ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ

ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΙΔΙΚΟΣ ΧΕΙΡΙΣΜΟΣ ΔΕΔΟΜΕΝΩΝ ΜΕΤΑΦΟΡΙΚΩΝ ΔΙΚΤΥΩΝ ΚΑΙ ΚΑΙΝΟΤΟΜΙΚΗ, ΠΡΟΗΓΜΕΝΗ ΣΧΕΤΙΚΗ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΒΡΥΖΑΛΑ ΑΝΔΡΕΑ

Επιβλέπων : Κουκούτσης Ηλίας
Επικ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΙΔΙΚΟΣ ΧΕΙΡΙΣΜΟΣ ΔΕΔΟΜΕΝΩΝ ΜΕΤΑΦΟΡΙΚΩΝ ΔΙΚΤΥΩΝ ΚΑΙ ΚΑΙΝΟΤΟΜΙΚΗ, ΠΡΟΗΓΜΕΝΗ ΣΧΕΤΙΚΗ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΒΡΥΖΑΛΑ ΑΝΔΡΕΑ

Επιβλέπων : Κουκούτσης Ηλίας
Επικ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13 Ιουλίου 2016.

.....	.....	.....
Ηλίας Κουκούτσης	Κ. Παπαοδυσσεύς	Γ. Καμπουράκης
Επικ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

.....

ΒΡΥΖΑΛΑΣ ΑΝΔΡΕΑΣ

Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στους γονείς μου Γιώργο και Ειρήνη

και στα αδέρφια μου Κωνσταντίνο και Ocelia

Περίληψη

Στην εργασία αυτή παρουσίαζεται ο δυσλειτουργικός τρόπος οργάνωσης της γεωσυσχετισμένης πληροφορίας που αφορά τα συγκοινωνιακά δίκτυα και τις επιπτώσεις που έχει στις εφαρμογές που σχετίζονται με τον σχεδιασμό μεταφορών. Βασική αιτία των παραπάνω αποτελεί η προβληματική εσωτερική αρχιτεκτονική των δομών δεδομένων των υπολογιστικών συστημάτων που προσπαθούν να υποστηρίξουν τέτοιες εφαρμογές.

Για την αντιμετώπιση των παραπάνω προβλημάτων παρουσιάζεται ένας νέος τρόπος οργάνωσης των γεωσυσχετισμένων δεδομένων κάνοντας χρήση μίας καινοτομικής εσωτερικής αρχιτεκτονικής της πληροφορίας η οποία προσφέρει νέες δυνατότητες όσον αφορά την πολυμορφία και τον όγκο της αποθηκευμένης πληροφορίας.

Εισάγεται η έννοια του ‘παράγωγου΄ δικτύου καθώς επίσης και νέες δυναμικές δομές κόμβου και συνδέσμου καθώς επίσης και μία προηγμένη δομή γειτνίασης. Επίσης γίνεται αναφορά στα αναγκαία εργαλεία λογισμικού που κατασκευάστηκαν για την βέλτιστη αναπαράσταση των δικτύων μεταφορών και παρουσιάζονται πρακτικές εφαρμογές των παραπάνω εργαλείων, αναδεικνύοντας πως η επεξεργασία, αποθήκευση και απεικόνιση της γεωγραφικά συσχετισμένης πληροφορίας μπορεί να πραγματοποιηθεί μέσω σαφώς πιο αποδοτικών διαδικασιών.

Λέξεις Κλειδιά: Γεωσυσχετισμένη πληροφορία, δομές δεδομένων για συγκοινωνιακά δίκτυα, GIS, Τελεολογία, Θέματα υπεράνω χαρτών, Εργαλεία λογισμικού για βέλτιστη αναπαράσταση δικτύων μεταφορών

Abstract

This paper presents the dysfunctional way that any geo-related information is organized and the impact it has on applications related to transportation planning. The main cause of these problems is the internal architecture of data structures of computer systems that try to support such applications. To address the above problems we shall present a new way of organizing geo-related data using an innovative interior data architecture which offers new possibilities in terms of diversity and volume of stored information. We will present the notion of the ‘produced’ network as well as new dynamic node and link structures as well as a new, innovative adjacency structure. Also we will adress the necessary software tools developed for the optimal representation of transport networks and we will present practical applications of these tools highlighting how the processing, storage and display of geo-related information may be performed through much more efficient procedures.

Keywords: Geo-related information, data structures for transport networks, GIS, Teleology, Themes over maps, Software tools for optimal representation of transport networks

Πίνακας περιεχομένων

Ευρετήριο Εικόνων

1ο Κεφάλαιο: Εισαγωγή

Η επεξεργασία, αποθήκευση και εξαγωγή ορθώς καθορισμένης και οργανωμένης γεωσυσχετισμένης πληροφορίας είναι μία από τις βασικές συνιστώσες για την αποτελεσματικό σχεδιασμό στον τομέα των συγκοινωνιών και μεταφορών. Η συνεχιζόμενη αύξηση του όγκου των δεδομένων σε συνδυασμό με τον πεπαλαιωμένο τρόπο οργάνωσης της πληροφορίας δημιουργεί σημαντικά προβλήματα στον σχεδιασμό, στην κατασκευή και στην διατήρηση συστημάτων γεωσυσχετισμένης πληροφορίας καθώς επίσης και στην ανάπτυξη εφαρμογών για τον σχεδιασμό μεταφορών. Αυτό έχεις ως αποτέλεσμα πληθώρα διαδικασιών που θα έπρεπε να πραγματοποιούνται μέσω αυτοματοποιημένων συστημάτων να έρχονται εις πέρας μέσω επίπονων διαδικασιών από τον χρήστη.

Στην παρούσα εργασία θα αναδείξουμε την ανεπάρκεια του συμβατικού μοντέλου κατασκευής δικτύων και θα παρουσιάσουμε ένα νέο, πληρέστερο μοντέλο για τον σχεδιασμό μεταφορών. Θα προτείνουμε ένα διαφορετικό τρόπο μοντελοποίησης δικτύων στο οποίο εισάγεται η έννοια του ‘παράγωγου’ δικτύου, δηλαδή η κατασκευή διαφορετικών δικτύων για κάθε υποεφαρμογή, βάση πλήρως καθορισμένων κανόνων. Θα προσπαθήσουμε να απεξαρτητοποιήσουμε την κυκλοφοριακή συνέχεια από την γεωγραφική κάνοντας χρήση πολυτροπικών δικτύων και θα παρουσιάσουμε βασικά εργαλεία λογισμικού για πρακτικές εφαρμογές πάνω στα δίκτυα.

Στο **Πρώτο** Κεφάλαιο θα αναφέρουμε βασικές έννοιες για τα δίκτυα μεταφορών, τα διαφορετικά μοντέλα αλλά και τα πιο διαδομένα γεωγραφικά συστήματα πληροφοριών για την αναπαράσταση αυτών των μοντέλων.

Στο **Δεύτερο** Κεφάλαιο, θα εξετάσουμε τον συνήθη τρόπο οργάνωσης της γεωσχετισμένης πληροφορίας. Θα αναλύσουμε τα προβλήματα που δημιουργούνται στον σχεδιασμό μεταφορών καθώς επίσης και τις ιδιαιτερότητες των εφαρμογών που αναφέρονται στα δίκτυα.

Εν συνεχεία στο **Τρίτο** Κεφάλαιο, θα αναδείξουμε την ανάγκη της χρήσης της τελεολογικής δομής δεδομένων και αρχιτεκτονικής γεωσυσχετισμένων πληροφοριών, ενώ θα αναλύσουμε την καινοτομική αρχιτεκτονική ‘Themes

Over Maps’ και τα πλεονεκτήματα που περιλαμβάνει η ενσωμάτωση της στον σχεδιασμό μεταφορών.

Στο **Τέταρτο** Κεφάλαιο, θα προτείνουμε ένα νέο μοντέλο παράστασης δικτύων για τον σχεδιασμό μεταφορών και θα παρουσιάσουμε τον προτεινόμενο τρόπο οργάνωσης της πληροφορίας. Θα αναλύσουμε λεπτομερώς τις δομές του κόμβου και του συνδέσμου καθώς επίσης και την προτεινόμενη καινοτομική δομή γειτνίασης (Adjacency Dictionary).

Στο **Πέμπτο** Κεφάλαιο, θα παρουσιάσουμε τις τεχνολογίες με τις οποίες εργαστήκαμε στο πλαίσιο της διπλωματικής εργασίας καθώς και τα αναγκαία εργαλεία λογισμικού που αναπτύξαμε για την αποτελεσματική εργασία στα δίκτυα μεταφορών. Σε αυτά περιλαμβάνονται τόσο ο σχεδιασμός και κατασκευή κατάλληλης βάσης δεδομένων για την αποθήκευση της πληροφορίας όσο και η ανάπτυξη αλγορίθμων για βασικές διαδικασίες στον σχεδιασμό μεταφορών.

Στο **‘Εκτο** Κεφάλαιο, θα εξετάσουμε τα παραπάνω εργαλεία σε πραγματικές περιπτώσεις σχεδιασμού μεταφορών. Θα δημιουργήσουμε ‘παράγωγα’ δίκτυα, θα αφαιρέσουμε τους αχρείαστους κόμβους, θα εισάγουμε καινούριους κόμβους σε ήδη υπάρχον δίκτυο και τέλος θα εφαρμόσουμε τον αλγόριθμο Dijkstra για την εύρεση βέλτιστης διαδρομής.

Τέλος στο **‘Εβδομο** Κεφάλαιο, θα παρουσιάσουμε τα συμπεράσματα μάς καθώς επίσης και τις προτάσεις για την συνέχιση της εργασίας.

2ο Κεφάλαιο: Προσκήνιο της Ερευνητικής και Αναπτυξιακής Δραστηριότητας στον τομέα των δικτύων μεταφορών

2.1 Βασικές έννοιες στα δίκτυα μεταφορών

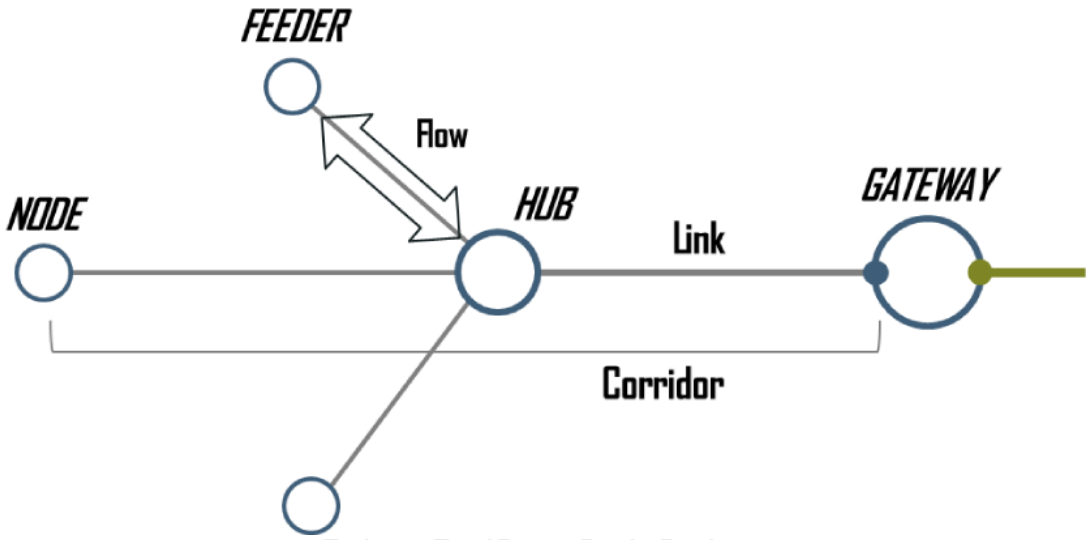
Ο όρος δίκτυο αναφέρεται στο πλαίσιο των διαδρομών μέσα σε ένα σύστημα από τοποθεσίες, που αναγνωρίζονται ως κόμβοι . Μια διαδρομή είναι μια ενιαία σύνδεση μεταξύ δύο κόμβων που αποτελούν μέρος ενός μεγαλύτερου δικτύου το οποίο μπορεί να αναφέρεται σε απτές διαδρομές, όπως δρόμους και σιδηροδρόμους , ή λιγότερο απτές διαδρομές όπως ο αέρας και η θάλασσα.

Τα συστήματα μεταφορών συνήθως παρουσιάζονται χρησιμοποιώντας δίκτυα ως μια αναλογία για τη δομή και τη ροή τους. Τα δίκτυα μεταφορών ανήκουν στην ευρύτερη κατηγορία των χωρικών δικτύων, καθώς ο σχεδιασμός και η εξέλιξή τους είναι φυσικώς περιορισμένη σε αντίθεση με τα μη - χωρικά δίκτυα, όπως η κοινωνική αλληλεπίδραση, την εταιρική οργάνωση και βιολογικά συστήματα, τα οποία συνήθως περιορίζονται από άλλους παράγοντες.

Η εδαφική δομή της κάθε περιοχής αντιστοιχεί σε ένα δίκτυο που περιλαμβάνει όλες τις οικονομικές αλληλεπιδράσεις του. Η υλοποίηση των δικτύων, ωστόσο, είναι σπανίως προμελετημένη, αλλά η ανάγκη για συνεχή βελτίωση έχει ως αποτέλεσμα να γίνονται επενδύσεις στον τομέα αυτό καθώς οι συνθήκες αλλάζουν. Η ρύθμιση των δικτύων είναι το αποτέλεσμα διαφόρων στρατηγικών, όπως η παροχή πρόσβασης και η κινητικότητα σε μια περιοχή, η ενίσχυση ενός ειδικού εμπορικού διαδρόμου κάνοντας έτσι μια συγκεκριμένη λειτουργία και το δίκτυό της, πιο συμφέρουσα σε σχέση με άλλες. Ένα δίκτυο μεταφορών μπορεί να αναφέρεται είτε σε ένα μόνιμο κομμάτι (π.χ. δρόμοι, σιδηρόδρομοι και κανάλια) ή σε μια προγραμματισμένη υπηρεσία (π.χ. αεροπορικές εταιρείες, δημόσια συγκοινωνία, τρένο). Μπορεί να επεκταθεί για να καλύψει διάφορα είδη συνδέσμων μεταξύ σημείων, κατά μήκος του οποίου μπορεί να λάβει χώρα κίνηση. Η σημασία ενός δικτύου σχετίζεται με τη συνδεσιμότητα του. Σύμφωνα με τον νόμο του Metcalfe η αξία ενός δικτύου είναι ανάλογη με το τετράγωνο των συνδεδεμένων κόμβων.

Στη γεωγραφία των μεταφορών, υπάρχουν διάφοροι τύποι δομών μεταφορών που συνδέονται με τα δίκτυα μεταφοράς με βασικά χαρακτηριστικά, τους κόμβους, τις συνδέσεις, τις ροές, κέντρα ή διαδρόμους.

Η δομή του δικτύου κυμαίνεται από κεντρομόλα σε φυγόκεντρα δίκτυα όσον αφορά την προσβασιμότητα που παρέχουν στις εκάστοτε τοποθεσίες. Ένα κεντρομόλο δίκτυο ευνοεί έναν περιορισμένο αριθμό τοποθεσιών, ενώ ένα φυγοκεντρικό δίκτυο δεν σχετίζεται με συγκεκριμένα γεωγραφικά πλεονεκτήματα.



Εικόνα 1 Παράδειγμα δομής δικτύου

Τις τελευταίες δεκαετίες έχουν κάνει την εμφάνιση τους συγκοινωνιακά κέντρα, μια έντονα κεντρομόλος μορφή, που αποτελεί μια δομή ενός προνομιούχου δικτύου για πολλούς τύπους υπηρεσιών μεταφορών, ιδίως για τις αεροπορικές μεταφορές. Παρά το γεγονός ότι τα δίκτυα hub-and-spoke* έχουν οδηγήσει σε βελτίωση της αποτελεσματικότητας του δικτύου, έχουν μειονεκτήματα που συνδέονται με την ευπάθειά τους σε διαταραχές και καθυστερήσεις σε κόμβους, ένα αποτέλεσμα της έλλειψης άμεσων συνδέσεων. Η εμφάνιση των δικτύων hub-and-spoke είναι μια μεταβατική μορφή της ανάπτυξης δικτύων με άξονα τον περιορισμένο όγκο ροής μέσω ενός περιορισμένου αριθμού διαδρομών. Όταν η κίνηση γίνεται επαρκής, άμεση point-to-point υπηρεσίες τείνουν να καθιερωθούν καθώς αποτυπώνουν καλύτερα την προτίμηση των χρηστών.

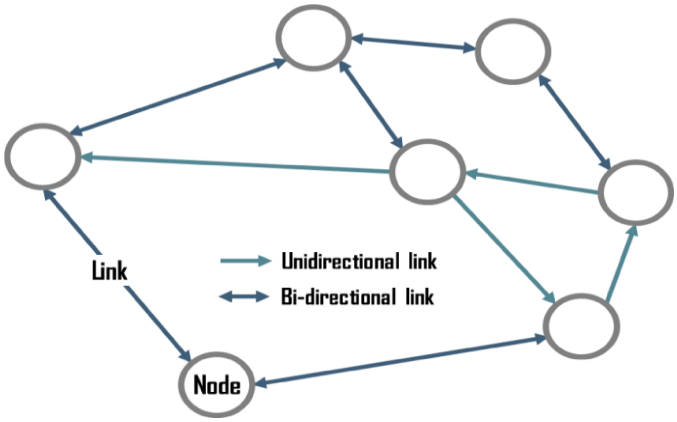
Τα δίκτυα μεταφορών είναι καλύτερα κατανοητά μέσω του επιπέδου χρήσης τους(π.χ. αριθμός επιβατών, τόνοι, οχήματα, χωρητικότητας) παρά με την τοπολογία τους η οποία μπορεί να βασίζεται σε μια δυαδική κατάσταση δηλαδή

την παρουσία ή απουσία συνδέσμων). Οι ανισότητες μεταξύ των τοποθεσιών μπορούν να μετρηθούν βάση της ποσότητας των συνδέσμων μεταξύ των κόμβων και των σχετικών εσόδων που προέρχονται από την ροής της κυκλοφορίας. Πολλές τοποθεσίες εντός ενός δικτύου έχουν υψηλότερη προσβασιμότητα από άλλες, γεγονός το οποίο συχνά συνδέεται με καλύτερες (οικονομικές) ευκαιρίες. Ωστόσο, η ενσωμάτωση οικονομικών διαδικασιών τείνουν να αλλάξουν τις ανισότητες μεταξύ των περιφερειών, κυρίως μέσω του επαναπροσανατολισμού της δομής και της ροής των δικτύων μεταφορών σε διακρατικό επίπεδο.

Η αποτελεσματικότητα ενός δικτύου μπορεί να μετρηθεί μέσω της θεωρίας γραφημάτων και της ανάλυσης του δικτύου. Αυτές οι μέθοδοι στηρίζονται επί της αρχής ότι η απόδοση ενός δικτύου εξαρτάται εν μέρει από την διαρρύθμιση των κόμβων και των συνδέσμων. Προφανώς κάποιες δομές δικτύων έχουν από συνδέσμους που εκπροσωπούν τις συνδέσεις μεταξύ αυτών των τοποθεσιών. Η διάταξη και συνδεσιμότητα του δικτύου που είναι γνωστή ως τοπολογία του, είναι μοναδική για κάθε δίκτυο μεταφοράς. Τα πιο βασικά στοιχεία μιας τέτοιας δομής είναι η γεωμετρία του δικτύου και το επίπεδο της υψηλότερο βαθμό πρόσβασης από τους άλλους, αλλά πρέπει να δοθεί ιδιαίτερη προσοχή στη βασική σχέση μεταξύ των εσόδων και των δαπανών συγκεκριμένων δικτύων μεταφοράς καθώς οι τιμές τείνουν να επηρεάζονται από τη δομή των δικτύων μεταφοράς έχοντας σημαντικές επιπτώσεις στο κόστος μεταφοράς [1].

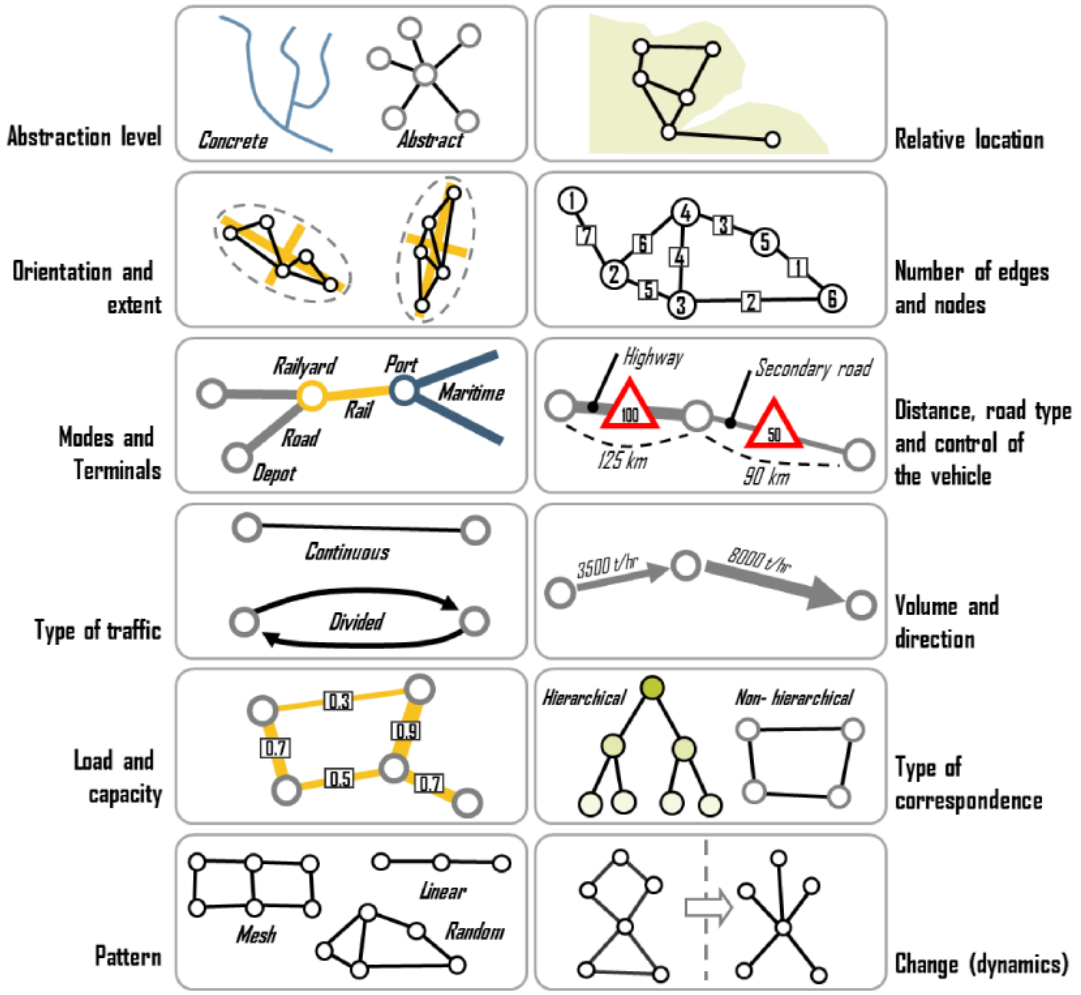
2.2 Τοπολογία και τυπολογία των δικτύων

Τα δίκτυα μεταφορών, όπως και πολλά δίκτυα, γενικά αποτελούνται από ένα σύνολο τοποθεσιών και μια σειρά συνδεσιμότητας.



Εικόνα 2 Παράδειγμα τοπολογίας δικτύου

Τα δίκτυα μεταφορών μπορούν να ταξινομηθούν σε συγκεκριμένες κατηγορίες ανάλογα με τις τοπολογικές ιδιότητες που τις περιγράφουν. Έτσι είναι δυνατόν να καθοριστεί μια βασική τυπολογία των δικτύων μεταφορών που σχετίζεται με τις γεωγραφικές ιδιότητες και τα δομικά χαρακτηριστικά του εκάστοτε δικτύου.



Εικόνα 3 Παράδειγμα τυπολογίας δικτύου

Η φυσική σημασία ενός δικτύου ποικίλλει σε συνάφεια ανάλογα με το μοντέλο στο οποίο αναφερόμαστε. Οι δρόμοι και οι σιδηροδρομικές γραμμές αποτελούνται από απτές υποδομές, ενώ θαλάσσιες και αεροπορικές μεταφορές παραμένουν αόριστα καθορισμένες, λόγω της υψηλότερης χωρικής ευελιξίας τους, εκτός από τους τερματικούς σταθμούς. Τα δίκτυα των ποταμιών συνήθως σχηματίζουν λεκάνες και μπορεί να χαρακτηριστούν ως δέντρα ή δενδρογράμματα. Ως εκ τούτου, υπάρχουν τρεις τύποι φυσικών χώρων στους οποίους καθορίζονται τα δίκτυα μεταφορών και όπου το καθένα αντιπροσωπεύει μια συγκεκριμένη λειτουργία εδαφικής κατοχής:

- **Σαφώς καθορισμένοι και οριοθετημένοι.** Ο χώρος που καταλαμβάνεται από το δίκτυο μεταφορών προορίζεται αυστηρά για την αποκλειστική χρήση του και μπορεί να εντοπιστεί σε ένα χάρτη. Μορφές ιδιοκτησίας μπορούν επίσης να καθοριστούν με σαφήνεια. Σημαντικά παραδείγματα αποτελούν τα οδικά, και σιδηροδρομικά δίκτυα.
- **Αόριστα καθορισμένοι και οριοθετημένοι.** Ο χώρος των δικτύων αυτών μπορεί να μοιραστεί με άλλα δίκτυα και δεν αποτελεί το αντικείμενο οποιασδήποτε συγκεκριμένης ιδιοκτησίας, παρά μόνο των δικαιωμάτων διέλευσης. Παραδείγματα αποτελούν το εναέριο δίκτυο και τα δίκτυα θαλάσσιων μεταφορών.
- **Χωρίς ορισμό.** Ο χώρος δεν έχει απτή έννοια, εκτός από την απόσταση που επιβάλλει με τους κόμβους να αποτελούν την βασική δομή. Μερικώς έλεγχος και ιδιοκτησία είναι δυνατή, αλλά θα πρέπει να υπάρξουν συμφωνίες για κοινή χρήση. Παραδείγματα αποτελούν το ραδιόφωνο, η τηλεόραση, το WiFi και τα δίκτυα κινητής τηλεφωνίας.

Τα δίκτυα παρέχουν ένα είδος υπηρεσιών μεταφορών που είναι άμεσα συσχετισμένα με το κόστος τους . Ένα βέλτιστο δίκτυο θα είναι ένα δίκτυο εξυπηρέτησης προς όλες τις πιθανές τοποθεσίες , αλλά μια τέτοια υπηρεσία θα έχει υψηλό κεφαλαιουχικό και λειτουργικό κόστος. Οι υποδομές των μεταφορών έχει καθιερωθεί πάνω σε ασυνεχή δίκτυα δεδομένου ότι πολλά δεν χτίστηκαν την ίδια χρονική περίοδο, από τον ίδιο φορέα ή με την ίδια τεχνολογία. Ως εκ τούτου, τα λειτουργικά δίκτυα σπάνια εξυπηρετούν όλα τα τμήματα της επικράτειας με άμεσο τρόπο. Τα δίκτυα χαρακτηρίζονται επίσης και ανάλογα με το συνολικές τους ιδιότητες:

- **Κανονικό δίκτυο.** Ένα δίκτυο όπου όλοι οι κόμβοι έχουν τον ίδιο αριθμό ακμών. Στο ίδιο πνεύμα, ένα τυχαίο δίκτυο είναι ένα δίκτυο που σχηματίζεται από τυχαίες διαδικασίες. Ενώ κανονικά δίκτυα τείνουν να συνδέονται με υψηλά επίπεδα χωρικής οργάνωσης (π.χ. ένα πλέγμα πόλης), τυχαία δίκτυα τείνουν να συνδέονται με ευκαιριακές ευκαιρίες ανάπτυξης , όπως η πρόσβαση σε έναν συγκεκριμένο πόρο.
- **Δίκτυα "μικρόκοσμου".** Ένα δίκτυο με πυκνές συνδέσεις μεταξύ στενών γειτόνων και λίγες, αλλά ζωτικής σημασίας συνδέσεις μεταξύ

μακρινών γειτόνων. Τέτοια δίκτυα είναι ιδιαίτερα ευάλωτα στις καταστροφικές αποτυχίες γύρω από μεγάλα κέντρα.

- **Δίκτυα χωρίς κλίμακα.** Ένα δίκτυο που έχει μια ισχυρή ιεραρχική διάσταση, με λίγες κορυφές που έχουν πολλές συνδέσεις και πολλές κορυφές που έχουν λίγες συνδέσεις. Τέτοια δίκτυα εξελίσσονται μέσα από την δυναμική των προτιμησιακών συνδεσιμότητας με την οποία οι νέοι κόμβοι που προστίθενται στο δίκτυο θα συνδεθούν κυρίως με μεγαλύτερους κόμβους αντί να συνδέονται τυχαία .

Η διερεύνηση των αλληλεξαρτήσεων μεταξύ διάφορων δικτύων μεταφορών, ιδίως όταν αυτά είναι διαφορετικής φύσης και δομής, είναι ιδιαιτέρως δύσκολη. Μερικές κρίσιμες πτυχές και προβλήματα που σχετίζονται με τις σχέσεις μεταξύ των δικτύων είναι ως εξής:

- **Συνεξέλιξη.** Διαφορετικά δίκτυα μεταφορών μπορούν να ακολουθήσουν παρόμοιες ή διαφορετικές διαδρομές με βάση την χωρική εγγύτητα και τη εξάρτηση της διαδρομής από την οικονομική ανάπτυξη μιας περιοχής. Στις βασικές περιοχές υπάρχει ευρύτερη ποικιλία δικτύων σε αντίθεση με απομακρυσμένες περιοχές.
- **Συμπληρωματικότητα.** Ορισμένες περιοχές μπορεί να είναι κεντρικές σε ένα δίκτυο, αλλά περιφερικές σε ένα άλλο, ανάλογα με την ειδικότητα, τη λειτουργία τους και με την κλίμακα της ανάλυσης (τερματικό, πόλη, περιοχή , χώρα). Η συμπληρωματικότητα μεταξύ των δικτύων μπορεί να μετρηθεί με βάση τον αριθμό των κοινών κόμβων και συνδέσμων.
- **Διαλειτουργικότητα.** Τυπικά, το φορτίο ρέει από ένα θαλάσσιο δίκτυο και μεταπηδά σε ένα οδικό δίκτυο ή από μια δομή χωρίς κλίμακα σε μία κανονική δομή, ακολουθώντας έτσι διαφορετικές τοπολογίες που δεν συνδυάζονται εύκολα. Τα τερματικά αέρος και θαλάσσης παραμένουν λίγα στον κόσμο, λόγω της δυσκολίας να συνδυαστούν και ενσωματωθούν εναέρια και θαλάσσια δίκτυα στις ίδιες φυσικές θέσεις.
- **Ευαισθησία.** Πώς αλλαγές σε ένα δίκτυο επηρεάζουν ένα άλλο δίκτυο, σε παγκόσμιο ή τοπικό επίπεδο. Αυτό είναι ιδιαίτερα σημαντικό για δύο δίκτυα τα οποία μοιράζονται κοινούς κόμβους, θέτοντας έτσι το πρόβλημα της επαναδρομολόγησης της ροής μέσα από εναλλακτικές διαδρομές και τοποθεσίες.

2.3 Μοντελοποίηση δικτύων μεταφορών

Η μοντελοποίηση των συστημάτων μεταφορών και η χρήση τους έχει δύο βασικούς σκοπούς:

- για την εκτίμηση των χαρακτηριστικών ενός υπάρχοντος συστήματος μεταφορών και την χρήση του, στην περίπτωση που είναι δύσκολο να παρατηρηθούν
- για την εκτίμηση των χαρακτηριστικών ενός υπάρχοντος συστήματος μεταφορών και την χρήση του, στην περίπτωση που αυτά δεν υφίστανται ακόμα

Ο δεύτερος σκοπός προκύπτει όταν το σύστημα που μας ενδιαφέρει δεν έχει ακόμη κατασκευαστεί ή περιλαμβάνει, αλλαγές οι οποίες δεν έχουν ενσωματωθεί, ή όταν πρέπει να διερευνηθούν πρωτοφανή πρότυπα χρήσης ενός υπάρχοντος συστήματος.

Η μοντελοποίηση εύκολα παρατηρήσιμων χαρακτηριστικών μίας υφιστάμενης κατάστασης έχει επίσης σκοπό, να βοηθήσει στην τεκμηρίωση της ίδιας της μοντελοποίησης. Ωστόσο, εκτός από την ικανότητα αναπαραγωγής μιας υπάρχουσας κατάστασης θα πρέπει να δίνεται και η δυνατότητα αναπαραγωγής παρατηρούμενων επιδράσεων μέσω συγκεκριμένων μεταβολών στο πλαίσιο της ήδη υπάρχουσας κατάστασης.

Μια βασική αρχή για την μοντελοποίηση είναι η διαχείριση ενός υπάρχοντος συστήματος. Πρώτον για να αυξηθεί η συλλογή δεδομένων με σκοπό την παρακολούθηση της απόδοσης του, αλλά κυρίως για να ενημερώσει τη λειτουργία των διαδικτυακών εργαλείων διαχείρισης, όπως ο έλεγχος των σημάτων ανταπόκρισης κυκλοφορίας και συστημάτων πληροφόρησης οδηγών.

Η δεύτερη βασική αρχή είναι ότι η αλλαγή μελετάται. Είτε για παρατήρηση πιθανής δυσλειτουργίας ή για ενσωμάτωση διαδικασιών προς εξυπηρέτηση του συστήματος. Η μοντελοποίηση ενός συστήματος μπορεί να βοηθήσει στη διάγνωση των προβλημάτων που σχετίζονται με αυτό και την ανάπτυξη προτάσεων για πιθανές αλλαγές. Επίσης η μοντελοποίηση ενός συστήματος με προτεινόμενες αλλαγές μπορεί να βοηθήσει στο να αποφασιστεί ποιες από τις αλλαγές θα ενσωματωθούν αλλά και τον τρόπο ενσωμάτωσής τους.

Όλα αυτά προφανώς απαιτούν μία σαφή ερμηνεία των αποτελεσμάτων της μοντελοποίησης έχοντας ως βάση την κατανόηση του πεδίου εφαρμογής και τα όριά της. Απαιτείται ισορροπία μεταξύ της επιθυμίας για ρεαλισμό και της αξιοπιστίας ενός συστήματος. Αυτό απαιτεί ένα υψηλό επίπεδο επικοινωνίας μεταξύ των ατόμων που κατασκευάζουν τα μοντέλα (συνήθως μαθηματικοί) και των χρηστών των μοντέλων (συνήθως ένα μίγμα από σχεδιαστές, μηχανικούς, οικονομολόγους και φορείς λήψης αποφάσεων) [2].

2.4 Η εξέλιξη της μοντελοποίησης των δικτύων μεταφορών

Το αρχικό κίνητρο για την μοντελοποίηση των συστημάτων μεταφορών και την χρήση τους, ήταν η ανάγκη των βιομηχανοποιημένων κοινωνιών να προσαρμοστούν στην ευρεία ιδιοκτησία και χρήση των αυτοκινήτων. Η ανάγκη αυτή έγινε σαφής στη δεκαετία του 1940 και του 1950 ταυτόχρονα με την έλευση των ψηφιακών υπολογιστών και την ανάπτυξη της επιστήμης του προγραμματισμού, κάνοντας εφικτή την μεγάλης κλίμακας αριθμητική μοντελοποίηση. Οι πρώιμες μορφές μοντελοποίησης δικτύων στον τομέα των μεταφορών σχετίζονταν με τις αστικές οδούς και βασιζόντουσαν στην κοινή λογική των ανθρώπων για την αντιμετώπιση απλών πρακτικών προβλημάτων με νέα εργαλεία και λίγη συνειδητοποίηση ότι η επιστήμη των μαθηματικών θα μπορούσε να τους βοηθήσει. Οι μαθηματικές εξελίξεις στον τομέα αυτό αναπτύσσονταν παράλληλα, αντλώντας ταυτόχρονα καινοτομίες στον μαθηματικό προγραμματισμό και στη θεωρία πιθανοτήτων αλλά ήταν μόνο κατά τη διάρκεια της δεκαετίας του 1960 όπου η θεωρία και η πρακτική εφαρμογή άρχισαν πραγματικά να συγκλίνουν.

Ο πυρήνας της μαθηματικής ανάπτυξης ήταν το πρόβλημα της ανάθεσης της κυκλοφορίας. Αυτό είχε διατυπωθεί αρχικά ως σταθερή κατάσταση κατανομής στα δρομολόγια μίας συγκεκριμένης μήτρας ζήτησης για τις μετακινήσεις οχημάτων μεταξύ των σημείων εισόδου και εξόδου προς ένα πρότυπο οδικό δίκτυο με χωρητικότητα που υπερβαίνει την καθορισμένη ζήτηση. Το υποτιθέμενο κόστος διέλευσης μέσω ενός συνδέσμου είναι ίδιο για όλα τα οχήματα και οι διαδρομές πρέπει να καθοριστούν με τέτοιο τρόπο έτσι ώστε ανάμεσα σε κάθε ζεύγος εισόδου-εξόδου όλες οι χρησιμοποιηθείσες διαδρομές

να έχουν το ίδιο κόστος και καμία μη χρησιμοποιηθείσα διαδρομή να μην έχει χαμηλότερο κόστος. Η σύνθεση αυτή αποτελεί το βασικό πρόβλημα.

Η εφαρμογή της μαθηματικής επιστήμης σε αυτό το πρόβλημα βοήθησε στο να αποσαφηνιστεί γρήγορα η διάκριση μεταξύ της *ισορροπίας χρήστη* (αποτέλεσμα του γεγονότος ότι οι ταξιδιώτες επιλέγουν τις διαδρομές τους ώστε να ελαχιστοποιηθεί το υποτιθέμενο κόστος του ταξιδιού τους) , και ενός συστήματος βέλτιστης οργάνωσης διαδρομών των ταξιδιωτών ώστε να ελαχιστοποιηθεί το συνολικό κόστος όλα τα ταξίδια τους.

Μία από τις πρώτες συνέπειες της διατύπωσης του βασικού προβλήματος ήταν η παραγωγή ειδικών διοδίων με χρεώσεις διέλευσης συνδέσμων κάνοντας την προκύπτον σύστημα ισορροπίας χρήστη βέλτιστο από άποψη κόστους. Το απαιτούμενο κόστος διοδίων για κάθε σύνδεσμο εξαρτάται κυρίως από την ροή του συνδέσμου πράγμα το οποίο σημαίνει ότι το κόστος αυτό μπορεί να προσεγγιστεί καλύτερα στην πράξη.

Οι βασικές γραμμές της γενίκευσης και επέκτασης του βασικού προβλήματος είναι οι ακόλουθες:

- Εξάρτηση του κόστους κάθε συνδέσμου από την ροή του προς άλλους συνδέσμους
- Διαφορετικά είδη οχημάτων σε κάθε δίκτυο
- Δίκτυα δημοσίων μέσων μεταφοράς και δίκτυα δρόμων καθώς επίσης και η επιλογή ανάμεσα στις δημόσιες μεταφορές και στα ιδιωτικά οχήματα
- Ενδιαφέρον για τη διαχείριση της κυκλοφορίας και την προσαρμογή της στο σύστημα προς μοντελοποίηση
- Διαφορές μεταξύ των ταξιδιωτών στην αντίληψη του κόστους
- Εξάρτηση της ζήτησης βάση της αντίληψης του κόστους των ταξιδιών και των χαρακτηριστικών των περιοχών που αντιπροσωπεύονται από σημεία εισόδου και εξόδου
- Διακύμανση της ζήτησης μέσα σε μία συγκεκριμένη περίοδο, συμπεριλαμβανομένης της προσωρινής υπερφόρτωσης
- Μεταβολή της αντίληψης του κόστους με το πέρασμα του χρόνου

- Εκτίμηση της κατανομής πιθανότητας του μοτίβου του ταξιδιού και όχι στην επιλογή ενός προκαθορισμένου μοτίβου

Για το βασικό πρόβλημα, ακριβείς συνθήκες για την ύπαρξη, τη μοναδικότητα και τη σταθερότητα της λύσης έχουν από καιρό καθιερωθεί μέσω της διατύπωσης του ως ένα πρόβλημα βελτιστοποίησης και ανισότητας και σταδιακά έχουν αναπτυχθεί πιο αποτελεσματικοί αλγόριθμοι για τον υπολογισμό της λύσης σε μεγάλης κλίμακας δίκτυα.

Η εφευρετικότητα στην κατασκευή του δικτύου, η ανάλυση της κίνησης στους δρόμους και στα δημόσια συστήματα μεταφορών, καθώς και η διερεύνηση των εξωτερικών επιδράσεων της κυκλοφορίας έχουν επιτύχει σε μεγάλο βαθμό ρεαλιστική ποσοτική περιγραφή των χαρακτηριστικών του δικτύου, συμπεριλαμβανομένων και αλγεβρικών συναρτήσεων υπολογισμού της απόδοσης ενός συνδέσμου. Ο τρόπος με τον οποίο διάφοροι οργανισμοί μπορούν να χρησιμοποιούν εργαλεία διαχείρισης είναι σε μεγάλο βαθμό κατανοητός και πολλά εμπειρικά στοιχεία έχουν συγκεντρωθεί σχετικά με τα χαρακτηριστικά ταξιδιώτη γεγονός που οδηγεί στο συμπέρασμα ότι ο ρεαλισμός των μοντέλων εξαρτάται κυρίως από τον βαθμό στον οποίο αυτά αναπαριστώνται με μαθηματικό τρόπο [3].

2.5 Μοντέλα δικτύων μεταφορών

2.5.1 Μικρο-οικονομικά μοντέλα χρήσης της γης

Τα μικρο-οικονομικά μοντέλα έχουν κοινά χαρακτηριστικά που καθορίζουν τη χρήση της γης, ως αποτέλεσμα του μηχανισμού της αγοράς, στην οποία τα νοικοκυριά και οι επιχειρήσεις ανταγωνίζονται για χώρο, δημιουργώντας ένα μοτίβο ισορροπίας του ενοικίου της γης. Την ίδια στιγμή, οι τιμές ισορροπίας επιτρέπουν την βέλτιστη κατανομή της γης προς τα νοικοκυριά και τις επιχειρήσεις, και αυτά, με τη σειρά τους μεγιστοποιούν τις διαδικασίες κοινής ωφέλειας. Το μοντέλο του Von Thunen είναι βασικό για την χωρική οικονομική θεωρία και έχει επεκταθεί με διάφορους τρόπους. Αυτό το μοντέλο εξηγεί την επίδραση του κόστους μεταφοράς για τη θέση των δραστηριοτήτων και των λειτουργιών της αγοράς γης. Το μοντέλο έχει επίσης επεκταθεί για να συμπεριλάβει διαδικασίες γύρω από το οικονομικό σύστημα [4].

2.5.2 θεωρία τυχαίας χρησιμότητας και διακριτά μοντέλα επιλογής

Το μοντέλο αυτό θεωρείται ως μια γέφυρα μεταξύ των μοντέλων της μικροοικονομικής θεωρίας και των μοντέλων χωρικής αλληλεπίδρασης. Αυτό επιτρέπει την ενοποίηση των θεωριών ή τουλάχιστον την χρήση ορισμένων αρχών και από τα δύο μοντέλα. Η ανάλυση των διακριτών μοντέλων επιλογής χρησιμοποιεί την αρχή της μεγιστοποίησης χρησιμότητας. Η διαμόρφωση μίας διαδικασίας επιλογής καθορίζεται ώστε να επιλέγει την λύση με την υψηλότερη χρησιμότητα μεταξύ εκείνων που διατίθενται. Ένα λειτουργικό μοντέλο αποτελείται από λειτουργίες παραμετροποιημένης χρησιμότητας όσον αφορά στην παρατήρηση ανεξάρτητων μεταβλητών και αγνώστων παραμέτρων, και οι τιμές τους υπολογίζονται από ένα δείγμα παρατηρήσιμων επιλογών που πραγματοποιήθηκαν από τους ιθύνοντες όταν έρχονται αντιμέτωποι με μια κατάσταση επιλογής. Οι πρώτες εφαρμογές των διακριτών μοντέλων επιλογής έγιναν για τη δυαδική επιλογή του τρόπου μετακίνησης. Μερικές από αυτές τις μελέτες εστιάζονται στην εκτίμηση της «αξίας του χρόνου» και στην σχέση μεταξύ του χρόνου ταξιδιού και του κόστους ταξιδιού. Αυτή η εκτίμηση έχει χρησιμοποιηθεί για να εκχωρήσει μια νομισματική αξία στα ταξιδιωτική εξοικονόμηση χρόνου κατά την αξιολόγηση εναλλακτικών συγκοινωνιακών έργων. Άλλοι ερευνητές έδωσαν έμφαση την ανάπτυξη μοντέλων για τις προβλέψεις των μεριδίων αγοράς των εναλλακτικών τρόπων μεταφοράς. Οι διαδικασίες κατασκευής διακριτών μοντέλων επιλογής στη δεκαετία του 1970 ήταν προσανατολισμένες προς τα μοντέλα επιλογής λειτουργίας με περισσότερες από δύο εναλλακτικές λύσεις και σε εφαρμογές που σχετίζονται με ταξίδια, όπως τον προορισμό του ταξιδιού, την συχνότητα ταξιδιών, της ιδιοκτησίας αυτοκινήτων, της τοποθεσίας των κατοικημένων περιοχών, και τη στέγαση. Μελέτες για την επιλογή του τρόπου μετακίνησης προς τους χώρους εργασίας έχουν χρησιμοποιήσει διαφορετικούς τύπους δεδομένων από διαφορετικές αστικές περιοχές, να ανέπτυξαν ένα πιο ολοκληρωμένο μοντέλο με κοινωνικοοικονομικές μεταβλητές και δοκιμάστηκε η ακρίβεια πρόβλεψης των μοντέλων με τα στοιχεία που είχαν στην διάθεση τους πριν και μετά τις αλλαγές στον τρόπο μεταφοράς [5].

2.5.3 Μοντέλα που περιλαμβάνουν μεταβλητές αστικού χαρακτήρα

Εκτός από τη χρήση της γης και τις μεταβλητές των συστήματος μεταφοράς, έχει προταθεί ότι η κατηγοριοποίηση βάση του μεγέθους των αστικών κέντρων, της συγκέντρωσης δραστηριοτήτων, και των γεωγραφικών συμπλεγμάτων μπορεί να ενισχύσει την μεταφορά πληροφορίας μεταξύ διαφορετικών μοντέλων.

Οι μελέτες δείχνουν ότι ο αστικός χαρακτήρα κάθε περιοχής φέρει σημαντική επιρροή στη ζήτηση της μετακίνησης και το μοντέλο μεταφοράς ποικίλλει ανάλογα με τα ύψος της ζήτησης και τις προδιαγραφές του μοντέλου. Τα συγκεκριμένα ευρήματα περιλαμβάνουν: (α) το είδος της συγκέντρωσης δραστηριότητας έχει σημαντική επίδραση στην συχνότητα ταξιδιών για τις μη μητροπολιτικές περιοχές και στην διάρκεια του ταξιδιού και για τις μητροπολιτικές και μη μητροπολιτικές περιοχές (α) η επιρροή του μεγέθους των αστικών περιοχών για την επιλογή λειτουργίας και διάρκειας του ταξιδιού είναι σημαντική για τις μητροπολιτικές περιοχές (γ) η επίδραση των γεωγραφικών χαρακτηριστικών στην ζήτηση της μετακίνησης έχει άμεσο αντίκτυπο στην διάρκεια του ταξιδιού και στην συχνότητα των ταξιδιών.

2.5.4 Ατομικές επιλογές προς τη γενική ισορροπία

Αυτή η προσέγγιση παρουσιάζει στατικά μοντέλα, κάτω υπό σταθερές συνθήκες, οι οποίες προβλέπουν όλες τις σημαντικές διαστάσεις της ταξιδιωτικής συμπεριφοράς, κάτω από μια ανάλυση της χωρικής δομής. Το βασικό πρόβλημα της ατομικής επιλογής της διαδρομής κάτω από μη αμφισβητούμενες συνθήκες, τόσο για τους επιβάτες αυτοκινήτων όσο και για επιβάτες μέσων μαζικής μεταφοράς, λύθηκε σε ατομικό επίπεδο από την εφαρμογή του "Minimum Cost Route" που αναπτύχθηκε από τον Speiss. Στην περίπτωση του στοχαστικού αυτοκινήτου, το πρόβλημα επιλογής διαδρομή λύθηκε από την εφαρμογή του αλγορίθμου "Stoch". Σε όλες τις περιπτώσεις, η έννοια του "εκπροσώπου ταξιδιώτη" εφαρμόστηκε με σκοπό την ανάκτηση της συνολικής ζήτησης διαδρομών για να λυθεί το πρόβλημα της μεγιστοποίησης χρησιμότητας του "εκπροσώπου ταξιδιώτη".

Στη συνέχεια, άλλες διαστάσεις της ζήτησης της μετακίνησης, συμπεριλαμβανομένων προορισμού, επιλογή λειτουργίας, επιλογή διαδρομής

(υπό την απουσία της κυκλοφοριακής συμφόρησης), συνδυάζονται.Η μοντελοποίηση της επιλογής διαδρομής γίνεται με πιο την ρεαλιστική παραδοχή της κυκλοφοριακής συμφόρησης του δικτύου. Σύμφωνα με την υπόθεση των διάφορων εξωτερικών παραγόντων που μπορούν να επηρεάσουν το μοντέλο, οι πτυχές της ζήτηση της μετακίνησης, τον προορισμό, της επιλογής διαδρομής δύνανται να μοντελοποιηθούν. Ένα συνδυασμένο μοντέλο ισορροπίας των αστικών μετακινήσεων και μεταφοράς προϊόντων, παράγεται από την ανάγκη για υποστήριξη μιας αστικής δραστηριότητας που αναλαμβάνεται από μεμονωμένους ταξιδιώτες. Έτσι παρουσιάζεται μια σαφής, πλήρης αναπαράσταση των αλληλεπιδρώντων συμπεριφορών των ταξιδιωτών, των προμηθευτών πρώτων υλών και των φορτωτών στο πλαίσιο του χωρικού ανταγωνισμού. Έχει αποδειχθεί ότι υπό γενικές συνθήκες, το μοντέλο έχει μια μοναδική λύση ενώ στη συνέχεια επεκτάθηκε σε περιπτώσεις πολλαπλών επιπέδων εμπορικής διαπραγμάτευσης και πολλαπλών εμπορευμάτων [6].

2.5.5 Από τη γενική ισορροπία σε δυναμικά μοντέλα σχεδιασμού / εξελικτικά μοντέλα

Η παραδοχή αυτής της συμπεριφοράς του μοντέλου είναι ότι η προσφορά και η ζήτηση δεν είναι σε τέλεια ισορροπία, διότι το κόστος της μετακίνησης και η εναλλαγή θέσεων εργασίας είναι υψηλό και η εκχώρηση της κυκλοφορίας μπορεί να μην είναι σε τέλεια ισορροπία, γιατί η γνώση δεν είναι τέλεια. Εδώ η ζήτηση σε ένα δεδομένο έτος εξαρτάται από τη ζήτηση του προηγούμενου έτους. Το μοντέλο αναδιανέμει ένα κλάσμα των εργασιακών διαδρομών κάθε χρόνο και συνδέει την μετεγκατάσταση ενός νοικοκυριού με τη λήψη μιας νέας θέσης εργασίας και λαμβάνει υπ’όψιν τις μεταβολές στην κατανομή. Το πλαίσιο μοντελοποίησης θεωρεί την ισορροπία και την εξέλιξη ως δύο πόλους με δύο ενδιάμεσους συνδυασμούς των μεθόδων, ανάλογα με το χρονικό καθεστώς της λήψης της απόφασης (day-to-day ή year-to-year). Day-to-day αποφάσεις περιλαμβάνουν την επιλογή διαδρομής, επιλογή λειτουργίας, επιλογή χρόνου αναχώρησης, και επιλογή προορισμού του ταξιδιού μη συσχετιζόμενα με την εργασία. Year-to-year αποφάσεις περιλαμβάνουν μετεγκατάσταση ή ταξίδι εργασίας, η ιδιοκτησία αυτοκινήτων. Οι αποφάσεις δεν αλληλοαποκλείονται, οπότε ενδεχόμενη year-to-year απόφαση μπορεί να επιφέρει αλλαγές στις day-

to-day αποφάσεις. Επιπλέον, οι μεταβλητές του συστήματος, όπως το δίκτυο, η χρήση γης και τα δημογραφικά στοιχεία ποικίλλουν σε ετήσια βάση. Το νέο στοιχείο εδώ είναι η απόφαση να μετεγκατάστασης. Αυτή η προσέγγιση έχει τη δυνατότητα να χρησιμοποιήσει παρατηρούμενα δεδομένα πιο εύκολα και έτσι περιορίζει τη μοντελοποίηση κυρίως σε αλλαγές στη συμπεριφορά ενώ προσθέτει περισσότερο ρεαλισμό στην έννοια του μοντέλου [7].

2.6 Χαρτογραφώντας το τοπίο των Γεωγραφικών Πληροφοριακών Συστημάτων (GIS)

Ένα σύστημα γεωγραφικών πληροφοριών ή γεωγραφικό σύστημα πληροφοριών (GIS) είναι ένα σύστημα σχεδιασμένο με σκοπό την συλλογή, αποθήκευση, το χειρισμό, την ανάλυση, τη διαχείριση, και την παρουσίαση κάθε είδους χωρικών ή γεωγραφικών δεδομένων.

Σε μια γενική έννοια, ο όρος περιγράφει κάθε πληροφοριακό σύστημα που ενσωματώνει, αποθηκεύει, επεξεργάζεται, αναλύει, διαμοιράζει και απεικονίζει γεωγραφικές πληροφορίες. Οι GIS εφαρμογές αποτελούν εργαλεία που επιτρέπουν στους χρήστες να δημιουργήσουν διαδραστικές ερωτήσεις (αναζητήσεις που έχουν δημιουργήσει οι χρήστες), να αναλύσουν χωρικές πληροφορίες, να επεξεργαστούν δεδομένα σε χάρτες, και να απεικονίσουν τα αποτελέσματα όλων αυτών των ενεργειών.

Ο όρος GIS είναι ένας ευρύς όρος που μπορεί να αναφέρεται σε έναν αριθμό διαφορετικών τεχνολογιών, διαδικασιών και μεθόδων. Είναι άμεσα συνδεδεμένο με πολλές λειτουργίες και έχει πολλές εφαρμογές που σχετίζονται με την μηχανική, τον προγραμματισμό, τη διαχείριση, τις μεταφορές, τις τηλεπικοινωνίες και τις επιχειρήσεις. Για αυτό τον λόγο οι GIS εφαρμογές μπορούν να αποτελέσουν το θεμέλιο για γεωγραφικές υπηρεσίες που βασίζονται στην ανάλυση και στην οπτικοποίηση. Τα GIS μπορούν να συσχετίσουν φαινομενικά άσχετες πληροφορίες, χρησιμοποιώντας την γεωγραφική θέση ως βασική μεταβλητή.

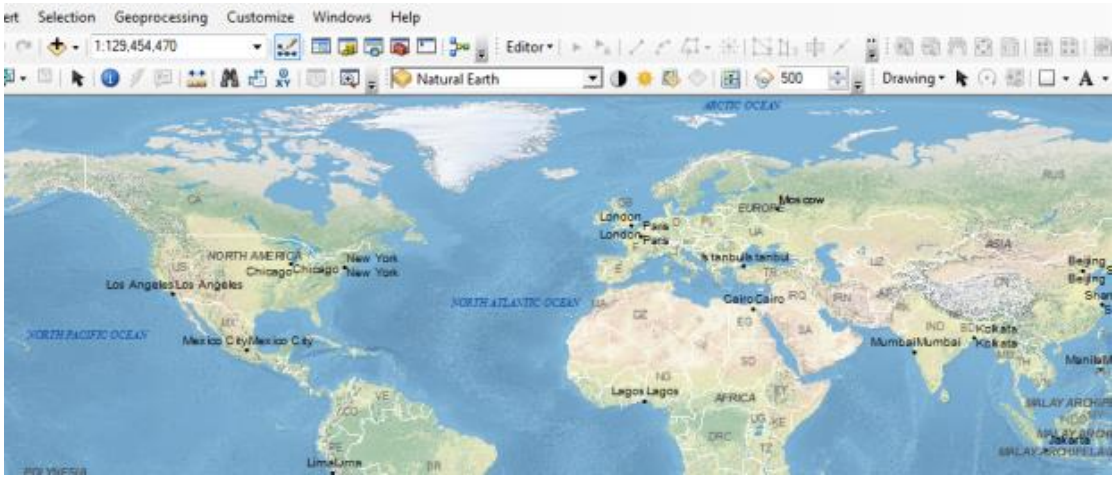
Τα γεωγραφικά συστήματα πληροφοριών (GIS) διαχειρίζονται χωρικά δεδομένα του πραγματικού κόσμου. Τα GIS συλλέγουν γεωγραφικά δεδομένα που είναι προϊόν έρευνας , εικόνες , ψηφιακές σαρώσεις , GPS και δεδομένα

μη επανδρωμένων εναέριων οχημάτων (UAV). Επίσης αποθηκεύουν και διαχειρίζονται τα εν λόγω δεδομένα σε χωρικές βάσεις δεδομένων , και επιτρέπουν να έρθουν στην επιφάνεια πρωτότυπες ιδέες. Ο κύριος σκοπός ενός GIS είναι η συσχέτιση των δεδομένων και η παραγωγή χαρτών και χαρτογραφικών απεικονίσεων των χωρικών δεδομένων. Χρησιμοποιούνται σε πολλές βιομηχανίες γεωεπιστημών , καθώς και την κατασκευή , τη γεωργία , από τους δήμους και τις επιχειρήσεις κοινής ωφελείας.

Οι επιλογές λογισμικού GIS φαντάζουν ατελείωτες . ArcGIS, QGIS, GRASS GIS, SuperGIS, SAGA GIS, JUMP GIS...

Το τοπίο των λογισμικών GIS αλλάζει συνεχώς και υπάρχει η δυνατότητα επιλογής μεταξύ εμπορικού λογισμικού αλλά και λογισμικού ανοικτού κώδικα.Παρακάτω ακλουθεί μία σύντομη παρουσίαση για τα βασικότερα λογισμικά GIS που χρησιμοποιούνται σήμερα στην βιομηχανία [8].

2.6.1 ArcGIS (ESRI)



Εικόνα 4 Περιβάλλον εργασίας ArcGIS

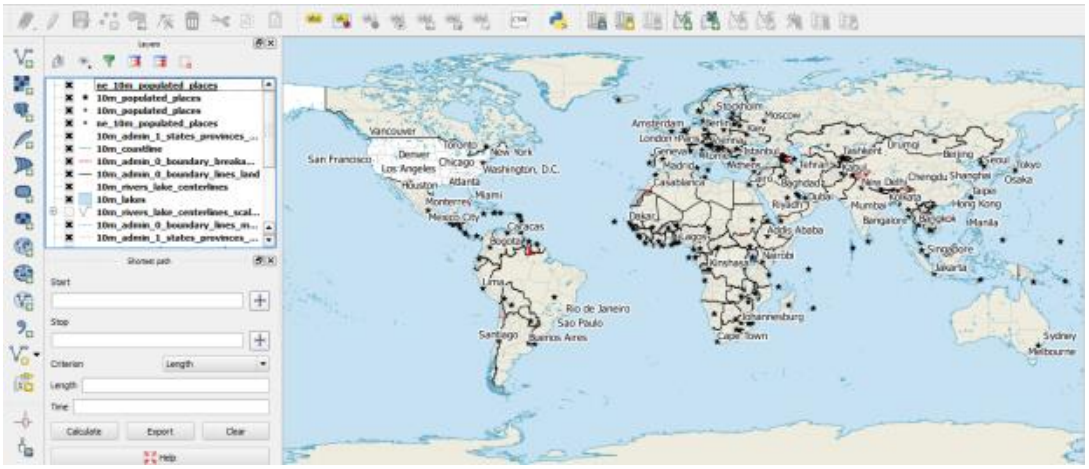
Το ArcGIS είναι το πιο διαδεδομένο λογισμικό GIS. Είναι τέτοια η επιρροή του στην βιομηχανία που, μερικές φορές, ο όρος ArcGIS χρησιμοποιείται (λανθασμένα) για να περιγράψει όλα τα GIS. Η ESRI βρίσκεται στην κορυφή ως η μεγαλύτερη εταιρεία λογισμικού GIS.

Το ArcGIS είναι, στις περισσότερες των περιπτώσεων, πολύ πιο γρήγορο και αποτελεσματικό σε σχέση με τα άλλα GIS και η ESRI ανεβάζει τον πήχη στο επόμενο επίπεδο στη βιομηχανία GIS, ενώ οι άλλες επιλογές λογισμικού GIS προσπαθούν συνεχώς να ανταπεξέλθουν στον ρυθμό που ορίζει αυτή. Μερικά από τα προϊόντα της ESRI είναι τα εξής:

ArcGlobe, ArcGIS Pro, ArcMap (Basic, Editor and ArcInfo), 3D Analyst, Spatial Analyst, Geostatistics, Network Analyst,

Η ESRI έχει την μεγαλύτερη κοινότητα χρηστών, τον μεγαλύτερο αριθμό ατόμων που χρησιμοποιούν σε επαγγελματικό επίπεδο τα προϊόντα της και τις περισσότερες ακαδημαϊκές έρευνες με σημαντική διαφορά από τον επόμενο. Η επιτυχία της ESRI οφείλεται στην λειτουργικότητα, την μεγάλη κοινότητα χρηστών, τη σταθερότητα, τη διαλειτουργικότητα, την επεκτασιμότητα, τη μοντελοποίηση, την διαδικτυακή χαρτογράφηση και στην συντήρηση δεδομένων.

2.6.2 QGIS (Quantum GIS)



Εικόνα 5 Περιβάλλον εργασίας QGIS

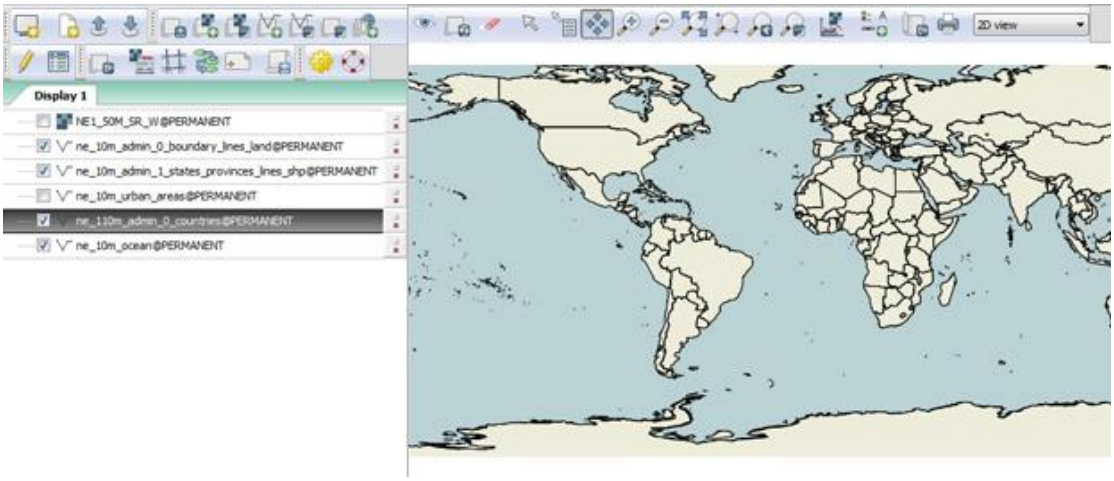
Η όλο και αυξανόμενη χρήση του QGIS αντιπροσωπεύει την πιο σημαντική επιλογή τεχνολογίας ανοικτού κώδικα στα GIS σήμερα. Το QGIS (πρώην Quantum GIS), μπορεί να δημιουργήσει, να επεξεργαστεί, να απεικονίσει, να αναλύσει και να δημοσιεύσει γεωχωρικές πληροφορίες χωρίς κόστος. Είναι ελεύθερο λογισμικό GIS που αναπτύχθηκε από μια κοινότητα αφοσιωμένων εθελοντών.

Ο αριθμός των plugins στο QGIS φτάνει τα 400. Θα μπορούσαμε θεωρητικά να δοκιμάσουμε ένα νέο plugin κάθε μέρα του χρόνου. Στο QGIS υπάρχει η δυνατότητα ενσωμάτωσης δεδομένων CAD και πρόσθεσης χαρτών από το OpenStreetMap και Bing.

Ποια είναι τα πλεονεκτήματα της χρήσης QGIS; Μεγάλη κοινότητα χρηστών, 64-bit γεωεπεξεργασία, μεγάλος αριθμός plugins, ποιότητα χαρτογραφίας.

Πιστεύεται ευρέως πως το QGIS είναι η καλύτερη επιλογή ανοικτού κώδικα με τεράστιες δυνατότητες.

2.6.3 GRASS GIS



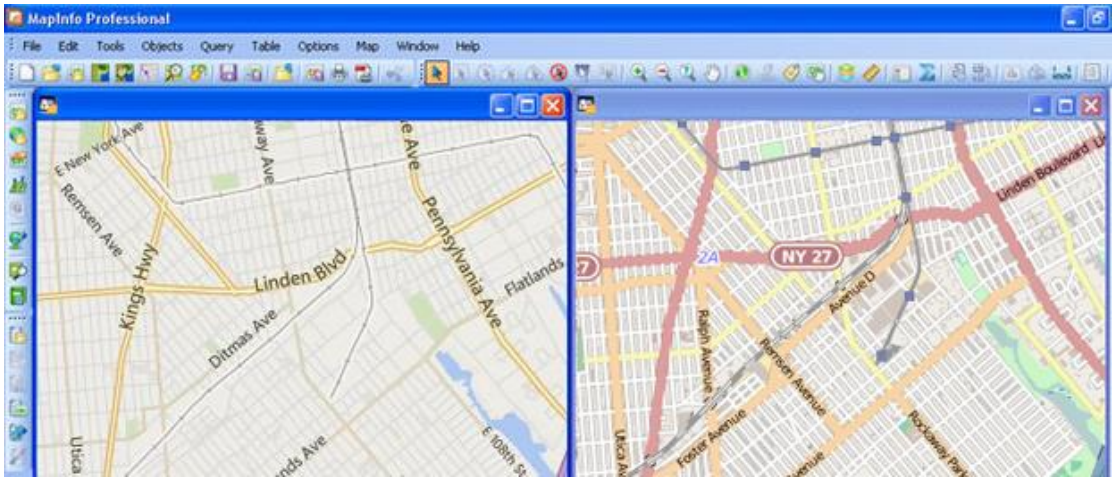
Εικόνα 6 Περιβάλλον εργασίας GRASS GIS

Το Grass (Geographic Resource Analysis Support System) αναπτύχθηκε από τον αμερικανικό στρατό και είναι μια δωρεάν εναλλακτική λύση αντί των εμπορικών λογισμικών GIS . Το συγκεκριμένο λογισμικό έχει πραγματοποιήσει αξιοσημείωτη εξέλιξη τα τελευταία χρόνια. Η χρήση του σε ακαδημαϊκές δημοσιεύσεις είναι εκπληκτική . Είναι δημοφιλής στους ακαδημαϊκούς κύκλους, καθώς ο κώδικας ανοιχτού λογισμικού μπορεί να υποστεί αλλαγές ανάλογα τις ανάγκες του χρήστη.

Το GRASS GIS προσφέρει 350 modules για GIS ανάλυση . Αυτά περιλαμβάνουν διαχείριση δεδομένων, επεξεργασία εικόνας, παραγωγή γραφικών ,χωρική μοντελοποίηση και οπτικοποίηση. Ακόμα και η NASA κάνει χρήση των εφαρμογών του GRASS GIS. Επίσης το GRASS GIS φέρνει προηγμένες γεωχωρικές έννοιες στο ευρύ κοινό.

Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα της GRASS GIS; Ιδανικό για δεδομένα LiDAR, εκτεταμένη τεκμηριωμένη βοήθεια, ανάλυση δικτύων, διαλειτουργικότητα ,δυσκολία στην εκμάθηση, δυσλειτουργικό UI.

2.6.4 MapInfo



Εικόνα 7 Περιβάλλον εργασίας MapInfo

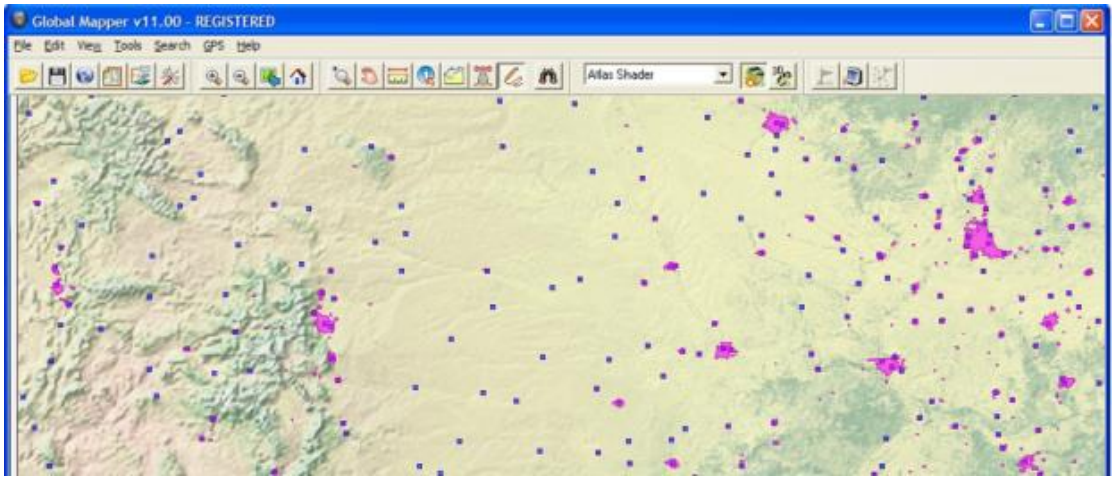
Το MapInfo Professional επιτρέπει την δημιουργία, διαχείριση και απεικόνιση δεδομένων GIS. Το MapInfo συνδέει τη γεωγραφία με τα δεδομένα, αλλά με χαμηλότερο κόστος και περιέχει λεπτομερή χαρτογράφηση, ανάλυση δεδομένων και αναγνώριση μοτίβων γεγονός που το καθιστά ιδιαίτερα δημοφιλές.

Το MapInfo GIS Suite επιτρέπει την δημιουργία, πρόσβαση και διαχείριση γεωχωρικών στοιχείων, οπτικοποίηση δεδομένων επιχειρηματικής ευφυΐας και διαμοιρασμό υψηλής ποιότητας διαδραστικών χαρτών γρήγορα και εύκολα.

Το MapInfo είναι ουσιαστικά από τους μοναδικούς ανταγωνιστές της ESRI στην αγορά των GIS.

Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα του MapInfo: Ευκολία στη χρήση, επεξεργασία 64-bit, καλύτερη διαχείριση πινάκων, side-by-side χαρτογράφηση, γεωλογική χαρτογράφηση, χαμηλό κόστος, μικρός αριθμός χαρτογραφικών δεδομένων, φτωχή υποστήριξη δεδομένων από άλλα λογισμικά.

2.6.5 Global Mapper



Εικόνα 8 Περιβάλλον εργασίας Global Mapper

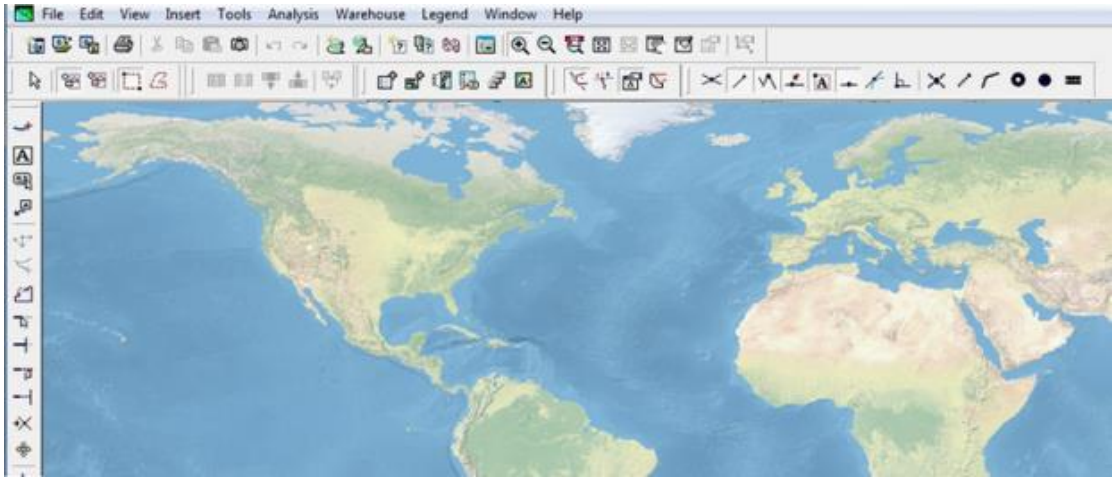
Το Global Mapper αναπτύχθηκε αρχικά από γεωγραφική υπηρεσία των Η.Π.Α. για την απεικόνιση τοπογραφικών χαρτών (.DRG) και ψηφιακών μοντέλων εδάφους (.DEM. Το λογισμικό αυτό εξελίχθηκε σε ένα από τα πιο αποδοτικά (από άποψη κόστους) εμπορικά προϊόντα. Όπως και οι περισσότερες επιλογές λογισμικού GIS, επιτρέπει στους χρήστες να προβάλουν, επεξεργαστούν, συγχωνεύσουν και να εξάγουν εκατοντάδες υποστηριζόμενες μορφές αρχείων.

Το Global Mapper είναι ένα προσιτό και εύκολο στη χρήση GIS που προσφέρει πρόσβαση σε μια τεράστια ποικιλία χωρικών δεδομένων και παρέχει ένα σωστό επίπεδο λειτουργικότητας για να ικανοποιήσει τόσο έμπειρους επαγγελματίες GIS όσο και αρχάριους χρήστες.

Όσον αφορά τις αξιολογήσεις του λογισμικού GIS, το Global Mapper αποτελεί ένα πολυεργαλείο στον τομέα της GIS ανάλυσης ενώ δεν έχει απομακρυνθεί πάρα πολύ μακριά από τον αρχικό του σκοπό της εργασίας με υψομετρικά δεδομένα. κύριες εφαρμογές του περιλαμβάνουν υπολογισμούς εμβαδών περιοχών, ανάλυση και οριοθέτηση λεκανών απορροής, ισοϋψείς καμπύλες, 3D προβολή, υποστήριξη GPS, χαρτογράφησης GIS, ψηφιοποίηση, κ.λπ.

Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα της εργασίας με την Global Mapper: Εργασία με δεδομένα ανύψωσης, δεδομένα LiDAR, χαμηλό κόστος, μεγάλη υποστήριξη δεδομένων από άλλα λογισμικά, κακή διάταξη εκτύπωσης, φτωχή υποστήριξη γεωβάσεων δεδομένων.

2.6.6 GeoMedia (InterGraph/Hexagon Geospatial)



Εικόνα 9 Περιβάλλον εργασίας GeoMedia

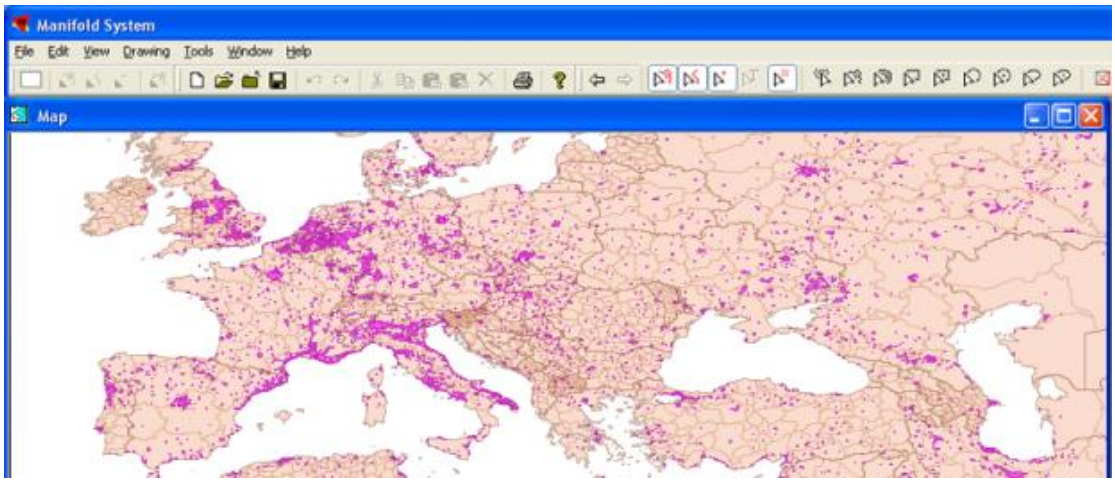
Με μια ιστορία 40 χρόνων, το Intergraph (τώρα Hexagon Geospatial) έχει εξελιχθεί σε μια προνομιακή λύση λογισμικού GIS για την ασφάλεια, την κυβέρνηση, τις δημόσιες υποδομές με γνώμονα την παροχή εφαρμογών GIS στον πραγματικό κόσμο.

ΤΟ GeoMedia είναι ένα ολοκληρωμένο και δυναμικό GIS το οποίο αποσπά πληροφορίες από γεωχωρικά δεδομένα και τα ενσωματώνει με τέτοιο τρόπο ώστε να είναι δυνατή η εξαγωγή χρήσιμης και αξιοποιήσιμης πληροφορίας.

Αποτελεί ένα ώριμο πρόγραμμα λογισμικού GIS για τη συλλογή δεδομένων, την ανάλυση ή τη διαχείριση της αξιοποιήσιμης πληροφορίας στον τομέα των μεταφορών, στις επιχειρήσεις κοινής ωφέλειας, στον τομέα των επικοινωνιών και της διαχείρισης καταστάσεων έκτακτης ανάγκης.

Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα της συνεργασίας με GeoMedia: διατήρηση δεδομένων, πολλαπλές διατάξεις, γρήγορη ανάλυση, on-the-fly προβολές, web-based χαρτογράφηση, σχετικά χαμηλής ποιότητας χάρτες και μοντέλα εδάφους.

2.6.7 Manifold System



Εικόνα 10 Περιβάλλον εργασίας Manifold System

Το Manifold είναι ένα από εκείνα τα πακέτα λογισμικού χαρτογράφησης με εύκολη διαδικασία εκμάθησης. Αποτελείται από μία desktop εφαρμογή , μια βιβλιοθήκη αντικειμένων για προγραμματιστές και ένα Internet Map Server. Παρέχει σχεδόν όλες τις λειτουργίες των εμπορικών GIS με σχεδόν μηδενικό κόστος. Μερικά από τα πλεονεκτήματα του είναι το έξυπνο UI και 64-bit προγραμματισμός που παρέχει. Αυτό που κάνει το Manifold ακόμη καλύτερο είναι το πολύ μικρό κόστος για ένα καλό σύνολο εργαλείων επεξεργασίας GIS και διαχείρισης δεδομένων.

Είναι ένας συνδυασμός χαρτογράφησης, CAD, DBMS και επεξεργασίας εικόνας. Το σήμα κατατεθέν του, είναι η δυνατότητα χρήσης οπτικής διεπαφής για την απεικόνιση, ανάλυση, διαχείριση και μετατροπή δεδομένων που δεν θα ήταν κατανοητά στην κλασσική μορφή DBMS με γραμμές και στήλες.

Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα της συνεργασίας με το Manifold: Σταθερό ,διαισθητικό GUI ,ευρύ φάσμα λειτουργιών προγραμματισμού, εγγενώς 64-bit, χαμηλή τιμή, πολύ μικρός αριθμώς χαρτογραφικών εργαλείων [9].

4ο Κεφάλαιο: Ευρύτερη άποψη για την κατασκευή ‘observatories’ πληροφοριών και βάσεων δεδομένων για την υποστήριξη του σχεδιασμού μεταφορών

Στο κεφάλαιο αυτό παρουσιάζονται, κατ' αρχήν, τα αποτελέσματα διερεύνησης της εσωτερικής αρχιτεκτονικής των δομών δεδομένων που συνήθως χρησιμοποιείται σε υπολογιστικά συστήματα που παρέχουν πληροφορία για την υποδομή των μεταφορικών δικτύων διακίνησης επιβατών και εμπορευμάτων. Τα υπολογιστικά αυτά συστήματα περιέχουν, μεταξύ άλλων και γεωσυσχετισμένη πληροφορία που οργανώνεται σε εμπορικά διαθέσιμα GIS και περιγράφει είτε δομικά και λειτουργικά χαρακτηριστικά της υποδομής, είτε στατιστική πληροφορία που αφορά την χρήση της υποδομής. Η πληροφορία αυτή τροφοδοτεί υπολογιστικά μοντέλα για πληθώρα εφαρμογών για τον σχεδιασμό ή την λειτουργία μεταφορικών συστημάτων (δρομολογήσεις, διαχείριση μεταφορικών στόλων, εκτιμήσεις στοιχείων για τα οποία δεν υπάρχουν μετρήσεις, προβλέψεις κυκλοφοριακών φόρτων, συγκριτικές μελέτες εναλλακτικών σεναρίων πολιτικής στις μεταφορές κ.ά.). Ένα χαρακτηριστικό σύστημα του τύπου αυτού είναι το ETIS (*E*uropean *T*ransport policy *I*nformation *S*ystem), το οποίο έχει σχεδιαστεί για την υποστήριξη αποφάσεων στις Ευρωπαϊκές Μεταφορές. Ο σκοπός της προαναφερθείσας διερεύνησης είναι να εξεταστεί η καταλληλότητα των δομών αυτών για την επίλυση σύγχρονων προβλημάτων που οι ειδικοί των μεταφορών καλούνται να επιλύσουν. Η διερεύνηση έγινε από ερευνητική ομάδα των Σχολών Ηλεκτρολόγων Μηχανικών ΕΜΠ και Πολιτικών Μηχανικών ΕΜΠ, μέλη των οποίων συμμετείχαν σε σημαντικό αριθμό Ευρωπαϊκών Ερευνητικών Προγραμμάτων για την μελέτη και την υλοποίηση του ETIS. Τα αποτελέσματα της διερεύνησης έδειξαν σημαντικές αδυναμίες της πρακτικά καθιερωμένης εσωτερικής αρχιτεκτονικής των δεδομένων και, για τον λόγο αυτό, βρίσκεται σε εξέλιξη η κατασκευή μιας δέσμης εργαλείων λογισμικού (α) για τον έλεγχο και

φιλτράρισμα των δεδομένων, (β) για την καλύτερη προσαρμογή των δεδομένων στις απαιτήσεις τυπικών εφαρμογών διερεύνησης μεταφορικών συστημάτων και (γ) για την εξαγωγή κατάλληλης πληροφορίας για χρήση σε αναπτυσσόμενα ή νέα υπολογιστικά μοντέλα. Εκτός από την δέσμη αυτών των εργαλείων λογισμικού, ευρίσκεται σε εξέλιξη η υλοποίηση μίας νέας, καινοτομικής συνολικής αρχιτεκτονικής δεδομένων και προτείνεται μια νέα μεθοδολογία εργασίας για τα συστήματα του τύπου ETIS. Η νέα αυτή αρχιτεκτονική και η σχετική μεθοδολογία εργασίας παρουσιάζουν σαφή πλεονεκτήματα σε σχέση με την συμβατική αρχιτεκτονική δεδομένων και καθιστούν τα σχετικά πληροφοριακά συστήματα ευέλικτα, εύρωστα και πλήρως ελέγξιμα [10].

4.1 Συνήθης οργάνωση της γεωσυσχετισμένης πληροφορίας

Στα περισσότερα μεγάλα Συστήματα Υποστήριξης Λήψης Αποφάσεων στον Τομέα των Συγκοινωνιών και των Μεταφορών, αλλά και εν λειτουργία εφαρμογές συγκοινωνιακών δικτύων, η γεωσυσχετισμένη πληροφορία οργανώνεται σε πολλές, ξεχωριστές γεωγραφικές βάσεις δεδομένων, συνήθως, ανάλογα με το σκοπό που κάθε μία από αυτές εξυπηρετεί. Για παράδειγμα, υπάρχει μια γεωγραφική βάση δεδομένων για την περιγραφή μόνο του οδικού δικτύου, μια άλλη για την περιγραφή μόνο του σιδηροδρομικού δικτύου, ακόμα μία για τους ηπειρωτικούς υδάτινους δρόμους, μία για την περιγραφή των οικοδομικών τετραγώνων κ.ά.

Κάθε βάση από αυτές, με τη σειρά της, περιέχει ένα ή περισσότερα επίπεδα (layers) γεωγραφικών αντικειμένων, μαζί με το σύνολο της πληροφορίας που συσχετίζεται με τα αντικείμενα αυτά. Ως παράδειγμα, στη βάση των οδικών δικτύων εμπεριέχεται ένα επίπεδο που ονομάζεται επίπεδο των κόμβων (nodes) και έχει όλους τους κόμβους του οδικού δικτύου (π.χ. διασταυρώσεις, μικρές πόλεις, μεγάλες πόλεις, σταθμούς, σημεία ενδιαφέροντος κ.τ.λ.) μαζί με όλη τη δομική, λειτουργική, περιγραφική και στατιστική πληροφορία που τους συνοδεύει. Οι κόμβοι αυτοί έχουν συνήθως διαφορετική σημασία, δηλαδή μεγάλες πόλεις από τις οποίες διέρχεται το οδικό δίκτυο μπορεί να συγκαταλέγονται στο ίδιο επίπεδο με τους οικισμούς και τα μικρά χωριά. Επί πλέον, οι κόμβοι του ίδιου επιπέδου είναι δυνατόν να έχουν διαφορετική

λειτουργικότητα για ένα δίκτυο. Για παράδειγμα, ένας κόμβος του δικτύου των δρόμων μπορεί να παριστάνει ολόκληρη την πόλη της Αθήνας και να συμπίπτει γεωγραφικά με την πλατεία Συντάγματος, ενώ ένας άλλος να παριστάνει έναν εμπορευματικό ή έναν επιβατικό σταθμό. Εν πολλοίς, το ίδιο επίπεδο της βάσης μπορεί περιλαμβάνει κόμβους ανεξαρτήτως των εφαρμογών, για τις οποίες οι κόμβοι αυτοί είναι αναγκαίοι.

Σε ένα άλλο επίπεδο της ίδιας βάσης βρίσκονται αποθηκευμένοι όλοι οι σύνδεσμοι (links) του οδικού δικτύου, τα άκρα των οποίων συμπίπτουν γεωγραφικά με τους κόμβους του προηγούμενου επιπέδου, μαζί με όλη την συσχετισμένη πληροφορία (δομική, λειτουργική ή στατιστική). Στο σημείο αυτό τονίζεται ότι, σχεδόν σε όλα τα συστήματα αυτά, η κυκλοφοριακή συνέχεια κάποιων συνδέσμων συμπεραίνεται έμμεσα από τη γεωγραφική συνέχεια των παραστάσεων των συνδέσμων αυτών [11].

Το χαρτογραφικό υπόβαθρο τηρείται ξεχωριστά και χρησιμοποιείται κατά περίπτωση. Τις περισσότερες, βέβαια, φορές, ως χαρτογραφικό υπόβαθρο χρησιμοποιείται κάποιος χάρτης ή κάποια γεωσυσχετισμένη εικόνα της περιοχής, π.χ. η εικόνα των χωρών ή η εικόνα των διοικητικών περιοχών ή η εικόνα από έναν δορυφόρο κ.τ.λ.

Πρόσθετη πληροφορία μπορεί να εισαχθεί και να τηρηθεί στο σύστημα με τη μορφή πινάκων, οι οποίοι σχετίζονται με τα γεωσυσχετισμένα δεδομένα και τις πληροφορίες με μοναδικά πεδία-κλειδιά. Τέλος, στο σύστημα ενδέχεται να υπάρχουν πινακοποιημένες μορφές γεωσυσχετισμένης πληροφορίας που δεν παρουσιάζονται στη μορφή επιπέδων γεωσυσχεστισμένων βάσεων δεδομένων, όπως αεροπορικές διαδρομές. Συχνά, σε μεγάλα συστήματα, οι πίνακες αυτοί μπορεί να είναι πολλών διαστάσεων, όπως παράδειγμα ροές εμπορευμάτων μεταξύ διοικητικών περιοχών της Ευρώπης, καταμερισμένες κατά την χρησιμοποιούμενη από το Eurostat κατηγοριοποίηση των εμπορευμάτων.

4.2 Ιδιαιτερότητες εφαρμογών των δικτύων και περιγραφή των προβλημάτων που έχουν εντοπιστεί στην αρχιτεκτονική της πληροφορίας της υποδομής μεταφορών.

Η χρήση, στην πράξη, της προηγούμενης αρχιτεκτονικής έχει δείξει ότι η οργάνωση του τρόπου αυτού οδηγεί σε μια σειρά από προβλήματα. Τα πιο βασικά είναι τα εξής:

- Σε πολλές εν λειτουργία εφαρμογές των συγκοινωνιακών δικτύων(ETIS, OpenStreetMap), λόγω του τρόπου ψηφιοποίησης ή/και οργάνωσης της αρχικής πληροφορίας, στα δίκτυα μεταφορών παρουσιάζονται πολλοί ενδιάμεσοι κόμβοι, οι οποίοι δεν είναι ούτε διασταυρώσεις, ούτε έχουν λειτουργική σημασία για τα δίκτυα αυτά. Οι επί πλέον αυτοί κόμβοι δημιουργούν δυσκολία στον υπολογιστικό χειρισμό των μοντέλων βελτιστοποίησης διαδρομών και είναι επιθυμητό να απαλειφθούν από την περιγραφή του δικτύου. Όμως, στην περίπτωση απαλοιφής κόμβων του δικτύου χάνεται η αρχική αρίθμηση των στοιχείων του δικτύου.
- Σε άλλες περιπτώσεις, οι υπάρχοντες κόμβοι του δικτύου πλεονάζουν σε σχέση με το πρόβλημα που πρέπει να διερευνηθεί. Παραδείγματος χάριν, σε ένα σιδηροδρομικό δίκτυο, στο οποίο περιγράφονται με ισοδύναμο τρόπο και οι ήσσονος σημασίας, μικροί σταθμοί, καθώς και οι τοπικές διαδρομές, δυσχεραίνεται η επίλυση προβλημάτων που αφορούν σε διεθνείς μεταφορές.
- Ένας λειτουργικός κόμβος που είναι σημαντικός για μια κατηγορία εφαρμογών μπορεί να μην αποτελεί λειτουργικό κόμβο σε άλλη κατηγορία εφαρμογών. Για παράδειγμα, όταν εξετάζουμε τη λειτουργία μιας σιδηροδρομικής υπηρεσίας που εξυπηρετεί μόνο μεγάλες πόλεις (Intercity), μία διασταύρωση αποτελεί κόμβο επιβίβασης/αποβίβασης επιβατών, μόνον εάν στη διασταύρωση αυτή προβλέπεται στάση του συρμού. Ανάλογα, όταν εξετάζουμε μεταφορές μοναδοποιημένων φορτίων, ένας σιδηροδρομικός σταθμός αποτελεί κόμβο μεταφόρτωσης μόνον εάν υπάρχει ο απαιτούμενος ειδικός εξοπλισμός για το χειρισμό των φορτίων. Θα πρέπει να σημειωθεί δε ότι οι λειτουργικοί κόμβοι ενός

δικτύου που αφορά επιβατικές ροές μπορεί να μην ταυτίζονται με τους λειτουργικούς κόμβους ενός δικτύου που αφορά την μεταφορά φορτίων.

- Σε πολλές περιπτώσεις, απαιτείται να προβάλλεται όλο το δίκτυο μεν, αλλά το υπολογιστικό μοντέλο να χρησιμοποιεί επιλεγμένα, ενδεχομένως μετασχηματισμένα, υποσύνολα του δικτύου.

Είναι εμφανές ότι οι ειδικοί χρήστες των πληροφοριακών συστημάτων που αφορούν μεταφορές έχουν ανάγκη νέων μεθοδολογιών και εργαλείων για την απαλοιφή των περιττών κόμβων (ή/και την πρόσθεση νέων). Η μείωση του πλήθους των κόμβων, ή η αναζήτηση ειδικών κόμβων για ειδικές εφαρμογές, πρέπει να γίνεται κατά το δυνατόν αυτόματα, δηλαδή με την σε μεγάλο βαθμό αυτόματη επιλογή των αναγκαίων κόμβων από το σύνολο των κόμβων της βάσης δεδομένων. Οι κόμβοι προς διατήρηση πρέπει να είναι:

- είτε κόμβοι σημαντικοί για την εκάστοτε υπό μελέτη εφαρμογή,
- είτε διασταυρώσεις του εκάστοτε δικτύου ή του συνδυασμού δικτύων.

Οι κόμβοι αυτοί πρέπει να επιλέγονται με βάση κριτήρια που θέτουν οι ειδικοί στις περιοχές των εφαρμογών ενδιαφέροντος. Στη συνέχεια, πρέπει να είναι δυνατή η άμεση συνένωση επί μέρους συνδέσμων του υπό καθαρισμό δικτύου σε γενικότερους, ώστε, τελικά να παραμένουν ως κόμβοι μόνο οι επιλεγέντες από τους ειδικούς και οι αναγκαίες διασταυρώσεις. Κατά τη διαδικασία αυτή, είναι απαίτηση των ειδικών στις μεταφορές να τηρείται αυστηρά η αντιστοίχηση κωδικών χαρακτηρισμών και ονομάτων των συνδέσμων του αρχικού δικτύου, με αυτούς του δικτύου που παράγεται από την διαδικασία ελαχιστοποίησης των κόμβων.

Είναι ακόμη προφανές, ότι, η παραγωγή του προκύπτοντος δικτύου πρέπει να γίνεται με τρόπο μη καταστροφικό για το αρχικό δίκτυο. Επομένως, το νέο παραγόμενο δίκτυο πρέπει, ουσιαστικά, να οργανώνεται ως διαφορετικό τμήμα της βάσης δεδομένων.

Η πράξη έχει αποδείξει ότι η προσπάθεια οργάνωσης και τήρησης μεγάλου αριθμού ομάδων δεδομένων, η καθεμία εκ των οποίων είναι αναγκαία για την επίλυση ενός προβλήματος ή μιας υποκατηγορίας προβλημάτων, καταλήγει στην δημιουργία πυρήνων πολυσχιδών, πολύπλοκων και πολυμορφικών

δεδομένων. Αυτό το φαινόμενο παρουσιάζεται συχνά σε συστήματα που πρέπει να καλύψουν ευρείες θεματικές περιοχές, με δύο κύρια αποτελέσματα:

- Η πολυμορφία αυξάνει σημαντικά την πολυπλοκότητα του πυρήνα δεδομένων και των αντίστοιχων προγραμμάτων και την δυσκολία ορθής τήρησης (επικαιροποίησης, αναβάθμισης) και εξέλιξης (θεματικής στροφής, αύξησης ή/και μείωσης του περιεχομένου και των αντίστοιχων υπηρεσιών κ.τ.λ.) του συστήματος.
- Η περαιτέρω προσπάθεια αύξησης των θεμάτων, με τα οποία το σύστημα ασχολείται, καταλήγει στο να καταστήσει το συνολικό σύστημα πρακτικά
- μη τηρήσιμο ή, ακόμη και μη ελέγξιμο. Η μόνη λύση στην περίπτωση αυτή είναι η εξ αρχής σχεδίαση και υλοποίηση ενός νέου συστήματος και η απόρριψη του παλαιού.

Επί πλέον, υπάρχουν εφαρμογές στις οποίες είναι απαραίτητη η χρήση πληροφορίας από διαφορετικές βάσεις δεδομένων. Για παράδειγμα, σε μία εφαρμογή δρομολόγησης συγκοινωνιών μπορεί είναι απαραίτητοι οι δρόμοι (για την κίνηση των επιβατών με λεωφορεία) και οι σιδηρόδρομοι (για την κίνηση των επιβατών με τρένα). Όπως περιγράφηκε προηγουμένως, συνήθης πρακτική επίλυσης του προβλήματος αυτού είναι η εξ αρχής δημιουργία μιας βάσης δεδομένων, η οποία θα εμπεριέχει, εκ νέου, δύο επίπεδα. Ένα επίπεδο θα περιέχει τους κόμβους, τόσο του οδικού δικτύου όσο και του σιδηροδρομικού που βρίσκονται στην εξεταζόμενη περιοχή, με όλη τη γεωσυσχετισμένη πληροφορία που διασυνδέεται με τους κόμβους αυτούς. Ένα δεύτερο θα περιέχει τους συνδέσμους, τα άκρα των οποίων θα συμπίπτουν με τους κόμβους του προηγούμενου επιπέδου. Είναι εμφανές από την περίπτωση αυτή, ότι δεν υπάρχει ουσιαστική εσωτερική διαλειτουργικότητα των δεδομένων.

Γεωσυσχετισμένες πληροφορίες που αφορούν τις ίδιες οντότητες ή είναι παρόμοιας μορφής, αλλά έχουν αναπτυχθεί για συγκεκριμένη εφαρμογή, μπορεί να είναι περιττές ή χωρίς λειτουργικότητα για κάποια άλλη εφαρμογή. Για παράδειγμα, η πληροφορία για το δίκτυο των δρόμων που έχει αναπτυχθεί για μία εφαρμογή που αφορά την κατασκευή και τη διατήρηση του οδικού δικτύου, θα πρέπει να έχει πολύ μεγαλύτερο βαθμό λεπτομέρειας (*level of detail*

ή *granularity*) από την αντίστοιχη πληροφορία μίας εφαρμογής δρομολόγησης και διαχείρισης στόλου, ακόμα κι αν και οι δύο εφαρμογές ενδιαφέρουν την ίδια γεωγραφική περιοχή και εξετάζουν το ίδιο δίκτυο δρόμων. Στις ίδιες εφαρμογές, διαφορετικού τύπου πληροφορία, που αφορά όμως την ίδια οντότητα, θα πρέπει να διασυνδέεται με το εξεταζόμενο δίκτυο, όπως π.χ. διαστάσεις και αντίστοιχες διατομές ανά συγκεκριμένες αποστάσεις, η ύπαρξη ή μη λωρίδας εκτάκτου ανάγκης, η ύπαρξη ή μη διαχωριστικής νησίδας, το έτος τελευταίας συντήρησης, το κόστος συντήρησης ανά περιοχή κ.ά.

Από τα προηγούμενα, γίνεται κατανοητό ότι διαφορετικές εφαρμογές, οι οποίες επιλύουν διαφορετικά προβλήματα, έχουν ανάγκη διαφορετικής παράστασης της πληροφορίας για τις ίδιες οντότητες (π.χ. οδικούς συνδέσμους, σιδηροδρομικούς συνδέσμους, κόμβους κ.ά.) και, συνεπώς, διαφορετική εσωτερική αρχιτεκτονική των δεδομένων.

Επίσης, ιδιαιτέρως σημαντικά προβλήματα προκύπτουν όταν το σύστημα διευρύνεται διαρκώς θεματικά, εξ αιτίας της πολυμορφίας που σχεδόν αναπόφευκτα αναπτύσσεται. Ο δε μεγάλος όγκος και η πολυπλοκότητα της οργανωμένης πληροφορίας που εμφανίζονται σε συστήματα μεγάλης έκτασης συστήματα (όπως το ETIS), καταλήγει στο να καταστήσει το σύστημα δύσκολα ελέγξιμο και τηρήσιμο ή ακόμα και μη διαχειρίσιμο. Σε πολλές περιπτώσεις της πράξης, μεγάλα συστήματα οργανωμένα με το συγκεκριμένο τρόπο κατέστησαν τελικά μη διαχειρίσιμα και αντικαταστάθηκαν από νέα, εξ αρχής σχεδιασθέντα και υλοποιηθέντα. Είναι προφανές ότι λύσεις του τύπου αυτού έχουν αρκετά αυξημένο κόστος σε χρόνο και χρήμα [12].

4.3 Περαιτέρω προβλήματα που δεν μπορεί να επιλύσει η συμβατική οργάνωση

Εκτός των προαναφερθέντων προβλημάτων, η οργάνωση των δεδομένων κατ' αυτόν τον τρόπο δεν προσφέρει τη δυνατότητα επίλυσης επί πλέον προβλημάτων που προκύπτουν από τη λειτουργικότητα των κόμβων, δηλαδή τη διαφορετική έννοια που έχει κάθε κόμβος, ανάλογα με την εφαρμογή στην οποία χρησιμοποιείται.

Τυπικά, κάθε κόμβος πρέπει να είναι είτε διασταύρωση, είτε κάποιο σημείο με σημασία για την εφαρμογή την οποία εξετάζουμε. Ως απλό παράδειγμα, στο

δίκτυο των σιδηροδρόμων, όλοι οι κόμβοι, στους οποίους υπάρχουν σταθμοί τρένων δεν εξυπηρετούν όλους τους διερχόμενους συρμούς (απλούς συρμούς, ταχείες, ιντερσίτι κ.ά.). Αν η εξεταζόμενη εφαρμογή αφορά μόνο στα τρένα-εξπρές, τότε λειτουργικοί κόμβοι του σχετικού δικτύου είναι μόνο οι διασταυρώσεις του συγκεκριμένου τύπου συρμού και οι σταθμοί που αυτό σταματά. Στο αναγκαίο, επομένως, δίκτυο για την εφαρμογή αυτή, θα πρέπει να υπάρχουν μόνο οι συγκεκριμένοι σταθμοί και οι διασταυρώσεις, ενώ οι σύνδεσμοι ανάμεσα στους κόμβους αυτούς πρέπει να ενοποιηθούν.

Σε ένα πιο σύνθετο παράδειγμα, όπως η μελέτη μεταφοράς εμπορευμάτων ή αγαθών μέσω δικτύου που περιλαμβάνει δρόμους και σιδηροδρόμους, το πρόβλημα που αναφέρθηκε στην προηγούμενη παράγραφο επιτείνεται σημαντικά, καθώς πρέπει να χρησιμοποιηθούν οντότητες από διαφορετικές βάσεις δεδομένων, όχι απαραίτητα κατάλληλα εναρμονισμένες. Στο ίδιο παράδειγμα, πρόσθετοι περιορισμοί στην κίνηση των οχημάτων (π.χ. η ύπαρξη εποχιακών συνδέσμων), μπορούν επίσης να διαφοροποιήσουν το δίκτυο. Ένας άλλος παράγοντας δε που μπορεί να διαφοροποιήσει το αναγκαίο δίκτυο είναι το γεγονός ότι διαφορετικά προϊόντα μπορεί να έχουν την ανάγκη να κινηθούν με διαφορετικό τρόπο στα δίκτυα.

Παραδείγματος χάριν, παλέτες στις οποίες φορτώνεται υψηλής αξίας, αλλά σύντομης διάρκειας ζωής λογισμικό, μπορεί να χρειαστεί να μετακινηθούν αεροπορικά, κάτι που δε συμβαίνει συνήθως σε συμβατικά εμπορευματοκιβώτια, τα οποία σχεδόν ποτέ δεν διακινούνται αεροπορικά. Γενικά, είναι πολύ δύσκολο να γίνει εξ αρχής μία εξαντλητικά αναλυτική μελέτη των σχετικών προβλημάτων και να βρεθούν όλες οι διαφορετικές πιθανές απαιτήσεις που αφορούν κόμβους και συνδέσμους δικτύων, για κάθε επί μέρους εφαρμογή. Επί πλέον, διαφορετικές εφαρμογές απαιτούν συχνά διαφορετικό τύπο γεωσυσχετισμένης πληροφορίας.

Για παράδειγμα, σε εφαρμογές μελέτης ρύπων, μπορεί να χρειάζεται, πέραν της συνηθισμένης πληροφορίας:

- Το υψόμετρο κάθε σημείου του δικτύου ως τρίτη διάσταση (πέραν των συνηθισμένων x, y), εάν χρησιμοποιούνται μοντέλα, τα οποία από την κλίση των δρόμων εκτιμούν την παραγόμενη από τα αυτοκίνητα ρύπανση.

- Το τρισδιάστατο, ψηφιακό μοντέλο του εδάφους,
- αλλά και ειδική δυναμική πληροφορία για τον καιρό (επικρατούσες και αναμενόμενες καιρικές συνθήκες, όπως της ύπαρξης βροχής ή του μοντέλου των ρευμάτων των αερίων μαζών), για τη μελέτη της συγκέντρωσης ή της πιθανής μετακίνησης των ρύπων.

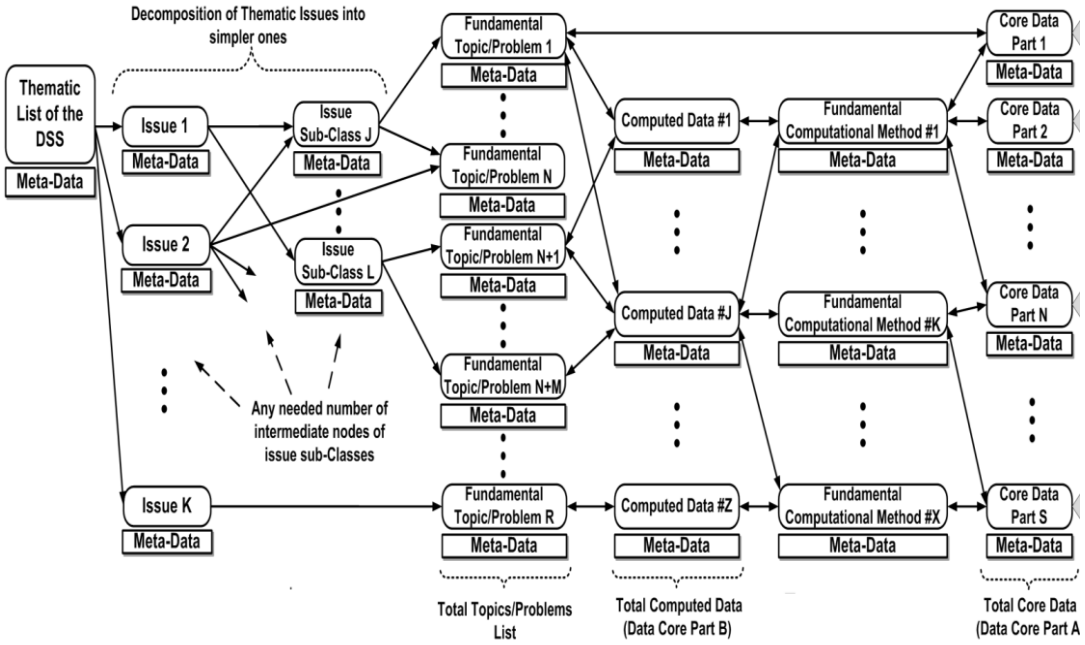
Ενώ, για εφαρμογές μελέτης θορύβου στους αυτοκινητόδρομους, απαραίτητη είναι η γνώση, όχι μόνο της ύπαρξης, αλλά της ακριβούς θέσης των πετασμάτων για τη μείωση του θορύβου, καθώς και, ενδεχομένως, άλλες πρόσθετες πληροφορίες (όρια ταχύτητας που επηρεάζουν τα επίπεδα θορύβου, διαστάσεις και υλικό κατασκευής πετασμάτων, έτος κατασκευής ή συντήρησής τους κ.τ.λ.).

Η δημιουργία των αναγκαίων, ειδικών παραστάσεων δικτύων αποτελεί σήμερα μια εξαιρετικά επίπονη διαδικασία, η οποία πραγματοποιείται είτε «με το χέρι», δηλαδή μη αυτόματα, και απαιτεί τη χρήση των δυνατοτήτων προγραμμάτων GIS, είτε, αν υπάρχει γνώση ειδικού προγραμματισμού τη δημιουργία ειδικών προγραμμάτων για την επιλογή, την εξαγωγή και το μετασχηματισμό των γεωσυσχετισμένων δεδομένων. Το αποτέλεσμα δε της διαδικασίας αυτής είναι, συνήθως, μια καινούρια βάση δεδομένων που εξυπηρετεί μόνο την εκάστοτε εφαρμογή και, η οποία, ως επί το πλείστον, δεν έχει πλέον αντιστοιχία με το αρχικό δίκτυο [13].

5ο Κεφάλαιο: Ανάγκη χρήσης τελεολογικής δομής μεταδεδομένων και αρχιτεκτονικής γεωσυσχετισμένων πληροφοριών.

5.1 Τελεολογία – Χάρτης Περιεχομένων του συστήματος

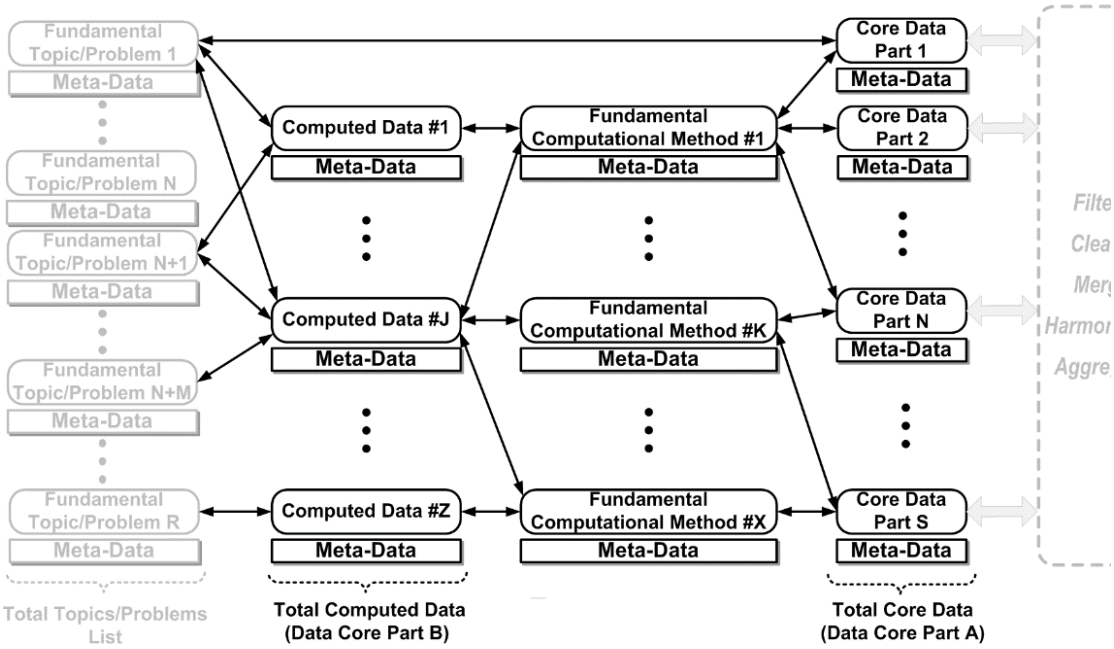
Όπως έγινε εμφανές, ένα σύστημα υποστήριξης λήψης αποφάσεων και χάραξης πολιτικής συγκοινωνιών και μεταφορών σε ευρωπαϊκό επίπεδο μπορεί γρήγορα να αποκτήσει έναν πολυσχιδή, πολύπλοκο και πολυμορφικό πυρήνα δεδομένων (και αντίστοιχων προγραμμάτων). Η βασική ιδέα για την αποτελεσματική διαχείριση του πακτωλού της πληροφορίας αυτής είναι η ομαλή αρχική ανάπτυξη και, στην συνέχεια, η διατήρηση υψηλού επίπεδου οργάνωσης του συστήματος μέσω της χρήσης μιας δομής μεταδεδομένων, την οποία η ομάδα εργασίας ονόμασε «Χάρτης Περιεχομένων του Συστήματος» ή «Τελεολογική Δομή».



Εικόνα 11 Ακυκλικός γράφος

Στο πρώτο μέρος της Τελεολογικής Δομής, ο κατάλογος με τα θέματα (Thematic List), σχετικά με τα οποία προσφέρει απαντήσεις ή υπηρεσίες το

σύστημα, αναλύεται σταδιακά σε επί μέρους θέματα (Issues) μέχρι τα στοιχειώδη θέματα ή προβλήματα (δηλαδή αυτά για τα οποία είναι πλήρως καθορισμένα τα ζητούμενα και το πώς αυτά υπολογίζονται), μέσω μιας δομής που μοιάζει με δέντρο αλλά στην πραγματικότητα είναι ακυκλικός γράφος (Εικόνα 11). Στο σημείο αυτό, όποια πιθανή πολυμορφία υπήρχε έχει πλήρως αρθεί μέσω κατάλληλης θεματικής ανάλυσης. Στο επόμενο τμήμα της Τελεολογικής Δομής απεικονίζεται η επιθυμητή πληροφορία, την οποία το σύστημα πρέπει να δώσει στους χρήστες του (την οποία ονομάζουμε «Υπολογισμένα Δεδομένα – Computed Data»), καθώς και ο τρόπος υπολογισμού αυτής της πληροφορίας και τα δεδομένα που είναι απαραίτητα για τον υπολογισμό της πληροφορίας αυτής.



Εικόνα 12 Κορμός Τελεολογικής Δομής

Στην *Εικόνα 12*, απεικονίζονται τα ακόλουθα:

- Κατ' αρχήν, οι περιγραφές της επιθυμητής πληροφορίας εξόδου (ή Ενδεικτών), που φαίνονται στην πρώτη στήλη (Computed Data). Η πληροφορία αυτή, είναι στην πραγματικότητα:
  1. Είτε αποτελέσματα της εκτέλεσης των προγραμμάτων που υλοποιούν τις Υπολογιστικές Μεθόδους (συμπεριλαμβανομένων των Υπολογιστικών Μοντέλων). Οι Υπολογιστικές Μέθοδοι ονομάζονται Θεμελιώδης Υπολογιστικές Μέθοδοι (Fundamental Computation Method), όπου, ο όρος Θεμελιώδης ενέχει και την έννοια του

«Στοιχειώδους» ή «Άτμητου», δεδομένου ότι οι μέθοδοι αυτές αντιστοιχούν στα Θεμελιώδη Προβλήματα .

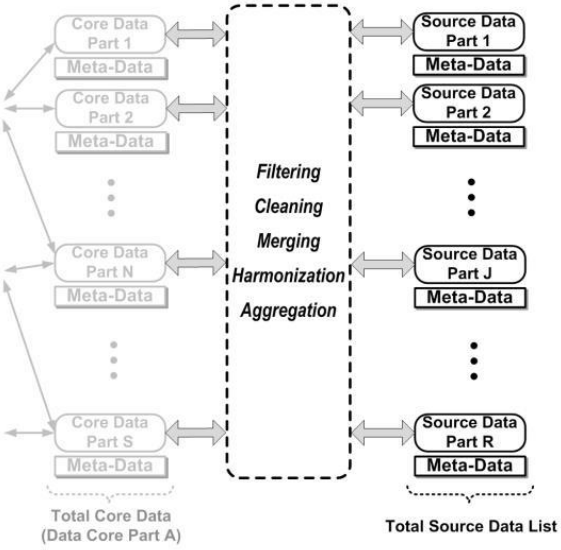
2. Είτε, σε ειδικές περιπτώσεις, πληροφορία, η οποία αποκτήθηκε από εξωτερικές πηγές, αλλά αποτελεί η ίδια Ενδεικτική Πληροφορία και πρέπει να δοθεί στους χρήστες χωρίς περεταίρω επεξεργασία. Τέτοια περίπτωση είναι, π.χ., οι μετρήσεις θορύβου σε ένα οδικό δίκτυο που μπορούν να δοθούν στους χρήστες και να αποτελέσουνε σαφή ενδεικτική πληροφορία για την ηχορύπανση στο δίκτυο αυτό.

Αρκετές φορές τα αποτελέσματα αυτά είναι ανάγκη να διατηρηθούν για ένα χρονικό διάστημα ή/και επ' αόριστον, οπότε οργανώνονται στο «Τμήμα Β του Κορμού Δεδομένων» (Data Core, Part B) της τοπικής βάσης δεδομένων.

- Οι περιγραφές των Υπολογιστικών Μεθόδων (Computational Methods), στις οποίες περιλαμβάνονται και τα Υπολογιστικά Μοντέλα. Οι Υπολογιστικές Μέθοδοι είναι στην πραγματικότητα υλοποιήσεις των αλγορίθμων που είναι αναγκαίοι για την επίλυση των σχετικών προβλημάτων. Δέχονται δε ως εισόδους δεδομένα που απεικονίζονται στην τελευταία στήλη της *Εικόνας 11* (Core Data).
- Δεξιά των Υπολογιστικών Μεθόδων απεικονίζονται:
  1. Οι περιγραφές των δεδομένων που είναι αναγκαία για την επίλυση των θεμελιωδών προβλημάτων. Τα δεδομένα αυτά τα ονομάζουμε Δεδομένα Κορμού (Core Data). Πρέπει δε να οργανώνονται σε τοπική βάση του συστήματος δεδομένων του συστήματος και να τηρούνται με τον ενδεδειγμένο τρόπο. Το τμήμα αυτό της τοπικής βάσης δεδομένων ονομάζεται «Τμήμα Α του Κορμού Δεδομένων» (Data Core, Part A).
  2. Σε ειδικές περιπτώσεις, όπως ήδη αναφέρθηκε, η πληροφορία, η οποία αποκτήθηκε από εξωτερικές πηγές, αλλά αποτελεί η ίδια Ενδεικτική Πληροφορία και πρέπει να δοθεί στους χρήστες χωρίς περεταίρω επεξεργασία. (Προφανώς, μόνο ένα αντίτυπο της πληροφορίας οργανώνεται στο Σύστημα, αλλά το αντίτυπο αυτό λειτουργεί τόσο ως Δεδομένα Κορμού όσο και, καταχρηστικά, ως Υπολογισμένα Δεδομένα.)

Πρέπει να τονιστεί ότι το περιεχόμενο των δομών είναι *η περιγραφή* δεδομένων και υπολογιστικών μεθόδων και όχι τα δεδομένα ή οι μέθοδοι αυτές κάθε αυτές. Αυτό βέβαια ισχύει για όλη την Τελεολογική Δομή ή Χάρτη Περιεχομένων του Συστήματος.

Τα δύο Τμήματα της Τελεολογικής Δομής που περιγράφηκαν μέχρι τώρα είναι υποχρεωτικά για κάθε σύστημα που οργανώνεται με τον τρόπο αυτό. Το τρίτο Τμήμα της Τελεολογικής Δομής, που παρουσιάζεται στο Σχήμα 3, είναι προαιρετικό μεν, αλλά ιδιαίτερα χρήσιμο στην περίπτωση συστημάτων που φέρνουν δεδομένα από πολλές εξωτερικές πηγές δεδομένων (το ETIS, πχ., χρειάζεται πληροφορία από περίπου 500 εξωτερικές βάσεις δεδομένων).



Εικόνα 13 Τρίτο τμήμα Τελεολογικής Δομής

Στην Εικόνα αυτή απεικονίζονται ενέργειες που γίνονται πάνω στα εξωτερικά σύνολα δεδομένων, προκειμένου είτε τα δεδομένα αυτά καθ' αυτά είτε τα δεδομένα της επεξεργασίας τους να έρθουν σε κατάλληλη μορφή, ώστε να χρησιμοποιηθούν από το σύστημα που η Τελεολογική Δομή περιγράφει. Τέτοιες ενέργειες είναι ο έλεγχος και ο καθαρισμός των δεδομένων, πιθανό φιλτράρισμα για επιλογή δεδομένων, πιθανή συγχώνευση δεδομένων, πιθανή εναρμόνιση δεδομένων κ.τ.λ.

Όπως φαίνεται στα σχήματα που περιγράφουν την Τελεολογική Δομή ή Χάρτη Περιεχομένων του Συστήματος, σε κάθε περιγραφή υποσυνόλου μεταδεδομένων της δομής αυτής είναι δυνατή η επισύναψη οποιωνδήποτε επί πλέον μεταδεδομένων, τα οποία οι σχεδιαστές ή οι χρήστες του συστήματος

πιστεύουν ότι είναι απαραίτητο ή χρήσιμο να έχουν. Τα επί πλέον αυτά μεταδεδομένα μπορεί να χρησιμοποιηθούν από:

- Το σύστημα αυτό καθ' εαυτό για την επίτευξη κατά το δυνατόν αυτόματης λειτουργίας (πχ. επιλογή κατάλληλων χαρτών για ένα συγκεκριμένο θέμα, επιλογή Ενδεικτών και δεδομένων για το θέμα αυτό, επιλογή επί πλέον έμμεσα γεωσυσχετισμένων δεδομένων, επιλογή δυνατών μεθόδων επεξεργασίας των δεδομένων και αντίστοιχης οπτικοποίησης, κ.τ.λ.).
- Τους χρήστες του συστήματος. Στην κατηγορία αυτή ανήκουν πληροφορίες που έχουν σχέση με τον τρόπο παραγωγής των δεδομένων, με ειδικά χαρακτηριστικά και ιδιότητες των δεδομένων, με την καταλληλόλητα των δεδομένων για συγκεκριμένες εργασίες, με την ποιότητα συνόλων δεδομένων, με τον τρόπο επιλογής εναλλακτικών συνόλων δεδομένων κ.τ.λ.

Η Τελεολογική Δομή πρέπει να συμπεριληφθεί στο πληροφοριακό σύστημα και, στην πραγματικότητα, να αποτελεί το αρχικό σημείο πρακτικά όλων των λειτουργιών του συστήματος.

Τμήμα της αρχιτεκτονικής αυτής έχει κατ' αρχήν δοκιμαστεί στο πλαίσιο του προγράμματος ETIS-Agent και έχει επιδειχθεί σε εκπροσώπους Στατιστικών Υπηρεσιών και Υπουργείων Μεταφορών της Ευρωπαϊκής Ένωσης με πολύ καλά αποτελέσματα. Η κατασκευή μιας σύγχρονης πλατφόρμας, βασισμένης στην προτεινόμενη αρχιτεκτονική, που θα έχει πληροφορία για την υποδομή των μεταφορικών δικτύων (supply) και τις καταγεγραμμένες ή επιθυμητές ροές επιβατών και εμπορευμάτων (demand) είναι σε εξέλιξη στο Εργαστήριο Συστημάτων Πολυθεματικής και Γεωσυσχετισμένης Πληροφορίας του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΕΜΠ, στο οποίο ανήκουν, μεταξύ άλλων, όλοι οι συγγραφείς της παρούσας εργασίας [10]

5.2 Αρχιτεκτονική 'Θέματα υπεράνω χαρτών'

Από τα προηγούμενα έγινε κατανοητό ότι αν εξετάσουμε μακροσκοπικά τον συνήθη τρόπο οργάνωσης της γεωσυσχετισμένης πληροφορίας στα εξεταζόμενα συστήματα, θα παρατηρήσουμε ότι τις περισσότερες φορές

βασίζεται σε ένα σύστημα GIS, στο οποίο αποθηκεύεται ένα σύνολο χαρτών, που περιλαμβάνουν (α) τις σημαντικές οντότητες, συνήθως ως ακολουθίες στοιβαγμένων επιπέδων πληροφορίας και (β) επί πλέον πληροφορία που αποθηκεύεται ως χαρακτηριστικά (attributes) των οντοτήτων αυτών. Η υπόλοιπη πληροφορία που αφορά τις εφαρμογές ή τις υπηρεσίες που παρέχει το σύστημα και συνδέεται με τις οντότητες αυτές, προέρχεται είτε από τοπικές βάσεις δεδομένων, είτε ενδεχόμενα χωρικά διασπαρμένες στο διαδίκτυο εξωτερικές βάσεις δεδομένων. Η αρχιτεκτονική αυτή καλείται από την ομάδα εργασίας Αρχιτεκτονική «Χάρτες Υπεράνω Θεμάτων». Αρχικό στοιχείο οργάνωσης του συστήματος αυτού είναι οι Χάρτες (και οι σχετικές σημαντικές οντότητες) και όχι το θέμα ή η εφαρμογή, για την οποία οι Χάρτες και οι οντότητες είναι αναγκαίοι.

Αν και η αρχιτεκτονική αυτή είναι αρκετά ισχυρή, έχει ένα άνω όριο ευρωστίας. Όταν ο όγκος, η πολυθεματικότητα, και, επομένως, η πολυμορφία της πληροφορίας αυξάνεται, η λειτουργία του συστήματος επιβαρύνεται και το ίδιο το σύστημα τείνει να γίνεται δύσχρηστο και δύσκολα διαχειρίσιμο (έως και μη διαχειρίσιμο), όπως πολλές φορές έχει αποδειχθεί στην πράξη.

Αντίθετα, η χρήση της Τελεολογικής Δομής για την οργάνωση του συστήματος μπορεί να αυξήσει σημαντικά τόσο την ευελιξία, όσο την ευρωστία του συστήματος. Αρχικό στοιχείο οργάνωσης του συστήματος στην περίπτωση αυτή είναι ο Χάρτης Περιεχομένων του Συστήματος. Αυτό σημαίνει ότι η Θεματική Λίστα, δηλαδή οι εφαρμογές και οι υπηρεσίες που παρέχονται από το σύστημα ή τα προβλήματα που επιλύονται με τη βοήθεια του συστήματος έχουν τον κυρίαρχο ρόλο στην οργάνωση της γεωσυσχετισμένης πληροφορίας. Αυτή η αρχιτεκτονική που προτάθηκε από τα μέλη του Εργαστηρίου, στα οποία ανήκουν και οι συγγραφείς, ονομάστηκε Αρχιτεκτονική «Θέματα Υπεράνω Χαρτών». Ο Χάρτης Περιεχομένων του Συστήματος ή Τελεολογική Δομή οργανώνεται σε μία κατάλληλη βάση μεταδεδομένων (την οποία για συντομία ονομάζουμε Μετά-Βάση), που αποτελεί την αρχή κάθε ομάδας λειτουργιών του συστήματος. Η Τελεολογική Δομή προσδιορίζει για κάθε κατηγορία εφαρμογών:

- Το σύνολο της πληροφορίας που θα οργανωθεί σε έναν GIS Server, συμπεριλαμβανομένων των χαρτών που θα χρησιμοποιηθούν, οι οποίοι

μπορεί να διαφέρουν για διαφορετικές κατηγορίες εφαρμογών. Η πληροφορία αυτή είναι:

1. Το χαρτογραφικό υπόβαθρο
2. Οι σημαντικές για την κατηγορία εφαρμογών οντότητες
3. Η άμεσα συνδεδεμένη σημαντική, συνήθως στατικού χαρακτήρα για τις οντότητες αυτές

- Η υπόλοιπη, έμμεσα γεωσυσχετισμένη πληροφορία διασυνδέεται με τις σημαντικές οντότητες που περιγράφει η Τελεολογική Δομή, όπως αναφέρθηκε στην αρχή του Κεφαλαίου.

Δεδομένης της υψηλής δυνατότητας οργάνωσης που η Τελεολογική Δομή καθιστά δυνατή, είναι εφικτή η κατανομή της έμμεσα γεωσυσχετισμένης πληροφορίας σε διεσπαρμένες χωρικά βάσεις δεδομένων. Η κατανομή μπορεί να γίνει με οποιοδήποτε κριτήριο, αλλά προτιμάται αυτό που η Ευρωπαϊκή Κοινότητα ονομάζει αρχή της εντοπιότητας (subsidiarity principle). Στην συγκεκριμένη περίπτωση, η αρχή αυτή εκφράζει την επιθυμία επί μέρους σύνολα δεδομένων να ευρίσκονται κοντά στον τόπο παραγωγής τους, δηλαδή τα σύνολα αυτά δεδομένων να οργανώνονται και τηρούνται από αυτούς που τα δημιούργησαν και τα επικαιροποιούν, αν αυτό είναι δυνατό. Με τις σημερινές διαδικτυακές δυνατότητες και τις υψηλές δυνατότητες οργάνωσης που επιτρέπει η Τελεολογική Δομή, γεωγραφική κατανομή σύμφωνη με το κριτήριο αυτό είναι δυνατή για σημαντικό αριθμό συνόλων δεδομένων [14].

Με τη χρήση της Αρχιτεκτονικής «Θέματα Υπεράνω Χαρτών» έχουμε τα ακόλουθα πλεονεκτήματα:

1. Το Πληροφοριακό Σύστημα (ή, συντομογραφικά, ΠΣ) μπορεί άνετα να αντιμετωπίσει την διαφοροποίηση και την πολυπλοκότητα δεδομένων και των μεθόδων επεξεργασίας αυτών, καθώς και να επιλύσει το πρόβλημα της πολυμορφίας δεδομένων.
2. Το ΠΣ μπορεί να αυξάνεται διαρκώς, να επικαιροποιείται και να αναβαθμίζεται διαρκώς και να είναι πολυθεματικό με πολύ σοβαρά μειωμένο κίνδυνο να καταστεί ανεξέλεγκτο (unmanageable).
3. Ο πυρήνας δεδομένων και προγραμμάτων του ΠΣ μπορεί, επίσης, να μειωθεί κατά βούληση με την επιλεκτική αφαίρεση υποθεμάτων και των συσχετισμένων με αυτά δεδομένων και προγραμμάτων, χωρίς να

υπάρχει κίνδυνος να αφαιρεθούν δεδομένα ή προγράμματα που χρειάζονται για θέματα ή προβλήματα που θα παραμείνουν στο σύστημα.

4. Το ΠΣ μπορεί να αυξηθεί θεματικά ή να υποστεί σημαντική θεματική στροφή, δεδομένου ότι η Τελεολογική Δομή επιτρέπει την ολοκλήρωση νέων θεμάτων ή προβλημάτων και των αντίστοιχων δεδομένων και προγραμμάτων με τρόπο ορθογώνιο ως προς την ήδη υπάρχουσα πληροφορία. Ο όρος ορθογώνιος δηλώνει ότι η νέα πληροφορία μπορεί να προστεθεί στο σύστημα χωρίς επικάλυψη με την ήδη υπάρχουσα. Επίσης, πρέπει να επιτρέπει την μέγιστη δυνατή επαναχρησιμοποίηση του πυρήνα δεδομένων και προγραμμάτων.
5. Το ΠΣ μπορεί να χειριστεί εύκολα τόσο την μη γεωσυσχετισμένη πληροφορία όσο και την γεωσυσχετισμένη. Στην περίπτωση που στο ίδιο σύστημα πρέπει να συνυπάρξουν εφαρμογές που έχουν ανάγκη διαφορετικών χαρτών και διαφορετικών σημαντικών γεωσυσχετισμένων οντοτήτων, η Τελεολογική Δομή μπορεί να επιτρέψει τον μερικό ή πλήρη διαχωρισμό των αναγκαίων δεδομένων, και, επομένως, την αποφυγή λανθασμένης χρήσης τους ή/και λανθασμένης οργάνωσης και τήρησής τους.
6. Επιτρέπεται δυναμικά η πρόσθεση και ομαλή ολοκλήρωση στο ΠΣ νέας πληροφορίας που παράγεται από τους χρήστες (είτε σε δική τους περιοχή, είτε, εάν οι χρήστες είναι εξουσιοδοτημένοι, στην βάση του ΠΣ).
7. Είναι δυνατόν να γίνουν ακόμη και σοβαρές αλλαγές στον πυρήνα δεδομένων ή/και προγραμμάτων του συστήματος με ελάχιστη διαταραχή της λειτουργίας του, δηλαδή χωρίς να χρειάζεται εκτεταμένη διακοπή της λειτουργίας του.
8. Είναι εφικτή η διαρκής εξέλιξη του συστήματος, με τα προφανή κέρδη σε χρήμα και χρόνο, δεδομένου ότι δεν χρειάζεται η εξ αρχής κατασκευή νέων εξελιγμένων μορφών του συστήματος. Με την έννοια αυτή, το σύστημα αυτό θα μπορούσε να χαρακτηριστεί αειφόρο (sustainable).

Αρχική μορφή της αρχιτεκτονικής αυτής έχει επίσης δοκιμαστεί στο πλαίσιο του προγράμματος ETIS-Agent και έχει επιδειχθεί σε εκπροσώπους Στατιστικών Υπηρεσιών και Υπουργείων Μεταφορών της Ευρωπαϊκής Ένωσης με πολύ

καλά αποτελέσματα. Η αναφερθείσα στα προηγούμενα σύγχρονη πλατφόρμα, που θα έχει πληροφορία για την υποδομή των μεταφορικών δικτύων και τις καταγεγραμμένες ή επιθυμητές ροές επιβατών και εμπορευμάτων και είναι υπό ανάπτυξη στο Εργαστήριο Συστημάτων Πολυθεματικής και Γεωσυσχετισμένης Πληροφορίας της ΣΗΜΜΥ ΕΜΠ, θα χρησιμοποιεί, επίσης, την προτεινόμενη Αρχιτεκτονική «Θέματα Υπεράνω Χαρτών» [15].

Ένα ενδεικτικό παράδειγμα για τις απαιτήσεις διαμόρφωσης δικτύων αποτελεί μια εργασία[13] με αντικείμενο την εύρεση βέλτιστων διαδρομών σε δίκτυα οδικών και συνδυασμένων εμπορευματικών μεταφορών. Στην σχετική διερεύνηση χρησιμοποιήθηκαν τα συγκοινωνιακά δίκτυα του ETIS (έκδοση 2012) και ελήφθησαν υπ' όψη οι κανονισμοί για το ωράριο εργασίας των οδηγών φορτηγών, οι καθυστερήσεις λόγω συνοριακών ελέγχων και (στην περίπτωση των συνδυασμένων θαλάσσιων διαδρομών) οι χρόνοι αναμονής των φορτηγών στους λιμένες μέχρι την αναχώρηση του αντίστοιχου πλοίου. Διαμορφώθηκε μια σειρά από σενάρια διαδρομών μεταξύ των πόλεων της Αθήνας και του Μονάχου στην Γερμανία. Χρησιμοποιήθηκε ο αλγόριθμος Dijkstra] με εναλλακτικά κριτήρια βελτιστοποίησης τον συντομότερο χρόνο και το μικρότερο λειτουργικό κόστος και αξιοποιήθηκε το γεγονός ότι για την μετάβαση από την Ελλάδα στην Γερμανία, τα φορτηγά πρέπει να περάσουν από συγκεκριμένους συνοριακούς σταθμούς στην περίπτωση της χρήσης αυτοκινητόδρομων ή να χρησιμοποιήσουν τις ναυτιλιακές γραμμές Ελλάδας-Ιταλίας. Η μεθοδολογία που ακολουθήθηκε ήταν:

1. Προσδιορισμός των χωρών πιθανής διέλευσης του φορτηγού οχήματος και αντίστοιχος περιορισμός του δικτύου.
2. Σε κάθε επιλεγείσα χώρα, λύση του επιμέρους προβλήματος βέλτιστης δρομολόγησης στο δίκτυο μεταξύ όλων των συνοριακών σημείων (συμπεριλαμβανομένων και των λιμένων).
3. Σε κάθε επιλεγείσα χώρα, λύση του επιμέρους προβλήματος βέλτιστης δρομολόγησης στο δίκτυο μεταξύ όλων των συνοριακών σημείων (συμπεριλαμβανομένων και των λιμένων).
4. Αντικατάσταση του αρχικού δικτύου της κάθε επιλεγείσας χώρας με το απλοποιημένο δίκτυο βέλτιστων διαδρομών ανάμεσα στα συνοριακά σημεία.

5. Επίλυση του συνολικού προβλήματος στο απλοποιημένο δίκτυο που πρέκυψε

Τα αποτελέσματα (διαδρομές) που προέκυψαν επιβεβαιώθηκαν ως προς την ορθότητα τους μέσω συνεντεύξεων με επαγγελματίες οδηγούς που εκτελούν μεταφορές μεταξύ Ελλάδας και Γερμανίας.

Η ανωτέρω περιγραφείσα διαμόρφωση του απλοποιημένου δικτύου απαίτησε σημαντικό κόπο και χρόνο που θα μπορούσε να είχε αποφευχθεί αν υπήρχε μία ευέλικτη δομή και κατάλληλα εργαλεία, σαν αυτά που προτείνονται στην παρούσα εργασία, για την επιθυμητή διαμόρφωση του αρχικού δικτύου, την υποβοήθηση του υπολογισμού των διαφόρων σεναρίων, την οργάνωση των αποτελεσμάτων και την τεκμηρίωση των επιμέρους βημάτων της επίλυσης του προβλήματος. Η κατασκευή των εργαλείων αυτών και της σχετικής γενικότερης πληροφοριακής δομής βρίσκεται υπό εξέλιξη.

6ο Κεφάλαιο: Πρόταση για νέο, πληρέστερο μοντέλο παράστασης των μεταφορικών δικτύων για τον σχεδιασμό μεταφορών

6.1 Πρόταση νέας οργάνωσης των παραστάσεων των μεταφορικών δικτύων

Από την εμπειρία και την εργασία των μελών του Εργαστηρίου Συστημάτων Πολυθεματικής και Γεωσυσχετισμένης Πληροφορίας της ΣΗΜΜΥ ΕΜΠ, προέκυψε ότι στις περισσότερες επί μέρους εφαρμογές των συστημάτων υποστήριξης λήψης αποφάσεων και χάραξης πολιτικής στις μεταφορές, όπως π.χ. των εφαρμογών για τις οποίες το ETIS καλείται να δώσει δεδομένα, είναι αναγκαίος ο διαχωρισμός της σχετικής γεωσυσχετισμένης πληροφορίας σε τρεις κατηγορίες:

- Το χαρτογραφικό υπόβαθρο, πάνω στο οποίο οπτικοποιείται οποιαδήποτε περαιτέρω πληροφορία. Το υπόβαθρο συνήθως οργανώνεται ως ένα ή περισσότερα επίπεδα μιας βάσης δεδομένων σε ένα GIS.
- Οι ειδικές γεωσυσχετισμένες οντότητες (objects), οι οποίες έχουν ιδιαίτερη σημασία για την συγκεκριμένη εφαρμογή και χρησιμοποιούνται ως *συνδετικοί κρίκοι* (linking objects) με την υπόλοιπη γεωσυσχετισμένη πληροφορία. Παραδείγματος χάριν, σε θέματα σχεδίασης μεταφορών οι πλέον σημαντικές οντότητες είναι τα δίκτυα μεταφορών και οι διοικητικές περιοχές, στις οποίες χωρίζεται η περιοχή ενδιαφέροντος. Με τα δίκτυα και τις διοικητικές περιοχές διασυνδέεται σημαντικός όγκος υπόλοιπων πληροφοριών, που ονομάζεται από τους συγγραφείς *έμμεσα γεωσυσχετισμένη πληροφορία*. Είναι προτιμότερο και οι σημαντικές οντότητες να οργανώνονται σε ένα GIS.
- Η έμμεσα γεωσυσχετισμένη πληροφορία, όπως αυτή ορίστηκε στο σημείο (2). Η πληροφορία αυτή είναι συνήθως μεγάλου όγκου και είναι προτιμότερο να μην οργανωθεί στο GIS που οργανώνεται η πληροφορία

της κατηγορίας (1) και (2), αλλά σε ξεχωριστές, πιθανόν εξωτερικές, βάσεις δεδομένων ή αποθήκες δεδομένων (data warehouses).

Η πράξη έχει καταδείξει ότι η διαφοροποίηση των επί μέρους προβλημάτων, μελετών ή εφαρμογών οδηγεί στην καθοριστική διαφοροποίηση της πληροφορίας των προηγούμενων κατηγοριών. Συνήθως, ωστόσο, στα προβλήματα, στων οποίων την επίλυση ένα μεγάλης έκτασης σύστημα υποστήριξης λήψης αποφάσεων και χάραξης πολιτικής στις μεταφορές επιβατών και εμπορευμάτων (όπως είναι το ETIS) καλείται να υποβοηθήσει, οι βασικές οντότητες περιλαμβάνουν τα *μεταφορικά δίκτυα* και τις *διοικητικές περιοχές*. Επομένως, στις δομές που προτείνονται, δίνεται ιδιαίτερη έμφαση στις οντότητες αυτές [10].

Για την επίλυση των προβλημάτων που αναφέρθηκαν στο προηγούμενο κεφάλαιο προτείνεται μια νέα εσωτερική αρχιτεκτονική της πληροφορίας που επικεντρώνεται στα τρία κύρια ακόλουθα σημεία:

1. Τον εμπλουτισμό της δυνατότητας προσδιορισμού θέσης πάνω στα δίκτυα με τον ευρύτερα χρησιμοποιούμενο στην πράξη τρόπο του καθορισμού της απόστασης από συγκεκριμένα σημεία αναφοράς πάνω σε μία καθορισμένη διαδρομή. Η παρουσίαση των αναγκαίων για το σκοπό αυτό δομών γίνεται στη συνέχεια του παρόντος κεφαλαίου.
2. Τη δυναμική αντιμετώπιση της διαφοροποίησης των μεταφορικών δικτύων που παρουσιάζεται στις περισσότερες από τις εφαρμογές των συστημάτων υποστήριξης λήψης αποφάσεων και χάραξης πολιτικής στις μεταφορές επιβατών και εμπορευμάτων. Η αντιμετώπιση αυτή βασίζεται κατ' αρχήν στα εξής:
    - Τη χρήση μιας νέας, καινοτομικής βάσης δεδομένων για την περιγραφή των δικτύων, όπου οργανώνεται η πληροφορία όλων των δυνατών κόμβων ενός συγκεκριμένου δικτύου, όλων των δυνατών συνδέσμων του ιδίου δικτύου και η *συνδεσιμότητα* αυτών των κόμβων και συνδέσμων.
    - Το δίκτυο που είναι αναγκαίο για κάθε επί μέρους εφαρμογή προκύπτει από την συνδεσιμότητα κόμβων και συνδέσμων και τη χρήση ειδικών κανόνων, που ο ειδικός-χρήστης θέλει να χρησιμοποιήσει στη συγκεκριμένη εφαρμογή. Δηλαδή, το αναγκαίο

δίκτυο συντίθεται «κατά ζήτηση» (on demand) ως παράγωγο της συνολικής βάσης δεδομένων, με τη χρήση των προαναφερθέντων κανόνων και οργανώνεται σε ένα GIS, ώστε να μπορεί να χρησιμοποιηθεί στη συνέχεια για να γίνουν οι αναγκαίοι υπολογισμοί ή/και να εφαρμοστούν τα αναγκαία υπολογιστικά μοντέλα. Προφανώς, οι κόμβοι και οι σύνδεσμοι του δικτύου αυτού πρέπει να χρησιμοποιηθούν για την οργάνωση της κατάλληλης διασυνδεδεμένης πληροφορίας.

- Τη χρήση της καινοτομικής «Τελεολογικής Δομής» ή «Χάρτη Περιεχομένων του Συστήματος» για την αντιμετώπιση της διαφοροποίησης της συνολικής πληροφορίας που είναι απαραίτητη για κάθε επί μέρους εφαρμογή, ώστε το προκύπτον πληροφοριακό σύστημα να μπορεί να αντιμετωπίσει την πολυθεματικότητα, τη διαφοροποίηση και πολυπλοκότητα των δεδομένων και τον τελικά μεγάλο όγκο δεδομένων που παρουσιάζονται σε συστήματα σαν το ETIS.
- Την ανάπτυξη επί μέρους εργαλείων λογισμικού για ειδικές διαδικασίες που δεν καλύπτονται από τις προηγούμενες συστηματικές δομές και απαιτούν ενέργειες όπως καθαρισμό (cleansing), φιλτράρισμα, επιλογή με ειδικά κριτήρια κ.τ.λ. στοιχείων από συγκεκριμένα υποσύνολα δεδομένων. Η ανάγκη αυτών των εργαλείων λογισμικού παρουσιάζεται επανειλημμένα στην πράξη.

3. Την οργάνωση των πολυθεματικών και μεγάλου όγκου συστημάτων γεωσυσχετισμένης πληροφορίας με βάση την Αρχιτεκτονική «Θέματα Υπεράνω Χαρτών» .

Στη συνέχεια του κεφαλαίου θα παρουσιαστεί ο τρόπος, με τον οποίον εμπλουτίζεται η δυνατότητα προσδιορισμού θέσης πάνω στα μεταφορικά δίκτυα με τον ευρύτερα χρησιμοποιούμενο στην πράξη τρόπο του καθορισμού της απόστασης από συγκεκριμένα σημεία αναφοράς πάνω σε μία καθορισμένη διαδρομή [14].

Αναγκαία για τη νέα αυτή δυνατότητα είναι η κατασκευή μίας δομής κατάλληλης για την περιγραφή διαδρομών (routes). Οι διαδρομές αυτές μπορεί να είναι είτε καθιερωμένες (όπως π.χ. στην Ελλάδα η Π.Α.Θ.Ε ή η Εγνατία), είτε ειδικές

διαδρομές που έχουν ανάγκη οι χρήστες του συστήματος (π.χ. η διαδρομή ενός λεωφορείου ή το δρομολόγιο ενός οχήματος εξυπηρέτησης των εργαζομένων μιας επιχείρησης). Για τη συγκεκριμένη δομή δεδομένων χρησιμοποιείται μια ειδική παράσταση διατεταγμένης ακολουθίας των περιλαμβανομένων συνδέσμων, που οργανώνεται ως μια διπλά συνδεδεμένη λίστα και η οποία περιλαμβάνει και την εσωτερική γεωμετρία αυτή καθ' εαυτή των συνδέσμων, δηλαδή την ακολουθία των κορυφών (vertices) των στοιχειωδών ευθυγράμμων τμημάτων με τα οποία παριστάνεται ο κυκλοφοριακός σύνδεσμος. Στην εσωτερική γεωμετρία, όμως, συμπεριλαμβάνεται και η απόσταση κάθε κορυφής από τα δύο άκρα του συνδέσμου στον οποίον ανήκει, ώστε να διευκολυνθεί ο χρησιμοποιούμενος αλγόριθμος προσδιορισμού θέσης. Η παράσταση των διαδρομών είναι πάντα αμφίδρομη, ώστε να επιτρέπει κίνηση προς οποιαδήποτε κατεύθυνση.

Ο αλγόριθμος προσδιορισμού σημείου με βάση τη χιλιομετρική απόσταση από σημείο αναφοράς περιλαμβάνει κατ' αρχήν τον προσδιορισμό των συνδέσμων της εμπλεκομένης διαδρομής και της φοράς διάσχισης της διαδρομής, δεδομένου ότι ο καθορισμός αυτός είναι μεν μονοσήμαντος, αλλά όχι τυποποιημένος. Παραδείγματος χάριν, μπορεί να δοθούν οι οδικές διαδρομές «Αθήνα-Μέγαρα», ή «Αθήνα-Πάτρα», ή «Κόρινθος-Κιάτο», ή «Αθήνα-Λαμία», ή «Αθήνα-Θεσ/νίκη» που όλες ανήκουν στην πολύ ευρύτερη Π.Α.Θ.Ε. (εθνική οδός Πατρών – Αθηνών – Θεσσαλονίκης – Ευζώνων) και έχουν διαφορετικές φορές. Στη συνέχεια, χρησιμοποιείται η γεωγραφική πληροφορία των επί μέρους συνδέσμων της κάθε διαδρομής, καθώς και το αρχικό σημείο και η φορά της διαδρομής, για να υπολογιστεί ταχύτητα το στοιχειώδες ευθύγραμμο τμήμα πάνω στο οποίο βρίσκεται το υπό προσδιορισμό σημείο. Απλοί γεωμετρικοί υπολογισμοί δίνουν τις συντεταγμένες (x, y) του ζητούμενου σημείου, στις οποίες δημιουργείται μία νέα κορυφή στοιχειωδών ευθυγράμμων τμημάτων (vertex). Οποιαδήποτε περαιτέρω αναφορά στο συγκεκριμένο σημείο γίνεται με χρήση της τριάδας (ταυτότητα (ID) του συνδέσμου πάνω στο οποίο βρίσκεται το σημείο, άκρο αναφοράς του συνδέσμου αυτού, απόσταση από το προηγούμενο άκρο αναφοράς) ή/και με το ήδη υπολογισμένο ζεύγος συντεταγμένων (x, y) του σημείου. Στα επόμενα θα καλούμε την προηγούμενη τριάδα προσδιορισμού του σημείου ως *«χιλιομετρική αναφορά»*.

Σημειακά χαρακτηριστικά των δικτύων προσδιορίζονται με χρήση της χιλιομετρικής αναφοράς. Ως παράδειγμα μπορούμε να θεωρήσουμε τα σημεία αλλαγής του αριθμού των λωρίδων ενός αυτοκινητοδρόμου (οι οποίες μπορεί να αλλάζουν αρκετές φορές μέσα σε ένα σύνδεσμο), τα σημεία αλλαγής των ορίων ταχύτητας κ.τ.λ. Με σημεία των δικτύων που προσδιορίζονται με χρήση της χιλιομετρικής αναφοράς συνηθίζεται να διασυνδέεται περαιτέρω έμμεσα γεωσυσχετισμένη πληροφορία, όπως π.χ. τα ατυχήματα πάνω σε μία διαδρομή.

Προτείνονται δε επί πλέον δυναμικές δομές, που χρησιμοποιούν την χιλιομετρική αναφορά σημείων για την παράσταση:

- Διατεταγμένων συνόλων σημειακών χαρακτηριστικών (δομικών, λειτουργικών, στατιστικών ή άλλων).
- Διατεταγμένης ή μη περιοχής ενός συνδέσμου με την οποία διασυνδέεται ένα ή περισσότερα επί μέρους χαρακτηριστικά του συνδέσμου, καλούμενων στο εξής «μερικών χαρακτηριστικών» (partial attributes).
- Διατεταγμένης ή μη περιοχής μιας διαδρομής με την οποία διασυνδέεται ένα ή περισσότερα στοιχεία δεδομένων.
- Διατεταγμένης ακολουθίας συνεχόμενων περιοχών μιας διαδρομής. Με κάθε μια από τις περιοχές αυτές διασυνδέεται ένα ή περισσότερα στοιχεία δεδομένων.
- Διατεταγμένης ακολουθίας *μη* συνεχόμενων περιοχών μιας διαδρομής. Με κάθε μια από τις περιοχές αυτές διασυνδέεται ένα ή περισσότερα στοιχεία δεδομένων.

Οι προτεινόμενες δομές έχουν τέτοια μορφή, ώστε να μην εισάγουν νέους κόμβους στο δίκτυο και να επιτρέπουν γενικότερες λογικές και αριθμητικές πράξεις πάνω στα χιλιομετρικά προσδιορισμένα χαρακτηριστικά των συνδέσμων (όπως, για παράδειγμα, η εύρεση των τμημάτων ενός αυτοκινητοδρόμου που έχουν όριο ταχύτητας 70 km/h και μόνο δύο λωρίδες ανά κατεύθυνση, ή η εύρεση του συνόλου των ατυχημάτων που συμβαίνει από ένα σημείο του αυτοκινητοδρόμου μέχρι ένα άλλο). Επί πλέον δε, η μορφή των δομών αυτών επιτρέπει την δυναμική πρόσθεση πολλών διαφορετικών

σημειακών ή επί μέρους χαρακτηριστικών, με ελάχιστο φόρτο του υπολογιστικού συστήματος.

Οι προαναφερθείσες δομές χαρακτηρίστηκαν ως δυναμικές, διότι ο προσδιορισμός νέων διαδρομών και σημειακών ή μερικών χαρακτηριστικών των συνδέσμων ή των διαδρομών είναι δυνατός καθ' όλη τη διάρκεια της ζωής του αντίστοιχου πληροφοριακού συστήματος.

Με χρήση δε της περιγραφόμενης στο επόμενο κεφάλαιο Τελεολογικής Δομής δίνεται η δυνατότητα σε κάθε εξουσιοδοτημένο χρήστη του συστήματος να δημιουργεί τη δική του ιδιωτική περιοχή πληροφορίας και να εργάζεται σε αυτήν, προσδιορίζοντας και χρησιμοποιώντας τις διαδρομές, για τις οποίες ο ίδιος ενδιαφέρεται, χωρίς να χάνει τη δυνατότητα χρήσης των καθιερωμένων ή/και των ευρέως χρησιμοποιούμενων διαδρομών [15].

6.2 Προτεινόμενες Δομές Κόμβου – Συνδέσμου

Προκειμένου να απεικονιστεί με συνεκτικό, ισχυρό τρόπο ένα δίκτυο μεταφορών, πρέπει να καθορίσουμε τον κόμβο και τον σύνδεσμο ως δομές δεδομένων. Οι προαναφερόμενες δομές δεδομένων θα πρέπει να είναι ισότιμες όσον αφορά την εκπροσώπηση των δεδομένων και θα πρέπει να υπάρχει η δυνατότητα μετατροπής του δικτύου, χωρίς απώλειες, από τη μία μορφή στην άλλη.

Μια σημαντική πτυχή των δομών δεδομένων μας είναι ο τρόπος τα με τον οποίο παρουσιάζονται τα δεδομένα. Εκτός από την αποθήκευση των δεδομένων, είναι σημαντικό να έχουμε μία αναπαράσταση η οποία να επιτρέπει την εφαρμογή πληθώρα διαδικασιών στα δεδομένα του δικτύου. Ιδανικά η αναπαράσταση αυτή δεν θα πρέπει να έχει υψηλό ΄κόστος΄ σε υπολογιστική πολυπλοκότητα. Σε ένα δίκτυο μεταφορών μας ενδιαφέρουν ιδιαίτερα οι εξής διαδικασίες:

- Εισαγωγή ενός κόμβου στο δίκτυο
- Διαγραφή κόμβου από το δίκτυο
- Προσπέλαση του δικτύου με διάφορους τρόπους(για παράδειγμα κάποιος μπορεί να θέλει να βρει το συντομότερο δρόμο, από χιλιομετρικής απόψεως, μεταξύ αφετηρίας και προορισμού ενώ κάποιος

άλλος μπορεί να ενδιαφέρεται να βρει την διαδρομή με το συνολικό ελάχιστο κόστος συμπεριλαμβανομένων του χρόνου και της απόστασης

Επιπλέον, μας ενδιαφέρει να αναπαριστούμε, να διατηρούμε και να επεξεργαζόμαστε συνεχώς όλο και μεγαλύτερο όγκο δεδομένων που αφορούν μεταφορικά δίκτυα. Έτσι, οι δομές που θα κατασκευάσουμε πρέπει να έχουν τέτοια μορφή ώστε να μπορούν να καλύψουν μελλοντικές ανάγκες. Για παράδειγμα, ένας κόμβος μπορεί να αναπαριστά ένα σταθμό τραίνου ενώ μετά το πέρασμα κάποιων μηνών να προστεθεί και σταθμός λεωφορείου. Οποιαδήποτε πληροφορία λοιπόν πρόκειται να προστεθεί στις δομές μας δεν πρέπει να αυξάνει την πολυπλοκότητα των δεδομένων ή να το επηρεάζει τις λειτουργίες του.

Επομένως οι βασικές μας προδιαγραφές είναι οι εξής:

- Σωστή αναπαράσταση χωρίς απώλειες
- Δυνατότητα μετατροπής από την μία δομή στην άλλη
- Προσαρμοστικότητα, όσον αφορά μελλοντικές εισαγωγές δεδομένων

Ας παρουσιάσουμε λοιπόν τις δύο δομές δεδομένων, μία για κόμβους και μία για συνδέσμους.

Κόμβος

Ο κόμβος θα πρέπει να είναι μία δομή με τα εξής χαρακτηριστικά:

- Το id του κόμβου, το οποίο είναι μοναδικό για κάθε κόμβο
- Την γεωμετρία του δηλαδή το ζευγάρι x, y του κόμβου που καθορίζουν την θέση του κόμβου στον χάρτη. Προς το παρόν περιέχει μόνο x, y ενώ μελλοντικά πρέπει να προστεθεί και z.
- Μία μεταβλητή που θα καθορίζει την σημαντικότητα του κόμβου
- Μία μεταβλητή που θα καθορίζει την κλάση του κόμβου (για παράδειγμα η κλάση μπορεί να αναφέρεται στον τύπο του κόμβου π.χ. επίγειος, θαλάσσιος)
- Μία μεταβλητή που καθορίζει πόσοι σύνδεσμοι ξεκινούν από αυτό τον κόμβο
- Και μία μεταβλητή που καθορίζει πόσοι σύνδεσμοι καταλήγουν σε αυτό τον κόμβο

Σύνδεσμος

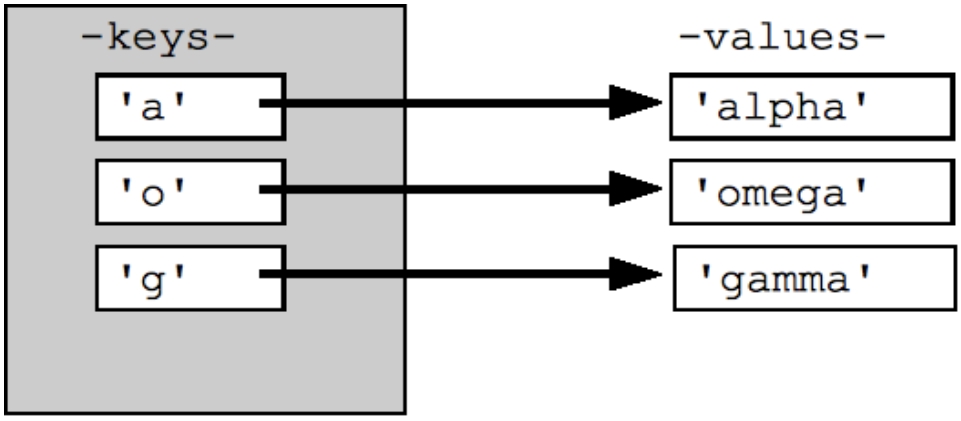
Ο σύνδεσμος θα πρέπει να είναι μία δομή με τα εξής χαρακτηριστικά:

- Το id του συνδέσμου, το οποίο είναι μοναδικό για κάθε σύνδεσμο
- Το id του κόμβου που έχει ως αφετηρία
- Το id του κόμβου που έχει ως προορισμό
- Το κόστος εκκίνησης από τον κόμβο-αφετηρία προς τον κόμβο-προορισμό. Το κόστος εκκίνησης μπορεί κάλλιστα να αναπαριστά ένα χρηματικό κόστος π.χ. διόδια αλλά μπορεί επίσης να αφορά χρονικό κόστος π.χ. ο απαιτούμενος χρόνος ενός επιβάτη για μετεπιβίβαση σε άλλο αεροπλάνο
- Το κόστος διάβασης του συνδέσμου. Είτε χρηματικό, είτε χρονικό.
- Μία μεταβλητή που θα καθορίζει την σημαντικότητα του συνδέσμου
- Μία μεταβλητή που θα καθορίζει την κλάση του συνδέσμου και μας βοηθάει όπως στην περίπτωση των κόμβων στην κατηγοριοποίηση των συνδέσμων σύμφωνα με κοινά χαρακτηριστικά πλήρως καθορισμένα από εμάς. Για παράδειγμα, η κλάση ΄1΄ μπορεί να αναφέρεται σε συνδέσμους που ανήκουν στο οδικό δίκτυο ενώ κλάση ΄2΄ να αναφέρεται στις ακτοπλοϊκές γραμμές κ.ο.κ.
- Μία μεταβλητή που θα καθορίζει αν ο σύνδεσμος είναι διπλής κατευθύνσεως
- σε περίπτωση που είναι διπλής κατευθύνσεως μία το κόστος εκκίνησης από τον κόμβο-προορισμό προς τον κόμβο αφετηρία
- Την γεωμετρία του δηλαδή όλα τα ζευγάρια x, y που καθορίζουν την θέση του συνδέσμου στον χάρτη

Προφανώς αυτά τα χαρακτηριστικά που αναφέρονται παραπάνω μπορούν να παραμετροποιηθούν και/ή να προστεθούν στο μέλλον περισσότερα.

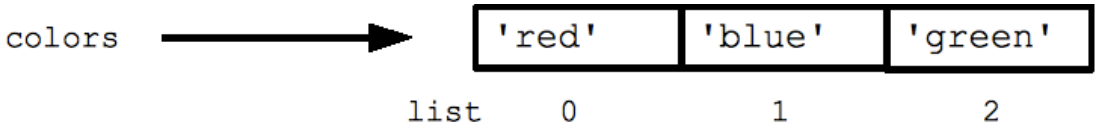
6.3 Δομή Γειτνίασης

Αφού λοιπόν έχουμε καθορίσει τις δομές του κόμβου και του συνδέσμου πρέπει καθορίσουμε και μία δομή η οποία ουσιαστικά θα αποτελεί την πλήρη αναπαράσταση του δικτύου μας. Για αυτό τον σκοπό λοιπόν κατασκευάζουμε το adjacency dictionary ή αλλιώς το dictionary γειτνίασης. Το dictionary είναι μία δομή της Python που αποτελείται από ένα ζευγάρι κλειδιού και τιμής.



Εικόνα 14 Python Dictionary

Στην δική μας δομή το κλειδί είναι το id του κάθε κόμβου ενώ η τιμή που αντιστοιχεί στον κάθε κόσμο κλειδί είναι ένας αριθμός από λίστες Python.



Εικόνα 15 Python List

Η πρώτη λίστα που συναντάμε στο adjacency dictionary είναι η λίστα που περιέχει όλες τις βασικές και προσωπικές πληροφορίες του κόμβου κλειδί όπως αυτά περιγράφηκαν στην δομή του κόμβου. Αμέσως μετά, η δεύτερη λίστα χωρίζεται σε δύο υπο-λίστες, όπου η πρώτη υπολίστα έχει τα χαρακτηριστικά των συνδέσμων που έχουν σαν αφετηρία τον κόμβο κλειδί ενώ η δεύτερη υπο-λίστα περιέχει τα id των κόμβων που έχουν σαν προορισμό τον κόμβο κλειδί.

Βασικά πλεονεκτήματα της παραπάνω δομής:

- Σχετική απεξάρτηση του δικτύου από την βάση δεδομένων. Είναι δυνατή η επεξεργασία και μετατροπή του δικτύου χωρίς καμία αλλαγή στο αρχικό μας δίκτυο το οποίο είναι αποθηκευμένο στην βάση δεδομένων. Έτσι είναι εύκολη η δημιουργία παράγωγων δικτύων για διαφορετικές εφαρμογές ανάλογα με τις επιθυμίες του κάθε χρήστη. Μετά από οποιαδήποτε διαδικασία και κινούμενοι αντίστροφα μπορούμε να αποθηκεύσουμε το παράγωγο δίκτυο σε μία καινούρια βάση δεδομένων ή σε αρχεία κατάλληλα για τα GIS.

- Υψηλή ταχύτητα προσπέλασης της δομής καθώς οποιοδήποτε dictionary της Python σώνεται προσωρινά στην μνήμη Ram του υπολογιστή γεγονός που κάνει την δομή αυτή ιδιαίτερα αποδοτική.
- Δυνατότητα επέκτασης και παραμετροποίησης. Η παραπάνω δομή μπορεί να ανταπεξέλθει στην συνεχόμενη αύξηση των δεδομένων ενός δικτύου ενώ οποιαδήποτε επέκταση θεωρηθεί αναγκαία στο μέλλον, είναι εύκολα πραγματοποιήσιμη.

Στην επόμενη σελίδα ακολουθεί κατάλληλο σχήμα ώστε η παραπάνω δομή να γίνει πιο κατανοητή.

Εικόνα 16 Adjacency Dictionatry

7ο Κεφάλαιο: Πρόταση και υλοποίηση αναγκαίων εργαλείων λογισμικού για την αποτελεσματική εργασία στα δίκτυα μεταφορών.

Από τα προηγούμενα σαφώς προκύπτει ότι, σε πολλά προβλήματα που έχουν σχέση με τα δίκτυα μεταφορών, είναι αναγκαία μια διαφορετική οργάνωση της περιγραφής της πληροφορίας. Επί πλέον δε, όπως ήδη αναφέρθηκε, πέραν της χρήσης νέων δομών δεδομένων, χρειάζονται πρόσθετα εργαλεία λογισμικού, ώστε να υπάρχει η δυνατότητα της επιλεκτικής αφαίρεσης ή/και πρόσθεσης κόμβων σε ένα δίκτυο και της συνεπαγόμενης διαίρεσης ή συνένωσης συνδέσμων.

Στην τρέχουσα εργασία της ερευνητικής ομάδας, επιχειρείται η ανάπτυξη μιας καινοτομικής *δυναμικής* μεθοδολογίας (και της αντίστοιχης εσωτερικής αρχιτεκτονικής των δεδομένων) για τον χειρισμό των γεωσυσχετισμένων (και μη) δεδομένων που έχουν σχέση με τα δίκτυα μεταφορών, η οποία δίνει στους χρήστες αυξημένες δυνατότητες και αντιμετωπίζει με επιτυχία την πλειοψηφία των ανιχνευθέντων στα προηγούμενα προβλημάτων. Τα βασικά σημεία της μεθοδολογίας αυτής είναι:

- Η γεωσυσχετισμένη πληροφορία των λειτουργικών κόμβων και συνδέσμων της υποδομής μεταφορών (κυρίως για προβλήματα σχεδίασης μεταφορών) είναι αποθηκευμένη σε μία γενικότερη, ειδικής δομής βάση δεδομένων, ενώ η τελική παράσταση στον υπολογιστή του εκάστοτε μεταφορικού δικτύου, πάνω στο οποίο πρέπει να εφαρμοστούν τα υπολογιστικά μοντέλα επίλυσης προβλημάτων μεταφορών, πρέπει να προκύπτει ως μία παράγωγη, μερική εικόνα της γεωσυσχετισμένης πληροφορίας της βάσης αυτής.
- Ο τρόπος προσδιορισμού της παράστασης του παράγωγου, τελικού δικτύου γίνεται:
    1. Με την χρήση ειδικών κανόνων για την επιλογή των λειτουργικών κόμβων που θα συμμετάσχουν στο τελικό δίκτυο και

2. αντίστοιχων κανόνων για τους συνδέσμους που προκύπτουν από την ομαδοποίηση επί μέρους συνδέσμων του αρχικού δικτύου, ώστε να υπάρχουν ενιαίοι (δηλαδή μη κατακερματισμένοι) σύνδεσμοι ανάμεσα στους κόμβους που επελέγησαν στο προηγούμενο βήμα και στις διασταυρώσεις του δικτύου.
3. Οι κανόνες επιλογής λειτουργικών κόμβων εξαρτώνται από την εκάστοτε εφαρμογή. Η επιλογή των κατάλληλων κόμβων δεν είναι τετριμμένη εργασία και πρέπει να γίνεται από ειδικούς στον συγκεκριμένο τομέα των μεταφορών.

- Προκειμένου να αντιμετωπιστούν προβλήματα, τα οποία προκύπτουν από την πιθανή συσσώρευση μεγάλου όγκου μη τυπικής, πολύμορφης πληροφορίας (δηλαδή πληροφορίας επιθυμητής μεν, που μπορεί ωστόσο να πάρει πολλές διαφορετικές μορφές ανάλογα με τα προβλήματα που πρέπει να επιλυθούν, ή με τις ανάγκες πολλών διαφορετικών χρηστών), καθώς και να καταστεί δυνατή πιθανά επιθυμητή θεματική στροφή ή εξέλιξη, εάν χρησιμοποιείται ως οδηγός και ελεγκτής της λειτουργίας του συνολικού συστήματος μία κατάλληλη δομή μεταδεδομένων, η «Τελεολογική Δομή» ή «Χάρτης Περιεχομένων του Συστήματος», η οποία περιγράφει με συγκεκριμένο τρόπο το περιεχόμενο του Πληροφοριακού Συστήματος. Λόγω της ευρύτητας του θέματος της Τελεολογικής Δομής, η περιγραφή της δομής αυτής γίνεται στο επόμενο κεφάλαιο.

Η ανάγκη χρησιμοποίησης νέων δομών δεδομένων παρουσιάζεται ακόμη στην προσπάθεια εφαρμογής πάνω στα υπάρχοντα μεταφορικά δίκτυα τυπικών αλγορίθμων εύρεσης βέλτιστών διαδρομών (όπως ο αλγόριθμος του Dijkstra, ο αλγόριθμος "A* search", ο αλγόριθμος του Viterbi, οι αλγόριθμοι για την εύρεση των "K-best" διαδρομών κ.ά.). Αυτό συμβαίνει επειδή οι πιο συνηθισμένες παραστάσεις δικτύων μεταφορών (που στην πραγματικότητα αποτελούν γράφους) στηρίζονται σε μια δομή, η οποία ονομάζεται μήτρα συνδεσιμότητας (connectivity matrix) και δεν είναι κατάλληλη για την αποτελεσματική εφαρμογή των προαναφερθέντων αλγορίθμων. Ως παράδειγμα, στην περίπτωση του ευρύτατα χρησιμοποιούμενου στην πράξη αλγορίθμου του Dijkstra, η μήτρα συνδεσιμότητας πρέπει να μετασχηματιστεί

σε μια δομή περιγραφής του γράφου (δικτύου) που ονομάζεται κατάλογος γειτνίασης (adjacency list), ενώ για την μείωση του χρόνου υπολογισμού πρέπει να χρησιμοποιηθούν δομές προτεραιότητας κόμβων του τύπου των δυαδικών ή κ-δικών σωρών (binary or k-ary heaps) ή των σωρών Fibonacci (Fibonacci heaps) [10] [14].

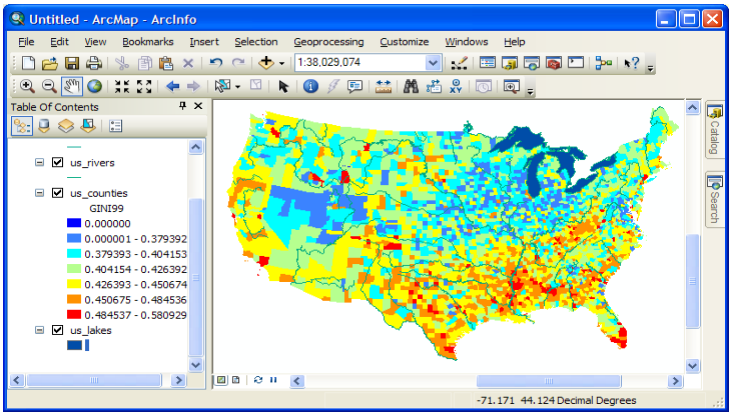
Στην συνέχεια θα αναφέρουμε τα εργαλεία τα οποία κατασκευάστηκαν στο πλαίσιο της διπλωματικής εργασίας. Πρώτα όμως πρέπει να αναφέρουμε ποιές τεχνολογίες λογισμικού που χρησιμοποιήσαμε για την κατασκευή των συγκεκριμένων εργαλείων.

7.1 ESRI ArcGIS Software

Το **ArcGIS** αποτελεί ένα τελευταίας τεχνολογίας, καινοτόμο και τέλεια δομημένο λογισμικό GIS. Αναπτύχθηκε για πρώτη φόρα στην δεκαετία του 1970 και βελτιώνεται συνεχώς τα τελευταία 40 χρόνια.

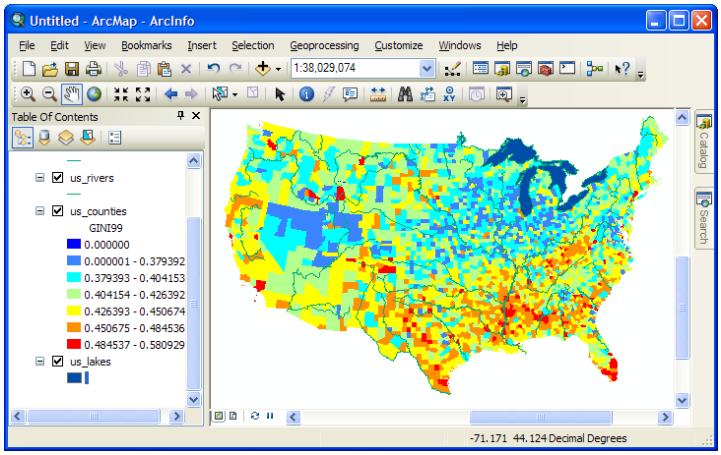
Το **ArcMap** είναι το κύριο περιβάλλον του λογισμικού GIS με ένα προσαρμόσιμο περιβάλλον εργασίας για τον χρήστη με εργαλεία χωρικής ανάλυσης και χαρτογράφησης.

Αποτελεί το βασικό λογισμικό επειδή στόχος του είναι η γρήγορη χωρική ανάλυση, η ψηφιοποίηση και η ανάπτυξη χαρτών σε ένα 2D περιβάλλον.



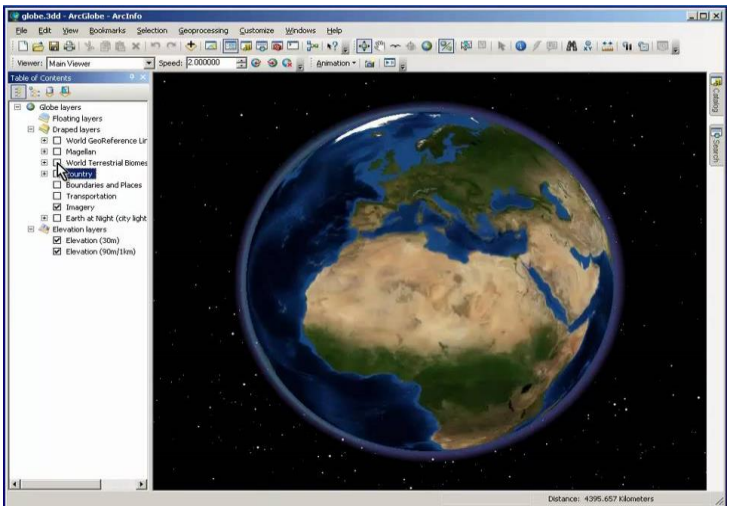
Εικόνα 17 ArcMap

Από τους βασικούς σκοπούς των GIS λογισμικών είναι η διαχείριση δεδομένων. Το **ArcCatalog** την επεξεργασία και κατασκευή βάσεων δεδομένων και μεταδεδομένων. Τα υποστηριζόμενα αρχεία είναι τα shapefiles, geodatabases και αρχεία raster.



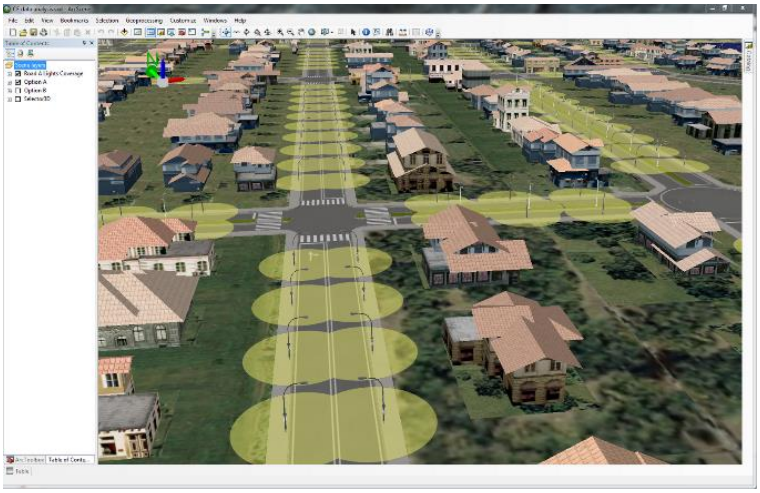
Εικόνα 18 ArcCatalog

Το **ArcGlobe** είναι μια παγκόσμιο τρισδιάστατο περιβάλλον οπτικοποίησης και ανάλυσης το οποίο ειδικεύεται σε παγκόσμια σύνολα δεδομένων και σε ευρύτερες περιοχές μελέτης.



Εικόνα 19 ArcGlobe

Το **ArcScene** είναι μία 3D λειτουργία της σουίτας εφαρμογών του Esri ArcGIS που ειδικεύεται σε μικρές περιοχές μελέτης.



Εικόνα 20 ArcScene

Επιμέρους εργαλεία του ArcGIS [16]:

- Το Analyst 3D το οποίο παρέχει εργαλεία για την μοντελοποίηση της επιφάνειας και 3D απεικόνιση
- Το ArcScan υποστηρίζει τη δημιουργία αυτοματοποιημένων χαρακτηριστικών από raster εικόνες.
- Η επέκταση Geostatistical Analysis η οποία παρέχει ένα ευρύ φάσμα χωρικών αναλυτικών εργαλείων για την διερεύνηση και πρόβλεψη αβεβαιοτήτων.
- Τα εργαλεία The Network Analysis υποστηρίζουν τη διατήρηση και την ανάλυση των δεδομένων δικτύων.
- Η The Spatial Analyst επέκταση παρέχει εργαλεία χωρικής ανάλυσης με δεδομένα ράστερ.

Στο πλαίσιο της διπλωματικής εργασίας έγινε χρήση κυρίως του ArcMap με σκοπό την εξαγωγή κατάλληλου Shapefile το οποίο χρησιμοποιήθηκε κατά την διαδικασία των προσομοιώσεων και της πρακτικής εφαρμογής των εργαλείων. Συγκεκριμένα κάναμε χρήση των δεδομένων της European Transport policy Information System **(ETIS)** για τους σιδηροδρόμους της Ελλάδας και εξαγάγαμε ένα Shapefile που περιείχε μέρος του σιδηροδρομικού δικτύου της κεντρικής Πελοποννήσου.

7.2 Microsoft SQL

Η **SQL** (από το **Structured Query Language**) είναι μία γλώσσα υπολογιστών στις βάσεις δεδομένων, που σχεδιάστηκε για τη διαχείριση δεδομένων, σε ένα

σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (Relational Database Management System, RDBMS) και η οποία, αρχικά, βασίστηκε στη σχεσιακή άλγεβρα. Η γλώσσα περιλαμβάνει δυνατότητες ανάκτησης και ενημέρωσης δεδομένων, δημιουργίας και τροποποίησης σχημάτων και σχεσιακών πινάκων, αλλά και ελέγχου πρόσβασης στα δεδομένα. Η SQL ήταν μία από τις πρώτες γλώσσες για το σχεσιακό μοντέλο του *Edgar F. Codd*, στο σημαντικό άρθρο του το 1970, και έγινε η πιο ευρέως χρησιμοποιούμενη γλώσσα για τις σχεσιακές βάσεις δεδομένων.

Η SQL αναπτύχθηκε στην IBM από τους Andrew Richardson, Donald C. Messerly και Raymond F. Boyce, στις αρχές της δεκαετίας του 1970. Αυτή η έκδοση, αποκαλούμενη αρχικά SEQUEL, είχε ως σκοπό να χειριστεί και να ανακτήσει τα στοιχεία που αποθηκεύτηκαν στο πρώτο RDBMS της IBM, το System R. .

Το πρώτο σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS) ήταν το RDMBS που αναπτύχθηκε στο MIT, στις αρχές της δεκαετίας του 1970 και η Ingres, που αναπτύχθηκε το 1974 στο Πανεπιστήμιο Μπέρκλεϋ. Η Ingres εφάρμοσε μία γλώσσα διατύπωσης ερωτήσεων γνωστή ως QUEL, το οποίο αντικαταστάθηκε αργότερα στην αγορά από την SQL.

Προς το τέλος της δεκαετίας του 70 η Relational Software (τώρα Oracle Corporation) είδε τη δυνατότητα αυτών που περιγράφηκαν από Codd, Chamberlin, και Boyce και ανάπτυξε την SQL βασισμένο στο RDBMS, με τις φιλοδοξίες πώλησης του στο Αμερικανικό ναυτικό, την Κεντρική Υπηρεσία Πληροφοριών και άλλες Αμερικανικές Υπηρεσίες. Το καλοκαίρι του 1979, η Relational Software εισήγαγε την πρώτη διαθέσιμη στο εμπόριο εφαρμογή του SQL και νίκησε την IBM με τη διάθεση του πρώτου εμπορικού RDBMS για μερικές εβδομάδες [17].

Στο πλαίσιο της διπλωματικής εργασίας έγινε χρήση της MySQL για την κατασκευή μία βάσης δεδομένων για την υποστήριξη του νέου μοντέλου που αναφέρθηκε στο κεφάλαιο 6. Η βάση αυτή περιέχει δύο tables. Το πρώτο περιέχει το σύνολο τον κόμβων του δικτύου ενώ το δεύτερο περιέχει το σύνολο των συνδέσμων.

Το table των **κόμβων** αποτελείται από τις εξής στήλες:

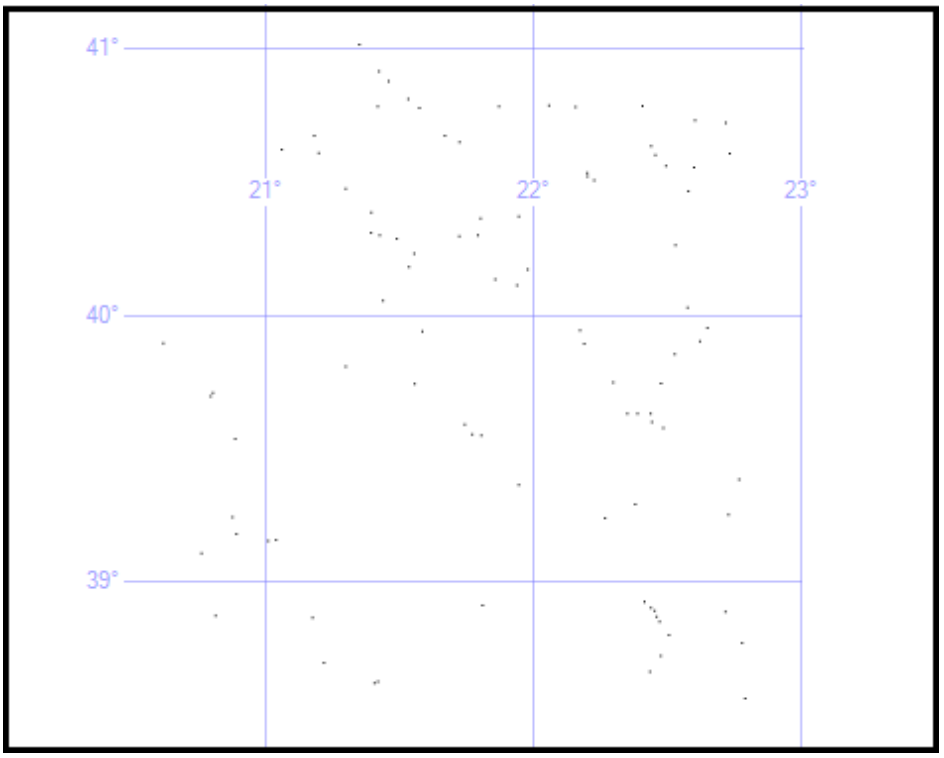
- FID. Μεταβλητή που χρησιμοποιείται για την απαρίθμηση των γραμμών του table.
- ID. Το ID του κόμβου.
- Visibility. Μεταβλητή σημαντικότητας του κόμβου.
- CLASS. Μεταβλητή που καθορίζει την κλάση του κόμβου.
- LinksExiting. Μεταβλητή που καθορίζει πόσοι σύνδεσμοι εξέρχονται από τον κόμβο.
- LinksEntering. Μεταβλητή που καθορίζει πόσοι σύνδεσμοι εισέρχονται στον κόμβο.
- GEOMETRY. Οι συντεταγμένες του κόμβου

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
4	4	103454	1	4	NULL	NULL	0xE6100000010C226E4E2503824440855CA96741583540
5	5	124933	1	1	NULL	NULL	0xE6100000010C077DE9EDCF7143407C2AA73D25B73640
6	6	103234	1	5	NULL	NULL	0xE6100000010C764F1E166A6F4340C4EDD0B018753640
7	7	124929	1	3	NULL	NULL	0xE6100000010C6A8995D1C80344400F99F221A8923640
8	8	124930	1	3	NULL	NULL	0xE6100000010C0A12DBDD03FA4340DB368C82E0A53640
9	9	124931	1	3	NULL	NULL	0xE6100000010CECA2E8818FF3434044C362D4B59E3640
10	10	124932	1	5	NULL	NULL	0xE6100000010C5AF5B9DA8AED4340D8F0F44A59863640
11	11	168092	1	3	NULL	NULL	0xE6100000010C69C4CC3E8FDF434089B48D3F51793640
12	12	103256	1	2	NULL	NULL	0xE6100000010CF3AE7AC03C50434031CF4A5AF1693540
13	13	186325	1	1	NULL	NULL	0xE6100000010C739EB12FD9744340799109F835CE3540
14	14	103141	1	3	NULL	NULL	0xE6100000010C55A52DAEF147444002B9C491077E3640
15	15	103137	1	4	NULL	NULL	0xE6100000010C5D8AABCABE3B444065C746205E933640
16	16	103180	1	4	NULL	NULL	0xE6100000010C0FD6FF39CC214440768A558330873640
17	17	103257	1	4	NULL	NULL	0xE6100000010CD252793BC27343409AEAC9FCA36F3640
18	18	103140	1	4	NULL	NULL	0xE6100000010CAD4ECE50DC5D444066BFEE74E7993640
19	19	103138	1	1	NULL	NULL	0xE6100000010C74620FED63514440BCE9961DE26F3640
20	20	103208	1	2	NULL	NULL	0xE6100000010CB073D3669C264440B22FD978B0C93540
21	21	103214	1	3	NULL	NULL	0xE6100000010CF5D896016711444090F9804067DA3540
22	22	103215	1	4	NULL	NULL	0xE6100000010CEAD2DB9E8B704440CAC4AD8218743540

Εικόνα 21 Δεδομένα Κόμβων σε SQL βάση δεδομένων

Στην παραπάνω εικόνα βλέπουμε τον πίνακα αποτελεσμάτων για του κόμβους ενός μέρους του σιδηροδρομικού δικτύου της Πελοποννήσου. Τα δεδομένα για τις στήλες ID, GEOMETRY προέρχονται από την ETIS ενώ οι για τις στήλες VISIBILITY και CLASS εισάγαμε δικά μας τυχαία δεδομένα. Οι στήλες LinksExiting και LinksEntering είναι αρχικά κενές ενώ υπάρχει κατάλληλο εργαλείο για να υπολογίζει τις στήλες αυτές. Παρακάτω βλέπουμε και τα χωρικά αποτελέσματα για τους κόμβους αυτούς.



Εικόνα 22 Χωρική αναπαράσταση Κόμβων σε SQL βάση δεδομένων

Το table των **συνδέσμων** αποτελείται από τις εξής στήλες:

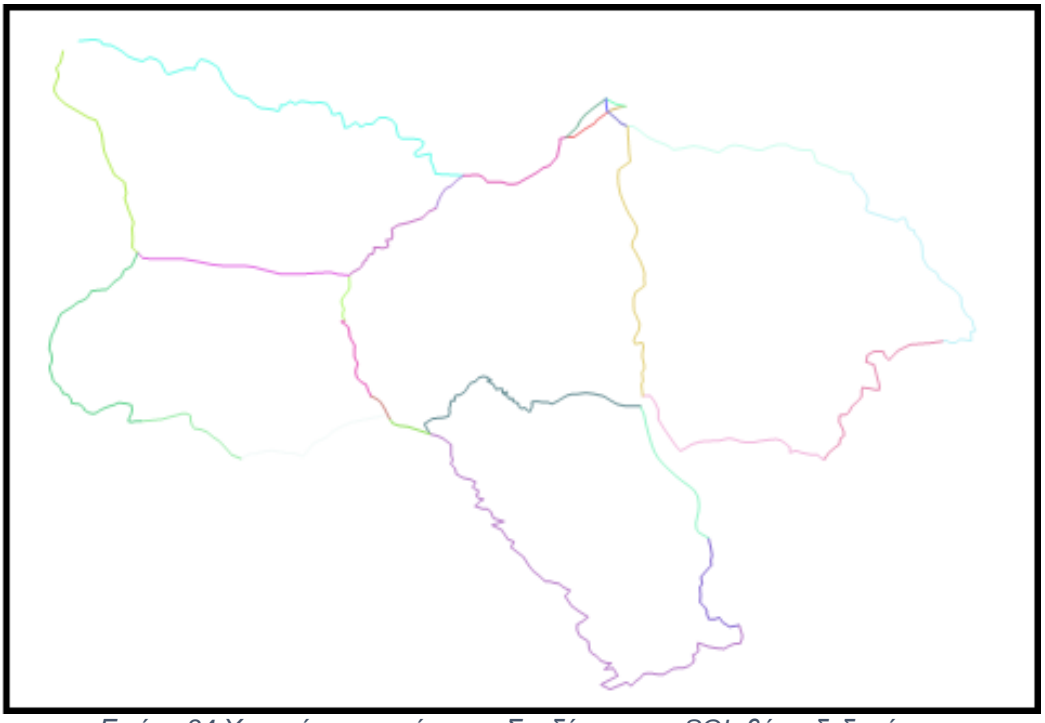
- FID. Μεταβλητή που χρησιμοποιείται για την απαρίθμηση των γραμμών του table.
- LinkID. Το ID του συνδέσμου.
- Visibility. Μεταβλητή σημαντικότητας του συνδέσμου.
- FromNodeID. Το ID του κόμβου που αποτελεί την αφετηρία του συνδέσμου.
- ToNodeID. Το ID του κόμβου που αποτελεί τον προορισμό του συνδέσμου.
- StartCost. Το κόστος εκκίνησης για την μετάβαση από τον κόμβο-αφετηρία προς τον κόμβο-προορισμό.
- TravCost. Το κόστος διάβασης του συνδέσμου.
- Class. Μεταβλητή που καθορίζει την κλάση του συνδέσμου.
- TwoWay. Μεταβλητή που καθορίζει αν ο σύνδεσμος είναι διπλής κατευθύνσεως.
- StartCost2. Το κόστος εκκίνησης για την μετάβαση από τον κόμβο-προορισμό προς τον κόμβο-αφετηρία.
- GEOMETRY. Τα ζεύγη συντεταγμένων που αποτελούν τον σύνδεσμο.

Στην παρακάτω εικόνα βλέπουμε τον πίνακα αποτελεσμάτων για του συνδέσμους που ενώνουν τους κόμβους που είδαμε προηγουμένως. Τα δεδομένα για τις στήλες LinkID, FromNodeID, ToNodeID, GEOMETRY προέρχονται από την ETIS ενώ οι για τις στήλες VISIBILITY, StartCost, TravCost, CLASS, TwoWay και StartCost2 εισάγαμε δικά μας τυχαία δεδομένα. Στην συνέχεια απεικονίζονται και τα χωρικά αποτελέσματα για τους συνδέσμους του δικτύου.

Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	1	1005603	1	104393	104388	13	18	1	1	17	0xE6100000010418000000287E8CB96B714240EF0390DAC48...
2	2	1005572	1	104374	104376	15	10	2	1	13	0xE6100000010404000000050E449D235BF424067B5C01E1361...
3	3	1005599	1	104392	104393	14	20	1	1	17	0xE6100000010415000000088D7F50B7689424032E9EFA5F06C...
4	4	1005571	1	104374	104373	15	15	2	1	19	0xE6100000010407000000050E449D235BF424067B5C01E1361...
5	5	1005594	1	104377	168100	20	13	4	1	15	0xE6100000010425000000DBA6785C548B4240D4D7F335CB6...
6	6	1005595	1	104387	104388	14	11	1	1	12	0xE6100000010464000000040C05AB56B84424072FE26142220...
7	7	1080256	1	104379	168100	14	11	5	1	16	0xE6100000010427000000AE0FEB8D5A954240C824236761D...
8	8	1005570	1	104376	104363	18	13	5	1	15	0xE6100000010425000000032B08EE387BC424013D21A834E6...
9	9	1005605	1	104387	104392	11	18	3	1	17	0xE6100000010043E00000040C05AB56B84424072FE26142220...
10	10	1004087	1	103286	103298	11	14	5	1	17	0xE6100000010041E0000004339D1AE42CA4240800D8810579...
11	11	1005574	1	104379	104363	17	10	2	1	15	0xE6100000010432000000AE0FEB8D5A954240C824236761D...
12	12	1005609	1	104397	104398	18	20	5	1	16	0xE6100000010412000000C91CCBBBEA7F42405E85949F54D...
13	13	1037504	1	125089	104375	10	19	4	1	17	0xE6100000010413000000DF5339ED299942405DFA97A4320...
14	14	1004072	1	103287	103288	13	17	2	1	13	0xE6100000010466000000FD69A33A1DCC42402CD7DB662A...
15	15	1037509	1	103288	125088	10	11	1	1	20	0xE6100000010407000009087BEBB95B342402D0ABB287A2...
16	16	1037510	1	104378	125089	16	18	2	1	13	0xE6100000010040E000000B804E09F52A14240DC2DC901BB0...
17	17	1005618	1	104400	104397	19	16	2	1	12	0xE6100000010412000000F1B913ECBF864240098A1F63EEB6...
18	18	1005575	1	104373	103288	11	12	4	1	12	0xE6100000010417000000014E8137992BA424082531F48DE51...
19	19	1005576	1	104362	104374	17	14	5	1	11	0xE6100000010404000000231631EC30C0424020425C397B67...
20	20	1005611	1	103298	104378	18	15	1	1	13	0xE6100000010416000000AF7D01BD70A54240293DD34B8CB...
21	21	1005612	1	104398	104387	17	10	5	1	14	0xE6100000010406000000A9FB00A436874240CAFACDC4741...
22	22	1005613	1	104375	104398	10	14	4	1	10	0xE6100000010406000000452921585588424020ED7F80B50A...
23	23	1005566	1	104362	104372	16	14	1	1	12	0xE6100000010403000000231631EC30C0424020425C397B67...
24	24	1005569	1	104376	104377	17	14	4	1	17	0xE6100000010430000000032B08EE387BC424013D21A834E6...
25	25	1005615	1	104400	103298	12	12	2	1	16	0xE6100000010043F000000F1B913ECBF864240098A1F63EEB6...
26	26	1037506	1	125088	104378	10	15	3	1	13	0xE6100000010041E0000000B410E4A98AD4240DBDD03745F2...
27	27	1005564	1	104372	104373	17	13	1	1	15	0xE6100000010406000000B30C71AC8BC14240FBAF73D3666...
28	28	1005565	1	104372	104374	14	11	2	1	13	0xE6100000010404000000B30C71AC8BC14240FBAF73D3666...

Εικόνα 23 Δεδομένα Συνδέσμων σε SQL βάση δεδομένων



Εικόνα 24 Χωρική αναπαράσταση Συνδέσμων σε SQL βάση δεδομένων

Όπως παρατηρούμε οι στήλες των tables στην βάση δεδομένων έχουν άμεση συσχέτιση με τα χαρακτηριστικά του μοντέλου που περιγράφηκαν λεπτομερώς στο προηγούμενο κεφάλαιο.

7.3 Python

Η **Python** είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού η οποία δημιουργήθηκε από τον Ολλανδό Γκβίντο βαν Ρόσσουμ (Guido van Rossum) το 1990. Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά της και η ευκολία χρήσης της και το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα απ'ότι θα ήταν δυνατόν σε γλώσσες όπως η C++ ή η Java. Διακρίνεται λόγω του ότι έχει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές συνηθισμένες εργασίες και για την ταχύτητα εκμάθησης της.

Οι διερμηνευτές της Python είναι διαθέσιμοι για εγκατάσταση σε πολλά λειτουργικά συστήματα, επιτρέποντας στην Python την εκτέλεση κώδικα σε ευρεία γκάμα συστημάτων. Χρησιμοποιώντας εργαλεία τρίτων, όπως το Py2exe ή το Pyinstaller, ο κώδικας της Python μπορεί να πακεταριστεί σε αυτόνομα εκτελέσιμα προγράμματα για μερικά από τα πιο δημοφιλή λειτουργικά συστήματα, επιτρέποντας τη διανομή του βασισμένου σε Python

λογισμικού για χρήση σε αυτά τα περιβάλλοντα χωρίς να απαιτείται εγκατάσταση του διερμηνευτή της Python.

Η Python αναπτύσσεται ως ανοιχτό λογισμικό (open source) και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation. Η γλώσσα χρησιμοποιεί μεταγλωττιστή (compiler) για την δημιουργία του εκτελέσιμου κώδικα και σχετίζεται με τις γλώσσες προγραμματισμού Tcl, Perl, Scheme, Java και Ruby, καθώς και με την ABC η οποία υπήρξε η αρχική πηγή έμπνευσης για τη δημιουργία της. Ένα ιδιαίτερο χαρακτηριστικό της γλώσσας είναι η χρήση κενών διαστημάτων (whitespace) για τον διαχωρισμό των συντακτικών δομών που προγράμματος, σε αντίθεση με την πρακτική σε άλλες γλώσσες όπου για τον ίδιο σκοπό χρησιμοποιούνται ειδικά σύμβολα (πχ αγκύλες). Αυτό, σε συνδυασμό με το ότι χρησιμοποιεί πλήρεις αγγλικές λέξεις στη θέση συμβόλων, καθιστούν τον κώδικα της Python ευανάγνωστο από όσους έχουν βασική γνώση των αγγλικών.

Στο πλαίσιο της διπλωματικής έγινε χρήση της Python 2.7 για την κατασκευή των απαραίτητων εργαλείων για εφαρμογές πάνω στα μεταφορικά δίκτυα. Στην συνέχεια παρουσιάζονται τα συγκεκριμένα εργαλεία ενώ ολόκληρος ο κώδικας βρίσκεται στο τέλος της εργασίας σε ειδικό παράρτημα [18].

7.4 Εργαλεία λογισμικού για εφαρμογές στα δίκτυα μεταφορών

1. **Adjacency_Dictionary_From_Database**. Παίρνει τα δεδομένα από την βάση δεδομένων και χτίζει το Adjacency Dictionary.
2. **Adjacency_Dictionary_To_Database**. Παίρνει τα δεδομένα από το Adjacency_Dictionary (μετά από πιθανές αλλαγές) και δημιουργεί μία καινούρια βάση δεδομένων με το παράγωγο δίκτυο.
3. **Adjacency_Dictionary_To_Shapefile**. Παίρνει τα δεδομένα από το Adjacency_Dictionary (μετά από πιθανές αλλαγές) και δημιουργεί ένα Shapefile με το παράγωγο δίκτυο για απεικόνιση σε GIS.
4. **Calculate_Power_Of_Nodes_In_Database**. Υπολογίζει πόσοι σύνδεσμοι εισέρχονται και εξέρχονται από έναν κόμβο και ενημερώνει την βάση δεδομένων.

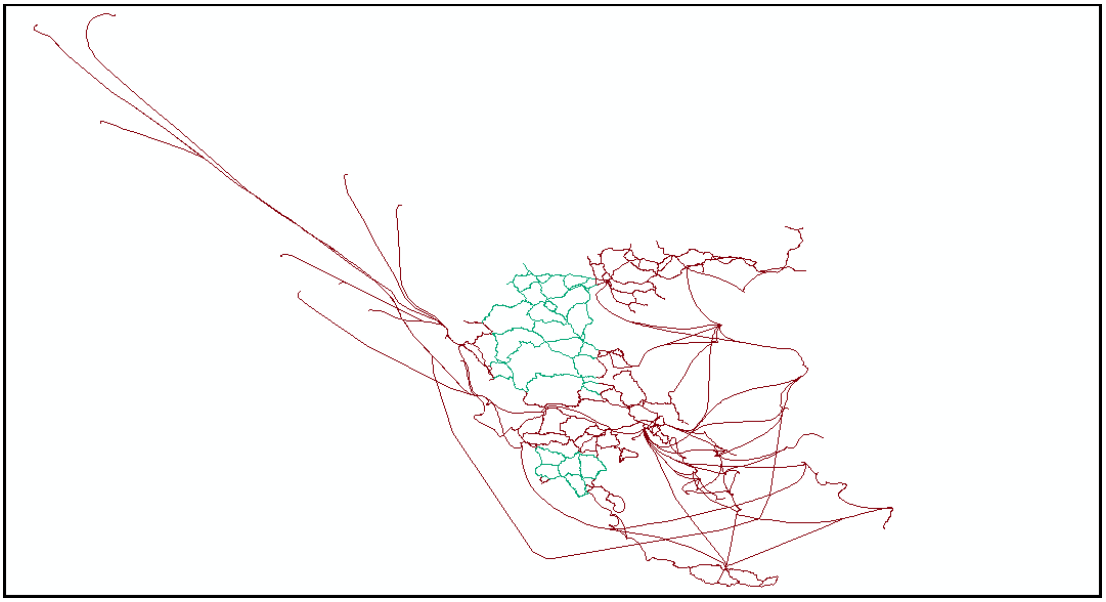
5. **Change_Visibility_Of_Link_In_Database_By_Class**. Αλλάζει την μεταβλητή σημαντικότητας ενός συνδέσμου, βάση της κλάσης του, και ενημερώνει την βάση δεδομένων.
6. **Change_Visibility_Of_Link_In_Database_By_ID**. Αλλάζει την μεταβλητή σημαντικότητας ενός συνδέσμου, βάση του ID του, και ενημερώνει την βάση δεδομένων.
7. **Change_Visibility_Of_Link_In_Dictionary_By_Class**. Αλλάζει την μεταβλητή σημαντικότητας ενός συνδέσμου, βάση της κλάσης του, μέσα στο Adjacency Dictionary αφήνοντα ανέπαφη την βάση δεδομένων.
8. **Change_Visibility_Of_Link_In_Dictionary_By_ID**. Αλλάζει την μεταβλητή σημαντικότητας ενός συνδέσμου, βάση του ID του, μέσα στο Adjacency Dictionary αφήνοντα ανέπαφη την βάση δεδομένων.
9. **Change_Visibility_Of_Node_In_Database_By_Class**. Αλλάζει την μεταβλητή σημαντικότητας ενός κόμβου, βάση της κλάσης του, και ενημερώνει την βάση δεδομένων.
10. **Change_Visibility_Of_Node_In_Database_By_Distance**. Αλλάζει την μεταβλητή σημαντικότητας ενός κόμβου, βάση της απόστασης του από έναν κόμβο-κέντρο που ορίζει ο χρήστης, και ενημερώνει την βάση δεδομένων.
11. **Change_Visibility_Of_Node_In_Database_By_ID**. Αλλάζει την μεταβλητή σημαντικότητας ενός κόμβου, βάση του ID του, και ενημερώνει την βάση δεδομένων.
12. **Change_Visibility_Of_Node_In_Dictionary_By_Class**. Αλλάζει την μεταβλητή σημαντικότητας ενός κόμβου, βάση της κλάσης του, μέσα στο Adjacency Dictionary αφήνοντα ανέπαφη την βάση δεδομένων.
13. **Change_Visibility_Of_Node_In_Dictionary_By_ID**. Αλλάζει την μεταβλητή σημαντικότητας ενός κόμβου, βάση του ID του, μέσα στο Adjacency Dictionary αφήνοντα ανέπαφη την βάση δεδομένων.
14. **Check_For_Connectivity_Of_Route_From_Dictionary**. Ελέγχει εάν υπάρχει γεωγραφικό κενό (ασυνέχεια) στην διαδρομή που έχει ορίσει ο χρήστης
15. **Copy_Sql_Table**. Δημιουργεί ένα αντίγραφο της βάσης δεδομένων.
16. **Create_Route_In_Database**. Δημιουργεί μία διαδρομή βάση των προτιμήσεων του χρήστη ενώνει οποιαδήποτε ασυνέχεια βρεθεί και

κατασκευάζει ένα καινούριο table στην βάση δεδομένων που περιέχει την διαδρομή.

17. **Delete_Links_In_Database_By_Class**. Διαγράφει συνδέσμους, βάση της κλάσης τους και ενημερώνει την βάση δεδομένων.
18. **Delete_Links_In_Database_By_ID**. Διαγράφει συνδέσμους, βάση των ID τους και ενημερώνει την βάση δεδομένων.
19. **Delete_links_In_Dictionary_By_Class**. Διαγράφει συνδέσμους, βάση της κλάσης τους, μέσα στο Adjacency Dictionary αφήνοντας ανέπαφη την βάση δεδομένων.
20. **Delete_links_In_Dictionary_By_ID**. Διαγράφει συνδέσμους, βάση των ID τους, μέσα στο Adjacency Dictionary αφήνοντα ανέπαφη την βάση δεδομένων.
21. **Finds_Nodes_To_Remove_From_Database**. Βρίσκει του κόμβους οι οποίοι είναι περιττοί σε ένα δίκτυο (και αποτελούν κόστος πολυπλοκότητας και υπολογισμού για οποιαδήποτε διαδικασία) και πρέπει να αφαιρεθούν.
22. **Geometry_Extraction_To_Database**. Εξάγει τα δεδομένα από ένα Shapefile με συνδέσμους, κατασκευάζει τα tables στην βάση δεδομένων και εισάγει τα δεδομένα στις κατάλληλες στήλες.
23. **Insert_Node_To_Database**. Εισάγει ένα καινούριο κόμβο στην βάση δεδομένων και χωρίζει στα δύο τον σύνδεσμο, μέσα στον οποίο εισήχθη ο νέος κόμβος.
24. **Insert_Node_To_Dictionary**. Εισάγει ένα καινούριο κόμβο στο Adjacency Dictionary και χωρίζει στα δύο τον σύνδεσμο, μέσα στον οποίο εισήχθη ο νέος κόμβος αφήνοντας ανέπαφη την βάση δεδομένων.
25. **Removing_Nodes_From_Dictionary**. Αφαιρεί από το Adjacency Dictionary τους περιττούς κόμβους συνδέοντας καταλλήλως τους εναπομείναντες κόμβους ώστε να μην αλλάξει μορφή το αρχικό δίκτυο.

8ο Κεφάλαιο: Εφαρμογή σε πραγματικές περιπτώσεις σχεδιασμού μεταφορών.

Στο κεφάλαιο αυτό θα εξετάσουμε την εφαρμογή ορισμένων εργαλείων που αναφέρθηκαν στο προηγούμενο κεφάλαιο και θα πραγματοποιήσουμε προσομοιώσεις πάνω σε δίκτυα που θα περιλαμβάνουν πραγματικά, αλλά και 'dummy' δεδομένα. Πρώτα θα εξάγουμε ένα παράγωγο δίκτυο από ένα ήδη υπάρχον ευρύτερο δίκτυο. Έπειτα θα προσπαθήσουμε να αφαιρέσουμε τους αχρείαστους κόμβους από ένα δίκτυο και τέλος, θα εισάγουμε ένα καινούριο κόμβο μέσα σε ένα συγκεκριμένο σύνδεσμο του δικτύου. Τα δίκτυα πάνω στα οποία θα εργαστούμε είναι ουσιαστικά υπό-Shapefiles από ένα ευρύτερο Shapefile. Το βασικό μας Shapefile προέρχεται από την ETIS και περιέχει δεδομένα για το σιδηροδρομικό δίκτυο της Ελλάδος. Οι σύνδεσμοι που έχουν διαφορετικό χρώμα είναι τα δίκτυα που θα χρησιμοποιήσουμε για τις προσομοιώσεις. Ακολουθεί σχετική εικόνα:



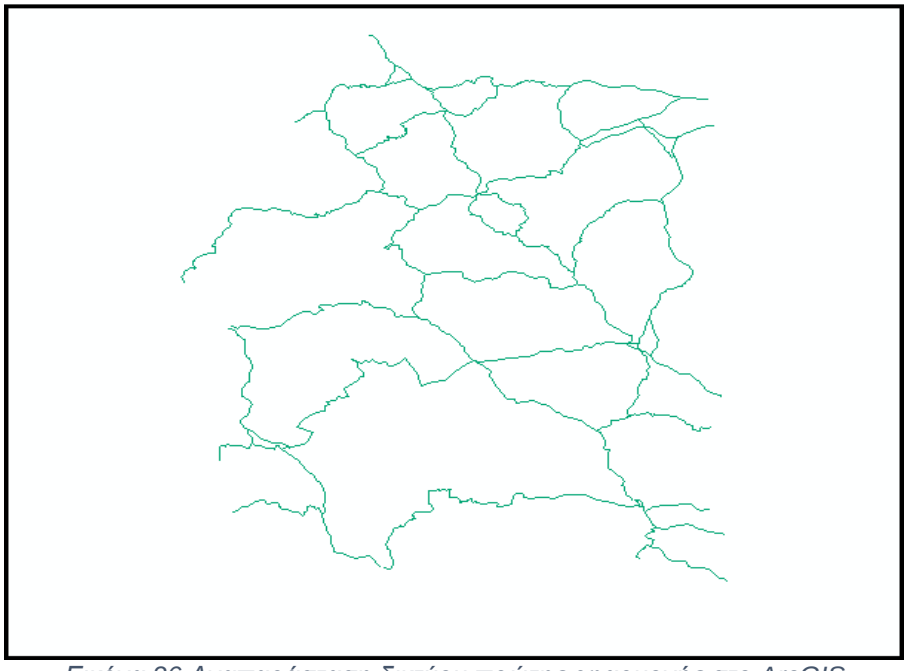
Εικόνα 25 Αναπράσταση σιδηροδρομικού δικτύου Ελλάδος στο ArcGIS

8.1 Δημιουργία παράγωγου δικτύου από ήδη υπάρχον δίκτυο

Ας θεωρήσουμε ότι θέλουμε να μεταφέρουμε παλέτες οι οποίες περιέχουν πακέτα με όλα τα προϊόντα της ESRI. Κάθε τέτοιο πακέτο μπορεί να κοστολογηθεί στις δεκάδες χιλιάδες ευρώ. Καταλαβαίνουμε λοιπόν πως μία

ολόκληρη παλέτα από παρόμοια πακέτα έχει κόστος όσο όλο το βάρος της σε χρυσό. Για την μετακίνηση ενός τόσο πολύτιμου φορτίου έχουμε αποφασίσει να μην κάνουμε χρήση των ακτοπλοϊκών γραμμών, καθώς θέλουμε το φορτίο να φτάσει στον προορισμό του το ταχύτερο δυνατό με την μέγιστη ασφάλεια. Είναι σκόπιμο λοιπόν να αφαιρέσουμε από το δίκτυο μας τους θαλάσσιους συνδέσμους και να δημιουργήσουμε ένα πιο απλό δίκτυο το οποίο πληρεί καλύτερα τις ανάγκες μας και είναι πιο αποδοτικό καθώς απαιτεί λιγότερη υπολογιστική ισχύ για οποιαδήποτε εφαρμογή. Το δίκτυο που θα χρησιμοποιήσουμε, για την προσομοίωση, είναι κομμάτι του σιδηροδρομικού δικτύου της κεντρικής και βορειοδυτικής Ελλάδας αλλά στην δικιά μας περίπτωση το δίκτυο δεν είναι αποκλειστικά σιδηροδρομικό αλλά κάθε σύνδεσμος έχει την δική του κλάση που καθορίζει το είδος του. Π.χ. κλάση ‘1’ έχουν οι σύνδεσμοι που αντιπροσωπεύουν τους αστικούς δρόμους ,κλάση ‘2’ οι εθνικές οδοί, κλάση ‘3’ οι σιδηρόδρομοι, κλάση ‘4’ οι ακτοπλοϊκές γραμμές και κλάση ‘5’ οι αερογραμμές. Ουσιαστικά εμείς πρέπει να δημιουργήσουμε ένα καινούριο δίκτυο αφού έχουμε αφαιρέσει τους συνδέσμους με κλάση ‘3’.

Αφού κάνουμε export το αρχικό δίκτυο μέσω του ArcMap σε ένα ξεχωριστό shapefile η εικόνα του δικτύου μας είναι η εξής:

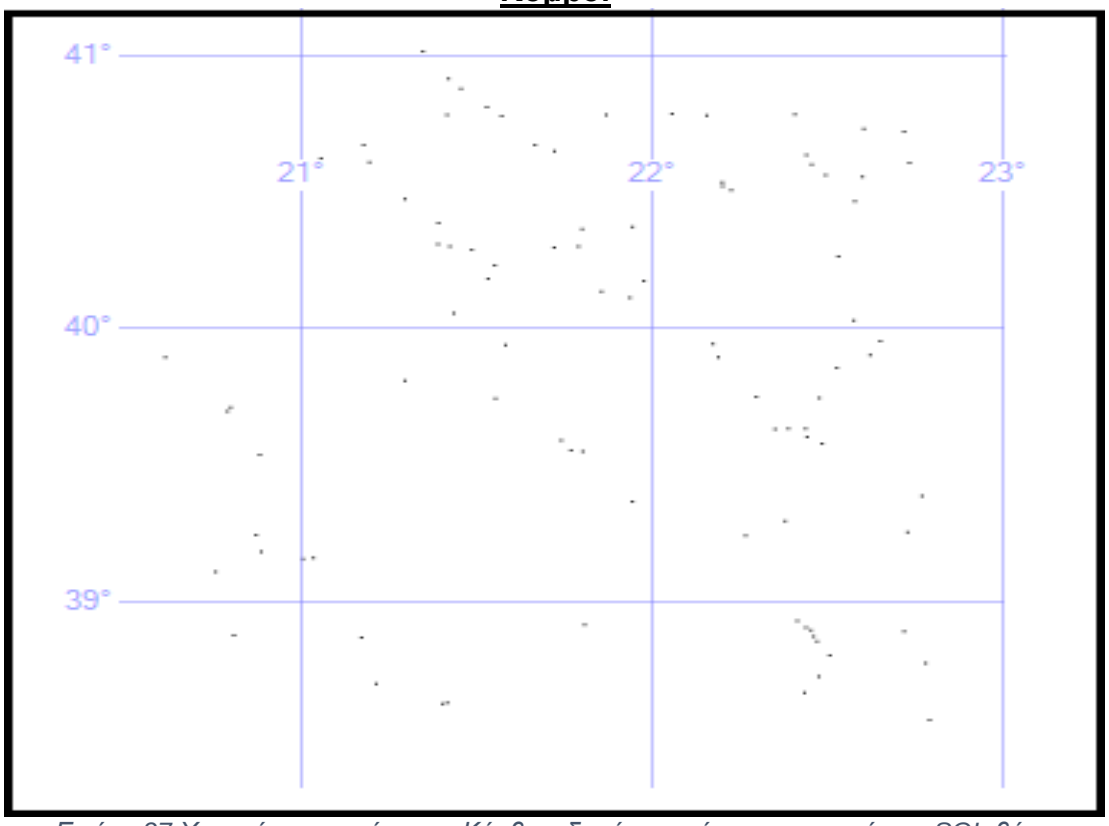


Εικόνα 26 Αναπαράσταση δικτύου πρώτης εφαρμογής στο ArcGIS

Πρώτη μας κίνηση είναι η μετάβαση από Shapefile στην δικιά μας βάση δεδομένων. Κάνοντας χρήση του Geometry_Extraction_To_Database.py

μεταφέρουμε το δίκτυο στην βάση δεδομένων όπου δημιουργούνται δύο tables, ένα για κόμβους και ένα για συνδέσμους.

Κόμβοι



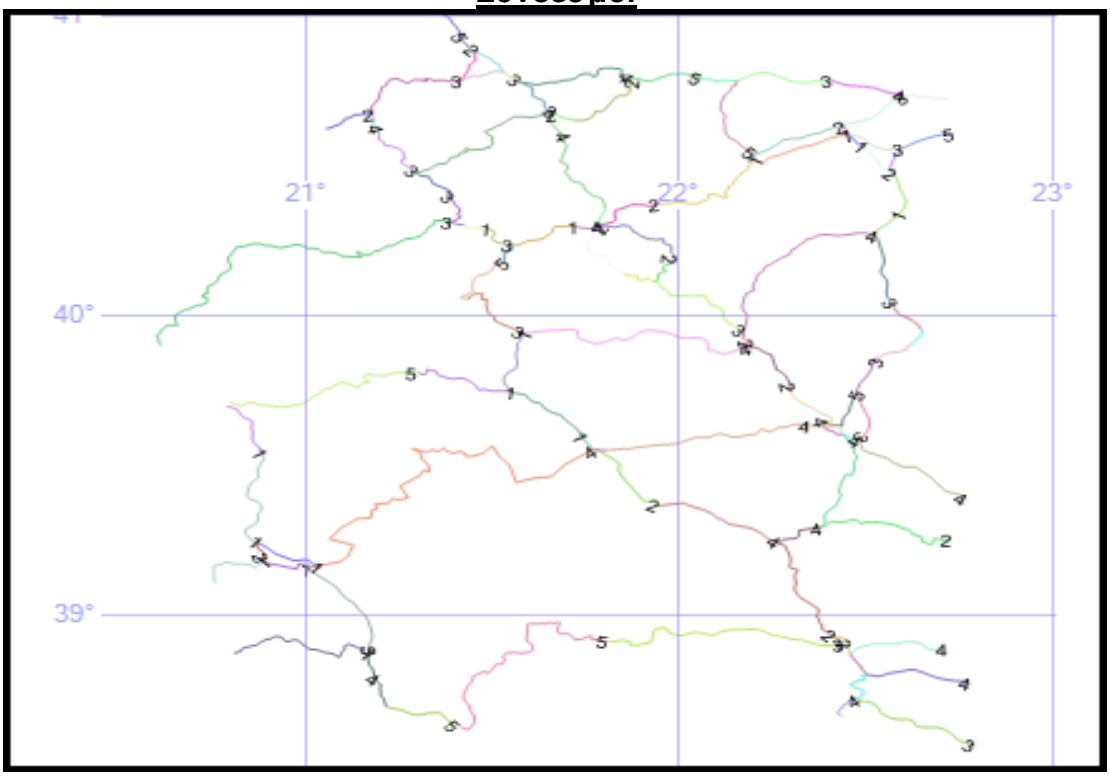
Εικόνα 27 Χωρική αναπαράσταση Κόμβων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
4	4	103454	1	4	NULL	NULL	0xE6100000010C226E4E2503824440855CA96741583540
5	5	124933	1	1	NULL	NULL	0xE6100000010C077DE9EDCF7143407C2AA73D25B73640
6	6	103234	1	5	NULL	NULL	0xE6100000010C764F1E166A6F4340C4EDD0B018753640
7	7	124929	1	3	NULL	NULL	0xE6100000010C6A8995D1C80344400F99F221A8923640
8	8	124930	1	3	NULL	NULL	0xE6100000010C0A12DBDD03FA4340DB368C82E0A53640
9	9	124931	1	3	NULL	NULL	0xE6100000010CECA2E8818FF3434044C362D4B59E3640
10	10	124932	1	5	NULL	NULL	0xE6100000010C5AF5B9DA8AED4340D8F0F44A59863640
11	11	168092	1	3	NULL	NULL	0xE6100000010C69C4CC3E8FDF434089B48D3F51793640
12	12	103256	1	2	NULL	NULL	0xE6100000010CF3AE7AC03C50434031CF4A5AF1693540
13	13	186325	1	1	NULL	NULL	0xE6100000010C739EB12FD9744340799109F835CE3540
14	14	103141	1	3	NULL	NULL	0xE6100000010C55A52DAEF147444002B9C491077E3640
15	15	103137	1	4	NULL	NULL	0xE6100000010C5D8AABCABE3B444065C746205E933640
16	16	103180	1	4	NULL	NULL	0xE6100000010C0FD6FF39CC214440768A558330873640
17	17	103257	1	4	NULL	NULL	0xE6100000010CD252793BC27343409AEAC9FCA36F3640
18	18	103140	1	4	NULL	NULL	0xE6100000010CAD4ECE50DC5D444066BFEE74E7993640
19	19	103138	1	1	NULL	NULL	0xE6100000010C74620FED63514440BCE9961DE26F3640
20	20	103208	1	2	NULL	NULL	0xE6100000010CB073D3669C264440B22FD978B0C93540
21	21	103214	1	3	NULL	NULL	0xE6100000010CF5D896016711444090F9804067DA3540
22	22	103215	1	4	NULL	NULL	0xE6100000010CEAD2DB9E8B704440CAC4AD8218743540

Εικόνα 28 Δεδομένα Κόμβων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων

Σύνδεσμοι



Εικόνα 29 Χωρική αναπαράσταση Συνδέσμων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων (Στους Συνδέσμους αναγράφεται και η Κλάση του καθενός)

Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	1	1004139	1	103331	124934	20	17	4	1	12	0xE610000001041500000514EB4AB9066434025EA059FE680...
2	2	1004285	1	103219	103454	12	15	5	1	15	0xE6100000010408000000F243A51133754440022B8716D96A...
3	3	1037249	1	124933	103234	18	19	4	1	10	0xE610000001041700000077DE9EDCF7143407C2AA73D25B...
4	4	1037245	1	124929	124930	14	16	2	1	13	0xE6100000010408C0000006A8995D1C80344400F99F221A892...
5	5	1037247	1	124931	124932	16	10	3	1	19	0xE6100000010408000000ECA2E8818FF3434044C362D4B59...
6	6	1037248	1	124932	168092	11	19	3	1	17	0xE6100000010408A0000005AF5B9DA8AED4340D8F0F44A598...
7	7	1004039	1	103256	186325	14	19	5	1	20	0xE610000001047400000F3AE7AC03C50434031CF4A5AF169...
8	8	1003871	1	103141	103137	16	13	2	1	17	0xE6100000010408E00000055A52DAEF147444002B9C491077...
9	9	1080150	1	103180	124929	10	17	3	1	18	0xE610000001041600000000FD6FF39CC214440768A55833087...
10	10	1251699	1	186325	103257	11	20	3	1	10	0xE610000001044900000739EB12FD9744340799109F835CE...
11	11	1003869	1	103140	103138	17	17	3	1	12	0xE6100000010408000000AD4ECE50DC5D444066BFEE74E7...
12	12	1003982	1	103208	103214	11	18	2	1	13	0xE610000001041700000B073D3669C264440B22FD978B0C...
13	13	1003983	1	103215	103216	13	20	2	1	13	0xE610000001040700000FAD2DB9F8B704440CAC4AD82187...
14	14	1037246	1	124930	124931	18	13	4	1	19	0xE610000001040600000A12DBDD03FA4340DB368C82E0...
15	15	1003965	1	103208	103210	12	20	4	1	10	0xE610000001040500000B073D3669C264440B22FD978B0C...
16	16	1003966	1	103211	103182	13	15	3	1	18	0xE610000001042200000E6AE25E4830E4440338AE59656EF...
17	17	1003967	1	103212	103208	12	18	1	1	12	0xE610000001040800000B9347EE1952E44402AC6F99B50C...
18	18	1003968	1	103133	103208	16	17	2	1	11	0xE6100000010408E0000004EEFE2FDB82F444016F71F990EF1...
19	19	1003969	1	103210	103213	17	13	1	1	20	0xE610000001041400000000DE2033BFE254440107860000E1B7...

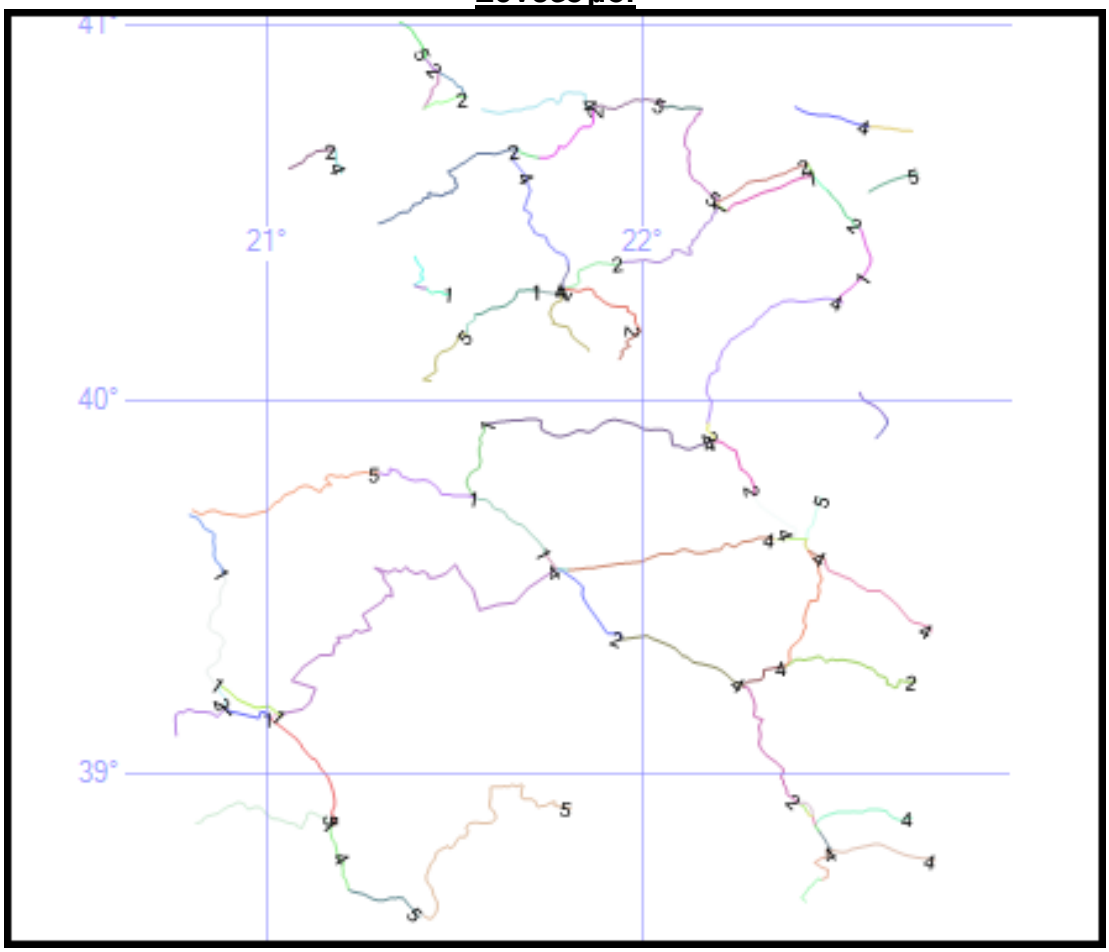
Εικόνα 30 Δεδομένα Συνδέσμων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων

Έπειτα τρέχουμε το Adjacency_Dictionary_From_Database.py και μεταβαίνουμε από βάση δεδομένων στην Adjacent δομή που παρουσιάσαμε στο κεφάλαιο 6. Συνεχίζοντας τρέχουμε το

Delete_links_In_Dictionary_By_Class.py και αφαιρούμε από την δομή μας οποιοδήποτε σύνδεσμο έχει κλάση ‘3’, δηλαδή εκείνους που έχουμε ορίσει εμείς ότι θα είναι οι ακτοπλοϊκές γραμμές. Δημιουργήσαμε λοιπόν μέσα στην δομή μας ένα νέο δίκτυο βάση των προτιμήσεων μας και το οποίο πληρεί τις ανάγκες μας καλύτερα απ’ότι το αρχικό δίκτυο. Τέλος, μπορούμε μέσω του Adjacency_Dictionary_To_Database.py να δημιουργήσουμε δύο καινούρια tables στην βάση δεδομένων για το παράγωγο δίκτυο ή να μεταβούμε στο ArcGIS και να απεικονίσουμε εκεί το καινούριο δίκτυο αφού πρώτα το αποθηκεύσουμε σε δύο Shapefiles (κόμβοι και σύνδεσμοι) τρέχοντας το Adjacency_Dictionary_To_Shapefile.py.

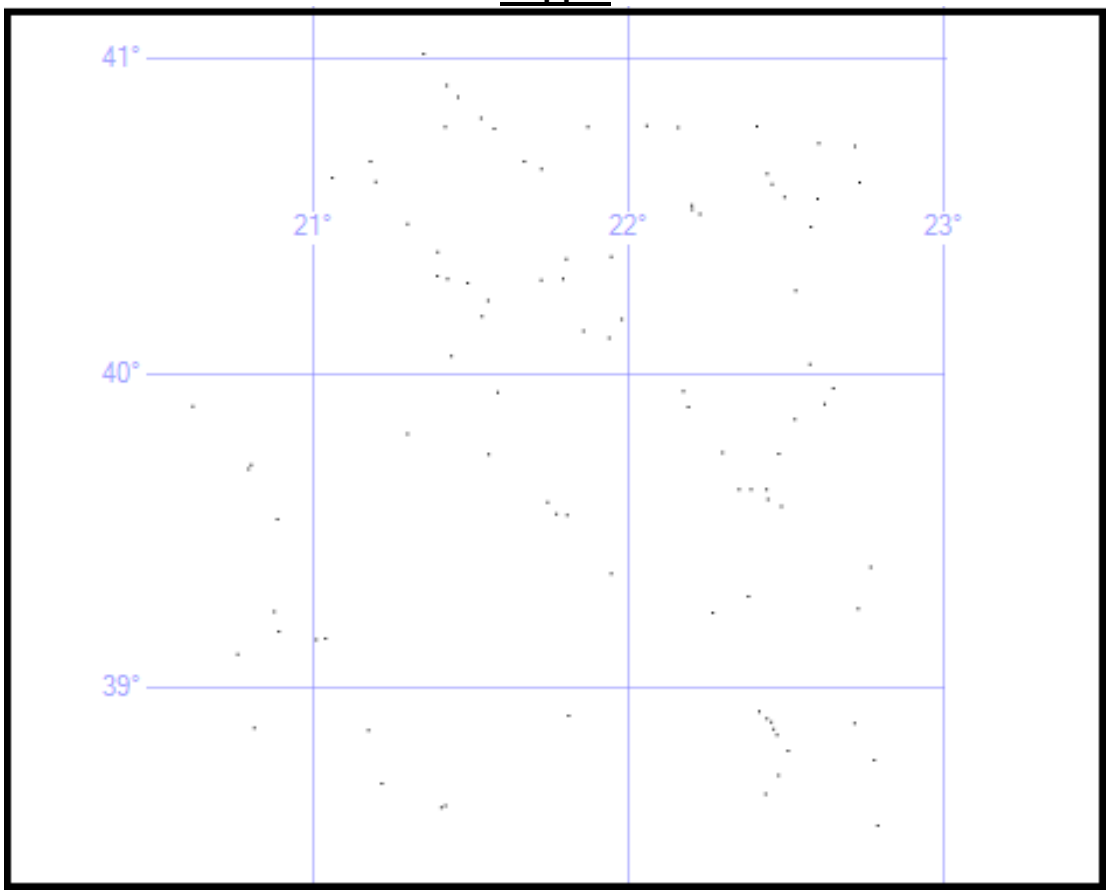
Παρακάτω απεικονίζεται το δίκτυο μας στην βάση δεδομένων οπού φαίνεται ότι πλέον δεν υπάρχουν σύνδεσμοι με κλάση ‘3’.

Σύνδεσμοι



Εικόνα 31 Χωρική αναπαράσταση Συνδέσμων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων μετά την αφαίρεση των κατάλληλων συνδέσμων

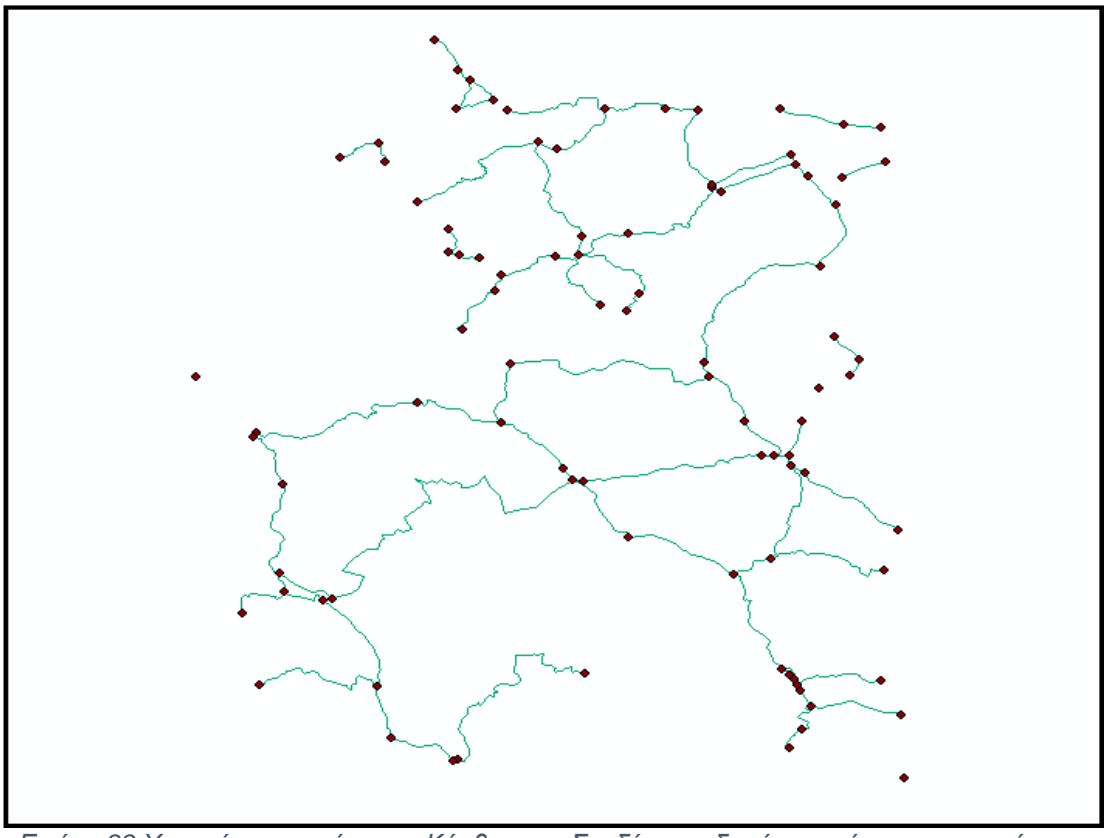
Κόμβοι



Εικόνα 32 Χωρική αναπαράσταση Κόμβων δικτύου πρώτης εφαρμογής σε SQL βάση δεδομένων μετά την αφαίρεση των κατάλληλων συνδέσμων

Αν παρατηρήσουμε τους κόμβους του αρχικού δικτύου και εκείνους του παράγωγου θα δούμε ότι είναι ακριβώς ίδιοι, γεγονός απολύτως λογικό, αφού σκοπός μας ήταν να αφαιρεθούν μόνο σύνδεσμοι και καθόλου κόμβοι.

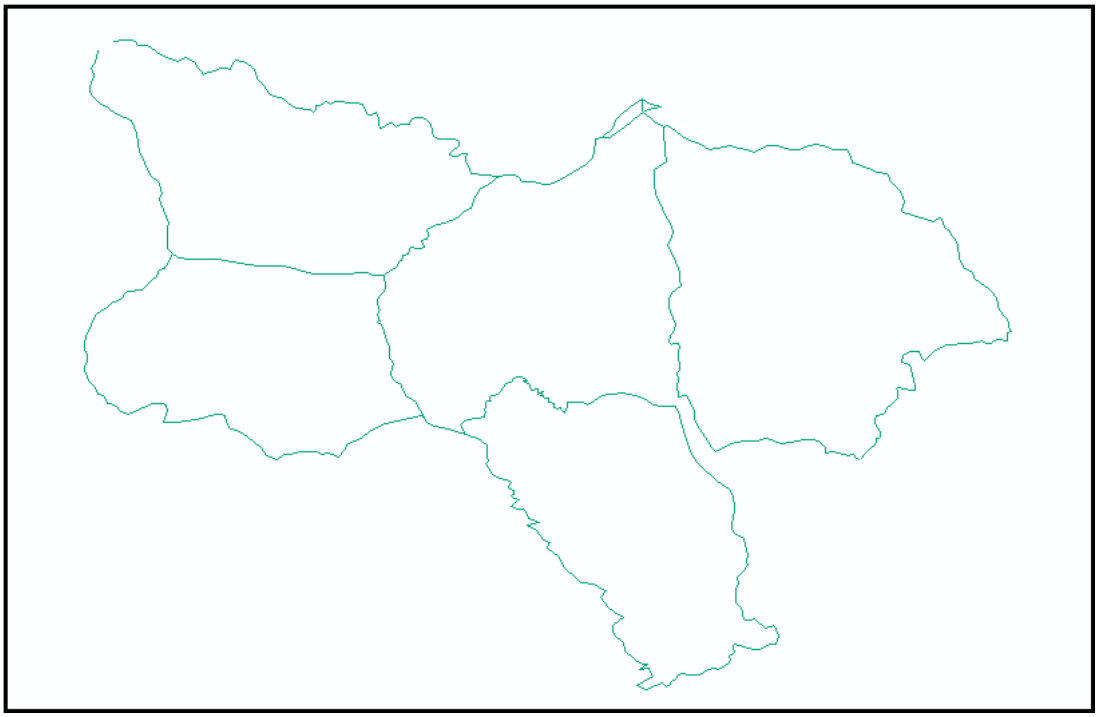
Αντίστοιχα μπορούμε να απεικονίσουμε το δίκτυο μέσω του ArcMap στο ArcGis όπου φαίνονται τα Shapefile των κόμβων και των συνδέσμων τα οποία είναι σε άμεση αντιστοιχία με τα χωρικά αποτελέσματα της βάσης δεδομένων:



Εικόνα 33 Χωρική αναπαράσταση Κόμβων και Συνδέσμων δικτύου πρώτης εφαρμογής στο ArcGIS μετά την αφαίρεση των κατάλληλων συνδέσμων

8.2 Αφαίρεση αχρείαστων κόμβων από το δίκτυο

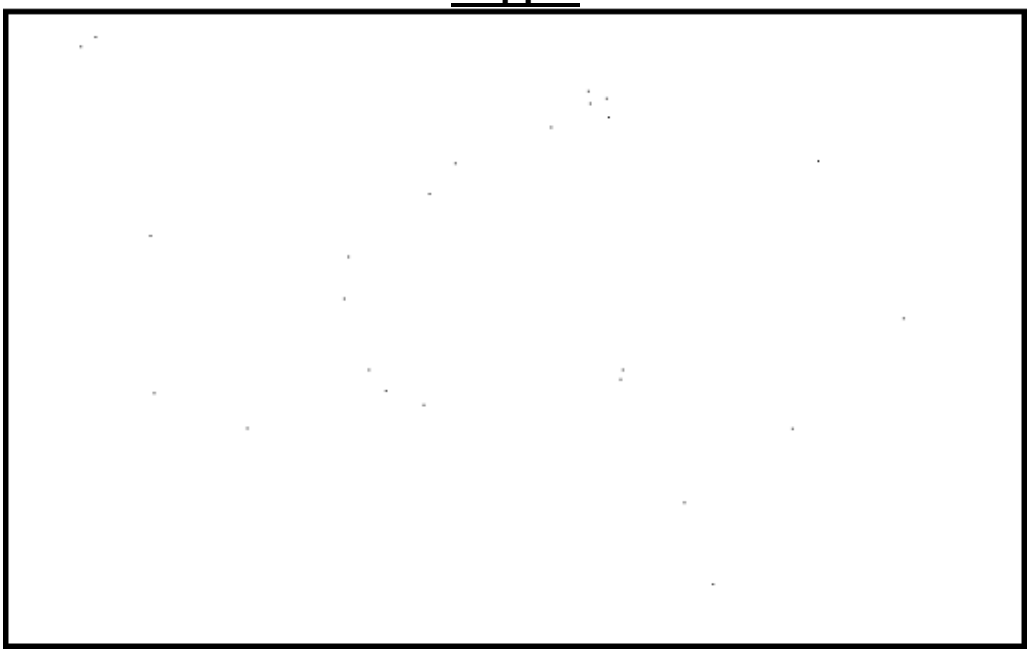
Βασική μας μέριμνα στις εφαρμογές πάνω στα δίκτυα μεταφορών είναι η ελαχιστοποίηση της πολυπλοκότητας και η μεγιστοποίηση της αποδοτικότητας του συστήματος μας. Αυτό περιλαμβάνει δύο συνιστώσες. Από την μία πρέπει οι αλγόριθμοι που θα κατασκευάσουμε να είναι βέλτιστοι, από άποψη πολυπλοκότητας, και από την άλλη το δίκτυο που θα παράξουμε και πάνω στο οποίο θα εργαστούμε να περιέχει μόνο την αναγκαία πληροφορία, χωρίς περιττά δεδομένα. Ένας τρόπος να πετύχουμε το τελευταίο είναι να αναγνωρίσουμε ποιοι κόμβοι είναι περιττοί στο δίκτυο μας και να αφαιρεθούν καταλλήλως χωρίς καταστροφικές συνέπειες. Το δίκτυο πάνω στο οποίο θα εργαστούμε απεικονίζεται παρακάτω:



Εικόνα 34 Αναπαράσταση δικτύου δεύτερης εφαρμογής στο ArcGIS

Όπως και στην προηγούμενη προσομοίωση, πρώτο μας βήμα είναι η μετάβαση από Shapefile στην βάση δεδομένων τρέχοντας το Geometry_Extraction_To_Database.py. Έτσι, μέσα στην βάση δεδομένων δημιουργούνται τα tables για τους κόμβους και για τους συνδέσμους.

Κόμβοι



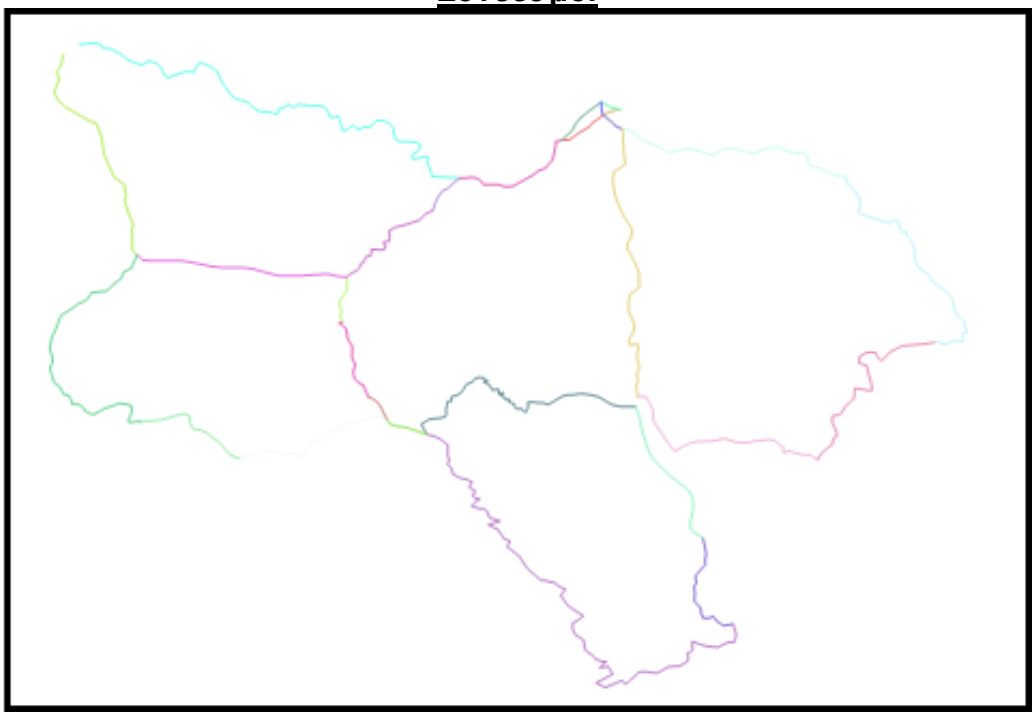
Εικόνα 35 Χωρική αναπαράσταση Κόμβων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων

Results | Spatial results | Messages

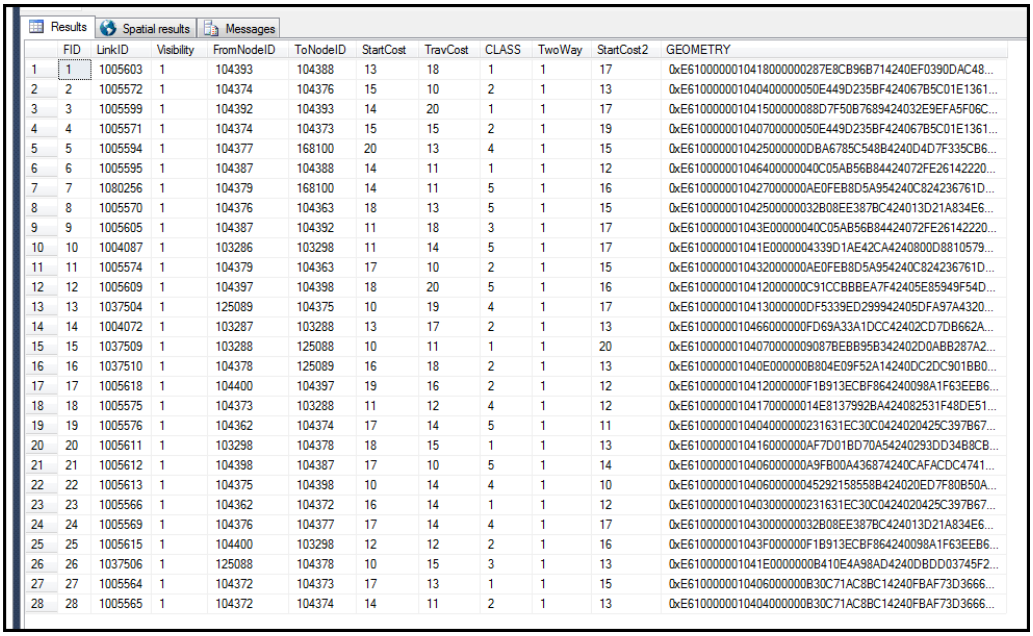
	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	1	104393	1	3	NULL	NULL	0xE6100000010C287E8CB96B714240EF0390DAC4853640
2	2	104388	1	5	NULL	NULL	0xE6100000010C9BC761307F6142406FA0C03BF9903640
3	3	104374	1	5	NULL	NULL	0xE6100000010C50E449D235BF424067B5C01E13613640
4	4	104376	1	2	NULL	NULL	0xE6100000010C32B08EE387BC424013D21A834E683640
5	5	104392	1	1	NULL	NULL	0xE6100000010C88D7F50B7689424032E9EFA5F06C3640
6	6	104373	1	5	NULL	NULL	0xE6100000010C14E8137992BA424082531F48DE513640
7	7	104377	1	5	NULL	NULL	0xE6100000010CDBA6785C548B4240D4D7F335CB6D3640
8	8	168100	1	2	NULL	NULL	0xE6100000010C6364C91CCB7F4240986A662D05B03640
9	9	104387	1	4	NULL	NULL	0xE6100000010C40C05AB56B84424072FE261422203640
10	10	104379	1	2	NULL	NULL	0xE6100000010CAE0FEB8D5A954240C824236761DB3640
11	11	104363	1	3	NULL	NULL	0xE6100000010C00E31934F4B34240425DA45016BA3640
12	12	103286	1	3	NULL	NULL	0xE6100000010C4339D1AE42CA4240800D8810579A3540
13	13	103298	1	2	NULL	NULL	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
14	14	104397	1	1	NULL	NULL	0xE6100000010CC91CCBBBEA7F42405E85949F54DB3540
15	15	104398	1	2	NULL	NULL	0xE6100000010CA9FB00A436874240CAFACDC474113640
16	16	125089	1	1	NULL	NULL	0xE6100000010CDF5339ED299942405DFA97A432013640
17	17	104375	1	4	NULL	NULL	0xE6100000010C45292158558B424020ED7F80B50A3640
18	18	103287	1	3	NULL	NULL	0xE6100000010CFD69A33A1DCC42402CD7DB662AA03540
19	19	103288	1	5	NULL	NULL	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640
20	20	125088	1	1	NULL	NULL	0xE6100000010C0B410E4A98AD4240DBDD03745F223640
21	21	104378	1	4	NULL	NULL	0xE6100000010CB804E09F52A14240DC2DC901BB023640
22	22	104400	1	3	NULL	NULL	0xE6100000010CF1B913ECBF864240098A1F63EEB63540
23	23	104362	1	2	NULL	NULL	0xE6100000010C231631EC30C0424020425C397B673640
24	24	104372	1	2	NULL	NULL	0xE6100000010CB30C71AC8BC14240FBAF73D366603640

Εικόνα 36 Δεδομένα Κόμβων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων

Σύνδεσμοι



Εικόνα 37 Χωρική αναπαράσταση Συνδέσμων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων



Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	1	1005603	1	104393	104388	13	18	1	1	17	0xE6100000010418000000287E8CB96B714240EF0390DAC48...
2	2	1005572	1	104374	104376	15	10	2	1	13	0xE6100000010404000000050E449D235BF424067B5C01E1361...
3	3	1005599	1	104392	104393	14	20	1	1	17	0xE6100000010415000000088D7F50B7689424032E9EFA5F06C...
4	4	1005571	1	104374	104373	15	15	2	1	19	0xE6100000010407000000050E449D235BF424067B5C01E1361...
5	5	1005594	1	104377	168100	20	13	4	1	15	0xE6100000010425000000DBA6785C548B4240D4D7F335CB6...
6	6	1005595	1	104387	104388	14	11	1	1	12	0xE6100000010464000000040C05AB56B84424072FE26142220...
7	7	1080256	1	104379	168100	14	11	5	1	16	0xE6100000010427000000AE0FEB8D5A954240C824236761D...
8	8	1005570	1	104376	104363	18	13	5	1	15	0xE6100000010425000000032B08EE387BC424013D21A834E6...
9	9	1005605	1	104387	104392	11	18	3	1	17	0xE6100000010443E00000040C05AB56B84424072FE26142220...
10	10	1004087	1	103286	103298	11	14	5	1	17	0xE6100000010441E0000004339D1AE42CA4240800D8810579...
11	11	1005574	1	104379	104363	17	10	2	1	15	0xE6100000010432000000AE0FEB8D5A954240C824236761D...
12	12	1005609	1	104397	104398	18	20	5	1	16	0xE6100000010412000000C91CCBBBEA7F42405E85949F54D...
13	13	1037504	1	125089	104375	10	19	4	1	17	0xE6100000010413000000DF5339ED299942405DFA97A4320...
14	14	1004072	1	103287	103288	13	17	2	1	13	0xE6100000010466000000FD69A33A1DCC42402CD7DB662A...
15	15	1037509	1	103288	125088	10	11	1	1	20	0xE6100000010407000009087BEBB95B342402D0ABB287A2...
16	16	1037510	1	104378	125089	16	18	2	1	13	0xE6100000010440E000000B804E09F52A14240DC2DC901BB0...
17	17	1005618	1	104400	104397	19	16	2	1	12	0xE6100000010412000000F1B913ECBF864240098A1F63EEB6...
18	18	1005575	1	104373	103288	11	12	4	1	12	0xE6100000010417000000014E8137992BA424082531F48DE51...
19	19	1005576	1	104362	104374	17	14	5	1	11	0xE6100000010404000000231631EC30C0424020425C397B67...
20	20	1005611	1	103298	104378	18	15	1	1	13	0xE6100000010416000000AF7D01BD70A54240293DD34B8CB...
21	21	1005612	1	104398	104387	17	10	5	1	14	0xE6100000010406000000A9FB00A436874240CAFACDC4741...
22	22	1005613	1	104375	104398	10	14	4	1	10	0xE6100000010406000000452921585588424020ED7F80B50A...
23	23	1005566	1	104362	104372	16	14	1	1	12	0xE6100000010403000000231631EC30C0424020425C397B67...
24	24	1005569	1	104376	104377	17	14	4	1	17	0xE6100000010430000000032B08EE387BC424013D21A834E6...
25	25	1005615	1	104400	103298	12	12	2	1	16	0xE6100000010443F000000F1B913ECBF864240098A1F63EEB6...
26	26	1037506	1	125088	104378	10	15	3	1	13	0xE6100000010441E0000000B410E4A98AD4240DBDD03745F2...
27	27	1005564	1	104372	104373	17	13	1	1	15	0xE6100000010406000000B30C71AC8BC14240FBAF73D3666...
28	28	1005565	1	104372	104374	14	11	2	1	13	0xE6100000010404000000B30C71AC8BC14240FBAF73D3666...

Εικόνα 38 Δεδομένα Συνδέσμων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων

Όπως παρατηρούμε στο table των κόμβων οι στήλες LinkExiting και LinksEntering είναι κενές. Οι στήλες αυτές περιέχουν πληροφορίες για το πόσοι σύνδεσμοι εισέρχονται και εξέρχονται από έναν κόμβο. Ουσιαστικά μας βοήθανε να καθορίσουμε αν ένας κόμβος είναι διασταύρωση ή όχι. Οι περιπτώσεις όπου ένας κόμβος δεν είναι διασταύρωση είναι τρείς:

1. Στον κόμβο να εισέρχεται μόνο ένας σύνδεσμος και να εξέρχεται από αυτόν επίσης μόνο ένας. Οι σύνδεσμοι πρέπει είτε να είναι και οι δύο μονής ή και οι δύο διπλής κατεύθυνσης
2. Στον κόμβο να εισέρχονται μόνο δύο σύνδεσμοι διπλής κατεύθυνσης αλλά να μην εξέρχεται κανένας
3. Από τον κόμβο να εξέρχονται μόνο δύο σύνδεσμοι διπλής κατεύθυνσης αλλά να μην εισέρχεται κανένας

Επόμενο βήμα λοιπόν είναι να υπολογίσουμε τις κενές στήλες στο table των κόμβων. Τρέχουμε λοιπόν το Calculate_Power_Of_Nodes_In_Database.py και το table των κόμβων παίρνει την εξής μορφή:

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	1	104393	1	3	1	1	0xE6100000010C287E8CB96B714240EF0390DAC4853640
2	2	104388	1	5	0	2	0xE6100000010C9BC761307F6142406FA0C03BF9903640
3	3	104374	1	5	2	2	0xE6100000010C50E449D235BF424067B5C01E13613640
4	4	104376	1	2	2	1	0xE6100000010C32B08EE387BC424013D21A834E683640
5	5	104392	1	1	1	1	0xE6100000010C88D7F50B7689424032E9EFA5F06C3640
6	6	104373	1	5	1	2	0xE6100000010C14E8137992BA424082531F48DE513640
7	7	104377	1	5	1	1	0xE6100000010CDBA6785C548B4240D4D7F335CB6D3640
8	8	168100	1	2	0	2	0xE6100000010C6364C91CCB7F4240986A662D05B03640
9	9	104387	1	4	2	1	0xE6100000010C40C05AB56B84424072FE261422203640
10	10	104379	1	2	2	0	0xE6100000010CAE0FEB8D5A954240C824236761DB3640
11	11	104363	1	3	0	2	0xE6100000010C00E31934F4B34240425DA45016BA3640
12	12	103286	1	3	1	0	0xE6100000010C4339D1AE42CA4240800D8810579A3540
13	13	103298	1	2	1	2	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
14	14	104397	1	1	1	1	0xE6100000010CC91CCBBBEA7F42405E85949F54DB3540
15	15	104398	1	2	1	2	0xE6100000010CA9FB00A436874240CAFACDC474113640
16	16	125089	1	1	1	1	0xE6100000010CDF5339ED299942405DFA97A432013640
17	17	104375	1	4	1	1	0xE6100000010C45292158558B424020ED7F80B50A3640
18	18	103287	1	3	1	0	0xE6100000010CFD69A33A1DCC42402CD7DB662AA03540
19	19	103288	1	5	1	2	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640
20	20	125088	1	1	1	1	0xE6100000010C0B410E4A98AD4240DBDD03745F223640
21	21	104378	1	4	1	2	0xE6100000010CB804E09F52A14240DC2DC901BB023640
22	22	104400	1	3	2	0	0xE6100000010CF1B913ECBF864240098A1F63EEB63540
23	23	104362	1	2	2	0	0xE6100000010C231631EC30C0424020425C397B673640
24	24	104372	1	2	2	1	0xE6100000010CB30C71AC8BC14240FBAF73D366603640

Εικόνα 39 Δεδομένα Κόμβων δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά τον υπολογισμό των στηλών 'LinksEntering' και ''LinksExiting

Έπειτα δίνουμε το δικαίωμα στην χρήστη να καθορίσει ποιους κόμβους θεωρεί εκείνος μη αναγκαίους αλλάζοντας την μεταβλητή σημαντικότητας (Visibility) είτε βάση της κλάσης του κόμβου είτε βάση συγκεκριμένων IDs.

Ας θεωρήσουμε ότι ο χρήστης επιλέγει να αλλάξει το 'Visibility' για τους κόμβους με κλάση '2' τρέχοντας το Change_Visibility_Of_Node_In_Database_By_Class.py. Η τελική μορφή του table των κόμβων είναι η ακόλουθη:

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	1	104393	1	3	1	1	0xE6100000010C287E8CB96B714240EF0390DAC4853640
2	2	104388	1	5	0	2	0xE6100000010C9BC761307F6142406FA0C03BF9903640
3	3	104374	1	5	2	2	0xE6100000010C50E449D235BF424067B5C01E13613640
4	4	104376	0	2	2	1	0xE6100000010C32B08EE387BC424013D21A834E683640
5	5	104392	1	1	1	1	0xE6100000010C88D7F50B7689424032E9EFA5F06C3640
6	6	104373	1	5	1	2	0xE6100000010C14E8137992BA424082531F48DE513640
7	7	104377	1	5	1	1	0xE6100000010CDBA6785C548B4240D4D7F335CB6D3640
8	8	168100	0	2	0	2	0xE6100000010C6364C91CCB7F4240986A662D05B03640
9	9	104387	1	4	2	1	0xE6100000010C40C05AB56B84424072FE261422203640
10	10	104379	0	2	2	0	0xE6100000010CAE0FEB8D5A954240C824236761DB3640
11	11	104363	1	3	0	2	0xE6100000010C00E31934F4B34240425DA45016BA3640
12	12	103286	1	3	1	0	0xE6100000010C4339D1AE42CA4240800D8810579A3540
13	13	103298	0	2	1	2	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
14	14	104397	1	1	1	1	0xE6100000010CC91CCBBBEA7F42405E85949F54DB3540
15	15	104398	0	2	1	2	0xE6100000010CA9FB00A436874240CAFACDC474113640
16	16	125089	1	1	1	1	0xE6100000010CDF5339ED299942405DFA97A432013640
17	17	104375	1	4	1	1	0xE6100000010C45292158558B424020ED7F80B50A3640
18	18	103287	1	3	1	0	0xE6100000010CFD69A33A1DCC42402CD7DB662AA03540
19	19	103288	1	5	1	2	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640
20	20	125088	1	1	1	1	0xE6100000010C0B410E4A98AD4240DBDD03745F223640
21	21	104378	1	4	1	2	0xE6100000010CB804E09F52A14240DC2DC901BB023640
22	22	104400	1	3	2	0	0xE6100000010CF1B913ECBF864240098A1F63EEB63540
23	23	104362	0	2	2	0	0xE6100000010C231631EC30C0424020425C397B673640
24	24	104372	0	2	2	1	0xE6100000010CB30C71AC8BC14240FBAF73D366603640

Εικόνα 40 Δεδομένα Κόμβων δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά την αλλάγη του 'Visibility' ορισμένων Κόμβων

Στην συνέχεια τρέχουμε το Adjacency_Dictionary_From_Database.py και κατασκευάζουμε το Adjacency Dictionary.

Τώρα που έχουμε συμπληρώσει πλήρως το table των κόμβων και έχουμε μεταβεί από την βάση δεδομένων στην Adjacent δομή, πρέπει να βρούμε τους κόμβους προς αφαίρεση.

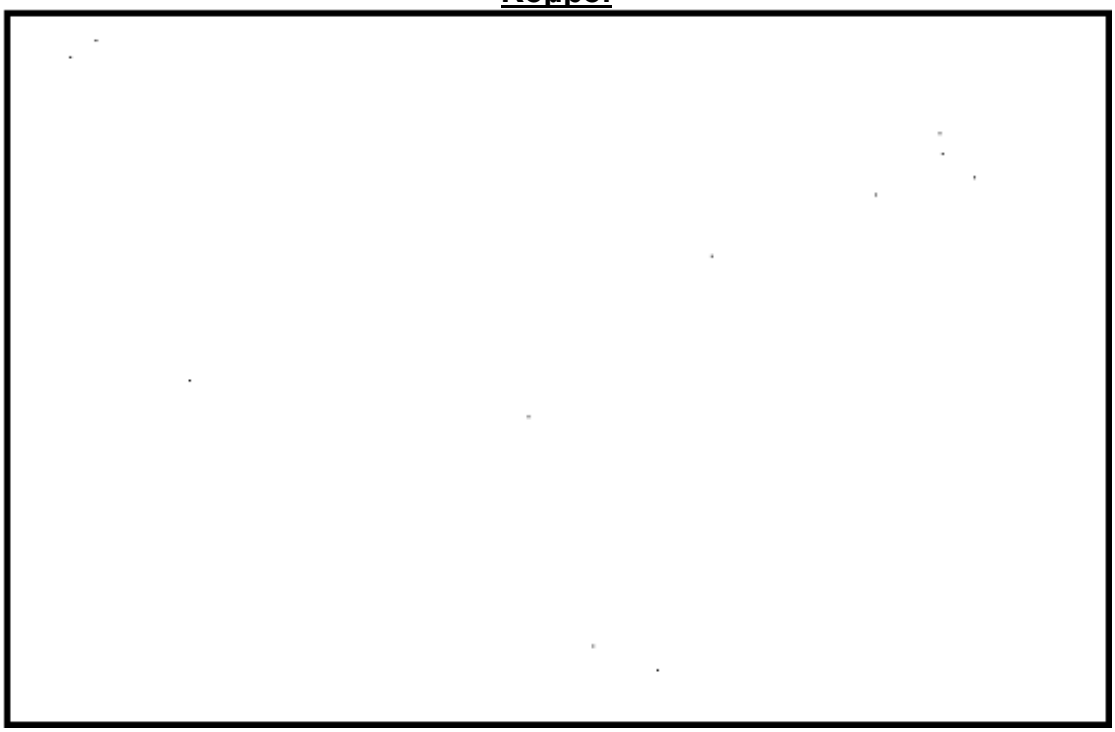
Ένας κόμβος είναι προς αφαίρεση όταν δεν είναι διασταύρωση και το 'Visibility' του είναι '0'. Για την διευκόλυνση της προσομοίωσης και για να έχουμε αρκετούς κόμβους προς αφαίρεση θεωρούμε ότι όλοι οι σύνδεσμοι είναι διπλής κατεύθυνσης και πως όλοι οι κόμβοι έχουν 'Visibility' '0'.

Τρέχοντας λοιπόν το Finds_Nodes_To_Remove_From_Database.py βρίσκουμε πως οι κόμβοι που πρέπει να αφαιρεθούν από το δίκτυο είναι οι: 104393, 104392, 104377, 104397, 125089, 104375, 125088, 104379, 104400, 104362, 104388, 168100, 104363.

Βάζοντας του κόμβους αυτούς σαν όρισμα στο Removing_Nodes_From_Dictionary.py αφαιρούνται οι κόμβοι αυτοί από την δομή μας, η οποία τελικά καταλήγει να περιέχει το παράγωγο δίκτυο.

Τέλος αυτό που μένει είναι να απεικονίσουμε το δίκτυο μας στην βάση δεδομένων μέσω του Adjacency_Dictionary_To_Database.py.

Κόμβοι



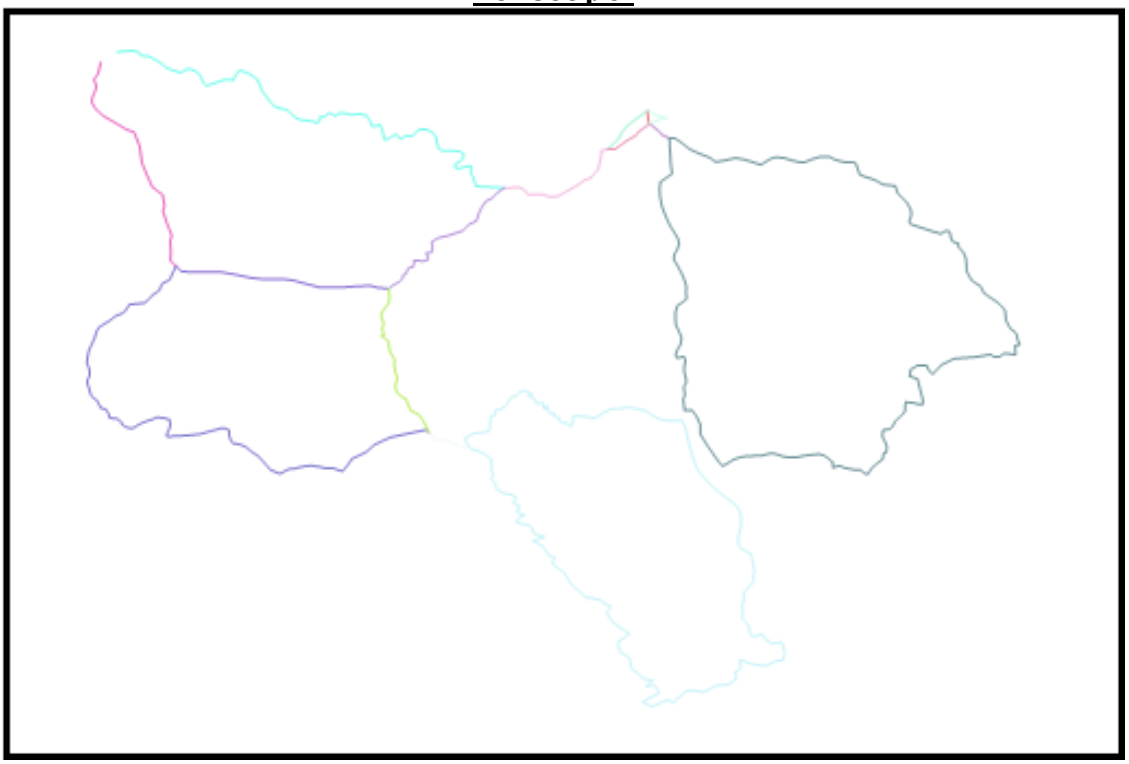
Εικόνα 41 Χωρική αναπαράσταση Κόμβων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά την αφαίρεση των αχρείαστων Κόμβων

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	0	103298	1	2	NULL	NULL	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
2	1	104372	1	2	NULL	NULL	0xE6100000010CB30C71AC8BC14240FBAF73D366603640
3	2	104373	1	5	NULL	NULL	0xE6100000010C14E8137992BA424082531F48DE513640
4	3	104374	1	5	NULL	NULL	0xE6100000010C50E449D235BF424067B5C01E13613640
5	4	104376	1	2	NULL	NULL	0xE6100000010C32B08EE387BC424013D21A834E683640
6	5	104378	1	4	NULL	NULL	0xE6100000010CB804E09F52A14240DC2DC901BB023640
7	6	104387	1	4	NULL	NULL	0xE6100000010C40C05AB56B84424072FE261422203640
8	7	104398	1	2	NULL	NULL	0xE6100000010CA9FB00A436874240CAFACDC474113640
9	8	103286	1	3	NULL	NULL	0xE6100000010C4339D1AE42CA4240800D8810579A3540
10	9	103287	1	3	NULL	NULL	0xE6100000010CFD69A33A1DCC42402CD7DB662AA03540
11	10	103288	1	5	NULL	NULL	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640

Εικόνα 42 Δεδομένα Κόμβων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά την αφαίρεση των αχρείαστων Κόμβων

Σύνδεσμοι



Εικόνα 43 Χωρική αναπαράσταση Συνδέσμων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά την αφαίρεση των αχρείαστων Κόμβων

Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	0	1005611	1	103298	104378	18	15	1	1	13	0xE6100000010416000000AF7D01BD70A54240293DD34B8CB...
2	1	999	1	103298	104398	16	48	2	1	16	0xE6100000010461000000AF7D01BD70A54240293DD34B8CB...
3	2	1005564	1	104372	104373	17	13	1	1	15	0xE6100000010406000000B30C71AC8BC14240FBAF73D3666...
4	3	1005565	1	104372	104374	14	11	2	1	13	0xE6100000010404000000B30C71AC8BC14240FBAF73D3666...
5	4	1005575	1	104373	103288	11	12	4	1	12	0xE6100000010417000000014E8137992BA424082531F48DE51...
6	5	1005572	1	104374	104376	15	10	2	1	13	0xE6100000010404000000050E449D235BF424067B5C01E1361...
7	6	1005571	1	104374	104373	15	15	2	1	19	0xE6100000010407000000050E449D235BF424067B5C01E1361...
8	7	999	1	104374	104372	11	28	5	1	12	0xE6100000010406000000050E449D235BF424067B5C01E1361...
9	8	999	1	104376	104376	18	61	5	1	17	0xE61000000104CF00000032B08EE387BC424013D21A834E6...
10	9	999	1	104378	104398	16	51	2	1	10	0xE6100000010425000000B804E09F52A14240DC2DC901BB0...
11	10	999	1	104387	104387	14	67	1	1	11	0xE61000000104CC00000040C05AB56B84424072FE26142220...
12	11	1005612	1	104398	104387	17	10	5	1	14	0xE6100000010406000000A9FB00A436874240CAFACDC4741...
13	12	1004087	1	103286	103298	11	14	5	1	17	0xE61000000104E0000004339D1AE42CA4240800D8810579...
14	13	1004072	1	103287	103288	13	17	2	1	13	0xE6100000010466000000FD69A33A1DCC42402CD7DB662A...
15	14	999	1	103288	104378	10	26	1	1	13	0xE6100000010424000000909 87BEBB95B342402D0ABB287A2...

Εικόνα 44 Δεδομένα Συνδέσμων δικτύου δεύτερης εφαρμογής σε SQL βάση δεδομένων, μετά την αφαίρεση των αχρείαστων Κόμβων

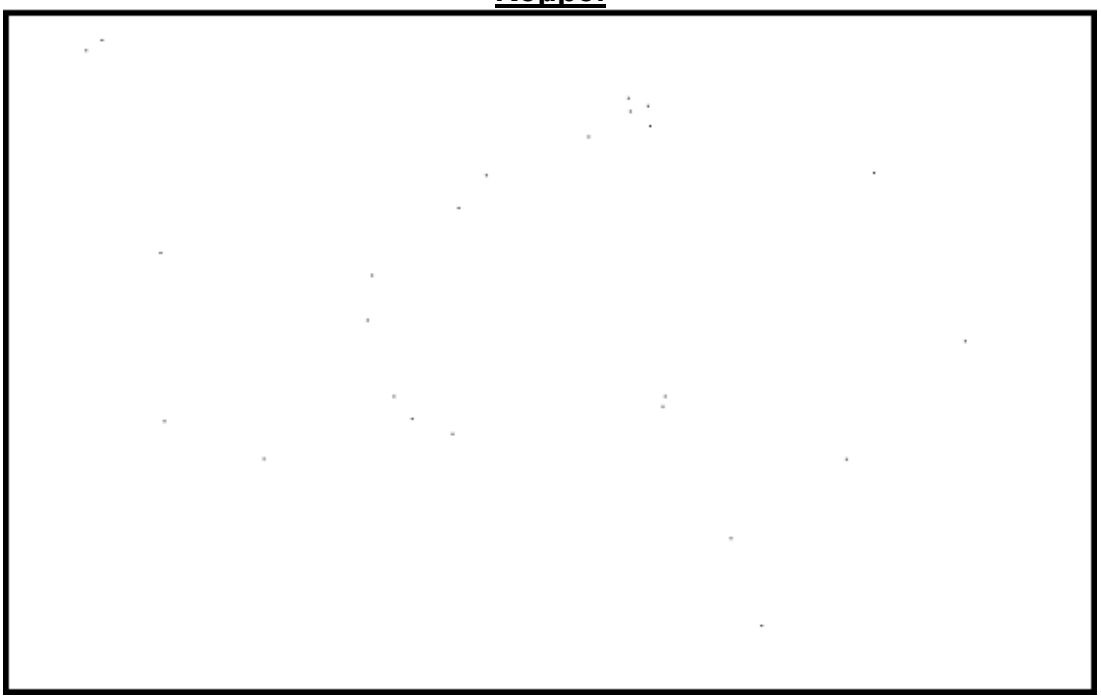
Συγκρίνοντας το αρχικό δίκτυο με το παράγωγο δίκτυο παρατηρούμε ότι ξεκινήσαμε με 24 κόμβους και μετά από την αφαίρεση 13 καταλήξαμε στους 11 ενώ από τους 28 συνδέσμους καταλήξαμε στους 15. Η γεωγραφική εικόνα του δικτύου μας παρέμεινε ίδια με το αρχικό, κάτι το οποίο ήταν όχι μόνο επιθυμητό αλλά άκρως απαραίτητο, ενώ τα κόστη εκκίνησης και μετάβασης υπολογίστηκαν καταλλήλως για τους εναπομείναντες συνδέσμους. Μόνο για την προσομοίωση αυτή χρησιμοποιήσαμε ένα κοινό LinkID για τους συνδέσμους. Συγκεκριμένα δώσαμε το ID ‘999’ στους καινούριους συνδέσμους

που κατασκευάσαμε ώστε να είναι πιο ευδιάκριτοι στον αναγνώστη. Καταλήξαμε λοιπόν σε ένα δίκτυο το οποίο περιέχει λιγότερους κόμβους και συνδέσμους γεγονός που το καθιστά σαφώς πιο αποδοτικό για οποιαδήποτε εφαρμογή θέλουμε να τρέξουμε σε αυτό.

8.3 Εισαγωγή νέου κόμβου στο δίκτυο

Μία επίσης σημαντική συνιστώσα στην οργάνωση των δικτύων μεταφορών είναι η δυνατότητα δημιουργίας νέων κόμβων και η ενσωμάτωση τους στο υπάρχον δίκτυο. Για διάφορους λόγους, οι κόμβοι που υπάρχουν σε ένα δίκτυο δεν εξυπηρετούν το ίδιο όλους τους χρήστες. Για κάποιους μπορεί το δίκτυο να είναι επαρκές αλλά για κάποιους άλλους να έχει ελλείψεις. Είναι κατανοητό λοιπόν πως για το βέλτιστο σχεδιασμό μίας διαδρομής το δίκτυο πρέπει να περιέχει όλους τους κόμβους που ο χρήστης θεωρεί αναγκαίους και στην περίπτωση που δεν υπάρχουν να κατασκευάζονται. Στην προσομοίωση αυτή θα δουλέψουμε πάνω σε ένα φαινομενικά ελλιπές δίκτυο και θα προσθέσουμε ένα καινούριο κόμβο σε συγκεκριμένη χιλιομετρική απόσταση ενός συνδέσμου. Το αρχικό μας δίκτυο είναι το ίδιο και με δίκτυο στο 8.2 και απεικονίζεται (στην βάση δεδομένων) παρακάτω:

Κόμβοι



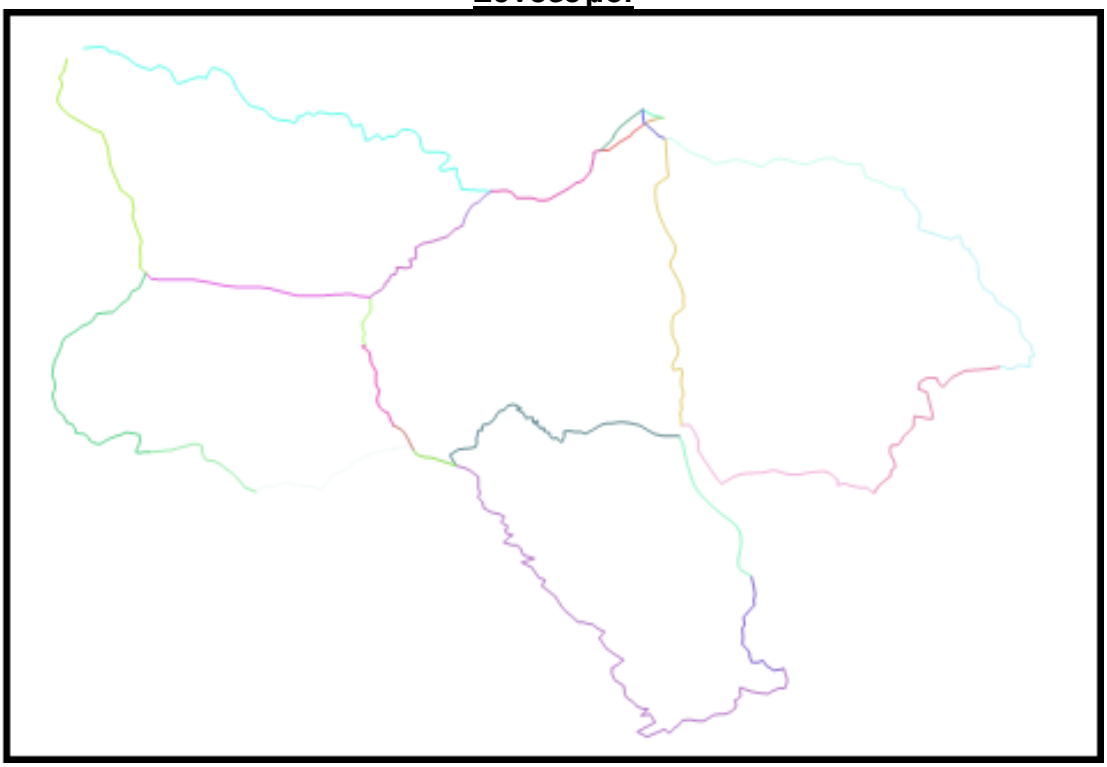
Εικόνα 45 Χωρική αναπαράσταση Κόμβων δικτύου τρίτηςεφαρμογής σε SQL βάση δεδομένων

Results | Spatial results | Messages

	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	1	104393	1	3	NULL	NULL	0xE6100000010C287E8CB96B714240EF0390DAC4853640
2	2	104388	1	5	NULL	NULL	0xE6100000010C9BC761307F6142406FA0C03BF9903640
3	3	104374	1	5	NULL	NULL	0xE6100000010C50E449D235BF424067B5C01E13613640
4	4	104376	1	2	NULL	NULL	0xE6100000010C32B08EE387BC424013D21A834E683640
5	5	104392	1	1	NULL	NULL	0xE6100000010C88D7F50B7689424032E9EFA5F06C3640
6	6	104373	1	5	NULL	NULL	0xE6100000010C14E8137992BA424082531F48DE513640
7	7	104377	1	5	NULL	NULL	0xE6100000010CDBA6785C548B4240D4D7F335CB6D3640
8	8	168100	1	2	NULL	NULL	0xE6100000010C6364C91CCB7F4240986A662D05B03640
9	9	104387	1	4	NULL	NULL	0xE6100000010C40C05AB56B84424072FE261422203640
10	10	104379	1	2	NULL	NULL	0xE6100000010CAE0FEB8D5A954240C824236761DB3640
11	11	104363	1	3	NULL	NULL	0xE6100000010C00E31934F4B34240425DA45016BA3640
12	12	103286	1	3	NULL	NULL	0xE6100000010C4339D1AE42CA4240800D8810579A3540
13	13	103298	1	2	NULL	NULL	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
14	14	104397	1	1	NULL	NULL	0xE6100000010CC91CCBBBEA7F42405E85949F54DB3540
15	15	104398	1	2	NULL	NULL	0xE6100000010CA9FB00A436874240CAFACDC474113640
16	16	125089	1	1	NULL	NULL	0xE6100000010CDF5339ED299942405DFA97A432013640
17	17	104375	1	4	NULL	NULL	0xE6100000010C45292158558B424020ED7F80B50A3640
18	18	103287	1	3	NULL	NULL	0xE6100000010CFD69A33A1DCC42402CD7DB662AA03540
19	19	103288	1	5	NULL	NULL	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640
20	20	125088	1	1	NULL	NULL	0xE6100000010C0B410E4A98AD4240DBDD03745F223640
21	21	104378	1	4	NULL	NULL	0xE6100000010CB804E09F52A14240DC2DC901BB023640
22	22	104400	1	3	NULL	NULL	0xE6100000010CF1B913ECBF864240098A1F63EEB63540
23	23	104362	1	2	NULL	NULL	0xE6100000010C231631EC30C0424020425C397B673640
24	24	104372	1	2	NULL	NULL	0xE6100000010CB30C71AC8BC14240FBAF73D366603640

Εικόνα 46 Δεδομένα Κόμβων δικτύου τρίτης εφαρμογής σε SQL βάση δεδομένων

Σύνδεσμοι



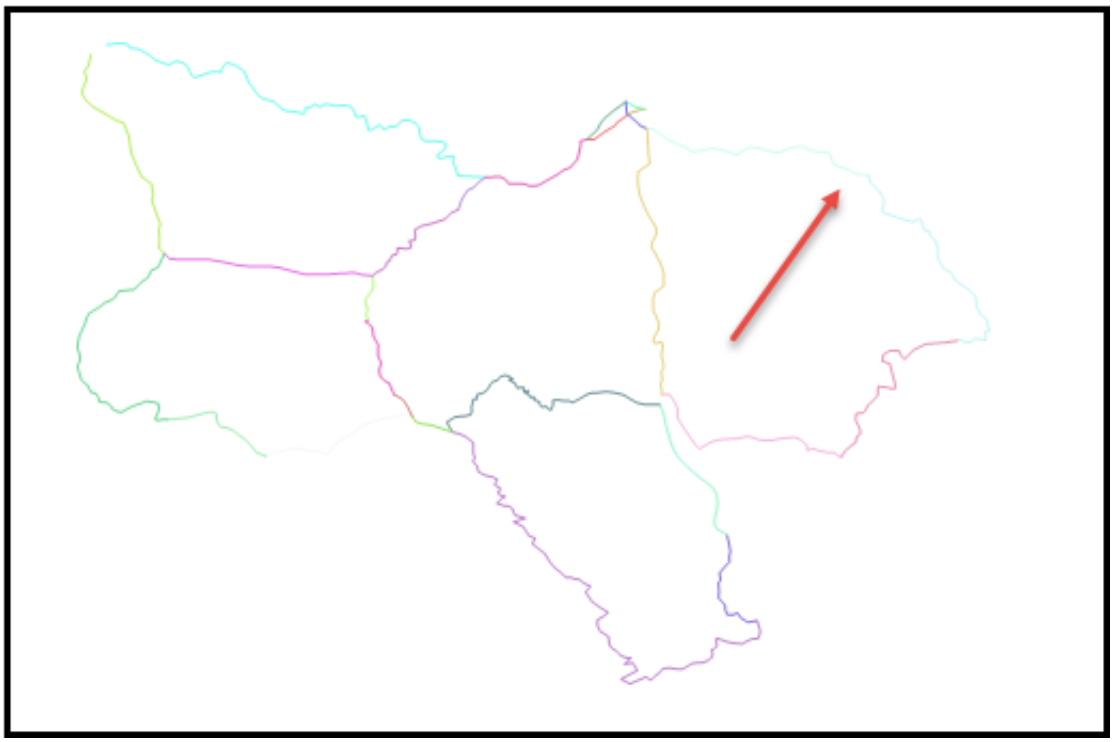
Εικόνα 47 Χωρική αναπαράσταση Συνδέσμων δικτύου τρίτης εφαρμογής σε SQL βάση δεδομένων

Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	1	1005603	1	104393	104388	13	18	1	1	17	0xE6100000010418000000287E8CB96B714240EF0390DAC48...
2	2	1005572	1	104374	104376	15	10	2	1	13	0xE6100000010404000000050E449D235BF424067B5C01E1361...
3	3	1005599	1	104392	104393	14	20	1	1	17	0xE6100000010415000000088D7F50B7689424032E9EFA5F06C...
4	4	1005571	1	104374	104373	15	15	2	1	19	0xE6100000010407000000050E449D235BF424067B5C01E1361...
5	5	1005594	1	104377	168100	20	13	4	1	15	0xE6100000010425000000DBA6785C548B4240D4D7F335CB6...
6	6	1005595	1	104387	104388	14	11	1	1	12	0xE6100000010464000000040C05AB56B84424072FE26142220...
7	7	1080256	1	104379	168100	14	11	5	1	16	0xE6100000010427000000AE0FEB8D5A954240C824236761D...
8	8	1005570	1	104376	104363	18	13	5	1	15	0xE6100000010425000000032B08EE387BC424013D21A834E6...
9	9	1005605	1	104387	104392	11	18	3	1	17	0xE6100000010430000040C05AB56B84424072FE26142220...
10	10	1004087	1	103286	103298	11	14	5	1	17	0xE6100000010410000004339D1AE42CA4240800D8810579...
11	11	1005574	1	104379	104363	17	10	2	1	15	0xE6100000010432000000AE0FEB8D5A954240C824236761D...
12	12	1005609	1	104397	104398	18	20	5	1	16	0xE6100000010412000000C91CCBBBEA7F42405E85949F54D...
13	13	1037504	1	125089	104375	10	19	4	1	17	0xE6100000010413000000DF5339ED299942405DFA97A4320...
14	14	1004072	1	103287	103288	13	17	2	1	13	0xE6100000010466000000FD69A33A1DCC42402CD7DB662A...
15	15	1037509	1	103288	125088	10	11	1	1	20	0xE6100000010407000000909087BEBB95B342402D0ABB287A2...
16	16	1037510	1	104378	125089	16	18	2	1	13	0xE610000001040E000000B804E09F52A14240DDC2DC901BB0...
17	17	1005618	1	104400	104397	19	16	2	1	12	0xE6100000010412000000F1B913ECBF864240098A1F63EEB6...
18	18	1005575	1	104373	103288	11	12	4	1	12	0xE6100000010417000000014E8137992BA424082531F48DE51...
19	19	1005576	1	104362	104374	17	14	5	1	11	0xE6100000010404000000231631EC30C0424020425C397B67...
20	20	1005611	1	103298	104378	18	15	1	1	13	0xE6100000010416000000AF7D01BD70A54240293DD34B8CB...
21	21	1005612	1	104398	104387	17	10	5	1	14	0xE6100000010406000000A9FB00A436874240CAFACDC4741...
22	22	1005613	1	104375	104398	10	14	4	1	10	0xE6100000010406000000452921585588424020ED7F80B50A...
23	23	1005566	1	104362	104372	16	14	1	1	12	0xE6100000010403000000231631EC30C0424020425C397B67...
24	24	1005569	1	104376	104377	17	14	4	1	17	0xE6100000010430000000032B08EE387BC424013D21A834E6...
25	25	1005615	1	104400	103298	12	12	2	1	16	0xE6100000010430000F1B913ECBF864240098A1F63EEB6...
26	26	1037506	1	125088	104378	10	15	3	1	13	0xE610000001041E0000000B410E4A98AD4240DBDD03745F2...
27	27	1005564	1	104372	104373	17	13	1	1	15	0xE6100000010406000000B30C71AC8BC14240FBAF73D3666...
28	28	1005565	1	104372	104374	14	11	2	1	13	0xE6100000010404000000B30C71AC8BC14240FBAF73D3666...

Εικόνα 48 Δεδομένα Συνδέσμων δικτύου τρίτης εφαρμογής σε SQL βάση δεδομένων

Αυτό που θα προσπαθήσουμε είναι να εισάγουμε ένα νέο κόμβο κλάσης ‘3’ με ID ‘200111’ στο σύνδεσμο με LinkID ‘1005570’ σε απόσταση 25.6 χιλιομέτρων από την αρχή του συνδέσμου. Στην εικόνα που ακολουθεί φαίνεται ο σύνδεσμος στον οποίο θα εισαχθεί ο καινούριος κόμβος:



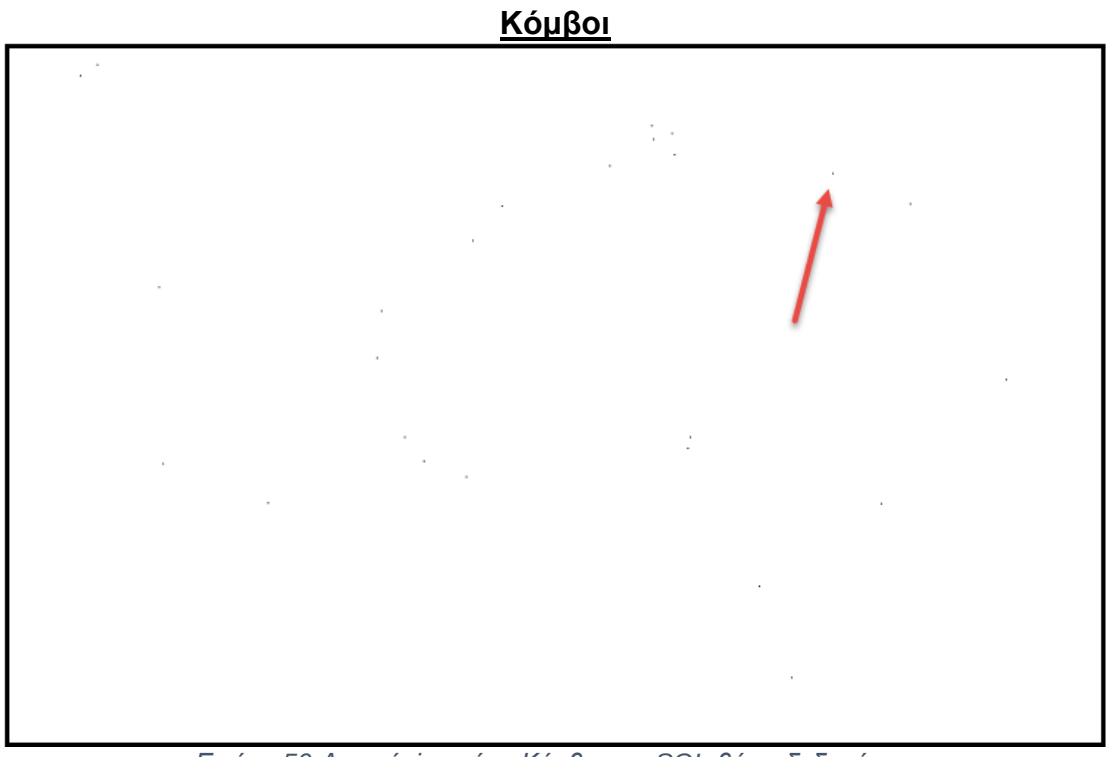
Εικόνα 49 Τοποθεσία εισαγωγής νέου Κόμβου

Εάν στην απόσταση που υποδεικνύει ο χρήστης υπάρχει ήδη σημείο που ανήκει στον σύνδεσμο τότε ο σύνδεσμος θα χωριστεί σε αυτό το σημείο και ο

καινούριος κόμβος θα πάρει τις συντεταγμένες του σημείου. Σε αντίθετη περίπτωση θα υπολογιστεί ένα καινούριο σημείο με καινούριες συντεταγμένες που θα ενσωματωθεί στον σύνδεσμο και θα αποτελέσει τον καινούριο κόμβο και το σημείο τομής του συνδέσμου.

Πρώτη μας κίνηση ως συνήθως είναι να τρέξουμε το Adjacency_Dictionary_From_Database.py και να κατασκευάσουμε το Adjacency Dictionary. ΄Έπειτα κάνοντας χρήση του Insert_Node_To_Dictionary.py και δίνοντας τα κατάλληλα ορίσματα που αναφέρθηκαν παραπάνω προσθέτουμε στην Adjacent δομή μας τον καινούριο κόμβο. Τέλος αυτό που πρέπει να κάνουμε είναι να απεικονίσουμε το δίκτυο μας και να μεταβούμε ή στην βάση δεδομένων ή στο ArcMap με τα Adjacency_Dictionary_To_Database.py, Adjacency_Dictionary_To_Shapefile.py.

Ας δούμε πρώτα το παράγωγο δίκτυο στην βάση δεδομένων:



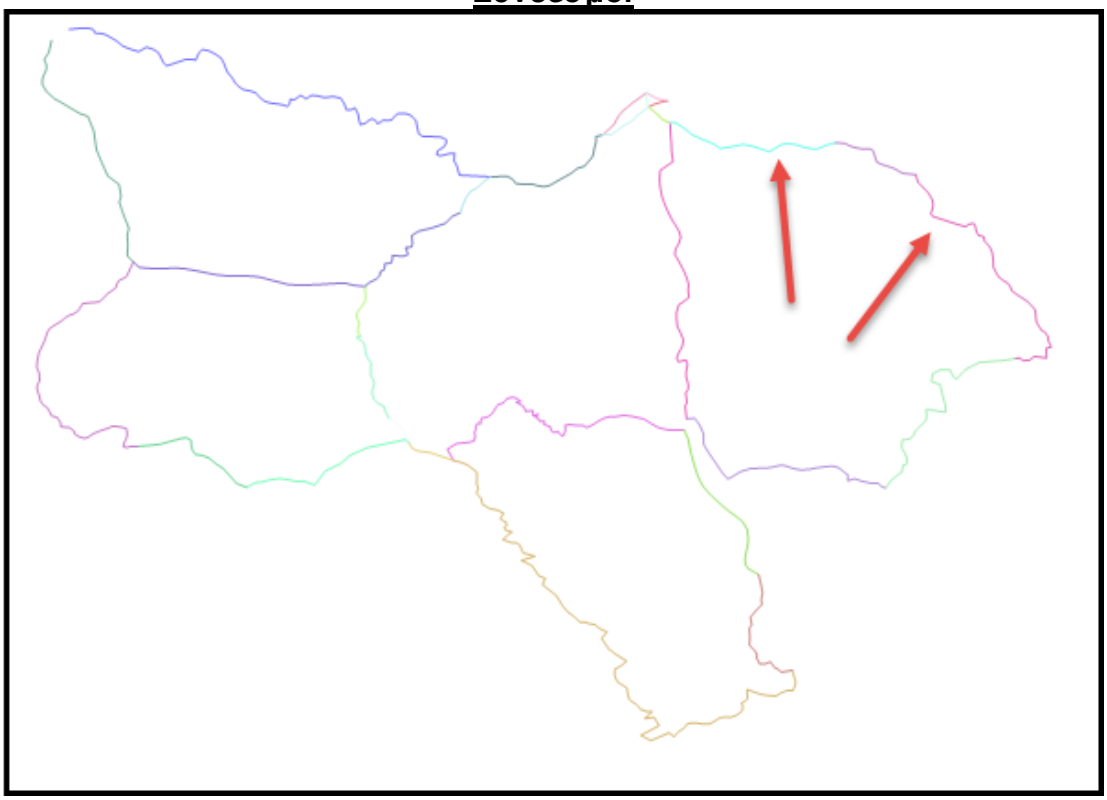
Εικόνα 50 Απεικόνιση νέου Κόμβου σε SQL βάση δεδομένων

Results | Spatial results | Messages

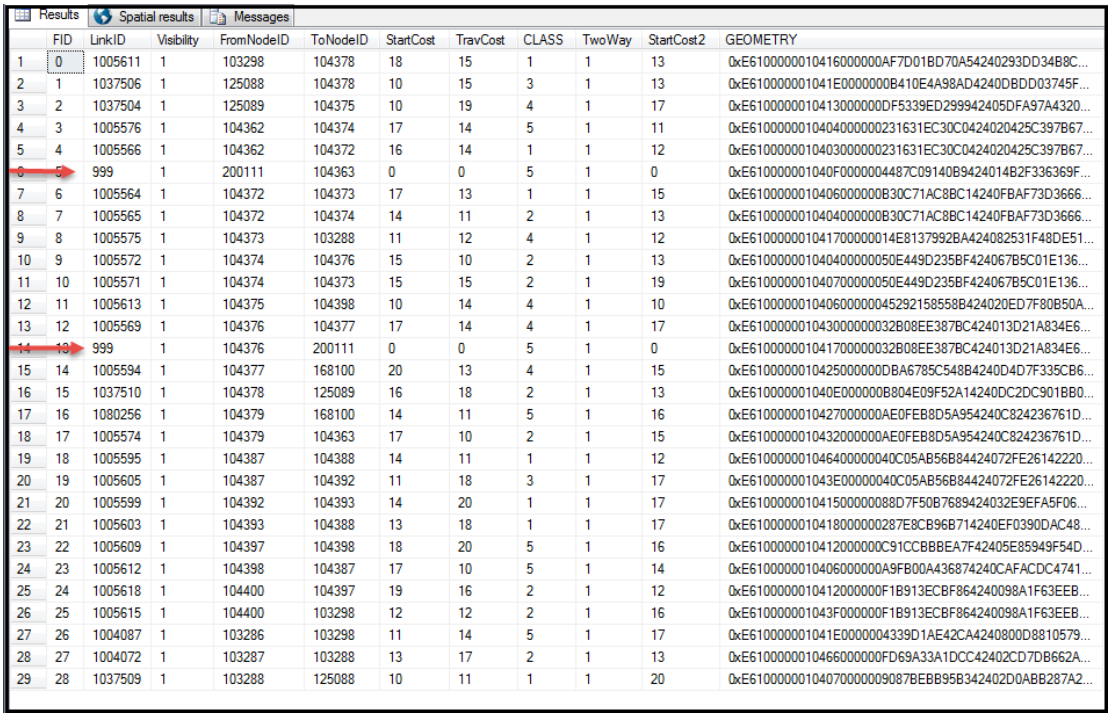
	FID	ID	VISIBILITY	CLASS	LinksExiting	LinksEntering	GEOMETRY
1	0	103298	1	2	NULL	NULL	0xE6100000010CAF7D01BD70A54240293DD34B8CB53540
2	1	125088	1	1	NULL	NULL	0xE6100000010C0B410E4A98AD4240DBDD03745F223640
3	2	125089	1	1	NULL	NULL	0xE6100000010CDF5339ED299942405DFA97A432013640
4	3	168100	1	2	NULL	NULL	0xE6100000010C6364C91CCB7F4240986A662D05B03640
5	4	104362	1	2	NULL	NULL	0xE6100000010C231631EC30C0424020425C397B673640
6	5	104363	1	3	NULL	NULL	0xE6100000010C00E31934F4B34240425DA45016BA3640
7	6	200111	1	3	NULL	NULL	0xE6100000010C14B2F336369F36404487C09140B94240
8	7	104372	1	2	NULL	NULL	0xE6100000010CB30C71AC8BC14240FBAF73D366603640
9	8	104373	1	5	NULL	NULL	0xE6100000010C14E8137992BA424082531F48DE513640
10	9	104374	1	5	NULL	NULL	0xE6100000010C50E449D235BF424067B5C01E13613640
11	10	104375	1	4	NULL	NULL	0xE6100000010C45292158558B424020ED7F80B50A3640
12	11	104376	1	2	NULL	NULL	0xE6100000010C32B08EE387BC424013D21A834E683640
13	12	104377	1	5	NULL	NULL	0xE6100000010CDBA6785C548B4240D4D7F335CB6D3640
14	13	104378	1	4	NULL	NULL	0xE6100000010CB804E09F52A14240DC2DC901BB023640
15	14	104379	1	2	NULL	NULL	0xE6100000010CAE0FEB8D5A954240C824236761DB3640
16	15	104387	1	4	NULL	NULL	0xE6100000010C40C05AB56B84424072FE261422203640
17	16	104388	1	5	NULL	NULL	0xE6100000010C9BC761307F6142406FA0C03BF9903640
18	17	104392	1	1	NULL	NULL	0xE6100000010C88D7F50B7689424032E9EFA5F06C3640
19	18	104393	1	3	NULL	NULL	0xE6100000010C287E8CB96B714240EF0390DAC4853640
20	19	104397	1	1	NULL	NULL	0xE6100000010CC91CCBBBEA7F42405E85949F54DB3540
21	20	104398	1	2	NULL	NULL	0xE6100000010CA9FB00A436874240CAFACDC474113640
22	21	104400	1	3	NULL	NULL	0xE6100000010CF1B913ECBF864240098A1F63EEB63540
23	22	103286	1	3	NULL	NULL	0xE6100000010C4339D1AE42CA4240800D8810579A3540
24	23	103287	1	3	NULL	NULL	0xE6100000010CFD69A33A1DCC42402CD7DB662AA035...
25	24	103288	1	5	NULL	NULL	0xE6100000010C9087BEBB95B342402D0ABB287A2C3640

Εικόνα 51 Δεδομένα νέου Κόμβου σε SQL βάση δεδομένων

Σύνδεσμοι



Εικόνα 52 Απεικόνιση των δύο συνδέσμων που δημιουργούνται μετά την εισαγωγή του νέου Κόμβου σε SQL βάση δεδομένων



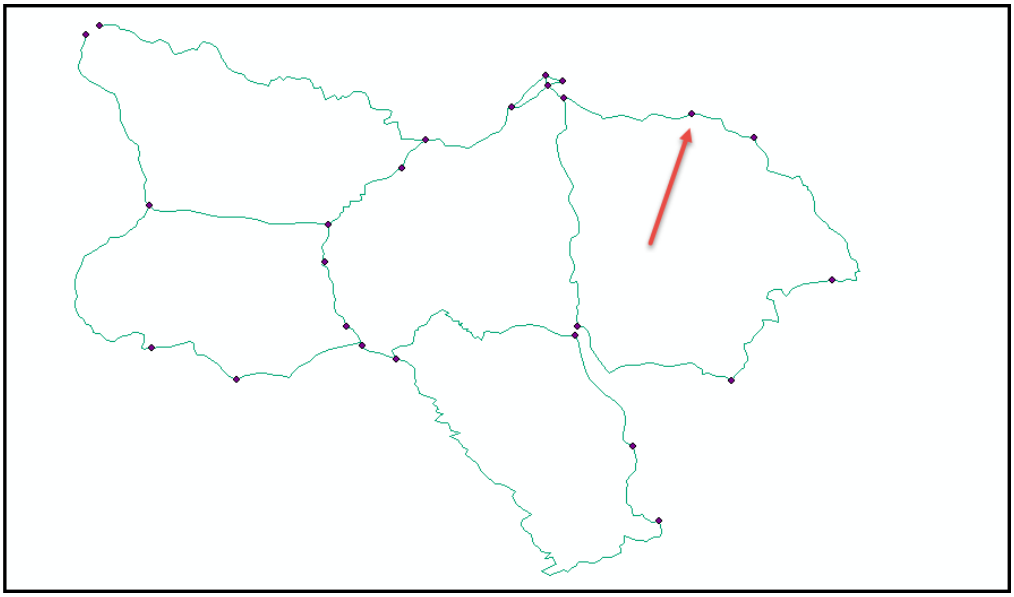
Results | Spatial results | Messages

	FID	LinkID	Visibility	FromNodeID	ToNodeID	StartCost	TravCost	CLASS	TwoWay	StartCost2	GEOMETRY
1	0	1005611	1	103298	104378	18	15	1	1	13	0xE6100000010416000000AF7D01BD70A54240293DD34B8C...
2	1	1037506	1	125088	104378	10	15	3	1	13	0xE6100000010041E0000000B410E4A98AD4240DBDD03745F...
3	2	1037504	1	125089	104375	10	19	4	1	17	0xE6100000010413000000DF5339ED299942405DFA97A4320...
4	3	1005576	1	104362	104374	17	14	5	1	11	0xE6100000010404000000231631EC30C0424020425C397B67...
5	4	1005566	1	104362	104372	16	14	1	1	12	0xE6100000010403000000231631EC30C0424020425C397B67...
6	5	999	1	200111	104363	0	0	5	1	0	0xE6100000010040F0000004487C09140B9424014B2F336369F...
7	6	1005564	1	104372	104373	17	13	1	1	15	0xE6100000010406000000B30C71AC8BC14240FBAF73D3666...
8	7	1005565	1	104372	104374	14	11	2	1	13	0xE6100000010404000000B30C71AC8BC14240FBAF73D3666...
9	8	1005575	1	104373	103288	11	12	4	1	12	0xE6100000010417000000014E8137992BA424082531F48DE51...
10	9	1005572	1	104374	104376	15	10	2	1	13	0xE6100000010404000000050E449D235BF424067B5C01E136...
11	10	1005571	1	104374	104373	15	15	2	1	19	0xE6100000010407000000050E449D235BF424067B5C01E136...
12	11	1005613	1	104375	104398	10	14	4	1	10	0xE6100000010406000000045292158558B424020ED7F80B50A...
13	12	1005569	1	104376	104377	17	14	4	1	17	0xE6100000010430000000032B08EE387BC424013D21A834E6...
14	13	999	1	104376	200111	0	0	5	1	0	0xE6100000010417000000032B08EE387BC424013D21A834E6...
15	14	1005594	1	104377	168100	20	13	4	1	15	0xE6100000010425000000DBA6785C548B4240D4D7F335CB6...
16	15	1037510	1	104378	125089	16	18	2	1	13	0xE6100000010040E000000B804E09F52A14240DC2DC901BB0...
17	16	1080256	1	104379	168100	14	11	5	1	16	0xE6100000010427000000AE0FEB8D5A954240C824236761D...
18	17	1005574	1	104379	104363	17	10	2	1	15	0xE6100000010432000000AE0FEB8D5A954240C824236761D...
19	18	1005595	1	104387	104388	14	11	1	1	12	0xE6100000010464000000040C05AB56B84424072FE26142220...
20	19	1005605	1	104387	104392	11	18	3	1	17	0xE6100000010430E00000040C05AB56B84424072FE26142220...
21	20	1005599	1	104392	104393	14	20	1	1	17	0xE6100000010415000000088D7F50B7689424032E9EFA5F06...
22	21	1005603	1	104393	104388	13	18	1	1	17	0xE6100000010418000000287E8CB96B714240EF0390DAC48...
23	22	1005609	1	104397	104398	18	20	5	1	16	0xE6100000010412000000C91CCBBBEA7F42405E85949F54D...
24	23	1005612	1	104398	104387	17	10	5	1	14	0xE6100000010406000000A9FB00A436874240CAFACDC4741...
25	24	1005618	1	104400	104397	19	16	2	1	12	0xE6100000010412000000F1B913ECBF864240098A1F63EEB...
26	25	1005615	1	104400	103298	12	12	2	1	16	0xE6100000010430F000000F1B913ECBF864240098A1F63EEB...
27	26	1004087	1	103286	103298	11	14	5	1	17	0xE6100000010410E00000004339D1AE42CA4240800D8810579...
28	27	1004072	1	103287	103288	13	17	2	1	13	0xE6100000010466000000FD69A33A1DCC42402CD7DB662A...
29	28	1037509	1	103288	125088	10	11	1	1	20	0xE6100000010407000000909087BEBB95B342402D0ABB287A2...

Εικόνα 53 Δεδομένα των δύο Συνδέσμων που δημιουργούνται μετά την εισαγωγή του νέου Κόμβου σε SQL βάση δεδομένων

Όπως παρατηρούμε ο νέος κόμβο έχει προστεθεί στο table των κόμβων και απεικονίζεται σωστά στα αντίστοιχα χωρικά αποτελέσματα. Επίσης ο σύνδεσμος, στον οποίο προστέθηκε ο κόμβος, έχει χωριστεί σε δύο μέρη στην κατάλληλη απόσταση.

Τέλος μπορούμε να απεικονίσουμε το δίκτυο μας στο ArcMap κατασκευάζοντας τα αντίστοιχα Shapefiles:

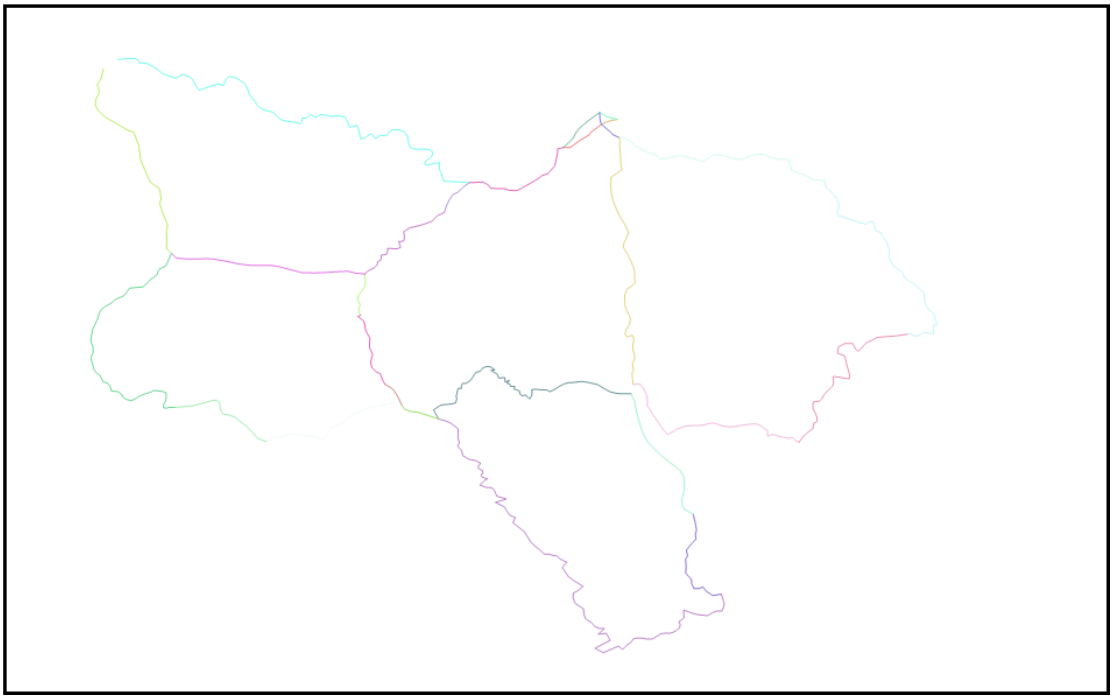


Εικόνα 54 Χωρική αναπαράσταση Κόμβων και Συνδέσμων δικτύου τρίτης εφαρμογής στο ArcGIS μετά την εισαγωγή του νέου Κόμβου

8.4 Αλγόριθμος Dijkstra

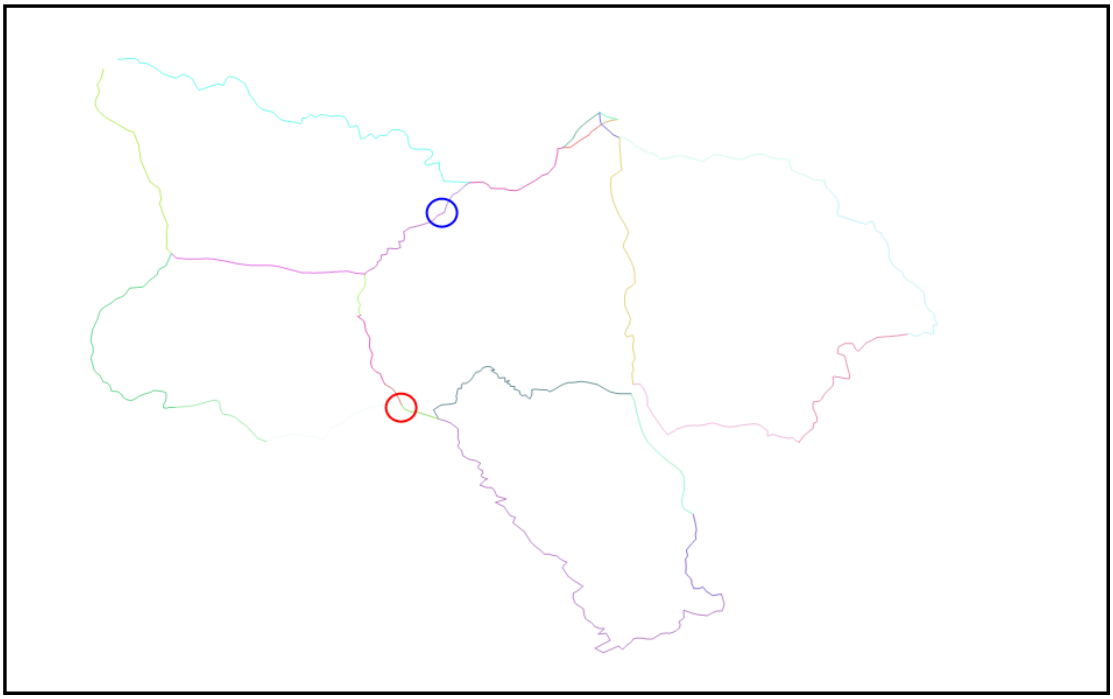
Ο **αλγόριθμος του Dijkstra** [19] πήρε το όνομά του από τον Ολλανδό Έντσγκερ Ντάικστρα, ο οποίος τον επινόησε το 1956 και τον δημοσίευσε το 1959. Πρόκειται για έναν αλγόριθμο εύρεσης συντομότερων διαδρομών (single-source shortest path problem) από κοινή αφετηρία σε έναν (κατευθυνόμενο ή μη) γράφο με μη αρνητικά βάρη στις ακμές. Ο αλγόριθμος του Dijkstra είναι άπληστος. Δηλαδή, σε κάθε βήμα επιλέγει την τοπικά βέλτιστη λύση, ώσπου στο τελευταίο βήμα συνθέτει μια συνολικά βέλτιστη λύση. Αν ο γράφος περιέχει αρνητικά βάρη, ο αλγόριθμος του Ντάικστρα δεν δίνει σωστό αποτέλεσμα. Για γράφους που μπορεί να έχουν αρνητικά βάρη στις ακμές, χρησιμοποιούνται πιο περίπλοκοι αλγόριθμοι, όπως αυτός των Bellman και Ford ή των Floyd-Warshall.

Ο αλγόριθμος του Ντάικστρα είναι πλέον ευρέως διαδεδομένος και χρησιμοποιείται σε πολλές εφαρμογές και κατά κόρον στα μεταφορικά δίκτυα. Εμείς θα χρησιμοποιήσουμε τον αλγόριθμο Dijkstra για να βρούμε την βέλτιστη διαδρομή μεταξύ δύο κόμβων. Ως βάρη χρησιμοποιούμε τις αποστάσεις μεταξύ των κόμβων. Το δίκτυο πάνω στο οποίο θα δουλέψουμε απεικονίζεται παρακάτω:

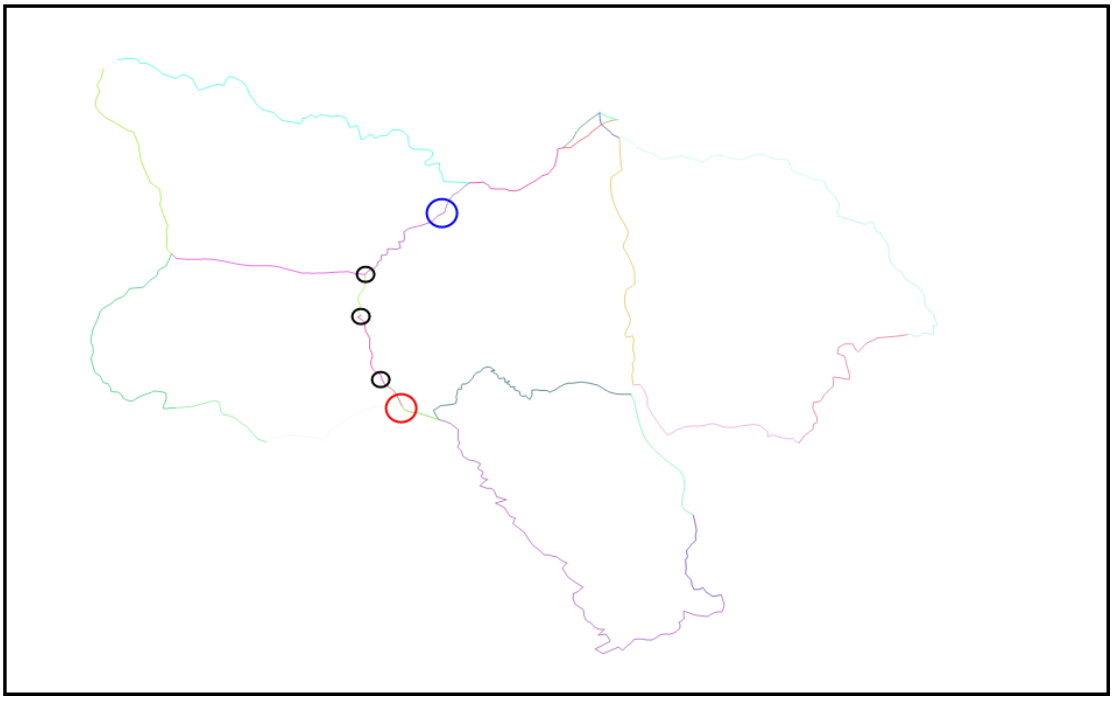


Εικόνα 55 Αναπαράσταση δικτύου τέταρτης εφαρμογής σε SQL βάση δεδεμένων

Πρώτη μας κίνηση ως συνήθως είναι να τρέξουμε το Adjacency_Dictionary_From_Database.py και να κατασκευάσουμε το Adjacency Dictionary. Αυτό που θα προσπαθήσουμε να κάνουμε είναι να βρούμε την βέλτιστη διαδρομή από τον κόμβο '104398' προς τον '125088'. Στην εικόνα 1 βλέπουμε την θέση των δύο κόμβων στο δίκτυο μας. Σε κόκκινο κύκλο είναι ο '104398' και σε γαλάζιο ο '125088'. Τρέχοντας μετά το Dijkstra.py μας επιστρέφεται μία λίστα με τους εξής κόμβους : [104398, 104375, 125089, 104378, 125088]. Στην εικόνα 2 απεικονίζονται οι παραπάνω κόμβοι και φαίνεται η βέλτιστη διαδρομή βάση του αλγορίθμου Dijkstra.



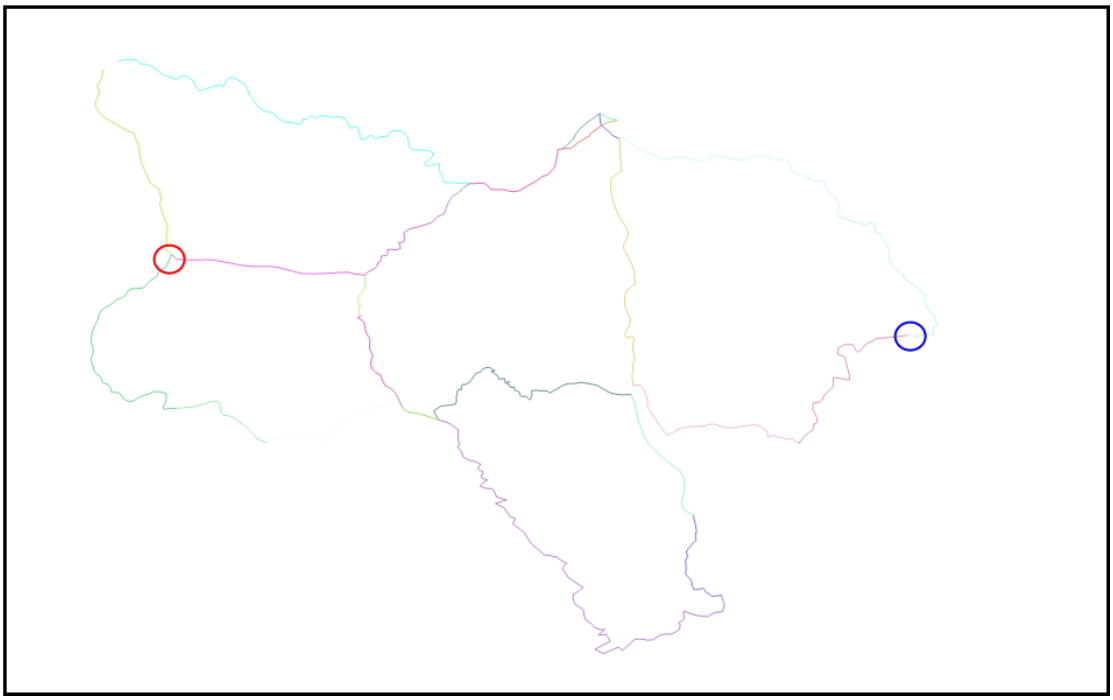
Εικόνα 56 Απεικόνιση Κόμβων αφετηρίας και τέρματος



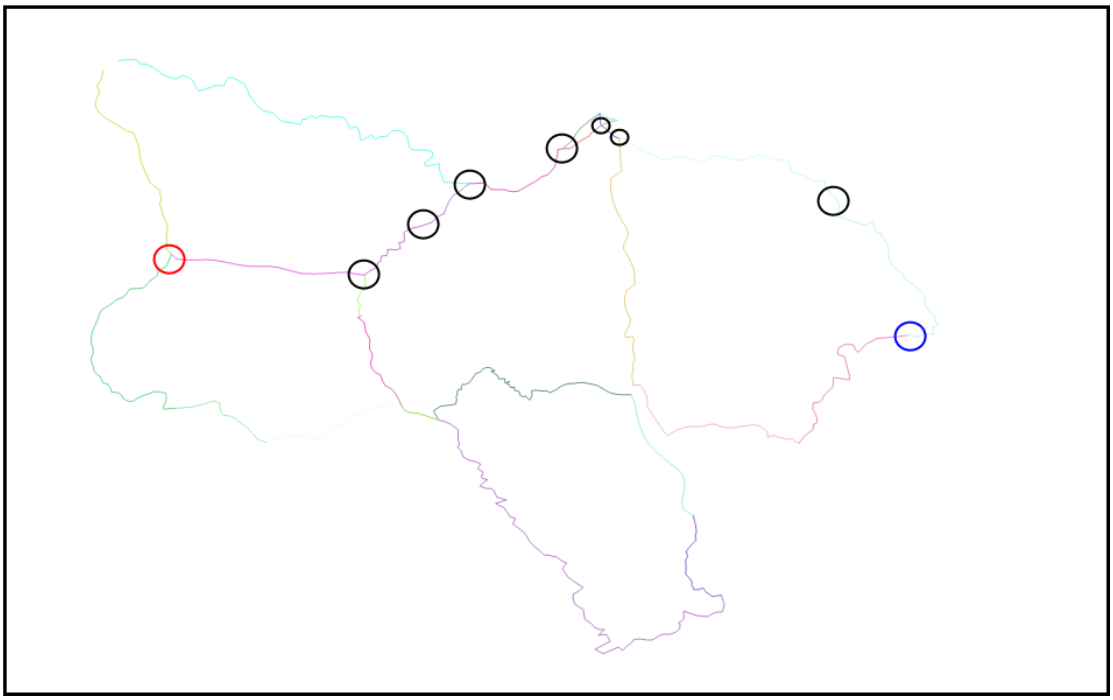
Εικόνα 57 Απεικόνιση Κομβών βέλτιστης προσπέλασης μεταξύ αφετηρίας και τέρματος

Κάνοντας μια δεύτερη δοκιμή, θα προσπαθήσουμε την βέλτιστη διαδρομή από τον κόμβο ‘103298’ προς τον ‘104379’. Στην εικόνα 3 βλέπουμε την θέση των δύο κόμβων στο δίκτυο μας. Σε κόκκινο κύκλο είναι ο ‘103298’ και σε γαλάζιο ο ‘104379’. Εκτελώντας τον αλγόριθμο Dijkstra στο δίκτυο μας επιστρέφεται η παρακάτω λίστα με κόμβους: [103298, 104378, 125088, 103288, 104373,

104374, 104376, 104363, 104379] η οποία αποτελεί την βέλτιστη διαδρομή και απεικονίζεται στην εικόνα 4.



Εικόνα 58 Απεικόνση Κόμβων αφετηρίας και τέρματος



Εικόνα 59 Απεικόνιση Κομβών βέλτιστης προσπέλασης μεταξύ αφετηρίας και τέρματος

Είδαμε λοιπόν πως χρησιμοποιώντας την Adjacent δομή που έχουμε παρουσιάσει σε προηγούμενο κεφάλαιο μπορούμε να τρέξουμε με επιτυχία τον αλγόριθμο Dijkstra ο οποίος σε κάθε περίπτωση μας υποδεικνύει την βέλτιστη

διαδρομή χρησιμοποιώντας ως βάρη των συνδέσμων τις αποστάσεις μεταξύ των κόμβων.

9ο Κεφάλαιο: Συμπεράσματα και προτάσεις για συνέχιση της εργασίας

Στο πλαίσιο της διπλωματικής εργασίας αναδείξαμε τα προβλήματα και τις ιδιαιτερότητες των εφαρμογών των μεταφορικών δικτύων που οφείλονται στην κακή οργάνωση της αποθηκευμένης πληροφορίας. Χαρακτηριστικά, είδαμε πως στις περισσότερες συμβατικές περιγραφές υποδομής δικτύων μεταφορών οι κόμβοι του δικτύου είναι σαφώς περισσότεροι από τους αναγκαίους. Αυτό, σε συνδυασμό με τον ενιαίο τρόπο περιγραφής της πληροφορίας μέσω ενός συνόλου επιπέδων (layers) ενός συστήματος GIS, καθιστά τα συστήματα σχεδιασμού μεταφορών λιγότερο αποδοτικά και σαφώς πιο πολύπλοκα.

Επισημάναμε την ανάγκη χρήσης της Τελεολογικής Δομής για την οργάνωση της συνολικής πληροφορίας, ώστε να αντιμετωπιστούν τα προβλήματα που προκύπτουν από το μεγάλο όγκο, την πολυπλοκότητα και την πολυμορφία των δεδομένων. Επί πλέον, παρουσιάστηκε μια καινοτομική προσέγγιση στην οργάνωση, αποθήκευση και τήρηση της γεωσυσχετισμένης πληροφορίας σε ένα ευρύτερο πληροφοριακό σύστημα. Η αρχιτεκτονική αυτή ονομάστηκε «Θέματα Υπεράνω Χαρτών». Με τη χρήση της νέας αυτής αρχιτεκτονικής ξεπερνάμε το έμμεσο όριο που θέτει η συμβατική αρχιτεκτονική όσον αφορά τον όγκο, την πολυμορφία και, κυρίως, την πολυθεματικότητα της πληροφορίας που αποθηκεύεται και τηρείται στο σύστημα.

Εισάγαμε την έννοια του ‘παράγωγου’ δικτύου δηλαδή, την «κατά ζήτηση» παραγωγή ενός υπο-δικτύου από ένα ευρύτερο το οποίο είναι ξεχωριστό για κάθε εφαρμογή και πλήρως καθορισμένο από τον χρήστη βάση κανόνων. Με τον τρόπο αυτό, υπάρχει σαφής αύξηση στην αποτελεσματικότητα και στην αποδοτικότητα οποιασδήποτε εργασίας πάνω στα δίκτυα μεταφορών.

Επίσης, παρουσιάσαμε ένα νέο πληρέστερο μοντέλο αναπαράστασης των δικτύων για την αποτελεσματική εργασία στα δίκτυα μεταφορών. Περιγράψαμε λεπτομερώς τις δομές του κόμβου και του συνδέσμου καθώς επίσης και την καινοτομική δομή γειτνίασης (Adjacency Dictionary). Το μοντέλο αυτό είναι προϊόν εργασίας και σκέψης των μελών του Εργαστηρίου Συστημάτων

Πολυθεματικής και Γεωσυσχετισμένης Πληροφορίας της Σχολής ΗΜΜΥ του ΕΜΠ.

Τέλος, προς απόδειξη των παραπάνω, κατασκευάσαμε μια σειρά εργαλείων επεξεργασίας της πληροφορίας, παρουσιάσαμε την δομή της βάσης δεδομένων για την αποδοτική αποθήκευση και ανάκληση της πληροφορίας και προχωρήσαμε σε πρακτικές εφαρμογές πάνω σε υπάρχοντα δίκτυα αποδεικνύοντας πως η επεξεργασία, αποθήκευση και απεικόνιση της γεωγραφικά συσχετισμένης πληροφορίας μπορεί να πραγματοποιηθεί μέσω σαφώς πιο αποδοτικών διαδικασιών.

Οι προβλεπόμενες κατευθύνσεις για περαιτέρω έρευνα και ανάπτυξη περιλαμβάνουν, μεταξύ άλλων:

- Επέκταση της βάσης δεδομένων με σκοπό την αποθήκευση περισσοτέρων δεδομένων. Ιδανικά προτείνεται η ερεύνα και ανάκτηση πραγματικής πληροφορίας για τα Ελληνικά και Ευρωπαϊκά μεταφορικά δίκτυα.
- Προσαρμογή της δομών του κόμβου και του συνδέσμου καθώς επίσης και της δομής γειτνίασης στην επεκταμένη βάση δεδομένων
- Χρήση των παραπάνω δομών για την κατασκευή σαφώς καθορισμένων διαδρομών (Routes)
- Ολοκλήρωση της προσπάθειας αποσύζευξης της συνέχειας συνδέσμων των δικτύων και της γεωγραφικής συνέχειας, κάτι που είναι αναγκαίο σε πολλές σύγχρονες εφαρμογές πολυτροπικών μεταφορών.
- Σύνθεση πολυτροπικών (multimodal) περιγραφών των δικτύων για τυπικές εφαρμογές πολυτροπικών και διατροπικών μεταφορών.
- Ανάλυση των αναγκών για τα δίκτυα μεταφορών και κατασκευή επιπλέον εργαλείων λογισμικού.
- Ενσωμάτωση την παραπάνω εργαλείων με τον περιβάλλον εργασίας χρήστη (UI) το οποίο είναι υπό κατασκευή από τα μέλη του Εργαστηρίου Συστημάτων Πολυθεματικής και Γεωσυσχετισμένης Πληροφορίας της Σχολής ΗΜΜΥ του ΕΜΠ.

Παραρτήματα

Κώδικας Python

Adjacency_Dictionary_From_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module takes node-centric table and a link-
centric table and creates
 an adjacency dictionary"""

# dbo.NODES ---> Columns ( FID ID VISIBILITY CLASS
JUNCTION)
# dbo.LINKS ---> Columns ( FID LinkID FromNodeID ToNodeID
StartCost TravCost TWOWAY StartCost2 )

import pyodbc
import csv
from prettytable import PrettyTable


class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class Conversion:

    def __init__(self, nodes, links, cnxn):
        self.nodes = nodes
        self.links = links
        self.cnxn = cnxn
        # Create two dictionaries that hold the
connections between the nodes
        self.list_of_dictionaries =
self.create_list_of_dictionaries(self.get_rows_of_table("
SELECT * FROM dbo."+self.links))
```

```
        # Create a dictionary with the personal data of
each node
        self.dictionary_of_personal_data =
self.create_dictionary_of_personal_data(self.get_rows_of_
table("SELECT * FROM dbo."+self.nodes))
        # Build the adjacency dictionary
        self.adjacency_dictionary =
self.build_dictionary(self.dictionary_of_personal_data,
self.list_of_dictionaries)
        # Print it in a pretty way
        self.print_dictionary(self.adjacency_dictionary)

    def get_rows_of_table(self, sql_string):
        """ This method gives us the desired rows of an
Sql table
        @Param Sql_String is the Sql string
        @Return are the desired rows """
        cursor = self.cnxn.cursor()
        cursor.execute(sql_string)
        rows = cursor.fetchall()
        return rows

    def string_to_list(self, input_string):
        """This method takes a string of coordinates and
puts them in a list
        as floats
        @Param input_string is the string(x y, x y, x y)
with the coordinates
        @Return is a list[[x, y], [x, y] ,[x ,y]] with
the same coordinates"""

        # Put all the letters in a list
        lst = []
        for letter in input_string:
            lst.append(letter)
        # Remove the unnecessary characters
        for i in range(0, 12):
            lst.pop(0)
        lst = lst[:-1]
        # Reform the String
        string_1 = "".join(lst)
        # Split the string on the spaces and put the
parts on a list
        lst_2 = string_1.split()
        # Putting the coordinates in pairs and then in a
list (For uniting links)
        # Create the list that we are going to use as a
base
        final_list = []
        # Create a temporary list
        temp_list = []
```

```
        i = 0
        # For every coordinate in the list
        for coord in lst_2:
            # Append the temporary list with the
coordinate
            temp_list.append(coord)
            # If the coordinate is in position 1,3,5,7...
            if i % 2 != 0:
                # Append the final list with the
temporary one
                final_list.append(temp_list)
                # And empty the temporary one
                temp_list = []
            i += 1
        # Create an empty list that we are going to store
the pairs of coordinates as floats
        final_list_2 = []
        # For every pair o coordinates in final list we
want to make a pair of float coordinates
        for part in final_list:
            # Create a temporary list
            temp_list_2 = []
            # For every 'string' coordinate
            for cord in part:
                # If the 'string' coordinate does not end
in comma
                if cord[-1] != ",":
                    # Append the list with the float of
the 'string' coordinate
                    temp_list_2.append(float(cord))
                # If the 'string' coordinate ends in
comma
                if cord[-1] == ",":
                    # Remove the comma
                    word = cord[:-1]
                    # Append the list with the the float
of the 'string' coordinate
                    temp_list_2.append(float(word))
                # Append the final list with the
temporary list
            final_list_2.append(temp_list_2)
        return final_list_2

    def create_list_of_dictionaries(self, rows):
        """ This method creates 2 dictionaries
        The first dictionary contains the ID of the nodes
where we can move to (from the Key node) and the required
        costs to do that.
        The key of the first dictionary will be a node
from the FromNodeID column.
        The value of the key for the first dictionary
```

```
will be a list of numerous 3 element sub-lists:
        FromNodeID--->[[LinkID, Visibility, ToNodeID,
StartCost, TravCost, Class, TwoWay, StartCost2, Geometry]
, ....repeat....]
        The second dictionary contains the ID of the
nodes where we can travel from, towards the Key node.
        The key of the second dictionary will be a node
from the ToNodeID column.
        The value of the key for the second dictionary
will be a list of nodes:
        ToNodeID--->[FromNodeID , ....repeat....]
        @Param rows is the rows of the LINKS table
        @Return is a list with the dictionaries """

        # Create empty Dictionaries
        temp_dict_0 = {}
        temp_dict_1 = {}

        rows_0 = self.get_rows_of_table("SELECT ID FROM
dbo."+self.nodes)
        # For every node
        for row in rows_0:
            # Create a key for both of the dictionaries
            temp_dict_0[row[0]] = []
            temp_dict_1[row[0]] = []
        # For each row of the of the LINKS table:
        for row in rows:
            # Building the Temp_Dict0:
            cursor = self.cnxn.cursor()
            # Get the coordinates of the link
            cursor.execute("SELECT GEOMETRY.STAsText()
FROM dbo."+self.links+" WHERE LinkID=?", (row[1],))
            row_0 = cursor.fetchone()
            string_geometry = row_0[0]
            # Put the coordinates in a list
            geometry_list =
self.string_to_list(string_geometry)
            cursor.close()
            # Building the temp_dict_0
            temp_dict_0[row[3]].append([row[1], row[2],
row[4], row[5], row[6], row[7], row[8], row[9],
geometry_list])
            # Building the Temp_Dict_1:
            temp_dict_1[row[4]].append(row[3])

        return [temp_dict_0, temp_dict_1]

    def create_dictionary_of_personal_data(self, rows):
        """ This method creates a dictionary for the
Nodes of the NODES table.
        The keys will be the ID of each node and the
```

```
value will be a 3 element list
        ID--->[VISIBILITY, CLASS, Geometry]
        @Param rows is the rows of the NODES table
        @Return is the dictionary """

        # Create an empty dictionary
        temp_dict = {}
        # For every row in the nodes table
        for row in rows:
            # Get the coordinate of the node
            cursor = self.cnxn.cursor()
            cursor.execute("SELECT GEOMETRY.Long,
GEOMETRY.Lat FROM "+self.nodes+" WHERE ID=?", (row[1],))
            row_0 = cursor.fetchone()
            cursor.close()
            # Put it in a list
            geometry = [row_0[1], row_0[0]]
            # Build the temp_dict
            temp_dict[row[1]] = [row[2], row[3],
geometry]

        return temp_dict

    def build_dictionary(self, node_data,
connections_dict):
        """This method builds the adjacency dictionary
        @Param node_data is the dictionary withe personal
information of each node
        @Param connections_dict is the list with the two
dictionaries that hold the connections between the
nodes"""

        # Create an empty dictionary
        final_dictionary = {}
        rows = self.get_rows_of_table("SELECT ID FROM
dbo."+self.nodes)
        # For every row of the nodes table
        for row in rows:
            # Build the dictionary
            final_dictionary[row[0]] =
[node_data[row[0]], [connections_dict[0][row[0]],
connections_dict[1][row[0]]]]

        return final_dictionary

    def print_dictionary(self, dict2):
        """This method takes a dictionary and prints it
in a pretty way
        @Param dict is the dictionary """
        i = 1
        for keys, values in dict2.items():
```

```
                print i
                k = PrettyTable([keys])
                k.add_row(["----->"])
                print k
                t = PrettyTable(['Visiblity', 'Class',
'Geometry'])
                t.add_row(values[0])
                f = PrettyTable(['LinkID', 'Visibility',
'ToNodeID', 'StartCost', 'TravCost', 'Class', 'TwoWay',
'StartCost2', 'Geometry'])
                for part in values[1][0]:
                    f.add_row(part)
                c = PrettyTable(['FromNodeID'])
                for part in values[1][1]:
                    c.add_row([part])
                print t
                print f
                print c
                print ""
                print ""
                i += 1


def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    # Create the adjacency Dictionary
    lis = Conversion(connection.nodes, connection.links,
connection.cnxn)


if __name__ == '__main__':
    main()
```

Adjacency_Dictionary_To_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module inserts the data of the adjacent
dictionary to a new LINKS TABLE"""
from Adjacency_Dictionary_From_Database import
Conversion, Connect
from Finds_Nodes_To_Remove_From_Database import
NodesToRemove
from Removing_Nodes_From_Dictionary import RemoveNodes
from Delete_links_In_Dictionary_By_Class import
DeleteLinksInDictByClass
```

```
class DictionaryToDatabase:

    def __init__(self, cnxn, dict, name_of_table,
name_of_table_2):
        self.cnxn = cnxn
        self.dict = dict
        # The name of the new table
        self.name_of_table_2 = name_of_table_2
        self.name_of_table = name_of_table
        # Create the new Links table
        self.create_tables(self.name_of_table,
self.name_of_table_2)
        # Insert the data
        self.insert_data(self.name_of_table,
self.name_of_table_2, self.dict)

    def list_to_string(self, lst):
            """This method takes a list of float
coordinates and forms the string to use as input to the
database
            @Param lst if the list of coordinates
            @Return is the desired string"""

            # Set i to 1 in order to be able to dictate
the first coordinate
            i = 1
            # Create an empty string
            string = ""
            # For every node in the list
            for node in lst:
                # If it is the first node
                if i == 1:
                    string = string+str(node[0])+"
"+str(node[1])
                # For every other node
                else:
                    string = string+", "+str(node[0])+"
"+str(node[1])
                i += 1

            return string

    def create_tables(self, name, name_2):
        """This method creates a new links table in our
database
        @Param name is the name of the new table"""
        cursor = self.cnxn.cursor()
        cursor.execute("CREATE TABLE "+name_2+" (FID int,
ID int, VISIBILITY int, CLASS int, LinksExiting int,
```

```
LinksEntering int, GEOMETRY geography)")

        cursor.execute("CREATE TABLE "+name+" (FID int,
LinkID int, Visibility int, FromNodeID int, ToNodeID int,
StartCost int, TravCost int, CLASS int, TwoWay int,
StartCost2 int, GEOMETRY geography)")
        cursor.commit()

    def insert_data(self, name, name_2, dict):
        """This method inserts tha data of the adjacent
dictionary to the new table
        @Param name is the name of the table
        @Param dict is the adjacent dictionary"""

        cursor = self.cnxn.cursor()
        counter = 0
        counter_2 = 0
        # For every key and value of the dictionary
        for key, values in dict.items():
            cursor.execute("INSERT INTO "+name_2+" (FID,
ID, VISIBILITY, CLASS) VALUES (?, ?, ?, ?)", (counter_2,
key, values[0][0], values[0][1]), )
            cursor.commit()
            cursor.execute("UPDATE "+name_2+" SET
GEOMETRY = geography::Point("+str(values[0][2][0])+",
"+str(values[0][2][1])+", 4326) WHERE FID=?",
(counter_2,))
            cursor.commit()
            counter_2 += 1
            # For every link that exit a certain node
            for link in values[1][0]:
                # Convert the geometry to string
                string = self.list_to_string(link[8])
                # Insert the data
                cursor.execute("INSERT INTO "+name+"
(FID, LinkID, Visibility, FromNodeID, ToNodeID,
StartCost, TravCost, CLASS, TwoWay, StartCost2)
VALUES(?,?,?,?,?,?,?,?,?,?)", (counter, link[0], link[1],
key, link[2], link[3], link[4], link[5], link[6],
link[7]), )
                cursor.commit()
                cursor.execute("UPDATE "+name+" SET
GEOMETRY=geography::STGeomFromText('LINESTRING("+string+"
)', 4326 ) WHERE FID=?", (counter), )
                cursor.commit()
                counter += 1


def main():
    """ main class here."""
    # Create a connection to the database and the desired
```

```
tables
    connection = Connect("NODES", "LINKS")
    # Create the adjacency Dictionary
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    adjacency_dictionary = first.adjacency_dictionary
    # Find the nodes we need to remove
    second = NodesToRemove(connection.nodes,
connection.links, connection.cnxn)
    nodes_to_remove = second.final_nodes_to_remove
    # Remove the nodes and adjust the dictionary
    third = RemoveNodes(adjacency_dictionary,
nodes_to_remove)
    # Insert the fixed links to a new table in our
database

    fourth = DictionaryToDatabase(connection.cnxn,
third.fixed_dictionary, "Fixed_links","Fixed_nodes")


if __name__ == '__main__':
    main()
```

Adjacency_Dictionary_To_Shapefile.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-

from Adjacency_Dictionary_From_Database import
Conversion, Connect
from Delete_links_In_Dictionary_By_Class import
DeleteLinksInDictByClass
import arcpy
import os

"""This module takes an adjacency dictionary and creates
a node shapefile and a links shapefile"""


class DictToShapefile:

    def __init__(self, nodes, links, dict):
        self.nodes = nodes
        self.links = links
        self.dict = dict
        # Create the two shapefiles
        self.create_shapefiles(self.nodes, self.links)
        # Add the required fields
        self.add_fields(self.nodes, self.links)
        # Add the data
```

```
        self.add_data(self.nodes, self.links, self.dict)

    def create_shapefiles(self, nds, lks):
        """This method creates two shapefiles. A
shapefile for points and a shapefile for polylines
        @Param nds is the name of the points shapefile
        @Param lks is the name of the polylines
shapefile"""

        # Create the links shapefile


arcpy.CreateFeatureclass_management(r"C:\Users\vriz\Deskt
op\ARGIS",  lks, "POLYLINE",
spatial_reference=arcpy.Describe(r"C:\Users\vriz\Desktop\
ARGIS\Diploma2.shp").spatialReference)
        # Create the nodes shapefile

arcpy.CreateFeatureclass_management(r"C:\Users\vriz\Deskt
op\ARGIS", nds, "POINT",
spatial_reference=arcpy.Describe(r"C:\Users\vriz\Desktop\
ARGIS\RailNode.shp").spatialReference)

    def add_fields(self, nds, lks):
        """This method add the required fields to our
shapefiles that we created before
        @Param nds is the name of the points shapefile
        @Param lks is the name of the polylines
shapefile"""

        # Set our arcpy environment
        arcpy.env.workspace =
r"C:\Users\vriz\Desktop\ARGIS"
        # Add the fields in the nodes shapefile
        arcpy.AddField_management(nds, 'NodeID',
'DOUBLE')
        arcpy.AddField_management(nds, 'Visibility',
'DOUBLE')
        arcpy.AddField_management(nds, 'Class', 'DOUBLE')
        arcpy.DeleteField_management(nds, "Id")
        # Add the fields in the links shapefile
        arcpy.AddField_management(lks, 'LinkID',
'DOUBLE')
        arcpy.AddField_management(lks, 'Visibility',
'DOUBLE')
        arcpy.AddField_management(lks, 'FromNodeID',
'DOUBLE')
        arcpy.AddField_management(lks, 'ToNodeID',
'DOUBLE')
        arcpy.AddField_management(lks, 'StartCost',
'DOUBLE')
```

```
        arcpy.AddField_management(lks, 'TravCost',
'DOUBLE')
        arcpy.AddField_management(lks, 'Class', 'DOUBLE')
        arcpy.AddField_management(lks, 'TwoWay',
'DOUBLE')
        arcpy.AddField_management(lks, 'StartCst_2',
'DOUBLE')
        arcpy.DeleteField_management(lks, "Id")

    def add_data(self, nds, lks, dict):
        """This method adds the data to our shapefiles
        @Param nds is the name of the points shapefile
        @Param lks is the name of the polylines shapefile
        @Param dict is the adjacency dictionary"""

        # Links feature Class
        fc = os.path.join(r"C:\Users\vriz\Desktop\ARGIS",
lks)
        # The fields
        fields = ['FID', "Shape@", 'LinkID',
'Visibility', 'FromNodeID', 'ToNodeID', 'StartCost',
'TravCost', 'Class', 'TwoWay', 'StartCst_2']
        # Nodes feature Class
        fc_2 =
os.path.join(r"C:\Users\vriz\Desktop\ARGIS", nds)
        # The fields
        fields_2 = ["Shape@", 'ID', 'Visibility',
'Class']

        # Cursor for the links shapefile
        rows = arcpy.InsertCursor(fc, fields)
        # Create an arcpy array that we are going to use
to input geometry data to our polylines
        array = arcpy.Array()
        # Cursor for the links shapefile
        rows_2 = arcpy.InsertCursor(fc_2, fields_2)

        # For every key and its values in our dictionary
        for key, values in dict.items():
            # Create a new row
            row_2 = rows_2.newRow()
            # Set the values
            row_2.setValue("NodeID", key)
            row_2.setValue('Visibility', values[0][0])
            row_2.setValue('Class', values[0][1])
            point = arcpy.Point(values[0][2][1],
values[0][2][0])
            row_2.setValue('Shape', point)
            # Insert the row
            rows_2.insertRow(row_2)
            # For every link in the dictionary
```

```
            for links in values[1][0]:
                # Create a new row
                row = rows.newRow()
                # Set the values
                row.setValue("LinkID", links[0])
                row.setValue('Visibility', links[1])
                row.setValue('FromNodeID', key)
                row.setValue('ToNodeID', links[2])
                row.setValue('StartCost', links[3])
                row.setValue('TravCost', links[4])
                row.setValue('Class', links[5])
                row.setValue('TwoWay', links[6])
                row.setValue('StartCst_2', links[7])
                # Set the Shape value for the polyline
                for cord in links[8]:
                    array.add(arcpy.Point(cord[0],
cord[1]))
                polyline = arcpy.Polyline(array)
                row.setValue('Shape', polyline)
                # Clear the array so that its empty for
the next polyline
                array.removeAll()
                # Insert the row
                rows.insertRow(row)


def main():


    connection = Connect("NN", "FF")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)

    arcgis = DictToShapefile("nodes.shp", "links.shp",
first.adjacency_dictionary)


if __name__ == '__main__':
    main()
```

Calculate_Power_Of_Nodes_In_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module calculates how many links exit a certain
node and how many links enter the same node, and informs
the
database"""

import pyodbc
```

```
class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class PowerOfNodes:

    def __init__(self, nodes, links, cnxn):
        self.nodes = nodes
        self.links = links
        self.cnxn = cnxn
        # Create two dictionaries that hold the
connections between the nodes
        self.list_of_dictionaries =
self.create_list_of_dictionaries(self.get_rows_of_table("
SELECT * FROM dbo."+self.links))
        # Calculate the power of the nodes

self.power_of_nodes_to_database(self.list_of_dictionaries
)

    def get_rows_of_table(self, sql_string):
        """ This method gives us the desired rows of an
Sql table
        @Param Sql_String is the Sql string
        @Return are the desired rows """
        cursor = self.cnxn.cursor()
        cursor.execute(sql_string)
        rows = cursor.fetchall()
        return rows

    def create_list_of_dictionaries(self, rows):
        """ This method creates 2 dictionaries
        The first dictionary contains the ID of the nodes
where we can move to (from the Key node) and the required
        costs to do that.
        The key of the first dictionary will be a node
from the FromNodeID column.
        The value of the key for the first dictionary
will be a list of numerous 3 element sub-lists:
        FromNodeID--->[ToNodeID , ....repeat....]
```

```
        The second dictionary contains the ID of the
nodes where we can travel from, towards the Key node.
        The key of the second dictionary will be a node
from the ToNodeID column.
        The value of the key for the second dictionary
will be a list of nodes:
        ToNodeID--->[FromNodeID , ....repeat....]
        @Param rows is the rows of the LINKS table
        @Return is a list with the dictionaries """

        # Create empty Dictionaries
        temp_dict_0 = {}
        temp_dict_1 = {}
        rows_0 = self.get_rows_of_table("SELECT * FROM
dbo."+self.nodes)
        # For every node
        for row in rows_0:
            # Create a key for both of the dictionaries
            temp_dict_0[row[1]] = []
            temp_dict_1[row[1]] = []
        # For each row of the rows:
        for row in rows:
            # Building the Temp_Dict0:
            # Append the list with the values for the
key: row[3] = FromNodeID
                temp_dict_0[row[3]].append(row[4])
            # Building the Temp_Dict_1:
            # Append the list with the values for the
key: row[4] = ToNodeID
                temp_dict_1[row[4]].append(row[3])

        return [temp_dict_0, temp_dict_1]

    def power_of_nodes_to_database(self, dict_list):
        """This method takes the dictionaries that hold
the connections between the nodes, calculates the power
of
        each node and informs the database
        @Param dict_list is the list with the 2
dictionaries"""

        # For every key and value of the dictionary that
holds the links that exit the key node
        for key, values in dict_list[0].items():
            # Inform the database (len(values) is the
power of the node)
            cursor = self.cnxn.cursor()
            cursor.execute("UPDATE NODES SET
LinksExiting=? WHERE ID=?", (len(values), key,))
            cursor.commit()
            cursor.close()
```

```
        # For every key and value of the dictionary that
holds the links that exit the key node
        for key, values in dict_list[1].items():
            # Inform the database (len(values) is the
power of the node)
            cursor = self.cnxn.cursor()
            cursor.execute("UPDATE NODES SET
LinksEntering=? WHERE ID=?", (len(values), key,))
            cursor.commit()
            cursor.close()


def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")

    lis = PowerOfNodes(connection.nodes,
connection.links, connection.cnxn)


if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Link_In_Database_By_Class.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
""" Takes an SQL table and changes the visibility of
links based on the users preferences """


from Adjacency_Dictionary_From_Database import Connect


class ChangeVisOfLink:

    def __init__(self, links, cnxn, cls):
        self.links = links
        self.cnxn = cnxn
        self.cls = int(cls)
        # Change the visibility of the links
        self.change_links_by_class(cls)

    def change_links_by_class(self, choice):
        """ This method changes the visibility of links
based on their class

        @Param choice is the class of links we want their
```

```
visibility to be changed"""
        cursor = self.cnxn.cursor()
        # Change the visibility of the links
        cursor.execute("UPDATE dbo."+self.links+" SET
Visibility=0 WHERE CLASS=?", (choice,))
        self.cnxn.commit()


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    # Change the visibility of the links
    ChangeVisOfLink(connection.links, connection.cnxn,
"1")

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Link_In_Database_By_ID.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
""" Takes an SQL table and changes the visibility of
links based on the users preferences """

from Adjacency_Dictionary_From_Database import Connect


class ChangeVisOfLinkByID:

    def __init__(self, links, cnxn, lst):
        self.links = links
        self.cnxn = cnxn
        self.list_to_change = lst
        # Change the visibility of the links
        self.change_links_by_id(self.list_to_change)

    def change_links_by_id(self, list_0):
        """This method takes a list of ids and changes
the visibility of those links
        @Param list_0 is the list with the ids"""

        # For each id in the list
        for link in list_0:
            self.cnxn.cursor().execute("UPDATE
dbo."+self.links+" SET Visibility=0 WHERE LinkID=?",
(link,))
            self.cnxn.commit()
```

```
def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    temp_list = []
    condition = True
    # As long as condition == True
    while condition:
        # Promt the user to give us the id of a link he
wants to change or to end the process
        temp_id = raw_input("Please type the id of the
link or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
    ChangeVisOfLinkByID(connection.links,
connection.cnxn, temp_list)

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Link_In_Dictionary_By_Class.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module changes the visibility of links in the
adjacency dictionary based on their class"""

from Adjacency_Dictionary_From_Database import
Conversion, Connect


class ChangeVisInDictByClass:

    def __init__(self, dict, cls):
        self.dict = dict
        self.cls = int(cls)
        # Change the visibility of the links
        self.change_visibility(self.dict, self.cls)

    def change_visibility(self, dictionary, cls):
        """This method changes the visibility of links
based on their class
```

```
        @Param dictionary is the adjacency dictionary
        @Param cls is the class of links we want theirs
visibility changed"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every link
            for link in values[1][0]:
                # If tha links class == cls
                if link[5] == cls:
                    # Set the visibility to zero
                    link[1] = 0


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    ChangeVisInDictByClass(first.adjacency_dictionary,
"2")


if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Link_In_Dictionary_By_ID.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module changes the visibility of links in the
adjacency dictionary based on their ID"""

from Adjacency_Dictionary_From_Database import
Conversion, Connect


class ChangeVisInDictByID:

    def __init__(self, dict, lst):
        self.dict = dict
        self.lst = lst
        # Change the visibility of the links
        self.change_visibility(self.dict, self.lst)

    def change_visibility(self, dictionary, lst):
        """This method changes the visibility of links
based on their ID
        @Param dictionary is the adjacency dictionary
        @Param lst is the list of links we want their
```

```
visibility changed"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every link
            for link in values[1][0]:
                # If tha link's ID is in the lst
                if link[0] in lst:
                    # Set the visibility to zero
                    link[1] = 0


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    temp_list = []
    condition = True
    # As long as condition == True
    while condition:
        # Promt the user to give us the id of a link he
wants to change or to end the process
        temp_id = raw_input("Please type the id of the
link or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
    ChangeVisInDictByID(first.adjacency_dictionary,
temp_list)

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Node_In_Database_By_Class.py

```
__author__ = 'vriz'
# !/usr/bin/env python
# -*-coding: utf-8-*-
"""This module takes node-SQL table and changes the
visibility of certain nodes
based on on the users preferences"""

import pyodbc
```

```
class Connect:
    """ Connects us to the database """

    def __init__(self, nodes, ):
        """ Connects us to the database and and assigns
the node table to the appropriate variable """
        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver = '{SQL
Server}', server = 'localhost', database =
'PracticeCopy1')
        self.nodes = nodes


class ChangeVisibilityByClass:
    """ Takes an SQL table creates an exact copy and then
changes the visibility of certain nodes based on on on
the
     users preferences """

    def __init__(self, nodes, cnxn, cls):
        self.nodes = nodes
        self.cnxn = cnxn
        self.cls = cls
        self.change_visibility_by_class(self.cls)

    def change_visibility_by_class(self, cls):
        """ This method changes the visibility of certain
nodes based on their class """

        self.cnxn.cursor().execute("UPDATE
dbo."+self.nodes+" SET VISIBILITY=? WHERE CLASS=?", (0,
cls,))
        self.cnxn.commit()


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES")
    # Changes the visibility of certain nodes
    ChangeVisibilityByClass(connection.nodes,
connection.cnxn, "2")

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Node_In_Database_By_Distance.py

```
__author__ = 'vriz'
# !/usr/bin/env python
```

```
#  -*-coding: utf-8-*-
"""This module takes node-centric table, a link-centric
table, a certain node id and a distance. Then changes the
visibility of the nodes that are in greater distance
(given distance) from the node (node id) and removes all
the
links that are associated with those nodes"""


import pyodbc
import math


class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database where the node-
table and link-table are located"""

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class ChangeVisibilityBasedOnDistance:

    def __init__(self, nodes, links, cnxn, node_id,
distance):
        self.nodes = nodes
        self.links = links
        self.cnxn = cnxn
        self.node_id = node_id
        self.distance = distance
        # Get the list of the distant nodes
        self.distant_nodes =
self.get_the_list_of_distant_nodes(self.node_id,
self.distance)

        # Change their visibility
        self.change_visibility(self.distant_nodes)

        # Get the nodes that are neighboring the main
node
        self.nodes_not_to_delete =
self.get_list_of_nodes_we_must_not_delete()

        # Remove the links
        self.remove_links(self.distant_nodes,
self.nodes_not_to_delete)
```

```
    def calculate_ds(self, point_1, point_2):
        """Calculate the great circle distance between
two points
        on the earth (specified in decimal degrees
        @Param point_1 is the first point (x, y)
        @Param point_2 is the second point (x, y))
        @Return is the distance between the two nodes in
Kilometers"""

        latitude_1 = point_1[0]
        longitude_1 = point_1[1]
        latitude_2 = point_2[0]
        longitude_2 = point_2[1]

        # convert decimal degrees to radians
        lon1, lat1, lon2, lat2 = map(math.radians,
[longitude_1, latitude_1, longitude_2, latitude_2])
        # haversine formula
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = math.sin(dlat/2)**2 + math.cos(lat1) *
math.cos(lat2) * math.sin(dlon/2)**2
        c = 2 * math.asin(math.sqrt(a))
        km = 6367 * c
        return km

    def get_the_list_of_distant_nodes(self, node_id,
distance):
        """This method finds the nodes of the distant
nodes
        @Param node_id is our main node
        @Param distance is the distance in which we want
our nodes to be located
        @Return is a list with the distant nodes"""

        # Insert a cursor
        cursor_1 = self.cnxn.cursor()
        # Select the row that holds the information of
our node
        cursor_1.execute("SELECT GEOMETRY.Long,
GEOMETRY.Lat FROM "+self.nodes+" WHERE ID=?", (node_id,))
        row = cursor_1.fetchone()
        primary_node_coord = [row[1], row[0]]
        # Close the cursor
        cursor_1.close()
        # Insert a second cursor
        cursor_2 = self.cnxn.cursor()
        # Select all the rows of the nodes table
        cursor_2.execute("SELECT ID FROM
dbo."+self.nodes)
```

```
        rows = cursor_2.fetchall()
        # Create an empty list
        nodes_list = []
        # For every row in rows
        for row in rows:
            # If the node id of the row is not the same
as the node_id (node id giver by the user)
            if row[0]!= node_id:
                # Set the coordinates of the node as the
secondary coordinates
                cursor_3 = self.cnxn.cursor()
                cursor_3.execute("SELECT GEOMETRY.Long,
GEOMETRY.Lat FROM "+self.nodes+" WHERE ID=?", (row[0],))
                rows = cursor_3.fetchone()
                secondary_node_coord = [rows[1], rows[0]]
                # If the distance between the two nodes
is greater than the distance given by the user
                if self.calculate_ds(primary_node_coord,
secondary_node_coord) >= distance:
                    # Append the nodes list
                    nodes_list.append(row.ID)
                cursor_3.close()
        # Close the cursor
        cursor_2.close()
        return nodes_list

    def get_list_of_nodes_we_must_not_delete(self):
        """This method finds the node that are neighbours
to our main node
         @Return is the list with those nodes"""

        cursor_5 = self.cnxn.cursor()
        # Get the all the rows tha begin or end with our
main node
        cursor_5.execute("SELECT * FROM
dbo."+self.links+" WHERE (FromNodeID=?) OR (ToNodeID=?)",
(self.node_id, self.node_id), )
        rows = cursor_5.fetchall()
        # Create an empty list
        nodes_list = []
        # For every row
        for row in rows:
            # if the FromNodeID is our main node
            if row[2] != self.node_id:
                # Then append the list with the ToNodeID
                nodes_list.append(row[2])
            # if the ToNodeID is our main node
            if row[3] != self.node_id:
                # Then append the list with the
FromNodeID
                nodes_list.append(row[3])
```

```
        # Return the list
        return nodes_list

    def change_visibility(self, nodes):
        """This method changes the visibility of the
distant nodes
        @Param nodes is the list with the distant
nodes"""

        # Insert a cursor
        cursor_3 = self.cnxn.cursor()
        # For every node is the distant nodes list
        for node in nodes:
            # Set the visibility to 2
            cursor_3.execute("UPDATE dbo."+self.nodes+"
SET VISIBILITY=2 WHERE ID=?", (node,))
            cursor_3.commit()
            # Close the cursor
        cursor_3.close()

    def remove_links(self, nodes, nodes_2):
        """This method removes all the links associated
with the distant nodes
        except the ones that connect to our main node
        @Param nodes is the distant nodes list"""

        # Insert a cursor
        nodes_3 = [item for item in nodes if item not in
nodes_2]
        cursor_4 = self.cnxn.cursor()
        # For every node is the distant nodes list
        for node in nodes_3:
            # Remove the links that have the node as
starting point or ending point
            cursor_4.execute("DELETE FROM
dbo."+self.links+" WHERE (FromNodeID=?) OR (ToNodeID=?)",
(node, node))
            cursor_4.commit()
        for node in nodes_2:
            cursor_4.execute("SELECT * FROM
dbo."+self.links+" WHERE (FromNodeID=?) OR (ToNodeID=?)",
(node, node))
            rows = cursor_4.fetchall()
            for row in rows:
                if row[2] != self.node_id and row[3] !=
self.node_id:
                    cursor_4.execute("DELETE FROM
dbo."+self.links+" WHERE LinkID=?", (row[1],))
                    cursor_4.commit()
        # Close the cursor
        cursor_4.close()
```

```
def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES","LINKS" )
    lis =
ChangeVisibilityBasedOnDistance(connection.nodes,
connection.links, connection.cnxn, 104377, 50)
    print lis.distant_nodes
    print lis.nodes_not_to_delete

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Node_In_Database_By_ID.py

```
__author__ = 'vriz'
# !/usr/bin/env python
# -*-coding: utf-8-*-
"""This module takes node-SQL table and changes the
visibility of certain nodes
based on on the users preferences"""

import pyodbc


class Connect:
    """ Connects us to the database """

    def __init__(self, nodes, ):
        """ Connects us to the database and and assigns
the node table to the appropriate variable """
        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver = '{SQL
Server}', server = 'localhost', database =
'PracticeCopy1')
        self.nodes = nodes


class ChangeVisibilityByID:
    """ Takes an SQL table creates an exact copy and then
changes the visibility of certain nodes based on on on
the
    users preferences """

    def __init__(self, nodes, cnxn, lst):
        self.nodes = nodes
        self.cnxn = cnxn
        self.lst = lst
```

```
        # Change the visibility of the nodes
        self.change_visibility_by_id(self.lst)



    def change_visibility_by_id(self, list_0):
        """This method takes a list of ids and sets the
visibility of those nodes to zero
        @Param list_0 is the list with the ids"""

        # For each id in the list
        for node in list_0:
            self.cnxn.cursor().execute("UPDATE
dbo."+self.nodes+" SET VISIBILITY=? WHERE ID=?", (0,
node,))
            self.cnxn.commit()


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES")
    temp_list = []
    condition = True
    # As long as condition == True
    while condition:
        # Promt the user to give us the id of the nodes
he wants their visibility to be changed or to end the
process
        temp_id = raw_input("Please type the id of the
node or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
    # Changes the visibility of certain nodes
    ChangeVisibilityByID(connection.nodes,
connection.cnxn, temp_list)

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Node_In_Dictionary_By_Class

```
__author__ = 'vriz'
# !/usr/bin/env python
```

```
#  -*-coding: utf-8-*-
"""This module changes the visibility of Nodes in the
adjacency dictionary based on their class"""

from Adjacency_Dictionary_From_Database import
Conversion, Connect


class ChangeVisOfNodeInDictByClass:

    def __init__(self, dict, cls):
        self.dict = dict
        self.cls = int(cls)
        # Change the visibility of the links
        self.change_visibility(self.dict, self.cls)

    def change_visibility(self, dictionary, cls):
        """This method changes the visibility of Nodes
based on their class
        @Param dictionary is the adjacency dictionary
        @Param cls is the class of Nodes we want theirs
visibility changed"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every Node
            if values[0][1] == cls:
                values[0][0] = 0



def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)

ChangeVisOfNodeInDictByClass(first.adjacency_dictionary,
"2")
    print first.adjacency_dictionary[104376]

if __name__ == '__main__':
    main()
```

Change_Visibility_Of_Node_In_Dictionary_By_ID

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module changes the visibility of Nodes in the
adjacency dictionary based on their ID"""
```

```
from Adjacency_Dictionary_From_Database import
Conversion, Connect


class ChangeVisOfNodesInDictByID:

    def __init__(self, dict, lst):
        self.dict = dict
        self.lst = lst
        # Change the visibility of the links
        self.change_visibility(self.dict, self.lst)

    def change_visibility(self, dictionary, lst):
        """This method changes the visibility of Nodes
based on their ID
        @Param dictionary is the adjacency dictionary
        @Param lst is the list of Nodes we want their
visibility changed"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every node that is in the list of node
we want their visibility changed
            if key in lst:
                # Set the visibility of the node to zero
                values[0][0] = 0


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    temp_list = []
    condition = True
    # As long as condition == True
    while condition:
        # Promt the user to give us the id of a link he
wants to change or to end the process
        temp_id = raw_input("Please type the id of the
link or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
```

```
ChangeVisOfNodesInDictByID(first.adjacency_dictionary,
temp_list)

if __name__ == '__main__':
    main()
```

Check_For_Connectivity_Of_Route_From_Dictionary.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module checks if the links given connect to each
other"""

from Adjacency_Dictionary_From_Database import
Conversion, Connect


class CheckForContinuation:

    def __init__(self, diction, lst):
        # The adjacency dictionary
        self.dict = diction
        # The list with the links
        self.lst = lst
        # Find the links in the dictionary
        self.list_of_links = self.find_links(self.dict,
self.lst)
        # Check them
        self.check =
self.check_for_connectivity(self.list_of_links)

    def find_links(self, diction, lst):
        """This method finds the links in the adjacency
dictionary based on the list of ids given
        @Param diction if the adjacency dictionary
        @Param lst is the list with the ids of the links
        @Return is a list with all the data of each
link"""

        # Create an empty list
        list_to_return = []
        # For every key and value in the adjacency
dictionary
        for key, values in diction.items():
            # For every link in the links exiting the
node
            for link in values[1][0]:
                # If the id of the link is in the list of
the list
```

```
                if link[0] in lst:
                    # Put the data of the link in the
list
                    list_to_return.append(link)

        return list_to_return

    def check_for_connectivity(self, lst):
        """This method checks if the links given connect
to each other
        @Param lst is the list with the all the data for
each link
        @Return is True or False"""

        # Create an empty list
        star_and_end_list = []
        # For every link in the list
        for part in lst:
            # Put the starting and ending point of each
link in the list
            star_and_end_list.append(part[8][0])
            star_and_end_list.append(part[8][-1])
        # Set a counter to zero
        count = 0
        # For every item in the list
        for i in range(0, len(star_and_end_list)):
            # Store the coordinate
            coord = star_and_end_list[i]
            # If the coordinate appears only one time in
the list
            if star_and_end_list.count(coord) == 1:
                # Add one to the counter
                count += 1
        # If there only two coordinates that appear one
time in the list
        if count == 2:
            # Retrun True
            return True
        else:
            # Return False
            return False


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    # Create the adjacency Dictionary
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    fifth =
```

```
CheckForContinuation(first.adjacency_dictionary,
[1005612, 1005609, 1005618, 1005615, 1005611, 1037506])

    print fifth.check


if __name__ == '__main__':
    main()
```

Copy_Sql_Table.py

```
__author__ = 'vriz'
# !/usr/bin/env python
# -*-coding: utf-8-*-
"""This module takes the name of an SQL table and creates
an exact copy of it"""

import pyodbc


class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class CopyTable:
    """Creates a copy of the desired table"""

    def __init__(self, nodes, cnxn, new_nodes):

        self.nodes = nodes
        self.cnxn = cnxn
        self.new_nodes = new_nodes
        self.cursor = self.cnxn.cursor()
        # Get the column names
        self.titles = self.get_column_names()
        # Create a copy of our table
        self.create_sql_table(self.nodes, self.new_nodes,
self.titles)

    def get_column_names(self):
        """This method returns a list that holds the
names of the columns of our table"""
        self.cursor.execute("SELECT * FROM "+self.nodes)
```

```
        return [i[0] for i in self.cursor.description]

    def create_sql_table(self, table, new_table, titles):
            """ This method creates an exact copy of a
SQL nodes-table
            @Param table is the table that is going to be
copied
            @Param new_table is the new table"""

            # Create an empty string
            sql_string = " "
            # Create an empty list
            type_list = []
            # For every column in our table
            for row in
self.cursor.columns(table='LINKS'):
                # Append the list with the variable type
of each column
                type_list.append(row.type_name)
            # Set a counter to zero
            count = 0
            for title in titles:
                # Set the correct SQL string for
execution
                sql_string = sql_string+" "+title+"
"+type_list[count]+","
                count += 1
            # Remove the last comma
            sql_string = sql_string[:-1]
            # Create the new table
            self.cursor.execute("CREATE TABLE
dbo."+new_table+" ("+sql_string+" )")
            # Inform the database so that we can see the
changes in the management studio
            self.cnxn.commit()
            # Select all the columns
            self.cursor.execute("SELECT * FROM
dbo."+table+" ORDER BY FID ASC")
            # Get all the rows of the table
            rows = self.cursor.fetchall()
            # For every row:
            # Create an empty string
            sql_string_2 = " "
            # For every row:
            for part in rows:
                # For every column of the table
                for i in range(0, len(self.titles)):
                    # Set the correct SQL string for the
values
                    sql_string_2 = sql_string_2 + "
"+str(part[i])+" ,"
```

```
                # Remove the last comma
                sql_string_2 = sql_string_2[:-1]
                print sql_string_2
                # Pass the values to the new table
                self.cursor.execute("INSERT INTO
dbo."+new_table+" VALUES("+sql_string_2+")")
                sql_string_2 = " "
                # Inform the database
                self.cnxn.commit()


def main():
    connection = Connect("NODES", "LINKS")
    CopyTable(connection.links, connection.cnxn,
"NEW_LINKS_2")

if __name__ == '__main__':
    main()
```

Create_Route_In_Database.py

```
__author__ = 'vriz'

import pyodbc
import math
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module takes list of links, creates a route made
of those links and an SQL table with the starting node,
ending node, length and geometry of the route"""

class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class CreateRoute:
    """Create the route based on the links given in a
list"""

    def __init__(self, nodes, links, cnxn, route_links):

        self.cnxn = cnxn
```

```
        self.nodes = nodes
        self.links = links
        self.route_links = route_links
        # Create the route
        self.route = self.make_route(self.route_links)
        # Calculate the distance
        self.route_distance =
self.calculate_length(self.route)
        # Find the starting point and ending node of the
route
        self.start_and_end =
self.start_and_end_node(self.route_links)
        # Insert the route to an SQL table

self.insert_route_to_database(self.route_distance,
self.start_and_end)

    def calculate_ds(self, point_1, point_2):
        """This method calculates the distance between
two points
        @Param point_1 is the first point
        @Param point_2 is the second point
        @Return is the distance in Km"""

        latitude_1 = point_1[0]
        longitude_1 = point_1[1]
        latitude_2 = point_2[0]
        longitude_2 = point_2[1]
        # convert decimal degrees to radians
        lon1, lat1, lon2, lat2 = map(math.radians,
[longitude_1, latitude_1, longitude_2, latitude_2])
        # haversine formula
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = math.sin(dlat/2)**2 + math.cos(lat1) *
math.cos(lat2) * math.sin(dlon/2)**2
        c = 2 * math.asin(math.sqrt(a))
        km = 6367 * c
        return km

    def calculate_min(self, distance_list):
        """This method takes a list and returns the
element with tha lowest value
        @Param distance_list is a list of distances
        @Return is the minimum of those distances"""

        # Set the first element as minimum
        min = distance_list[0]
        # For each remaining element of the list
        for i in range(1, len(distance_list)):
            # If the distance is smaller than the minimum
```

```
one
            if distance_list[i] < min:
                # Set that distance as minimum
                min = distance_list[i]

        return min

    def calculate_length(self, coordinates_list):
        """This method calculates the length of a
polyline
        @Param coordinates_list is the list that holds
the coordinates of the polyline
        @Return is the length of the polyline"""

        xy = coordinates_list
        # Set length to zero
        link_length = 0
        # For every coordinate of the polyline except the
last one
        for i in range(0, len(coordinates_list)-1):
            # Calculate the ds between a point and its
next one
            ds = self.calculate_ds(xy[i], xy[i+1])
            # Add it to the length
            link_length += ds
        return link_length

    def string_to_list(self, input_string):
        """This method takes a string of coordinates and
puts them in a list
        as floats
        @Param input_string is the string(x y, x y, x y)
with the coordinates
        @Return is a list[[x, y], [x, y] ,[x ,y]] with
the same coordinates"""

        # Put all the letters in a list
        lst = []
        for letter in input_string:
            lst.append(letter)
        # Remove the unnecessary characters
        for i in range(0, 12):
            lst.pop(0)
        lst = lst[:-1]
        # Reform the String
        string_1 = "".join(lst)
        # Split the string on the spaces and put the
parts on a list
        lst_2 = string_1.split()
        # Putting the coordinates in pairs and then in a
list (For uniting links)
```

```
        # Create the list that we are going to use as a
base
        final_list = []
        # Create a temporary list
        temp_list = []
        i = 0
        # For every coordinate in the list
        for coord in lst_2:
            # Append the temporary list with the
coordinate
            temp_list.append(coord)
            # If the coordinate is in position 1,3,5,7...
            if i % 2 != 0:
                # Append the final list with the
temporary one
                final_list.append(temp_list)
                # And empty the temporary one
                temp_list = []
            i += 1
        # Create an empty list that we are going to store
the pairs of coordinates as floats
        final_list_2 = []
        # For every pair o coordinates in final list we
want to make a pair of float coordinates
        for part in final_list:
            # Create a temporary list
            temp_list_2 = []
            # For every 'string' coordinate
            for cord in part:
                # If the 'string' coordinate does not end
in comma
                if cord[-1] != ",":
                    # Append the list with the float of
the 'string' coordinate
                    temp_list_2.append(float(cord))
                # If the 'string' coordinate ends in
comma
                if cord[-1] == ",":
                    # Remove the comma
                    word = cord[:-1]
                    # Append the list with the the float
of the 'string' coordinate
                    temp_list_2.append(float(word))
                # Append the final list with the
temporary list
            final_list_2.append(temp_list_2)
        return final_list_2

    def list_to_string(self, lst):
        """This method takes a list of float coordinates
and forms the string to use as input to the database
```

```
        @Param lst if the list of coordinates
        @Return is the desired string"""

        # Set i to 1 in order to be able to dictate the
first coordinate
        i = 1
        # Create an empty string
        string = ""
        # For every node in the list
        for node in lst:
            # If it is the first node
            if i == 1:
                string = string+str(node[0])+"
"+str(node[1])
            # For every other node
            else:
                string = string+", "+str(node[0])+"
"+str(node[1])
            i += 1

        return string

    def make_route(self, id_list):
        """This method takes a list of links and creates
a route
        @Param id_list is a list containing the id of the
links that will form the route
        @Return is a list that contains the coordinates
of the route"""

        # Create a cursor
        cursor = self.cnxn.cursor()
        # Get the coordinates of the first link in the
list
        cursor.execute("SELECT GEOMETRY.STAsText() FROM
dbo.LINKS WHERE LinkID=?", (id_list[0],))
        row = cursor.fetchone()
        string_geometry = row[0]
        # Get the coordinates of the first link convert
them to float coordinates and set the list of those
coordinates
        # as the route
        route = self.string_to_list(string_geometry)
        # For the rest links in the id_list
        for i in range(1, len(id_list)):
            #Create a cursor
            cursor_1 = self.cnxn.cursor()
            # Get the coordinates of the link
            cursor_1.execute("SELECT GEOMETRY.STAsText()
FROM dbo.LINKS WHERE LinkID=?", (id_list[i],))
            row = cursor_1.fetchone()
```

```
            string_geometry = row[0]
            # Get the coordinates of the link convert
them to float coordinates and set the list of those
coordinates
            # as the list_geometry (the link that we want
to add to our route)
            list_geometry =
self.string_to_list(string_geometry)
            # If the first coordinate of the route is the
same as the last coordinate of the link we want to add
            if route[0] == list_geometry[-1]:
                # Remove the last coordinate from the
link
                list_geometry = list_geometry[:-1]
                # Add the link to the route
                route = list_geometry + route
            # If the first coordinate of the route is the
same as the first coordinate of the link we want to add
            elif route[0] == list_geometry[0]:
                # Remove the first element of the link
                list_geometry = list_geometry[1:]
                # Reverse the link
                list_geometry.reverse()
                # Add the link to the route
                route = list_geometry + route
            # If the last coordinate of the route is the
same as the first coordinate of the link we want to add
            elif route[-1] == list_geometry[0]:
                # Remove the first coordinate from the
link
                list_geometry = list_geometry[1:]
                # Add the link to the route
                route = route + list_geometry
            # If the last coordinate of the route is the
same as the last coordinate of the link we want to add
            elif route[-1] == list_geometry[-1]:
                # Remove the last coordinate from the
link
                list_geometry = list_geometry[:-1]
                # Reverse the link
                list_geometry.reverse()
                # Add the link to the route
                route = route + list_geometry
            # If nothing of the above is happening it
means that at least two of the links given in the id_list
            #  do not connect with each other. In that
case:
            else:
                # First coordinate the route
                route_start = route[0]
                # Last coordinate of the route
```

```
                route_end = route[-1]
                # First coordinate of the link we want to
add to the route
                link_to_add_start = list_geometry[0]
                # Last coordinate of the link we want to
add to the route
                link_to_add_end = list_geometry[-1]
                # Calculate the distance of each possible
combination between the start and and of the route and
start
                # and end of the link we want to add to
the route
                distance_1 =
self.calculate_ds(route_start, link_to_add_start)
                distance_2 =
self.calculate_ds(route_start, link_to_add_end)
                distance_3 = self.calculate_ds(route_end,
link_to_add_start)
                distance_4 = self.calculate_ds(route_end,
link_to_add_end)
                # Find the minimum of the those 4
distances
                min = self.calculate_min([distance_1,
distance_2, distance_3, distance_4])
                # If the minimum distance is the one
between the start of the route and the start of the link
                #  we want to add
                if min == distance_1:
                    # Reverse the link we want to add
                    list_geometry.reverse()
                    # Add the link to the route
                    route = list_geometry + route
                # If the minimum distance is the one
between the start of the route and the end of the link
                #  we want to add
                elif min == distance_2:
                    # Add the link to the route
                    route = list_geometry + route
                # If the minimum distance is the one
between the end of the route and the start of the link
                #  we want to add
                elif min == distance_3:
                    # Add the link to the route
                    route = route + list_geometry
                # If the minimum distance is the one
between the end of the route and the end of the link we
want to add
                else:
                    # Reverse the link we want to add
                    list_geometry.reverse()
                    # Add the link to the route
```

```
                    route = route + list_geometry

        return route

    def start_and_end_node(self, id_list):
        """This method finds the id of the starting and
ending node of our route
        @Param id_list is a list containing the id of the
links that will form the route
        @Return is a list with the starting and ending
node"""

        # Create an empty list that we are going to put
all the possible nodes that could be the ones we are
looking for
        possible_nodes = []
        # Create a cursor
        cursor = self.cnxn.cursor()
        # The nodes we are looking are located in the
first link of the id_list of the last link of the id_list

        # Append the possible nodes list with the
FromNodeID and the ToNodeID of the first link
        cursor.execute("SELECT LinkID, FromNodeID,
ToNodeID FROM LINKS WHERE LinkID = ?", (id_list[0]))
        row = cursor.fetchone()
        possible_nodes.append(row[1])
        possible_nodes.append(row[2])
        # Append the possible nodes list with the
FromNodeID and the ToNodeID of the last link
        cursor.execute("SELECT LinkID, FromNodeID,
ToNodeID FROM LINKS WHERE LinkID = ?", (id_list[-1]))
        row = cursor.fetchone()
        possible_nodes.append(row[1])
        possible_nodes.append(row[2])
        start = None
        end = None
        # For every node in the possible nodes list
        for node in possible_nodes:
            # Create a cursor
            cursor = self.cnxn.cursor()
            # Get the coordinates of the node
            cursor.execute("SELECT GEOMETRY.Long,
GEOMETRY.Lat FROM NODES WHERE ID=?", (node,))
            row = cursor.fetchone()
            node_coord = [row[1], row[0]]
            # if the coordinates match the first
coordinates of the route
            if node_coord == self.route[0]:
                # Set the node as the starting node
                start = node
```

```
            # if the coordinates match the last
coordinates of the route
            elif node_coord == self.route[-1]:
                # Set the node as the ending node
                end = node
            else:
                pass
        return [start, end]

    def insert_route_to_database(self, distance, nodes):
        """This method creates an SQL table and puts the
route in it
        @Param distance is the length of the route
        @Param nodes is the list with starting and ending
nodes"""

        # Create the table
        cursor = self.cnxn.cursor()
        cursor.execute("CREATE TABLE Route (FID int,
FromNodeID int, ToNodeID int, Length float, GEOMETRY
geography)")
        cursor.commit()
        # Insert the data in the table
        cursor.execute("INSERT INTO Route (FID,
FromNodeID, ToNodeID, Length) VALUES(1, ?, ?, ?)",
(nodes[0], nodes[1], distance,))
        cursor.commit()
        # Form the string to use as input for the
geometry of the route
        sql_string = self.list_to_string(self.route)
        cursor = self.cnxn.cursor()
        # Insert the geometry to the table
        cursor.execute("UPDATE Route SET
GEOMETRY=geography::STGeomFromText('LINESTRING("+sql_stri
ng+")', 4326 )")
        cursor.commit()


def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    # Create the adjacency Dictionary
    lis = CreateRoute(connection.nodes, connection.links,
connection.cnxn, [1005612, 1005609, 1005618, 1005615,
1005611, 1037506])
    print lis.route_links
    print lis.route
    print lis.route_distance
    print lis.start_and_end
```

```
if __name__ == '__main__':
    main()
```

Delete_Links_In_Database_By_Class.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
""" Takes an SQL table and removes the non desired links
based on the users preferences """

import pyodbc


class Connect:
    """Connects us to the database"""
    def __init__(self, links):
        """ Connects us to the database and and assigns
and link table to the appropriate variable """
        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.links = links


class DeleteLinksByClass:
    """ Takes an SQL table and removes the non desired
    links base on the users preferences"""

    def __init__(self, links, cnxn, cls):
        self.links = links
        self.cnxn = cnxn
        self.cls = cls
        # Delete the links
        self.delete_links_by_class(cls)
        # Reset the FID column
        self.fid_reset()

    def delete_links_by_class(self, choice):
        """ This method removes the non-desired links
        @Param choice is the class of links that the user
wants to remove"""

        # Delete the links whom class==choice
        self.cnxn.cursor().execute("DELETE FROM
dbo."+self.links+" WHERE CLASS=?", (choice,))
        self.cnxn.commit()

    def fid_reset(self):
        """This method resets the Fid column of a SQL
```

```
table after we have
        deleted some rows"""

        cursor = self.cnxn.cursor()
        # Select all the columns and sort them by their
FID with ascending order
        cursor.execute("SELECT * FROM dbo."+self.links+"
ORDER BY FID ASC")
        # Get all the rows of the table
        rows = cursor.fetchall()
        # Create an empty list in which we are going to
store the FIDs of the remaining links
        fid_list = []
        # Fill the list with the FIDs
        for part in rows:
            fid_list.append(part.FID)
        # Set a counter to 0
        counter = 0
        # For every element in the fid_list:
        for part in fid_list:
            # Reset the FIDs of every link
            self.cnxn.cursor().execute("UPDATE
dbo."+self.links+" SET FID=? WHERE FID=?", (counter+1,
part,))
            self.cnxn.commit()
            counter += 1


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("LINKS")
    # Create a new table and then delete the links
    DeleteLinksByClass(connection.links, connection.cnxn,
"1")

if __name__ == '__main__':
    main()
```

Delete_Links_In_Database_By_ID.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
""" Takes an SQL table and creates a copy of it and
removes the non desired links based on the users
preferences """

import pyodbc
```

```
class Connect:
    """Connects us to the database"""
    def __init__(self, links):
        """ Connects us to the database and and assigns
and link table to the appropriate variable """
        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.links = links


class DeleteLinksByID:
    """ Takes an SQL table and creates a copy of it and
removes the non desired
    links base on the users preferences"""

    def __init__(self, links, cnxn, lst):
        self.links = links
        self.cnxn = cnxn
        self.lst = lst
        # Delete the links
        self.delete_links_by_id(self.lst)
        # Reset the FID column
        self.fid_reset()

    def delete_links_by_id(self, list_0):
        """This method takes a list of ids and deletes
the links
        with those ids
        @Param list_0 is the list with the ids"""

        # For each id in the list
        for link in list_0:
            self.cnxn.cursor().execute("DELETE FROM
dbo."+self.links+" WHERE LinkID=?", (link,))
            self.cnxn.commit()

    def fid_reset(self):
        """This method resets the Fid column of a SQL
table after we have
        deleted some rows"""

        cursor = self.cnxn.cursor()
        # Select all the columns and sort them by their
FID with ascending order
        cursor.execute("SELECT * FROM dbo."+self.links+"
ORDER BY FID ASC")
        # Get all the rows of the table
        rows = cursor.fetchall()
        # Create an empty list in which we are going to
store the FIDs of the remaining links
```

```
        fid_list = []
        # Fill the list with the FIDs
        for part in rows:
            fid_list.append(part.FID)
        # Set a counter to 0
        counter = 0
        # For every element in the fid_list:
        for part in fid_list:
            # Reset the FIDs of every link
            cursor.execute("UPDATE dbo."+self.links+" SET
FID=? WHERE FID=?", (counter+11005563, part,))
            self.cnxn.commit()
            counter += 1


def main():
    # Create a connection to the database and the desired
tables
    connection = Connect("LINKS")
    # Create an empty list
    temp_list = []
    # As long as condition == True
    condition = True
    while condition:
        # Promt the user to give us the id of a link he
wants to remove or to end the process
        temp_id = raw_input("Please type the id of the
link or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
    # Create a new table and then delete the links
    DeleteLinksByID(connection.links, connection.cnxn,
temp_list)

if __name__ == '__main__':
    main()
```

Delete_links_In_Dictionary_By_Class

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module deletes links from the adjacency
dictionary based on their class"""
```

```
from Adjacency_Dictionary_From_Database import
Conversion, Connect


class DeleteLinksInDictByClass:

    def __init__(self, dict, cls):
        self.dict = dict
        self.cls = int(cls)
        # Change the visibility of the links
        self.delete_links(self.dict, self.cls)

    def delete_links(self, dictionary, cls):
        """This method deletes links based on their class
        @Param dictionary is the adjacency dictionary
        @Param cls is the class of links we want to
remove"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every link
            for link in values[1][0]:
                # If tha link's class == cls
                if link[5] == cls:
                    # Go to the ToNodeID of the link and
remove the key node from the list of nodes that enter the
node
                    ToNodeId = link[2]

dictionary[ToNodeId][1][1].remove(key)
                    # Remove the link
                    values[1][0].remove(link)


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    DeleteLinksInDictByClass(first.adjacency_dictionary,
"3")


if __name__ == '__main__':
    main()
```

Delete_links_In_Dictionary_By_ID

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module deletes links in the adjacency dictionary
based on their ID"""

from Adjacency_Dictionary_From_Database import
Conversion, Connect


class DeleteLinksInDictByID:

    def __init__(self, dict, lst):
        self.dict = dict
        self.lst = lst
        # Delete the links
        self.delete_links(self.dict, self.lst)

    def delete_links(self, dictionary, lst):
        """This method removes links based on their ID
        @Param dictionary is the adjacency dictionary
        @Param lst is the list of links we want to
remove"""

        # For every key and value in the adjacency
dictionary
        for key, values in dictionary.items():
            # For every link
            for link in values[1][0]:
                # If tha link's ID is in the lst
                if link[0] in lst:
                    # Go to the ToNodeID of the link and
remove the key node from the list of nodes that enter the
node
                    ToNodeId = link[2]

dictionary[ToNodeId][1][1].remove(key)
                    # Remove the link
                    values[1][0].remove(link)


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    temp_list = []
    condition = True
    # As long as condition == True
    while condition:
        # Promt the user to give us the id of a link he
```

```
wants to change or to end the process
        temp_id = raw_input("Please type the id of the
link or type End to continue! :")
        # If the users does not want the process to end
        if temp_id != 'End':
            # Get the integer of the users input
            ids = int(temp_id)
            # Append the list with tha id
            temp_list.append(ids)
        # If the user types 'End'
        else:
            condition = False
    DeleteLinksInDictByID(first.adjacency_dictionary,
temp_list)

if __name__ == '__main__':
    main()
```

Finds_Nodes_To_Remove_From_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module detects the node that we need to remove
from our network"""

import pyodbc

class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class NodesToRemove:

    def __init__(self, nodes, links, cnxn):

        self.nodes = nodes
        self.links = links
        self.cnxn = cnxn
```

```
        # Find the possible nodes to remove based on how
many links entering and exiting a certain node
        self.possible_nodes_to_remove =
self.create_list_of_possible_nodes_to_remove(self.get_row
s_of_table("SELECT ID, LinksExiting, LinksEntering FROM
dbo."+self.nodes+" WHERE VISIBILITY=1"))
        # If the links entering or exiting the nodes have
the same two-way attribute put them in the final list
        self.final_nodes_to_remove =
self.create_final_list(self.possible_nodes_to_remove)

    def get_rows_of_table(self, sql_string):
        """ This method gives us the desired rows of an
Sql table
        @Param Sql_String is the Sql string
        @Return are the desired rows """
        cursor = self.cnxn.cursor()
        cursor.execute(sql_string)
        rows = cursor.fetchall()
        return rows

    def create_list_of_possible_nodes_to_remove(self,
rows):
        """This method finds the possible nodes we have
to remove from our network
        based on how many links entering and exiting a
certain node
        @Param rows is the rows of the NODES table
        @Return is the list with the possible nodes"""

        # Create an empty list
        list_to_return = [[], [], []]
        # For every row in the NODES table
        for row in rows:
            # If there is exactly one link exiting the
node and exactly one link entering the node
            if row[1] == 1 and row[2] == 1:
                # Put it the first sub list
                list_to_return[0].append(row[0])
            # If there are exactly two links exiting the
node and exactly zero links entering the node
            elif row[1] == 2 and row[2] == 0:
                # Put it the second sub list
                list_to_return[1].append(row[0])
            # If there are exactly zero links exiting the
node and exactly two links entering the node
            elif row[1] == 0 and row[2] == 2:
                # Put it the third sub list
                list_to_return[2].append(row[0])
            else:
                pass
```

```
        return  list_to_return

    def create_final_list(self, nodes_list):
        """This method finds the final nodes that we have
to remove from our network based on two-way attribute
        @Param nodes_list is the list with the possible
nodes to remvoe
        @Return is the final list with the nodes we have
to remove"""

        # Create an empty list
        final_list = [[], [], []]
        # For every node in the first sub list
        for node in nodes_list[0]:
            # Get the link that enter the node
            cursor_1 = self.cnxn.cursor()
            cursor_1.execute("SELECT TwoWay FROM LINKS
WHERE ToNodeID=?", (node,))
            row = cursor_1.fetchone()
            first = row[0]
            cursor_1.close()
            # Get the link that exits the node
            cursor_2 = self.cnxn.cursor()
            cursor_2.execute("SELECT TwoWay FROM LINKS
WHERE FromNodeID=?", (node,))
            row = cursor_2.fetchone()
            second = row[0]
            cursor_2.close()
            # If both links have the same value in the
two-way attribute
            if first == second :
                # Put it in the final list
                final_list[0].append(node)
        # For every node in the second sub list
        for node in nodes_list[1]:
            # Get both links that exit the node
            cursor_2 = self.cnxn.cursor()
            cursor_2.execute("SELECT TwoWay FROM LINKS
WHERE FromNodeID=?", (node,))
            rows = cursor_2.fetchall()
            cursor_2.close()
            i =1
            for row in rows:
                if i == 1:
                    first = row[0]
                else:
                    second = row[0]
                i +=1
            # If both links are bidirectional
            if first == 1 and second ==1:
                # Put the node in the final list
```

```
                final_list[1].append(node)
        # For every node in the third sub list
        for node in nodes_list[2]:
            # Get both links that enter the node
            cursor_3 = self.cnxn.cursor()
            cursor_3.execute("SELECT TwoWay FROM LINKS
WHERE ToNodeID=?", (node,))
            rows = cursor_3.fetchall()
            cursor_3.close()
            i =1
            for row in rows:
                if i == 1:
                    first = row[0]
                else:
                    second = row[0]
                i +=1
            # If both links are bidirectional
            if first == 1 and second ==1:
                # Put the node in the final list
                final_list[2].append(node)

        return final_list



def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")

    lis = NodesToRemove(connection.nodes,
connection.links, connection.cnxn)

    print lis.possible_nodes_to_remove

    print lis.final_nodes_to_remove



if __name__ == '__main__':
    main()
```

Geometry_Extraction_To_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
# -*-coding: utf-8-*-
import arcpy
import csv
import os
```

```
from prettytable import PrettyTable
import pyodbc
from random import randint
"""This module takes a polyline shape-file and creates
two Sql tables. A node table that contains all the nodes
that
 exist in the shape-file and a links table that contains
the connections between the nodes"""


class Connect:
    """ Connects us to the database """
    def __init__(self):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')


class InsertCursor:

    def __init__(self, shapefile):
        # Inserts a cursor in the desired fields of our
polyline.
        self.shapefile = shapefile
        # The path to our shape file
        self.path =
r"C:\Users\vriz\Documents\ArcGIS\AddIns\Desktop10.1"
        self.fc = os.path.join(self.path, self.shapefile)
        # The desired fields
        self.fields = ["FID", "Shape@", "FromNodeID",
"ToNodeID", "ID"]
        # The cursor
        self.cursor = arcpy.da.SearchCursor(self.fc,
self.fields)


class FindGeometry:
    """ Finds the geometry of each polyline in a shape
file, stores it to a list, saves it to a csv file
     and prints it in a pretty way """

    def __init__(self, cursor, cnxn,):
        self.cursor = cursor
        self.cnxn = cnxn
        self.create_tables()
        # Creates the list with the geometry of each
polyline
        self.final_list = self.geometry(self.cursor)
```

```
        # Saves it to csv file
        self.start_and_ending_points =
self.get_start_and_end(self.final_list[0])
        # self.save_to_csv(self.final_list,
raw_input("Name of file:"))
        # Print it in a pretty way

self.print_dictionary(self.start_and_ending_points)
        self.print_dictionary(self.final_list[0])

    def create_tables(self):
        """This method creates the two tables. The NODES
table and the LINKS table"""
        cursor = self.cnxn.cursor()
        try:
            # Create the NODES table
            cursor.execute("CREATE TABLE NODES (FID int,
ID int, VISIBILITY int, CLASS int, LinksExiting int,
LinksEntering int, GEOMETRY geography)")
            cursor.commit()
        except:
            pass
        try:
            # Create the LINKS table
            cursor.execute("CREATE TABLE LINKS (FID int,
LinkID int, Visibility int, FromNodeID int, ToNodeID int,
StartCost int, TravCost int, CLASS int, TwoWay int,
StartCost2 int, GEOMETRY geography)")
            cursor.commit()
        except:
            pass

    def list_to_string(self, lst):
        """This method takes a list of coordinates and
forms the sting to use as input in the geometry column in
our
        SQL tables
        @Param lst is the list with the coordinates
        @Return is the desired string"""

        # Set a counter to 1
        i = 1
        # Create an empty string
        string = ""
        # For every node in the list
        for node in lst:
            # If it's the first node
            if i == 1:
                string = string+str(node[0])+"
"+str(node[1])
            # For every other node
```

```
            else:
                string = string+", "+str(node[0])+"
"+str(node[1])
                i += 1
        return string

    def geometry(self, cur):
        """ Extracts the geometry of each polyline and
stores it to a list
        @Param cur is the cursor we insert to the desired
fields of the shape file
        @Return is a list with the geometries """
        nodes_list = []
        for row in cur:
            if row[2] not in nodes_list:
                nodes_list.append(row[2])
            if row[3] not in nodes_list:
                nodes_list.append(row[3])
        cur.reset()
        length = len(nodes_list)+1
        # Create an empty list (this is the one that will
be returned)
        list_final = []
        # For each row in our cursor
        for row in cur:
            # Create a temporary list that is going to be
emptied every time we change row
            lst = []
            # For each part in the row[1]='shape' (It
could be a multi part polyline)
            for part in row[1]:
                # For each vertex of the feature
                for pnt in part:
                    # reate list with X,Y coordinates of
the vertex
                    lst_pnts = [pnt.X, pnt.Y]
                    # Append the temporary list which at
the end contains every X,Y of the polyline
                    lst.append(lst_pnts)
            # Get starting coordinate
            start = lst[0]
            # Get ending coordinate
            end = lst[-1]
            coords = [start, end]
            cursor = self.cnxn.cursor()
            cursor.execute("INSERT INTO LINKS (FID,
LinkID, Visibility, FromNodeID, ToNodeID, StartCost,
TravCost, CLASS, TwoWay, StartCost2) VALUES (?, ?, 1, ?,
?, ?, ?, ?, 1, ?)", (row[0]+1, row[4], row[2], row[3],
randint(10,20), randint(10,20), randint(1,5),
randint(10,20)),)
```

```
            cursor.commit()
            coordinates = lst
            sql_string = self.list_to_string(coordinates)
            cursor.execute("UPDATE LINKS SET
GEOMETRY=geography::STGeomFromText('LINESTRING("+sql_stri
ng+")', 4326) WHERE LinkID=?", (row[4]),)
            cursor.commit()
            if nodes_list != []:
                if row[2] in nodes_list:
                    cursor.execute("INSERT INTO NODES
(FID, ID, VISIBILITY, CLASS) VALUES (?, ?, 1, ?)",
(length-len(nodes_list), row[2], randint(1, 5)),)
                    cursor.commit()
                    coord_1 = str(coords[0][0])
                    coord_2 = str(coords[0][1])
                    cursor.execute("UPDATE NODES SET
GEOMETRY = geography::Point("+coord_2+", "+coord_1+",
4326) WHERE ID=?", (row[2],))
                    cursor.commit()
                    nodes_list.remove(row[2])
                if row[3] in nodes_list:
                    cursor.execute("INSERT INTO NODES
(FID, ID, VISIBILITY, CLASS) VALUES (?, ?, 1, ?)",
(length-len(nodes_list), row[3], randint(1, 5)),)
                    cursor.commit()
                    coord_1 = str(coords[1][0])
                    coord_2 = str(coords[1][1])
                    cursor.execute("UPDATE NODES SET
GEOMETRY = geography::Point("+coord_2+", "+coord_1+",
4326) WHERE ID=?", (row[3],))
                    cursor.commit()
                    nodes_list.remove(row[3])
            # Every time before we change row we append
the final list with the temporary list
            list_final.append(lst)
        return [list_final, nodes_list]

    def get_start_and_end(self, geom):
        """This method takes list of geometries and
returns a list that holds the starting and ending point
of
        each geometry
        @Param geom is the list with the geometries
        @Return is the list with the starting and ending
points"""
        coords = []
        for lst in geom:
            point_1 = lst[0]
            point_2 = lst[-1]
            coords.append([point_1, point_2])
        return coords
```

```
    def print_dictionary(self, lst):
        """ This method takes a list and prints it in a
pretty way
        @Param list is the list with the geometry of each
polyline """
        count = 0
        for part in lst:
            k = PrettyTable(["FID "+str(count)])
            k.add_row(["----->"])
            print k
            t = PrettyTable(["Geometry"])
            for n in part:
                t.add_row([n])
            print t
            print ""
            count += 1

    def save_to_csv(self, lst, name_of_file):
        """ This method takes a list and saves it to a
csv file
        @Param list is the list with the geometry of each
polyline
        @Param Name_of_file is the name of the file in
which the dictionary will be saved """
        w = csv.writer(open(name_of_file, "w"))
        count = 0
        for part in lst:
            w.writerow(["FID "+str(count)+"--->", part])
            count += 1


def main():
    """ main class here """
    # Create an instance of the insert_cursor class for
the desired polyline
    connection = Connect()
    cur = InsertCursor("Diploma2.shp")
    # Use the cursor of the instance to calculate the
geometry
    check = FindGeometry(cur.cursor, connection.cnxn)


if __name__ == '__main__':
    main()
```

Insert_Node_To_Database.py

```
__author__ = 'vriz'
# !/usr/bin/env python
# -*-coding: utf-8-*-
```

```
"""This module takes data for a new node that is about to
be inserted to our grid. Creates a copy of our database
and
inserts the new node in the appropriate nodes-table.
After that, it splits the link in which we want the new
node
to be inserted, in two parts"""

import pyodbc
import math


class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class InsertNode:

    def __init__(self, nodes, links, cnxn, list_1,
list_2, distance):
        self.nodes = nodes
        self.links = links
        self.cnxn = cnxn
        self.cursor = self.cnxn.cursor()
        self.distance = distance

        # The id of the node
        self.node_id = list_1[0]

        # The list that holds the information for the
insert method
        self.nodes_info_list = list_1

        # Get the correct FID number for the row that is
about to be inserted
        self.fid = self.get_fid(self.nodes)

        # The list that holds the information for the
split_link method
        self.links_info_list = list_2

        # Get the length and the geometry of the target
```

```
link
        self.link_length_and_geometry =
self.calculate_link_length(self.links_info_list[0])

        # Find the point (X, Y) that the node will be
inserted as well as the geometries of the two new links
        # that will be created
        self.point_of_insertion =
self.calculate_point_of_insertion(self.link_length_and_ge
ometry[1], self.distance)

        # Insert the row
        self.insert(self.nodes_info_list)

        # Split the link
        self.split_link(self.links_info_list)

        # Reset the FID column
        self.fid_reset(self.links)

    def calculate_point_of_insertion(self,
coordinates_list, distance):
        """This method takes a list of coordinates, a
given distance and returns a list with 3 elements.
        First element is the point that the
coordinates_list will be split
        Second and third element hold the coordinates of
after we split the coordinates_list at the
        desired point
        @Param ooordinates_list is a list with the
geometry of a link
        @Param distance is a given distance based on
which the coordinates_list we be split in two
        @Return is the 3 element list"""

        xy = coordinates_list
        link_length = 0
        # For every coordinate of the polyline except the
last one
        for i in range(0, len(coordinates_list)-1):
            # Calculate the ds between a point and its
next one
            ds = self.calculate_ds(xy[i], xy[i+1])
            # Add it to the length
            link_length += ds
            # If the length at that point is equal or
larger than the distance given:
            if link_length > distance:

                start = coordinates_list[i]
                end = coordinates_list[i+1]
```

```
                tens = self.calculate_ds(start, end)/0.01
                for x in range(1, int(tens+1)):
                    possible_point =
self.calculate_possible_point(start, end, x, int(tens)+1)
                    limit = link_length -
self.calculate_ds(possible_point, end)
                    if limit < distance + 0.01 and limit
> distance - 0.01:

                        point_to_split = possible_point
                        coordinates_list.insert(i+1,
point_to_split)
                        point_index =
coordinates_list.index(point_to_split)

                        link_1 =
coordinates_list[0:point_index+1]
                        link_2 =
coordinates_list[point_index:len(coordinates_list)]

                        return [point_to_split, link_1,
link_2]
                    else:
                        pass
                return None

    def calculate_possible_point(self, point_1, point_2,
x, i):
        new_x = (point_1[0]*(i-x) + point_2[0]*x)/i
        new_y = (point_1[1]*(i-x) + point_2[1]*x)/i
        return [new_x, new_y]

    def string_to_list(self, input_string):
        """This method takes a string of coordinates and
puts them in a list
        as floats
        @Param input_string is the string(x y, x y, x y)
with the coordinates
        @Return is a list[[x, y], [x, y] ,[x ,y]] with
the same coordinates"""

        # Put all the letters in a list
        lst = []
        for letter in input_string:
            lst.append(letter)
        # Remove the unnecessary characters
        for i in range(0, 12):
            lst.pop(0)
        lst = lst[:-1]
        # Reform the String
        string_1 = "".join(lst)
```

```
        # Split the string on the spaces and put the
parts on a list
        lst_2 = string_1.split()
        # Putting the coordinates in pairs and then in a
list (For uniting links)
        # Create the list that we are going to use as a
base
        final_list = []
        # Create a temporary list
        temp_list = []
        i = 0
        # For every coordinate in the list
        for coord in lst_2:
            # Append the temporary list with the
coordinate
            temp_list.append(coord)
            # If the coordinate is in position 1,3,5,7...
            if i % 2 != 0:
                # Append the final list with the
temporary one
                final_list.append(temp_list)
                # And empty the temporary one
                temp_list = []
            i += 1
        # Create an empty list that we are going to store
the pairs of coordinates as floats
        final_list_2 = []
        # For every pair o coordinates in final list we
want to make a pair of float coordinates
        for part in final_list:
            # Create a temporary list
            temp_list_2 = []
            # For every 'string' coordinate
            for cord in part:
                # If the 'string' coordinate does not end
in comma
                if cord[-1] != ",":
                    # Append the list with the float of
the 'string' coordinate
                    temp_list_2.append(float(cord))
                # If the 'string' coordinate ends in
comma
                if cord[-1] == ",":
                    # Remove the comma
                    word = cord[:-1]
                    # Append the list with the the float
of the 'string' coordinate
                    temp_list_2.append(float(word))
                # Append the final list with the
temporary list
            final_list_2.append(temp_list_2)
```

```
        return final_list_2

    def list_to_string(self, geometry):
        """This method takes a list of coordinates and
builds a string that we be used to store the geometry
        in our database
        @Param geometry is a list of coordinates (floats)
        @Return is the string to use as input to the
database"""

        # Set a counter to 1
        i = 1
        # Create an empty string
        string = ""
        # For every node the geometry
        for node in geometry:
            # If it's the first node of the geometry
            if i == 1:
                string = string+str(node[0])+"
"+str(node[1])
            # For every other node
            else:
                string = string+", "+str(node[0])+"
"+str(node[1])
            i += 1
        return string

    def calculate_ds(self, point_1, point_2):
        """This method calculates the distance between
two points
        @Param point_1 is the first point
        @Param point_2 is the second point
        @Return is the distance in Km"""

        latitude_1 = point_1[0]
        longitude_1 = point_1[1]
        latitude_2 = point_2[0]
        longitude_2 = point_2[1]
        # convert decimal degrees to radians
        lon1, lat1, lon2, lat2 = map(math.radians,
[longitude_1, latitude_1, longitude_2, latitude_2])
        # haversine formula
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = math.sin(dlat/2)**2 + math.cos(lat1) *
math.cos(lat2) * math.sin(dlon/2)**2
        c = 2 * math.asin(math.sqrt(a))
        km = 6367 * c
        return km

    def calculate_length(self, coordinates_list):
```

```
        """This method calculates the length of a link
        @Param coordinates_list is the list that holds
the coordinates of the link
        @Return is the length of the link"""

        xy = coordinates_list
        # Set length to zero
        link_length = 0
        # For every coordinate of the polyline except the
last one
        for i in range(0, len(coordinates_list)-1):
            # Calculate the ds between a point and its
next one
            ds = self.calculate_ds(xy[i], xy[i+1])
            # Add it to the length
            link_length += ds
        return link_length

    def get_fid(self, table):
        """ This method finds the correct FID number for
our row
        we want to insert in the nodes-table
        @ Param new_table is the copy of the nodes-table
        @ Return is the FID number"""

        # Select everything from our table
        self.cursor.execute("SELECT * FROM dbo."+table+"
ORDER BY FID ASC")
        # Assign the rows
        rows = self.cursor.fetchall()
        # Set a counter to zero
        counter = 0
        # For every row of the table
        for row in rows:
            # Add one to our counter
            counter += 1
        return counter+1

    def insert(self, data_list):
        """This method inserts the row to our nodes-table
        @ Param new_table is the copy of the nodes table
        @ Param data_list is the list that holds all the
        information for the new node:
        data_list[1] is the Id of the new node
        data_list[2] is the visibility of the new node
        data_list[3] is the class of the new node
        data_list[4] is the junction variable for the new
node"""

        self.cursor.execute("INSERT INTO dbo.NODES (FID,
ID, VISIBILITY, CLASS, JUNCTION) VALUES(?, ?, ?, ?, ?)",
```

```
(self.fid, self.node_id, data_list[1], data_list[2],
data_list[3],))
        self.cnxn.commit()
        self.cursor.execute("UPDATE dbo.NODES SET
GEOMETRY =
geography::Point("+str(self.point_of_insertion[0][0])+",
"+str(self.point_of_insertion[0][1])+", 4326) WHERE
ID=?", (self.node_id,))
        self.cnxn.commit()

    def calculate_link_length(self, link_id):

        self.cursor.execute("SELECT GEOMETRY.STAsText()
FROM dbo.LINKS WHERE LinkID=?", (link_id,))
        row = self.cursor.fetchone()
        geometry = self.string_to_list(row[0])
        length = self.calculate_length(geometry)
        return [length, geometry]

    def split_link(self, data_list):
        """This method takes a links-table and data for a
node that is going to be
        inserted in link and splits the link in 2 parts
        @ Param new_table is the copy of the links-table
        @ Param data_list is a list that holds all the
required information :
        data_list[0] is the Id of the link we are about
to split in two
        data_list[1] is the Id of the first new link we
will create
        data_list[2] is the Id of the second new link we
will create
        data_list[3] is the start cost_2 for the first
new link
        data_list[4] is the start cost for the second new
link
        data_list[5] is the traverse cost for the first
new link
        data_list[6] is the traverse cost for the second
new link
        """
        # Get the row where the LinID is the one we want
to split in two
        self.cursor.execute("SELECT * FROM dbo.LINKS
WHERE LinkID=?", (data_list[0],))
        row = self.cursor.fetchone()
        # FromNodeID for the first link
        from_node_1 = row[2]
        # ToNodeID for the first link
        to_node_1 = self.node_id
        # The class for both of the new links
```

```
            link_class = row[6]
            # The TWOWAY attribute of the new links
            link_TWOWAY = row[7]
            # StartCost for the first new link
            start_cost = row[4]
            # FromNodeID for the second link
            from_node_2 = self.node_id
            # ToNodeID for the second link
            to_node_2 = row[3]
            # StarCost2 for the second new link
            start_cost_2 = row[8]
            # Find the appropriate FID for the new links we
are about to insert in our database
            self.cursor.execute("SELECT * FROM dbo.LINKS")
            rows = self.cursor.fetchall()
            counter = 0
            for part in rows:
                counter += 1
            # Insert first link
            self.cursor.execute("INSERT INTO dbo.LINKS (FID,
LinkID, FromNodeID, ToNodeID, StartCost, TravCost, CLASS,
TWOWAY, StartCost2) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
(counter+1, data_list[1], from_node_1, to_node_1,
start_cost, data_list[5], link_class, link_TWOWAY,
data_list[3], ))
            # Inform the database
            self.cnxn.commit()
            # Form the string that will be used as input for
the geometry column of our first new link
            string_to_database =
self.list_to_string(self.point_of_insertion[1])
            self.cnxn.cursor().execute("UPDATE LINKS SET
GEOMETRY=geography::STGeomFromText('LINESTRING("+string_t
o_database+")', 4326) WHERE LinkID=?", (data_list[1],))
            self.cnxn.commit()
            # Insert second link
            self.cursor.execute("INSERT INTO dbo.LINKS (FID,
LinkID, FromNodeID, ToNodeID, StartCost, TravCost, CLASS,
TWOWAY, StartCost2) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
(counter+2, data_list[2], from_node_2, to_node_2,
data_list[4], data_list[6], link_class, link_TWOWAY,
start_cost_2, ))
            # Inform the database
            self.cnxn.commit()
            # Form the string that will be used as input for
the geometry column of our second new link
            string_to_database_2 =
self.list_to_string(self.point_of_insertion[2])
            self.cnxn.cursor().execute("UPDATE LINKS SET
GEOMETRY=geography::STGeomFromText('LINESTRING("+string_t
o_database_2+")', 4326) WHERE LinkID=?", (data_list[2],))
```

```
        self.cnxn.commit()
        # Delete the link we have already split in two
        self.cursor.execute("DELETE FROM dbo.LINKS WHERE
LinkID=?", (data_list[0],))
        self.cnxn.commit()

    def fid_reset(self, table):
        """This method resets the Fid column of a SQL
table after we have
        deleted some rows
        @ Param new_table is the copy of the links-
table"""

        # Select all the columns and sort them by their
FID with ascending order
        self.cursor.execute("SELECT * FROM dbo.LINKS
ORDER BY FID ASC")
        # Get all the rows of the table
        rows = self.cursor.fetchall()
        # Create an empty list in which we are going to
store the FIDs of the remaining links
        fid_list = []
        # Fill the list with the FIDs
        for part in rows:
            fid_list.append(part.FID)
        # Set a counter to 0
        counter = 1
        # For every element in the fid_list:
        for part in fid_list:
            # Reset the FIDs of every link
            self.cursor.execute("UPDATE dbo."+table+" SET
FID=? WHERE FID=?", (counter, part,))
            self.cnxn.commit()
            counter += 1


def main():
    connection = Connect("NODES", "LINKS")
    # The numbers are for test purposes
    lis = InsertNode(connection.nodes, connection.links,
connection.cnxn, [5555, 1, 11, 0], [1005572, 11, 22, 101,
102, 1011, 1022], 2)
    print lis.link_length_and_geometry[0]
    print lis.link_length_and_geometry[1]
    print lis.point_of_insertion[0]
    print lis.point_of_insertion[1]
    print lis.point_of_insertion[2]

if __name__ == '__main__':
    main()
```

Insert_Node_To_Dictionary.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-
"""This module insert a node to the adjacency
dictionary"""

from Adjacency_Dictionary_To_Shapefile import
DictToShapefile
from Adjacency_Dictionary_From_Database import
Conversion, Connect
from Adjacency_Dictionary_To_Database import
DictionaryToDatabase
import math


class InsertNode:

    def __init__(self, node_id, link_id, distance, dict):
        # The id of the new node
        self.node_id = node_id
        # The id of the link in which the new node will
be inserted
        self.link_id = link_id
        # The distance from the start of the link in
which new node will be inserted
        self.distance = distance
        # The adjacency dictionary
        self.dict = dict
        # Retrieve the data of the link that we will
split in two and node key it belongs to
        self.data =
self.retrieve_link_data_and_key(self.link_id, self.dict)
        # Create the new links
        self.new_links =
self.create_the_new_links(self.data[0], self.distance)
        # Fix the dictionary
        self.fixed_dictionary =
self.insert_to_dictionary(self.new_links[0],
self.new_links[1], self.new_links[2], self.data[1],
self.dict)

    def calculate_ds(self, point_1, point_2):
        """This method calculates the distance between
two points
        @Param point_1 is the first point
        @Param point_2 is the second point
        @Return is the distance in Km"""

        latitude_1 = point_1[0]
```

```
        longitude_1 = point_1[1]
        latitude_2 = point_2[0]
        longitude_2 = point_2[1]
        # convert decimal degrees to radians
        lon1, lat1, lon2, lat2 = map(math.radians,
[longitude_1, latitude_1, longitude_2, latitude_2])
        # haversine formula
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = math.sin(dlat/2)**2 + math.cos(lat1) *
math.cos(lat2) * math.sin(dlon/2)**2
        c = 2 * math.asin(math.sqrt(a))
        km = 6367 * c
        return km

    def calculate_point_of_insertion(self,
coordinates_list, distance):
        """This method takes a list of coordinates, a
given distance and returns a list with 3 elements.
        First element is the point that the
coordinates_list will be split
        Second and third element hold the coordinates of
after we split the coordinates_list at the
        desired point
        @Param ooordinates_list is a list with the
geometry of a link
        @Param distance is a given distance based on
which the coordinates_list we be split in two
        @Return is the 3 element list"""

        xy = coordinates_list
        link_length = 0
        # For every coordinate of the polyline except the
last one
        for i in range(0, len(coordinates_list)-1):
            # Calculate the ds between a point and its
next one
            ds = self.calculate_ds(xy[i], xy[i+1])
            # Add it to the length
            link_length += ds
            # If the length at that point is equal or
larger than the distance given:
            if link_length > distance:
                # Split the link at that point
                point_to_split = coordinates_list[i]
                point_index =
coordinates_list.index(point_to_split)
                link_1 =
coordinates_list[0:point_index+1]
                link_2 =
coordinates_list[point_index:len(coordinates_list)]
```

```
                    break
                else:
                    pass

        return [point_to_split, link_1, link_2]

    def retrieve_link_data_and_key(self, id_of_link,
dictionary):
        """This methods finds the link we want the new
node to be inserted into and the key(node) of the
adjacency
         dictionary in which the link belongs
         @Param id_of_link is the id the link
         @Param dictionary is the adjacency dictionary
         @Return link is the data of the link
         @Return key is the key(node) of the adjacency
dictionary"""

        # For each key and its values in the dictionary
        for key, values in dictionary.items():
            # For every link exiting the node
            for link in values[1][0]:
                # If the id of the link matches the given
id
                if link[0] == id_of_link:
                    # Return the link and the key
                    return [link, key]

    def create_the_new_links(self, link_data, distance):
        """This method forms the new links after
splitting the link, that the new node will be inserted
into, in two
        @Param link_data is the all the data for the link
that is going to be split
        @Param distance is the desired distance in which
we want the new node to be inserted
        @Reurn new_node is a list with the personal data
of the new node
        @Return first_link is the first link that will
enter the new node
        @Return second_link is the second link that will
exit from the new node"""

        # Calculate the point of insertion and get the
geometry for the new links
        point_and_link_geometry =
self.calculate_point_of_insertion(link_data[8], distance)
        # Form the first link
        first_link = [999, link_data[1], self.node_id, 0,
0, link_data[5], link_data[6], 0,
point_and_link_geometry[1]]
```

```
        # Form the second link
        second_link = [999, link_data[1], link_data[2],
0, 0, link_data[5], link_data[6], 0,
point_and_link_geometry[2]]
        # The personal data for the new node
        new_node = [1, 3,
[point_and_link_geometry[0][1],point_and_link_geometry[0]
[0]]]
        return [new_node, first_link, second_link]

    def insert_to_dictionary(self, new_node, first_link,
second_link, key_node, dict):
        """This method inserts the new node to the
adjacency dictionary
        @Param new_node is the id of the new node
        @Param first_link is the data of the first link
that will enter the new node
        @Param second_link is the data for the second
link that will exit from the new node
        @Param key_node is the node that the link, that
was split, exits from
        @Return is the fixed adjacency dictionary"""

        # Remove the link that was split in two from the
key_node
        dict[key_node][1][0].remove(self.data[0])
        # Add the first link to the key_node
        dict[key_node][1][0].append(first_link)

        # Go to the node that self.link_id enters into
and remove the key_node from its FromNodeID list
        dict[second_link[2]][1][1].remove(key_node)
        # Add the new node to the same list
        dict[second_link[2]][1][1].append(self.node_id)
        # Create the new key in the adjacency dictionary
with all the required data
        dict[self.node_id] = [new_node, [[second_link],
[key_node]]]

        return dict


def main():
    connection = Connect("NODES", "LINKS")
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)

    sixth = InsertNode(200111, 1005570, 25.6,
first.adjacency_dictionary)

    print sixth.data
```

```
    print sixth.fixed_dictionary[125088]


    seventh = DictionaryToDatabase(connection.cnxn,
sixth.fixed_dictionary,"yeah1", "yeah2")
    arcgis = DictToShapefile("nodes.shp", "links.shp",
sixth.fixed_dictionary)
if __name__ == '__main__':
    main()
```

Removing_Nodes_From_Dictionary.py

```
__author__ = 'vriz'
# !/usr/bin/env python
#  -*-coding: utf-8-*-

"""This module takes a list of nodes and removes the from
the adjacency dictionary"""

import pyodbc
from Adjacency_Dictionary_From_Database import Conversion
from Finds_Nodes_To_Remove_From_Database import
NodesToRemove
from prettytable import PrettyTable


class Connect:
    """ Connects us to the database """
    def __init__(self, nodes, links):
        """ Connects us to the database and asks for the
names of the node=table and link-table """

        self.cnxn =
pyodbc.connect('Trusted_Connection=yes', driver='{SQL
Server}', server='localhost', database='PracticeCopy1')
        self.nodes = nodes
        self.links = links


class RemoveNodes:

    def __init__(self, dictionary, lst):
        # The adjacency dictionary
        self.dictionary = dictionary
        # The list with the nodes we have to remove
        self.lst = lst
        # Remove the nodes and store the fixed dictionary
        self.fixed_dictionary =
self.remove_unnecessary_nodes(self.dictionary, self.lst)
        # Print it in a pretty way
```

```
        self.print_dictionary(self.fixed_dictionary)

    def adjust_dict_0(self, node, dict):
        """This method takes a node that has exactly one
link exiting the node and exactly one link entering the
node
        removes it from the dictionary and adjust the
connections
        @Param nodes is the id of the node we want to
delete
        @Param if the adjacency dictionary"""

        # The id of the node that one of its exiting
links is the one that enters our node
        key_from = dict[node][1][1][0]
        # The link we are going to use as the second one
in line to form the new link
        second_link = dict[node][1][0][0]
        # The id of the node that one of its entering
links is the one that exits our node
        key_to = second_link[2]
        # Go to the dictionary of the node that the new
link will end, remove our node and add the node that the
        # new link will start
        dict[key_to][1][1].remove(node)
        dict[key_to][1][1].append(key_from)
        # For every link that exits the node that the new
link will start from
        for link in dict[key_from][1][0]:
            # If that link heads towards our node
            if link[2] == node:
                # This is the link that we are going to
use as first in line to form the new link
                first_link = link
        # For every link that exits the node that the new
link will start from
        for link in dict[key_from][1][0]:
            # If that link heads towards our node
            if link[2] == node:
                # Remove it
                dict[key_from][1][0].remove(link)
        dict[key_from][1][0].append([999, first_link[1],
second_link[2], first_link[3], first_link[4] +
second_link[4], first_link[5], first_link[6],
second_link[7], first_link[8] + second_link[8][1:]])

    def adjust_dict_1(self, node, dict, lst, temp_list):
        """This method takes a node that has exactly two
links exiting the node and exactly zero links entering
the
        node, remove it from the dictionary and adjust
```

```
the connections
        @Param node is the id of the node we want to
delete
        @Param dict is the adjacency dictionary
        @Param lst is the list with all the nodes we want
to delete
        @Param temp_list is a list that we are going to
store the nodes we cannot delete at this point"""

        # Store the first and second link that exit our
node
        first_link = dict[node][1][0][0]
        second_link = dict[node][1][0][1]
        # If both of our links head towards a node that
we have to delete
        if first_link[2] in lst[2] and second_link[2] in
lst[2]:
                # Store it in a list to delete it later
                temp_list.append(node)
        # If the first link heads towards a node we have
to delete then we have to direct the new link towards
that node
        elif first_link[2] in lst[2]:
            # Switch the first and second link
            tranfer = second_link
            first_link = tranfer
            second_link = first_link
            # The id of the node that the new link will
start from
            key_from = first_link[2]
            # The id of the node that the new link will
end to
            key_to = second_link[2]
            # Remove our node from the FromNode list of
the key_from and key_to node
            dict[key_from][1][1].remove(node)
            dict[key_to][1][1].remove(node)
            # Go to the dictionary of the key_to node and
append is with key_from node
            dict[key_to][1][1].append(key_from)
            # Remove the first coordinate of the geometry
of the first link
            start = first_link[8][1:]
            # and reverse it
            start.reverse()
            # Form the new link
            dict[key_from][1][0].append([999, 1, key_to,
first_link[7], first_link[4] + second_link[4],
first_link[5], first_link[6], second_link[7], start +
second_link[8]])
```

```
        elif second_link[2] in lst[2]:
            # The id of the node that the new link will
start from
            key_from = first_link[2]
            # The id of the node that the new link will
end to
            key_to = second_link[2]
            # Remove our node from the FromNode list of
the key_from and key_to node
            dict[key_from][1][1].remove(node)
            dict[key_to][1][1].remove(node)
            # Go to the dictionary of the key_to node and
append is with key_from node
            dict[key_to][1][1].append(key_from)
            # Remove the first coordinate of the geometry
of the first link
            start = first_link[8][1:]
            # and reverse it
            start.reverse()
            # Form the new link
            dict[key_from][1][0].append([999, 1, key_to,
first_link[7], first_link[4] + second_link[4],
first_link[5], first_link[6], second_link[7], start +
second_link[8]])
        else:
            # The id of the node that the new link will
start from
            key_from = first_link[2]
            # The id of the node that the new link will
end to
            key_to = second_link[2]
            # Remove our node from the FromNode list of
the key_from and key_to node
            dict[key_from][1][1].remove(node)
            dict[key_to][1][1].remove(node)
            # Go to the dictionary of the key_to node and
append is with key_from node
            dict[key_to][1][1].append(key_from)
            # Remove the first coordinate of the geometry
of the first link
            start = first_link[8][1:]
            # and reverse it
            start.reverse()
            # Form the new link
            dict[key_from][1][0].append([999, 1, key_to,
first_link[7], first_link[4] + second_link[4],
first_link[5], first_link[6], second_link[7], start +
second_link[8]])

    def adjust_dict_2(self, node, dict):
        """This method takes a node that has exactly zero
```

```
links exiting the node and exactly two links entering the
        node, remove it from the dictionary and adjust
the connections
        @Param node is the id of the node we want to
delete
        @Param if the adjacency dictionary"""

        # The key of the first node that one of its link
head towards our node
        first_key = dict[node][1][1][0]
        # The key of the second node that one of its link
head towards our node
        second_key = dict[node][1][1][1]
        # For every link that exit the first key node
        for link in dict[first_key][1][0]:
            # if that link heads towards our node
            if link[2] == node:
                # Store it as the first link
                first_link = link
                # and remove it
                dict[first_key][1][0].remove(first_link)
        # For every link that exit the second key node
        for link in dict[second_key][1][0]:
            # if that link heads towards our node
            if link[2] == node:
                # Store it as the second link
                second_link = link
                # and remove it

dict[second_key][1][0].remove(second_link)

        from_key = first_key
        to_key = second_key
        # Remove the last coordinate of the geometry of
the second link
        end = second_link[8][:-1]
        # Reverse it
        end.reverse()
        # Form the link
        dict[from_key][1][0].append([999, first_link[1],
to_key, first_link[3], first_link[4] + second_link[4],
first_link[5], first_link[6], second_link[3],
first_link[8] + end])
        # Add the from_key node in the FromNodes list of
the to_key node
        dict[to_key][1][1].append(from_key)

    def adjust_dict_3(self, node, dict):
        """This method takes a node that has exactly two
links exiting the node and exactly zero links entering
the
```

```
        node, remove it from the dictionary and adjust
the connections
        @Param node is the id of the node we want to
delete
        @Param dict is the adjacency dictionary"""

        first_link = dict[node][1][0][0]
        second_link = dict[node][1][0][1]

        # The id of the node that the new link will start
from
        key_from = first_link[2]
        # The id of the node that the new link will end
to
        key_to = second_link[2]
        # Remove our node from the FromNode list of the
key_from and key_to node
        dict[key_from][1][1].remove(node)
        dict[key_to][1][1].remove(node)
        # Go to the dictionary of the key_to node and
append is with key_from node
        dict[key_to][1][1].append(key_from)
        # Remove the first coordinate of the geometry of
the first link
        start = first_link[8][1:]
        # and reverse it
        start.reverse()
        # Form the new link
        dict[key_from][1][0].append([999, 1, key_to,
first_link[7], first_link[4] + second_link[4],
first_link[5], first_link[6], second_link[7], start +
second_link[8]])

    def remove_unnecessary_nodes(self, dict, lst):
        """This method removes nodes from the adjacency
dictionary
        @Param dict is the adjacency dictionary
        @Param lst is the list with the nodes we want to
remove"""

        # Create a temporary list that we are going to
store the nodes we will delete later
        temp_list = []
        # For every node in the first list of nodes we
want to remove
        for node in lst[0]:
            # Remove the nodes and adjust the dictionary
            self.adjust_dict_0(node, dict)
        # For every node in the second list of nodes we
want to remove
        for node in lst[1]:
```

```
            # Remove the nodes and adjust the dictionary
            self.adjust_dict_1(node, dict, lst,
temp_list)
        # For every node in the third list of nodes we
want to remove
        for node in lst[2]:
            # Remove the nodes and adjust the dictionary
            self.adjust_dict_2(node, dict)
        # For every node in the list of nodes we wanted
to delete later
        for node in temp_list:
            # If only one link exits the node and only
one link enters the node
            if len(dict[node][1][0]) == 1 and
len(dict[node][1][0]) == 1:
                self.adjust_dict_0(node, dict)
            # If only two links exit the node and zero
links enter the node
            elif len(dict[node][1][0]) == 2 and
len(dict[node][1][0]) == 0:
                self.adjust_dict_3(node, dict)
            # If only zero links exit the node and two
links enter the node
            elif len(dict[node][1][0]) == 0 and
len(dict[node][1][0]) == 2:
                self.adjust_dict_2(node, dict)
            else:
                pass
        # For every node we want to delete
        for part in lst:
            for node in part:
                    # Remove it as keys from the
dictionary
                    del dict[node]
        return dict

    def print_dictionary(self, dict2):
        """This method takes a dictionary and prints it
in a pretty way
        @Param dict is the dictionary """
        i = 1
        for keys, values in dict2.items():
                print i
                k = PrettyTable([keys])
                k.add_row(["----->"])
                print k
                t = PrettyTable(['Visiblity', 'Class',
'Geometry'])
                t.add_row(values[0])
                f = PrettyTable(['LinkID', 'Visibility',
'ToNodeID', 'StartCost', 'TravCost', 'Class', 'TwoWay',
```

```
'StartCost2', 'Geometry'])
                for part in values[1][0]:
                    f.add_row(part)
                c = PrettyTable(['FromNodeID'])
                for part in values[1][1]:
                    c.add_row([part])
                print t
                print f
                print c
                print ""
                print ""
                i += 1


def main():
    """ main class here."""
    # Create a connection to the database and the desired
tables
    connection = Connect("NODES", "LINKS")
    # Create the adjacency Dictionary
    first = Conversion(connection.nodes,
connection.links, connection.cnxn)
    second = NodesToRemove(connection.nodes,
connection.links, connection.cnxn)
    adjacency_dictionary = first.adjacency_dictionary
    nodes_to_remove = second.final_nodes_to_remove
    third = RemoveNodes(adjacency_dictionary,
nodes_to_remove)


if __name__ == '__main__':
    main()
```

Dijkstra.py

```
__author__ = 'vriz'

import math
from prettytable import PrettyTable
from Adjacency_Dictionary_From_Database import
Conversion, Connect
from priodict import priorityDictionary

class FixedDict:

    def __init__(self, dict):
        self.dict = dict
        self.fixed_dictionary =
self.fix_dictionary(self.dict)

        #self.print_dictionary(self.fixed_dictionary)
```

```
    def calculate_ds(self, point_1, point_2):
            """This method calculates the distance
between two points
            @Param point_1 is the first point
            @Param point_2 is the second point
            @Return is the distance in Km"""

            latitude_1 = point_1[0]
            longitude_1 = point_1[1]
            latitude_2 = point_2[0]
            longitude_2 = point_2[1]
            # convert decimal degrees to radians
            lon1, lat1, lon2, lat2 = map(math.radians,
[longitude_1, latitude_1, longitude_2, latitude_2])
            # haversine formula
            dlon = lon2 - lon1
            dlat = lat2 - lat1
            a = math.sin(dlat/2)**2 + math.cos(lat1) *
math.cos(lat2) * math.sin(dlon/2)**2
            c = 2 * math.asin(math.sqrt(a))
            km = 6367 * c
            return km

    def calculate_length(self, coordinates_list):
            """This method calculates the length of a
polyline
            @Param coordinates_list is the list that
holds the coordinates of the polyline
            @Return is the length of the polyline"""

            xy = coordinates_list
            # Set length to zero
            link_length = 0
            # For every coordinate of the polyline except
the last one
            for i in range(0, len(coordinates_list)-1):
                # Calculate the ds between a point and
its next one
                ds = self.calculate_ds(xy[i], xy[i+1])
                # Add it to the length
                link_length += ds
            return link_length

    def fix_dictionary(self, dict2):
        dict1 = {}
        for key, values in dict2.items():

             for link in values[1][0]:
                 try:
                     dict1[key].append([link[2],
```

```
self.calculate_length(link[8])])
                    except:
                        dict1[key] = [[link[2],
self.calculate_length(link[8])]]
                    try:
                        dict1[link[2]].append([key,
self.calculate_length(link[8])])
                    except:
                        dict1[link[2]] = [[key,
self.calculate_length(link[8])]]
        return dict1

    def print_dictionary(self, dict2):
        """This method takes a dictionary and prints it
in a pretty way
        @Param dict is the dictionary """
        i = 1
        for keys, values in dict2.items():
                print i
                k = PrettyTable([keys])
                k.add_row(["----->"])
                print k
                t = PrettyTable(['Visiblity', 'Class',
'Geometry'])
                t.add_row(values[0])
                f = PrettyTable(['LinkID', 'Visibility',
'ToNodeID', 'StartCost', 'TravCost', 'Class', 'TwoWay',
'StartCost2', 'Geometry'])
                for part in values[1][0]:
                    f.add_row(part)
                c = PrettyTable(['FromNodeID'])
                for part in values[1][1]:
                    c.add_row([part])
                print t
                print f
                print c
                print ""
                print ""
                i += 1


def main():
    def Dijkstra(G,start,end=None):

        D = {}  # dictionary of final distances
        P = {}  # dictionary of predecessors
        Q = priorityDictionary()   # est.dist. of non-
final vert.
        Q[start] = 0

        for v in Q:
```

```
                D[v] = Q[v]
                if v == end: break
                for w in G[v]:
                    vwLength = D[v] + G[v][w]
                    if w in D:
                        if vwLength < D[w]:
                            raise ValueError, \
          "Dijkstra: found better path to already-final
vertex"
                    elif w not in Q or vwLength < Q[w]:
                        Q[w] = vwLength
                        P[w] = v

            return (D,P)

        def shortestPath(G, start, end):
            """
            Find a single shortest path from the given start
vertex
            to the given end vertex.
            The input has the same conventions as Dijkstra().
            The output is a list of the vertices in order
along
            the shortest path.
            """

            D, P = Dijkstra(G, start, end)
            Path = []
            while 1:
                Path.append(end)
                if end == start: break
                end = P[end]
            Path.reverse()
            return Path

        """ main class here."""
        # Create a connection to the database and the desired
tables
        connection = Connect("NODES", "LINKS")
        # Create the adjacency Dictionary
        lis = Conversion(connection.nodes, connection.links,
connection.cnxn)

        test = FixedDict(lis.adjacency_dictionary)
        i = 1
        for key, values in test.fixed_dictionary.items():
            print str(key) + " ------->"
            print
            print values
            print "============"
            for link in values:
```

```
            print i
            i+=1

    dictionary = test.fixed_dictionary

    G = {}
    V = {}
    for key, values in dictionary.items():
        for pair in values:
            V[pair[0]] = pair[1]

        G[key] = V
        V = {}
    print len(G)

    for key, values in G.items():
        print key
        print values

    td = shortestPath(G, 103298, 104379)
    print td
    print "hi"

if __name__ == '__main__':
    main()
```

Priodict.py

```
from __future__ import generators
# Priority dictionary using binary heaps
# David Eppstein, UC Irvine, 8 Mar 2002

# Implements a data structure that acts almost like a
dictionary, with two modifications:
# (1) D.smallest() returns the value x minimizing D[x].
For this to work correctly,
#        all values D[x] stored in the dictionary must be
comparable.
# (2) iterating "for x in D" finds and removes the items
from D in sorted order.
#        Each item is not removed until the next item is
requested, so D[x] will still
#        return a useful value until the next iteration
of the for-loop.
# Each operation takes logarithmic amortized time.



class priorityDictionary(dict):
   def __init__(self):
```

```
      '''Initialize priorityDictionary by creating binary
heap of pairs (value,key).
Note that changing or removing a dict entry will not
remove the old pair from the heap
until it is found by smallest() or until the heap is
rebuilt.'''
      self.__heap = []
      dict.__init__(self)

   def smallest(self):
      '''Find smallest item after removing deleted items
from front of heap.'''
      if len(self) == 0:
         raise IndexError, "smallest of empty
priorityDictionary"
      heap = self.__heap
      while heap[0][1] not in self or self[heap[0][1]] !=
heap[0][0]:
         lastItem = heap.pop()
         insertionPoint = 0
         while 1:
            smallChild = 2*insertionPoint+1
            if smallChild+1 < len(heap) and
heap[smallChild] > heap[smallChild+1] :
               smallChild += 1
            if smallChild >= len(heap) or lastItem <=
heap[smallChild]:
               heap[insertionPoint] = lastItem
               break
            heap[insertionPoint] = heap[smallChild]
            insertionPoint = smallChild
      return heap[0][1]

   def __iter__(self):
      '''Create destructive sorted iterator of
priorityDictionary.'''
      def iterfn():
         while len(self) > 0:
            x = self.smallest()
            yield x
            del self[x]
      return iterfn()

   def __setitem__(self,key,val):
      '''Change value stored in dictionary and add
corresponding pair to heap.
Rebuilds the heap if the number of deleted items gets
large, to avoid memory leakage.'''
      dict.__setitem__(self,key,val)
      heap = self.__heap
      if len(heap) > 2 * len(self):
```

```
            self.__heap = [(v,k) for k,v in
self.iteritems()]
            self.__heap.sort()  # builtin sort probably
faster than O(n)-time heapify
        else:
            newPair = (val,key)
            insertionPoint = len(heap)
            heap.append(None)
            while insertionPoint > 0 and newPair <
heap[(insertionPoint-1)//2]:
                heap[insertionPoint] = heap[(insertionPoint-
1)//2]
                insertionPoint = (insertionPoint-1)//2
            heap[insertionPoint] = newPair

    def setdefault(self,key,val):
        '''Reimplement setdefault to pass through our
customized __setitem__.'''
        if key not in self:
            self[key] = val
        return self[key]
```

Βιβλιογραφία

[1] Richard E.Allsop (2006). Transport netwrorks and their use: how real can modelling get?

[2] Dr. Jean-Paul Rodrigue and Dr. Cesar Ducruet (2015). The Geography of Transportation Networks

[3] Center for Transportation Research and Education, Iowa State University (2007). Historical overview of transportation planning model development.

[4] Jianwei Zhang, Feixiong Liao, Theo Arentze, Harry Timmermans (2011). A multimodal transport network model for advanced traveler information systems

[5] Cesar Ducruet, Igor Lugo (2011). Structure and dynamics of transportation networks: models, nethods, and applications.

[6] A. Stewart Fotheringham, Micahel Wegener (1999): Spatial Models and GIS: New and Potential Models.

[7] Aruna Sivakumar (2007): MODELLING TRANSPORT: A Synthesis of Transport Modelling Methodologies.

[8] Geographic information system. Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Geographic_information_system

[9] Mapping out the GIS Software Landscape. Available at: http://gisgeography.com/mapping-out-gis-software-landscape

[10] Τσακαλίδου Παναγιώτα (2014): Μελέτη και Ανάπτυξη Δυναμικών Δομών Περιγραφής της Υποδομής Μεταφορών για την Αποτελεσματική Εφαρμογή Υπολογιστικών Μοντέλων.

[11] The European Transport-Policy Information System (ETIS): Overview and Assessment of the GIS Component, E. Koukoutsis, A. Ballis, Y. Koukoutsis, A. Koutoumanos, International Conference of the AcrGIS Users, Athens, 2006.

[12] MESUDEMO, Transport Data Bases: 'Data Architecture and Strategic Management' and 'Computer Models and Data Architecture', Koukoutsis, E., Marciani, M., Michalopoulos, M., Papaodysseus, C., common deliverable D4 and D8, project MESUDEMO, Transport Programme, 2000.

[13] Μπαλλής, Θ. (2012). *Ανάπτυξη εφαρμογής εύρεσης βέλτιστης διαδρομής σε περιβάλλον GIS - Εφαρμογή στο Ευρωπαϊκό δίκτυο συνδυασμένων μεταφορών.* Αθήνα: Τομέας Μεταφορών και Συγκοινωνιακής Υποδομής, Σχολής Πολιτικών Μηχανικών, ΕΜΠ

[14] Φωτόπουλος Ευάγγελος (2014): Λειτουργική Αναβάθμιση της Εσωτερικής Αρχιτεκτονικής της Γεωσυσχετισμένης Πληροφορίας Συστημάτων Υποστήριξης Αποφάσεων και Χάραξης Πολιτικής Συγκοινωνιών και Μεταφορών σε Ευρωπαϊκό Επίπεδο. Αναγκαία Σχετικά Εργαλεία Λογισμικού.

[15] Designing a flexible, highly adaptable and gracefully expandable Information System for the implementation of complex Decision Support Systems, E. Koukoutsis, C. Papaodysseus, N. V. Karadimas, A. Ballis, International Journal of Simulation Vol. 7, No 4-5, 2008.

[16] List of geographic information systems software. Retrieved from Wikipedia, the free encyclopedia available at: https://en.wikipedia.org/wiki/List_of_geographic_information_systems_software#cite_note-spatialserver.net-3

[17] SQL. Retrieved from Wikipedia, the free encyclopedia available at: https://en.wikipedia.org/wiki/SQL

[18] Python (programming language). Retrieved from Wikipedia, the free encyclopedia available at: https://en.wikipedia.org/wiki/Python

[19] Dijkstra's algorithm. (n.d.). Retrieved from Wikipedia, the free encyclopedia available at: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[20] ESRI ArcGIS Software review and Guide. Available at: http://gisgeography.com/esri-arcgis-software-review-guide

.