReader.Close();
          }
          perfLog.Write("After reading file");

          // Initialize timeTravels hash table
          var timeTravels = new System.Collections.Generic.Dictionary<string, TimeTravelInfo>();
            foreach (var toNode in nodes)
            {
              foreach (var fromNode in toNode.predecessors)
              {
                  foreach (var byMode in fromNode.availableModes.Intersect(toNode.availableModes))
                    {  //Get the modes that are available on both i and j
                      for (var t = searchFromTime; t < searchToTime + 1; t++ )
                      {//for every available time
                        var entry = new TimeTravelInfo();
                        //The travel time between nodes
                        entry.travelTime = GetTavelTime(fromNode, toNode, byMode, t, out entry. travelWait);
                        timeTravels.Add(KeyHelper.GetTimeTravelKey(fromNode.NodeId, toNode.NodeId, byMode.modeId, t), entry);
                      }
                  }
              }
            }


        Console.WriteLine("Searching from Node {0} to Node {1}.From time {2} to {3}.", searchFromNodeId+1,searchToNodeId+1,↙
searchFromTime,searchToTime);

        Console.Write("\tInitilizing final node's labels: ");
        var cnode = nodes[searchToNodeId]; //The current examined node
        var endingMode = modes[searchToModeId];

        foreach (var predecessor in cnode.predecessors.Union(new[] { cnode }))//including entry node
        {
          foreach (var mode in cnode.availableModes)
          {
            if (mode == endingMode)
            {
              //If this is the ending mode then the label is 0
              for (var t = searchFromTime ; t<= searchToTime;++t)
              {
                cnode.label.TryAdd(KeyHelper.GetLabelKey(predecessor.NodeId, mode.modeId, t), 0);
              }
            }else
            {
              //the label is the switching time
              for (var t = searchFromTime; t <= searchToTime; ++t)
              {
                var switchDelay = GetSwitchDelayHash(switchDelays,cnode, predecessor, cnode, mode, endingMode, t);
                //we will be storing only the non maxval values
                if (switchDelay<int.MaxValue) cnode.label.TryAdd(KeyHelper.GetLabelKey(predecessor.NodeId, mode.modeId, t),↙
switchDelay);
              }
            }
          }
        }
        Console.WriteLine("done.");

        perfLog.Write("After init");

        Console.WriteLine("\tScanning for best solution.");
        var seList = new ScanEligibleList();
        seList.addNode(cnode);//Add the last node in the list
        while (seList.Count>0)
        {
          var jNode = seList.getNextNode();//The target node is j
          Console.Write("\tChecking j node ({0}) : ",jNode.NodeDescr);
          //We will be looking the following path
          // x   y   modes
          //k->i->j  nodes
```

```
            foreach (var iNode in jNode.predecessors){
              foreach (var kNode in iNode.predecessors.Union(new [] {iNode})){ //include itself as an entry point
                //Get the modes that are available on both k and i
                foreach (var xMode in iNode.availableModes.Intersect(kNode.availableModes)){
                  //Get the modes that are available on both i and j
                  foreach (var yMode in iNode.availableModes.Intersect(jNode.availableModes)){
                    Parallel.For(searchFromTime,searchToTime+1, t=>{//for every available time
                     var switchDelay = GetSwitchDelayHash(switchDelays, iNode, kNode, jNode, xMode, yMode, t);
                     //TODO: possible skip loop if sd is inifinite
                     //In the paper τ = waitTime + travelTime
                     var waitTime = 0;//The time we will wait to climb aboard the yMode
                     //The travel time between nodes
                     var travelTime = GetTavelTimeHashed(timeTravels,iNode, jNode, yMode, t + switchDelay, out waitTime);
                    //The travel time between nodes
                     int label = int.MaxValue;
                     //if the move is feasible
                     if (travelTime < int.MaxValue && waitTime< int.MaxValue && switchDelay<int.MaxValue)
                     {
                              var clabel = jNode.getLabel(iNode.NodeId, yMode.modeId,t + switchDelay + travelTime + waitTime);
                              if (clabel<int.MaxValue)
                                       label = switchDelay + travelTime + waitTime + clabel;
                     }
                     if (iNode.setLabel(kNode.NodeId,xMode.modeId,t,label))//if we have a change in the label
                     {
                              seList.addNode(iNode);//add node to scan eligible
                              //and log the timepath entry
                              var tp = new TimePathEntry();
                              tp.bestGoTo = jNode;
                               tp.mode = yMode;
                              tp.switchDelay = switchDelay;
                              tp.travelTime = travelTime;
                              tp.waitTime = waitTime;
                              iNode.pushTimePathEntry(kNode.NodeId, xMode.modeId, t, tp);
                      }
                    });
                  }
                }
              }
            }
            Console.WriteLine("done.");
            perfLog.Write("Search loop");
          }
          Console.Write("Writing data to {0} : ",outputFileName);
          using (var output = new System.IO.StreamWriter(System.IO.File.OpenWrite(outputFileName)))
          {
            var targetNode = nodes[searchToNodeId];
            var startingNode = nodes[searchFromNodeId];
            var startingMode = modes[searchFromModeId];
            var targetMode = modes[searchToModeId];
            for (int startTime = searchFromTime; startTime <= searchToTime; ++startTime)
            {
              var iNode = startingNode;
              var xMode = startingMode;
              var t = startTime;//the travel time we will be looking for
              var kNode = iNode; //The previous node is the same as the start node
              var totalTravelTime = 0; //The total path travel time from begining to end
              output.WriteLine("---");
              while (iNode != targetNode)
              {
                var tp = iNode.getTimePathEntry(kNode.NodeId,xMode.modeId,t);
                if (tp!=null)
                {
                  output.WriteLine("At time {0} at {1} from {2} wait {3} to go to {4} by {5}.You need {6} to swap from {7}. The travel time will ↙
be {8}.",t,iNode.NodeDescr,kNode.NodeDescr,tp.waitTime,tp.bestGoTo.NodeDescr,tp.mode.modeDescr, ↙
tp.switchDelay,xMode.modeDescr,tp.travelTime);
                  totalTravelTime += tp.travelTime + tp.switchDelay + tp.waitTime;
                  kNode = iNode;
                  iNode = tp.bestGoTo;
                  t += tp.travelTime + tp.switchDelay + tp.waitTime;
                  xMode = tp.mode;
                }else
                {
                  break;
                }
              }
              if (iNode == targetNode){//if we found the end node
                if (xMode != targetMode)
                {
                  var finalSwitchDelay = GetSwitchDelayHash(switchDelays, targetNode, kNode, targetNode, xMode, targetMode, t);
                  output.WriteLine("At time {0} you reach the destination but you need {1} to swap from {2}.",  t,finalSwitchDelay, ↙
xMode.modeDescr);
```

```
                totalTravelTime += finalSwitchDelay;
                t += finalSwitchDelay;
              }
              output.WriteLine("You will reach the destination at {0}.", t);
            }
            output.WriteLine("Total travel time: {0}",totalTravelTime);
          }
        }
        Console.WriteLine("done.");

        //Finish
        perfLog.Write("Finished");
        perfLog.Close();

        Console.WriteLine("Finished in {0}.Press any key to finish.", perfLog.GetElapsedTime());

        Console.ReadKey();
      }

      private static void ShowVersion()
      {
        Console.WriteLine();
        Console.WriteLine("Version: 1.110518");
        Console.WriteLine();
        Console.WriteLine("Copyright (C) 2010 Chirstos Botsikas");
        Console.WriteLine();
        Console.WriteLine("Writen by Christos Botsikas <cbotsikas@gmail.com>.");
      }

      private static void ShowHelp()
      {
        Console.WriteLine();
        Console.WriteLine("Usage: <program> -fn fromNode [-ft fromTime] -tn toNode");
        Console.WriteLine("              [-n filename] [-debug filename]");
        Console.WriteLine();
        Console.WriteLine("Description:");
        Console.WriteLine("Calculate intermodal time-dependent least-time paths on");
        Console.WriteLine("multimodal transportation networks.");
        Console.WriteLine();
        Console.WriteLine("Options:");
        Console.WriteLine("\t-fn fromNode  Departure Node.");
        Console.WriteLine("\t-ft fromTime  Time to start searhing (minutes).");
        Console.WriteLine("\t           Default: 1");
        Console.WriteLine("\t-tn toNode    Arrival Node.");
        Console.WriteLine("\t-tt toTime    Time to arrive (minutes).");
        Console.WriteLine("\t           Default: 1440");
        Console.WriteLine("\t-n  filename  Network file name.");
        Console.WriteLine("\t           Default: 'network.txt'");
        Console.WriteLine("\t-o  filename  Output file name.");
        Console.WriteLine("\t           Default: 'solution.txt'");
        Console.WriteLine("\t-v          Display the version and exit.");
        Console.WriteLine("\t-h          Print this help.");
        Console.WriteLine("\t--help      Print this help.");
        Console.WriteLine("\t/?          Print this help.");
        Console.WriteLine();
        Console.WriteLine();
        Console.WriteLine("Mail bug reports and suggestions to <cbotsikas@gmail.com>.");
      }

      private static int GetTavelTimeHashed(System.Collections.Generic.Dictionary<string, TimeTravelInfo> hash, Node fromNode, Node ↙
toNode, ModeEntry mode, int atTime, out int waitingTime)
      {
        var key = KeyHelper.GetTimeTravelKey(fromNode.NodeId, toNode.NodeId, mode.modeId, atTime);
        if (hash.ContainsKey(key))
        {
          waitingTime = hash[key].travelWait;
          return hash[key].travelTime;
        }
        else
        {
          waitingTime = int.MaxValue;
          return int.MaxValue;
        }
      }


      private static int GetTavelTime(Node fromNode, Node toNode, ModeEntry mode, int atTime, out int waitingTime)
      {
        if (fromNode == toNode){//if the target is the init
          waitingTime = 0;
          return 0;
        }
```

```
        var request = toNode.timeTables.Where(
          sd =>
          sd.fromNode == fromNode && sd.mode == mode &&
          sd.fromTime <= atTime && sd.toTime >= atTime);

        if (request.Count() > 0)
        {
          waitingTime = 0;
          return request.Last().travelTime;//Peak the last one since it will be the most specific rule
        }else
        {
          //since there is no constant timetable, we look for the next available entry
          request = toNode.timeTables.Where(sd => sd.fromNode == fromNode && sd.mode == mode &&
                                sd.fromTime > atTime).OrderBy(sd => sd.fromTime);

          //Actually the next best case will be calculated later on
          if (request.Count()>0)
          {
            //we will pick the earliest departure time
            var timeTableEntry = request.First();
            waitingTime = timeTableEntry.fromTime - atTime;
            return timeTableEntry.travelTime;
          }else{//there is no such path
            waitingTime = int.MaxValue;
            return int.MaxValue;
          }
        }
      }

      private static int GetSwitchDelayHash(Dictionary<string, List<SwitchDelayInfo>> hash, Node atNode, Node fromNode, Node toNode,
ModeEntry fromMode, ModeEntry toMode, int atTime)
      {
        //Note: The following code allows every node to be an entry point for each available mode in the node
        //If the previous node is the same as the atnode this is an entry point
        if (atNode == fromNode && fromMode == toMode) return 0;
        var key = KeyHelper.GetSwitchKey(atNode.NodeId, fromNode.NodeId, toNode.NodeId,  fromMode.modeId, toMode.modeId);
        if (hash.ContainsKey(key) && hash[key]!=null)
        {
          var request = hash[key].Where(
          sd => sd.fromTime <= atTime && sd.toTime >= atTime);
          if (request.Count() > 0)
            return request.Last().swTime;//Peak the last one since it will be the most specific rule
        }
        return int.MaxValue;
      }
    }
}
```

Αρχείο InputFileHelper.cs

```
using System;
using System.Collections.Generic;

namespace Thesis.Net
{
  static class InputFileHelper
  {
    public static int ParseInt(string line)
    {
      if (string.IsNullOrEmpty(line))  throw new Exception("Null input");
      var substrings = line.Split(new[] { " " }, StringSplitOptions.RemoveEmptyEntries);
      int output;//the value we are looking for
      foreach (var substring in substrings)
      {
        //Return the first numeric input
        if (int.TryParse(substring,out output))
        {
          return output;
        }
      }
      //Otherwise we didn't find a number
      throw new Exception("Not found");
    }
```

```
    public static int[] ParseInts(string line,int numberOfInts)
    {
      if (numberOfInts<=0)  return new int[]{};//If we don't want any ints
      if (string.IsNullOrEmpty(line)) throw new Exception("Null input");
      var substrings = line.Split(new []{" "},StringSplitOptions.RemoveEmptyEntries);
      int tmpVal;//Keep the temporary parsed value
      var output = new List<int>(); //The output
      foreach (var substring in substrings)
      {
        //if it's numeric
        if (int.TryParse(substring, out tmpVal))
        {
          //add it to the list
          output.Add(tmpVal);
          //if we have the requested number, return
          if (output.Count==numberOfInts)
          {
            return output.ToArray();
          }
        }
      }
      //Otherwise we didn't find the number of ints wanted
      throw new Exception(String.Format("Found only {0} ints instead of {1}",output.Count,numberOfInts));
    }
  }
}
```

Αρχείο PerformanceLogger.cs

```
using System;
using System.Diagnostics;
using System.IO;

namespace Thesis.Net
{
  class PerformanceLogger
  {
    PerformanceCounter cpuCounter;
    private StreamWriter debugOutput = null;
    private System.Diagnostics.Stopwatch timer;
    public PerformanceLogger(string debugFile)
    {
      debugOutput = new System.IO.StreamWriter(System.IO.File.OpenWrite(debugFile));
      cpuCounter = new PerformanceCounter();
      cpuCounter.CategoryName = "Processor";
      cpuCounter.CounterName = "% Processor Time";
      cpuCounter.InstanceName = "_Total";
      //Start timing the process
      timer = new System.Diagnostics.Stopwatch();
      timer.Start();
    }

    public static void DisplayCategoryCounters(string category)
    {
      PerformanceCounterCategory mycat = new PerformanceCounterCategory(category);
      // Retrieve the counters.
      var instanceNames = mycat.GetInstanceNames();
      if (instanceNames.Length == 0)
      {
        DisplayCounters(mycat.GetCounters());
      }
      else
      {
        for (int i = 0; i < instanceNames.Length; i++)
        {
          DisplayCounters(mycat.GetCounters(instanceNames[i]));
        }
      }
    }

    private static void DisplayCounters(PerformanceCounter[] counters)
    {
      foreach (var performanceCounter in counters)
      {
        Console.WriteLine("{0} - {1}",performanceCounter.CategoryName, performanceCounter.CounterName);
      }
    }
```

```
        public void Write(string msg)
        {
            debugOutput.WriteLine("{3} {0} CPU: {1} % MEM: {2} KB",msg.Replace(" ","_"),cpuCounter.NextValue(),
GC.GetTotalMemory(false)/1024, timer.Elapsed );
        }

        public TimeSpan GetElapsedTime()
        {
            return timer.Elapsed;
        }

        public void Close()
        {
            timer.Stop();
            debugOutput.Close();
        }

        ~PerformanceLogger()
        {
            this.Close();
        }

    }
}
```

Αρχείο Structures.cs

```
using System;
using System.Collections.Generic;

namespace Thesis.Net
{
    class TimeTravelInfo
    {
        public int travelTime;
        public int travelWait;
    }

    class SwitchDelayInfo
    {
        public int fromTime;
        public int toTime;
        public int swTime;
    }

    class Node
    {
        public Node()
        {
            this.predecessors = new List<Node>();
            this.availableModes = new List<ModeEntry>();
            this.timeTables = new List<TimeTableEntry>();
            this.label = new System.Collections.Concurrent.ConcurrentDictionary<string, int>();
            this.timePath = new System.Collections.Concurrent.ConcurrentDictionary<string, TimePathEntry>();
            this.HasBeenIntoScanEligibleList = false;
        }

        public override string ToString()
        {
            return NodeDescr;
        }

        public int NodeId { get; set; }
        public string NodeDescr { get; set; }
        public List<Node> predecessors { get; set; }
        public List<ModeEntry> availableModes { get; set; }
        public List<TimeTableEntry> timeTables { get; set; }
        //The string key is
        //predecessorID_mode_time
        public System.Collections.Concurrent.ConcurrentDictionary<string, int> label { get; set; }
        //The string key is
        //predecessorID_mode_time
        public System.Collections.Concurrent.ConcurrentDictionary<string, TimePathEntry> timePath { get; set; }

        public bool HasBeenIntoScanEligibleList { get; set; }
```

```
public int getLabel(int predecessorId, int modelId, int time)
    {
      var key = KeyHelper.GetLabelKey(predecessorId, modelId, time);
      if (label.ContainsKey(key))
      {
        return label[key];
      }else
      {
        return int.MaxValue;
      }
    }

    /// <summary>
    /// Try to set the label
    /// </summary>
    /// <param name="predecessorId"></param>
    /// <param name="modelId"></param>
    /// <param name="time"></param>
    /// <returns>if the entry was modified</returns>
    public bool setLabel(int predecessorId, int modelId, int time, int newLabel)
    {
      if (newLabel == int.MaxValue)//if its the max value
        return false; //no change
      var key = KeyHelper.GetLabelKey(predecessorId, modelId, time);
      int clabel;
      bool keyExists = false;
      if (label.ContainsKey(key))
      {
        clabel = label[key];
        keyExists = true;
      }
      else
      {
        clabel =  int.MaxValue;
      }
      if (newLabel < clabel)
      {
        if (keyExists)
          label[key] = newLabel;
        else
          while (!label.TryAdd(key, newLabel)) {System.Threading.Thread.Sleep(100); }
        return true;
      }
      return false;
    }

    public void pushTimePathEntry(int predecessorId, int modelId, int atTime, TimePathEntry tp)
    {
      var key = KeyHelper.GetTimePathKey(predecessorId, modelId,atTime);
      if (timePath.ContainsKey(key))
      {
        timePath[key] = tp;
      }else{
        while(!timePath.TryAdd(key,tp)){System.Threading.Thread.Sleep(100);}
      }
    }

    public TimePathEntry getTimePathEntry(int predecessorId, int modelId, int atTime)
    {
      var key = KeyHelper.GetTimePathKey(predecessorId, modelId,atTime);
      if (timePath.ContainsKey(key))
      {
        return timePath[key];
      }else
      {
        return null;
      }
    }
  }

  class ScanEligibleList: System.Collections.Generic.List<Node>
  {
    private readonly object padlock = new object();
    public void addNode(Node node)
    {

      if (Contains(node)) return;//check before lock
      lock (padlock)
      {
```

```
            if (Contains(node))//check again
                return; //if the node exists skip it
            if (!node.HasBeenIntoScanEligibleList){//If this node has not been inside the list, add it to the end
                Add(node);
            }
            else{
                Insert(0, node);
            }
            node.HasBeenIntoScanEligibleList = true; //The node was added to the list
        }
    }

    public Node getNextNode()
    {
        lock (padlock)
        {
            if (Count > 0) //if we have nodes
            {
                var output = this[0]; //pop it
                Remove(output);
                return output;
            }
            return null; //otherwise null
        }
    }

}

static class KeyHelper
{

    public static string GetLabelKey(int predecessorId, int modeId, int time)
    {
        return String.Format("{0}_{1}_{2}", predecessorId, modeId, time);
    }
    public static string GetTimePathKey(int predecessorId, int modeId, int time)
    {
        return String.Format("{0}_{1}_{2}", predecessorId, modeId, time);
    }

    public static string GetTimeTravelKey(int fromNode, int toNode, int mode, int atTime)
    {
        return String.Format("{0}_{1}_{2}_{3}", fromNode, toNode, mode, atTime);
    }
    public static string GetSwitchKey(int atNode,int fromNode, int toNode, int fromMode, int toMode)
    {
        return String.Format("{0}_{1}_{2}_{3}_{4}", atNode,fromNode, toNode, fromMode, toMode);
    }
}

class TimeTableEntry
{
    public Node fromNode { get; set; }
    public ModeEntry mode { get; set; }
    public int fromTime { get; set; }
    public int toTime { get; set; }
    public int travelTime { get; set; }
}

class ModeEntry
{
    public int modeId { get; set; }
    public string modeDescr { get; set; }
    public override string ToString()
    {
        return modeDescr;
    }
}
class TimePathEntry
{
    public Node bestGoTo { get; set; }
    public ModeEntry mode { get; set; }
    public int switchDelay { get; set; }
    public int waitTime { get; set; }
    public int travelTime { get; set; }
}

}
```

Σύγκριση υλοποίησης FORTRAN και C#

Όσον αφορά στη σύγκριση μεταξύ των δύο υλοποιήσεων που έγιναν στα πλαίσια της διπλωματικής (σε Fortran και σε C#), τρέχοντας τα δύο προγράμματα, παρατηρήσαμε ότι η Fortran ολοκλήρωνε την διαδικασία σε υποπολλαπλάσιο χρόνο από ότι η C# ενώ από την άλλη πλευρά η C# απαιτούσε πολύ λιγότερη μνήμη από ότι η Fortran.

Η διαφορά στον υπολογιστικό χρόνο που απαιτείται από το .NET σε σχέση με την Fortran έγκειται τόσο στο γεγονός ότι η C# δεσμεύει την μνήμη καθώς προχωράει από κόμβο σε κόμβο ενώ στην Fortran η δέσμευση γίνεται μια φορά στην αρχή, όσο και στο γεγονός ότι ο compiler της Fortran πραγματοποιεί βελτιστοποιήσεις στις αριθμητικές πράξεις με αποτέλεσμα να πραγματοποιούνται περισσότερες πράξεις το δευτερόλεπτο σε σχέση με την C#. Και οι δύο εφαρμογές αξιοποιούσαν κατά μέσο όρο το 20% της επεξεργαστικής ισχύος του υπολογιστή αφού και τα δύο προγράμματα αξιοποιούσαν έναν μόνο επεξεργαστή από τους τέσσερις που ήταν διαθέσιμοι.

Αφού εντοπίστηκε η μεγάλη καθυστέρηση της C# σε σχέση με την Fortran έγινε αλλαγή του αλγόριθμου της C# με στόχο να αξιοποιήσει παράλληλη εκτέλεση σε πολλαπλούς πυρήνες. Η αλλαγή αυτή έγινε τροποποιώντας 10 γραμμές κώδικα ενώ για να γίνει κάτι αντίστοιχο στην Fortran θα έπρεπε να εισάγουμε μια νέα βιβλιοθήκη και να αλλάξουμε αρκετά τον υπάρχον κώδικα. Με αυτή την αλλαγή, το πρόγραμμα της C# κατάφερε να επιλύει το ίδιο πρόβλημα στον ίδιο χρόνο με την Fortran αξιοποιώντας σχεδόν το 100% της επεξεργαστικής ισχύς του υπολογιστή. Συνεπώς, παρόλο που η αλλαγή του κώδικα ήταν αρκετά μικρή και παρόλο που τελικά το πρόγραμμα σε C# κατάφερε να επιλύει τα ίδια προβλήματα σχεδόν στον ίδιο χρόνο, η απόδοση της C# θεωρείται αρκετά απογοητευτική αν αναλογιστεί κανείς ότι η Fortran τρέχει σε έναν μόνο πυρήνα, ενώ η C# απασχολούσε και τους τέσσερις.