ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση Map-Reduce σε Spatial Δεδομένα με τον Αλγόριθμο Dijkstra

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δημήτρη Ι. Θεοδωράκη

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση Map-Reduce σε Spatial Δεδομένα με τον Αλγόριθμο Dijkstra

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δημήτρη Ι. Θεοδωράκη

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 28 Νοεμβρίου 2011.

.....	.....	.....
Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Αναπλ. Καθηγητής Ε.Μ.Π.	Dieter Pfoser Ερευνητής Β ΙΠΣΥΠ

Αθήνα, Νοέμβριος 2011

.....
Δημήτρης Ι. Θεοδωράκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον Αλέξανδρο Εφεντάκη από το ΙΠΣΥΠ για την πολύτιμη βοήθεια που απλόχερα μου προσέφερε από την αρχή μέχρι το τέλος αυτής της διπλωματικής. Ακόμα θα ήθελα να ευχαριστήσω τον κ. Dieter Pfoser που επέβλεψε τη διπλωματική αυτή.

Επίσης θα ήθελα να ευχαριστήσω όλους του κοντινούς και αγαπημένους μου ανθρώπους, που πάντα με στηρίζουν σε ότι κι αν κάνω.

Αλλά πρώτα απ' όλα θα ήθελα να ευχαριστήσω τους γονείς μου, γιατί μου έχουν εμφυσήσει την αγάπη τους για τη γνώση και με στηρίζουν όλα αυτά τα χρόνια για να την αποκτήσω.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην υλοποίηση αλγορίθμου εύρεσης συντομότερων μονοπατιών σε τμηματοποιημένους γράφους με τη χρήση παραλληλοποίησης και map-reduce σε web browsers με Javascript. Ως αλγόριθμος συντομότερου μονοπατιού χρησιμοποιείται ο Dijkstra ενώ τα πειραματικά δεδομένα μας προκύπτουν από το γράφο του οδικού δικτύου της Δυτικής Ευρώπης, που διατίθενται για επιστημονική χρήση από το Ινστιτούτο Karlsruher. Η διαδικασία χωρίζεται σε τρία βασικά στάδια. Στο στάδιο της προ-επεξεργασίας δεδομένων τμηματοποιείται ο γράφος σε κελιά με τη βοήθεια του εργαλείου METIS. Στο δεύτερο στάδιο δημιουργείται ένα γράφος επικάλυψης, με τη βοήθεια της map-reduce διαδικασίας που εκτελείται από τα προγράμματα περιήγησης των χρηστών (Javascript), ενώ το τρίτο και τελευταίο στάδιο περιλαμβάνει την εύρεση του συντομότερου μονοπατιού με τη χρήση του γράφου επικάλυψης που κατασκευάστηκε στο δεύτερο στάδιο. Να σημειωθεί ότι η παρούσα εργασία ασχολείται μονάχα με τα δύο πρώτα στάδια ενώ το τελευταίο αναφέρεται για λόγους συνέπειας.

Λέξεις - Κλειδιά

map-reduce, Javascript, υπολογισμός συντομότερου μονοπατιού, Dijkstra, METIS, οδικά δίκτυα, γράφοι, multilevel graph partitioning, database, graph clustering, graph overlay

Abstract

This diploma thesis is focused on the implementation of a single source shortest path algorithm for partitioned graphs, using parallel computing and map-reduce on web browsers with Javascript. Dijkstra 's algorithm is used for shortest path calculations whereas the dataset used for the experiments comes from a road graph of Western Europe that the Institute of Karlsruher offers for research. The process is divided in three basic steps. In the first step we partition graph using the METIS tool. In the second step we construct an overlay graph by running the map-reduce process via web visitors. In the third step we calculate the actual shortest path by using the overlay graph that was constructed in the previous steps. It should be mentioned that this diploma thesis deals only with the first two steps. The last step is only mentioned for consequence.

Keywords

map-reduce, Javascript, shortest path calculation, Dijkstra, METIS, road maps, graphs, multilevel graph partitioning, database, graph clustering, graph overlay

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Ο υπολογισμός βέλτιστων διαδρομών σε δίκτυα μεταφοράς μεταξύ δύο σημείων έναρξης και τερματισμού, είναι ένα από τα εκθέματα πρακτικής εφαρμογής των αλγορίθμων. Συχνά χρησιμοποιούμε αυτά τα συστήματα, όταν σχεδιάζουμε ταξίδια με το αυτοκίνητο ή τα μέσα μεταφοράς [14]. Υπάρχουν επίσης πολλές εφαρμογές, όπως ο σχεδιασμός των logistic ή η εξομοίωση κίνησης, που απαιτούν την επίλυση τεράστιου αριθμού ερωτημάτων συντομότερου μονοπατιού σε δίκτυα μεταφοράς. Ο Dijkstra συντομότερου μονοπατιού [17] είναι ο βασικός αλγόριθμος για την επίλυση τέτοιων προβλημάτων. Αν και εκτελείται σχεδόν σε γραμμικό χρόνο, η ταχύτητα του σε υπολογισμούς ηπειρωτικού μεγέθους είναι περιοριστική. Έτσι ήδη από τη δεκαετία του '90 έχουν ξεκινήσει ακαδημαϊκές έρευνες για την επιτάχυνση του αλγορίθμου Dijkstra, τις οποίες παρακίνησε μία έρευνα για το χρονοδιάγραμμα των τρένων [18]. Μεγάλη ώθηση στις έρευνες έδωσε η δημοσίευση δικτύων μεταφοράς ηπειρωτικού μεγέθους το 2005. Το Ευρωπαϊκό δίκτυο έγινε διαθέσιμο για επιστημονική χρήση από την εταιρία PTV AG, ενώ το Αμερικάνικο δίκτυο έγινε διαθέσιμο από δημόσια γεωγραφικά δεδομένα [33].

Το σύνολο αυτών των τεχνικών χωρίζουν την όλη διαδικασία σε δύο βήματα. Αρχικά γίνεται ένας προϋπολογισμός στα δεδομένα του δικτύου. Αυτή η διαδικασία ανάλογα με τη μέθοδο, μπορεί να απαιτεί από λίγα λεπτά μέχρι και μερικές ώρες. Είναι πολύ σημαντικό ότι αυτό το βήμα γίνεται μονάχα μία φορά. Στο δεύτερο βήμα εκτελείται ο υπολογισμός του συντομότερου μονοπατιού, δεδομένου της πληροφορίας που έχει προϋπολογιστεί στο πρώτο βήμα. Το δεύτερο βήμα απαιτεί συνήθως μερικά εκατοστά του δευτερολέπτου, και συνεπώς μπορεί να εφαρμοστεί σε πρακτικά συστήματα. Συγκριτικά με την εκτέλεση ενός απλού Dijkstra σε ολόκληρο τον αρχικό γράφο, οι τεχνικές αυτές εκτελούν το ερώτημα 10^2 μέχρι 10^6 φορές γρηγορότερα. Οι τεχνικές που έχουν προταθεί μέχρι στιγμής στηρίζονται σε *πολυεπίπεδες μεθόδους διαχωρισμού (seperator multi-level)* [18], στα *edge flags* [25], ενώ η πρώτη ιεραρχική μέθοδος *HHs* κατάφερε να χειριστεί με ευκολία τα οδικά δίκτυα της Ευρώπης και της Αμερικής [30]. Η διαδικασία θα επιταχυνθεί ακόμα περισσότερο το 2006 από τον Muller (*High Performance Multilevel*) [26], που θα προϋπολογίσει με

επιθετικό τρόπο τα κομμάτια που απαιτούνται για την πολυεπίπεδη αναζήτηση με τη μέθοδο διαχωρισμού. Δαπανώντας πολύ υψηλό χώρο και χρόνο για τον προϋπολογισμό, αυτή η τεχνική επιταχύνει περίπου 400.000 φορές τον κλασικό αλγόριθμο Dijkstra. Ανεξάρτητα αναπτύχθηκε η τεχνική των *κόμβων διέλευσης* (*transit nodes*) [10]. Αυτή η τεχνική θα επιταχύνει τον κλασικό Dijkstra έξι τάξεις μεγέθους.

Παρ' όλο που αυτές οι τεχνικές επιταχύνουν εντυπωσιακά την εκτέλεση των ερωτημάτων, ο χρόνος προϋπολογισμού παραμένει αρκετά σημαντικός. Συγκεκριμένα η τελευταία τεχνική που αναφέραμε, απαιτεί περίπου 230 λεπτά προϋπολογισμού, περισσότερες από 3.5 ώρες.

1.1 Σκοπός

Σκοπός της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός συστήματος που θα επιταχύνει τη διαδικασία προϋπολογισμού που αναφέρθηκε παραπάνω. Συγκεκριμένα θα παραλληλοποιεί τη διαδικασία και θα την εκτελεί σε επισκέπτες ιστοσελίδων με τη χρήση Javascript. Η μέθοδος που υλοποιήθηκε σε αυτή την εργασία, στηρίζεται σε ιεραρχική δρομολόγηση και σε μεθόδους διαχωρισμού. Έτσι το στάδιο του προϋπολογισμού έχει σκοπό τη δημιουργία ενός γράφου επικάλυψης. Ο γράφος αυτός είναι μια μικρογραφία του αρχικού γράφου και τα ερωτήματα συντομότερου μονοπατιού εκτελούνται σε αυτόν. Ο γράφος αυτός περιέχει χιλιάδες κόμβους αντί των εκατομμυρίων που διέθετε ο αρχικός. Κατά συνέπεια τα ερωτήματα θα εκτελούνται σε αυτόν πολύ γρηγορότερα.

1.2 Περιγραφή Προβλήματος

Το συνολικό πρόβλημα χωρίζεται σε τρία βασικά στάδια. Το πρώτο στάδιο είναι *ανεξάρτητο των βαρών* του γράφου. Ο αρχικός γράφος διαιρείται σε μικρότερα «κελιά» (graph partitioning) με τη χρήση μεθόδων διαχωρισμού. Στην πράξη αυτό γίνεται σχετικά σπάνια αφού στηρίζεται μονάχα στην τοπολογία του γράφου. Στο δεύτερο στάδιο της υλοποίησης δημιουργείται ο γράφος επικάλυψης που αναφέραμε παραπάνω. Οι υπολογισμοί αυτοί εξαρτώνται από τα βάρη του αρχικού γράφου. Έτσι όποτε γίνεται κάποια αλλαγή σε αυτά, πρέπει να εκτελεστεί ξανά ο αλγόριθμος. Η δημιουργία του γράφου επικάλυψης γίνεται με ανεξάρτητους υπολογισμούς σε κάθε «κελί». Γι' αυτό μπορεί να παραλληλοποιηθεί εύκολα το βήμα αυτό. Τέλος στο τρίτο στάδιο εκτελείται το ερώτημα για τον υπολογισμό της συντομότερης απόστασης, χρησιμοποιώντας την πληροφορία που τα δύο πρώτα στάδια έχουν παράγει. Το ερώτημα πρέπει να εκτελείται τόσο γρήγορα ώστε να απαντιέται σε πραγματικό χρόνο. Να σημειωθεί εδώ πως η παρούσα εργασία πραγματεύεται μονάχα τα δύο πρώτα στάδια ενώ το τρίτο αναφέρεται για λόγους συνέπειας.

Στο επόμενο κεφάλαιο θα δούμε θεωρητικά πως γίνεται η διαίρεση του αρχικού γράφου σε «κελιά» αλλά και πώς δημιουργείται ο γράφος επικάλυψης. Στο κεφάλαιο 3 θα δούμε την αρχιτεκτονική του συστήματος που υλοποιήθηκε, ενώ στο κεφάλαιο 4 παρουσιάζονται τα πειραματικά αποτελέσματα. Τέλος στο κεφάλαιο 5 γίνεται μία σύνοψη της έρευνας και των μελλοντικών επεκτάσεων της.

Κεφάλαιο 2

Περιγραφή προβλήματος

Σε αυτό το κεφάλαιο θα περιγράψουμε με λεπτομέρεια κάθε βήμα που απαιτείται για τον προϋπολογισμό των δεδομένων (δημιουργία γράφου επικάλυψης). Στην πρώτη ενότητα παρουσιάζεται ο τρόπος με τον οποίο διαιρέθηκε ο αρχικός γράφος και δημιουργήθηκαν τα αντίστοιχα «κελιά», τόσο σε θεωρητικό αλλά και σε πρακτικό επίπεδο για την εκτέλεση των πειραμάτων. Στη συνέχεια παρουσιάζεται με λεπτομέρεια ο αλγόριθμος που εκτελείται για την ανάλυση κάθε κελιού, όπου γίνεται μία αναλυτική παρουσίαση του αλγορίθμου Dijkstra συντομότερου μονοπατιού. Ενώ στην τελευταία ενότητα η έννοια του map-reduce αλλά και ο τρόπος που υλοποιήθηκε στην παρούσα εργασία, σε επισκέπτες ιστοσελίδων με Javascript.

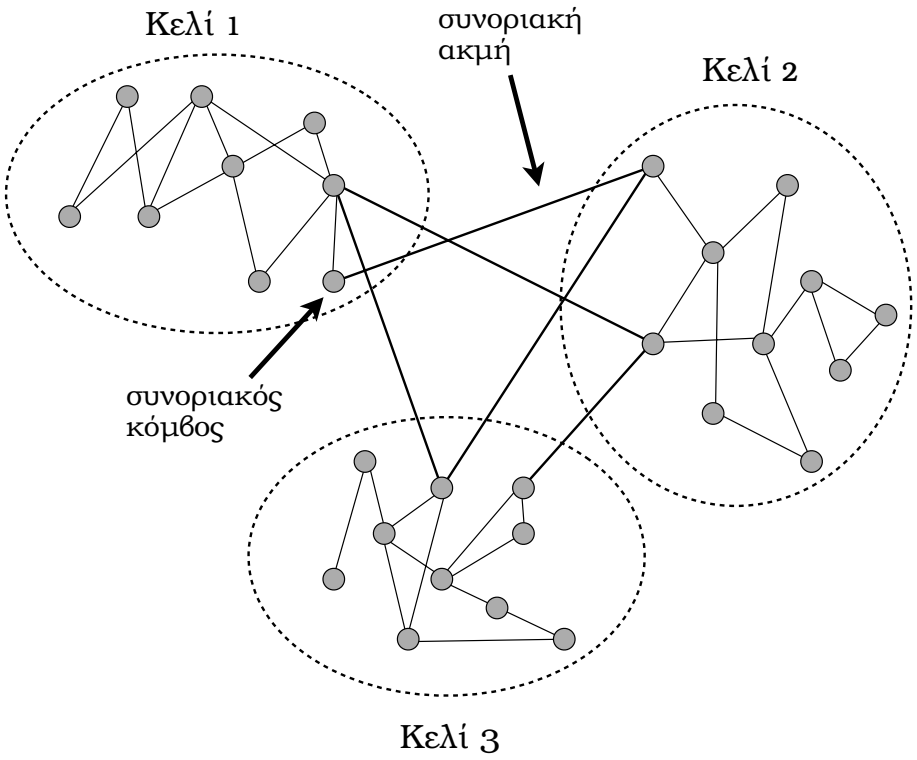
2.1 Διαίρεση γράφου

Το πρώτο βήμα για την κατασκευή του γράφου επικάλυψης, περιλαμβάνει τη χρήση μεθόδων διαχωρισμού για τη διαίρεση του αρχικού γράφου G σε τμήματα C. Το κριτήριο βελτιστοποίησης αυτής της διαδικασίας είναι να υπάρχουν οι λιγότερες δυνατές συνοριακές ακμές[1] (edge-cut - οι οποίες γραφικά απεικονίζονται στο σχήμα 2.1) μεταξύ των τμημάτων. Έτσι δημιουργούμε ένα πλέγμα από υπογράφους όπως φαίνεται στο σχήμα 2.1. Κάθε τέτοιος υπογράφος θα αναφέρεται ως «κελί». Η διαδικασία αυτή γίνεται πολύ σπάνια και δεν υπάρχει λόγος βελτιστοποίησης της. Παρακάτω γίνεται θεωρητική ανάλυση του προβλήματος καθώς και του εργαλείου που χρησιμοποιήθηκε για τη διαίρεση του γράφου.

2.1.1 Περιγραφή προβλήματος

Δεδομένου ενός γράφου $G(V, E)$, όπου το V υποδηλώνει το σύνολο των κόμβων και E το σύνολο των ακμών. Η βασική (αβαρής) έκδοση του προβλήματος διαίρεσης

[1] Λέμε ότι μία ακμή είναι συνοριακή ακμή όταν συνδέει κόμβους που ανήκουν σε διαφορετικά κελιά. Αντίστοιχα οι κόμβοι αυτοί ονομάζονται συνοριακοί κόμβοι.



Σχήμα 2.1: Εικονική αναπαράσταση δημιουργίας κελιών. Μπορεί εύκολα να δει κανείς τους συνοριακούς κόμβους και τις συνοριακές ακμές.

γράφου (partition problem) είναι: Δεδομένου G και ακεραίου $k > 1$, διαίρεσε το V σε k τμήματα (υποσύνολα) $V_1, V_2, ..., V_k$ τέτοια ώστε τα σύνολα να είναι ξένα μεταξύ τους, να έχουν ίσο μέγεθος και ο αριθμός των ακμών που έχουν άκρα σε διαφορετικά τμήματα (V_i, V_j) να είναι ελάχιστος. Σε πρακτικές εφαρμογές ένα μικρό σφάλμα ε επιτρέπεται δημιουργώντας το παρκάτω κριτήριο ισορροπίας:

$$\max_i |V_i| \leq (1 + \varepsilon)\frac{|V|}{k} \tag{2.1}$$

Στη γενικότερη (με βάρη) έκδοση, οι κόμβοι και οι ακμές μπορούν να έχουν βάρος. Το πρόβλημα διαίρεσης του γράφου σε αυτή την περίπτωση πραγματεύεται τη διαίρεση του G σε k ξένα τμήματα τέτοια ώστε να έχουν περίπου ίσο βάρος και το μέγεθος των ακμών που «κόβουν» (edge cuts) να ελαχιστοποιείται. Το μέγεθος ενός «κοψίματος» (cut) υπολογίζεται ως το άθροισμα των βαρών των ακμών που περιέχει, ενώ το βάρος ενός τμήματος είναι το άθροισμα των βαρών των κόμβων του τμήματος.

2.1.2 Το εργαλείο METIS

Για τη διαίρεση του γράφου χρησιμοποιήθηκε το εργαλείο METIS [20], που έχει χρησιμοποιηθεί με επιτυχία σε παρόμοιες εφαρμογές [8, 21, 9, 29]. Το METIS

είναι ένα σύνολο προγραμμάτων για διαίρεση γράφων, βρόχων με πεπερασμένα στοιχεία, καθώς και την παραγωγή fill reducing ταξινομήσεις για αραιούς πίνακες. Ο αλγόριθμος που υλοποιεί το METIS στηρίζεται σε:

- πολυεπίπεδα αναδρομικά - διχοτομικά
- πολυεπίπεδα k-διαδρομών
- πολλαπλών περιορισμών

σχήματα διαίρεσης που έχουν αναπτύξει.

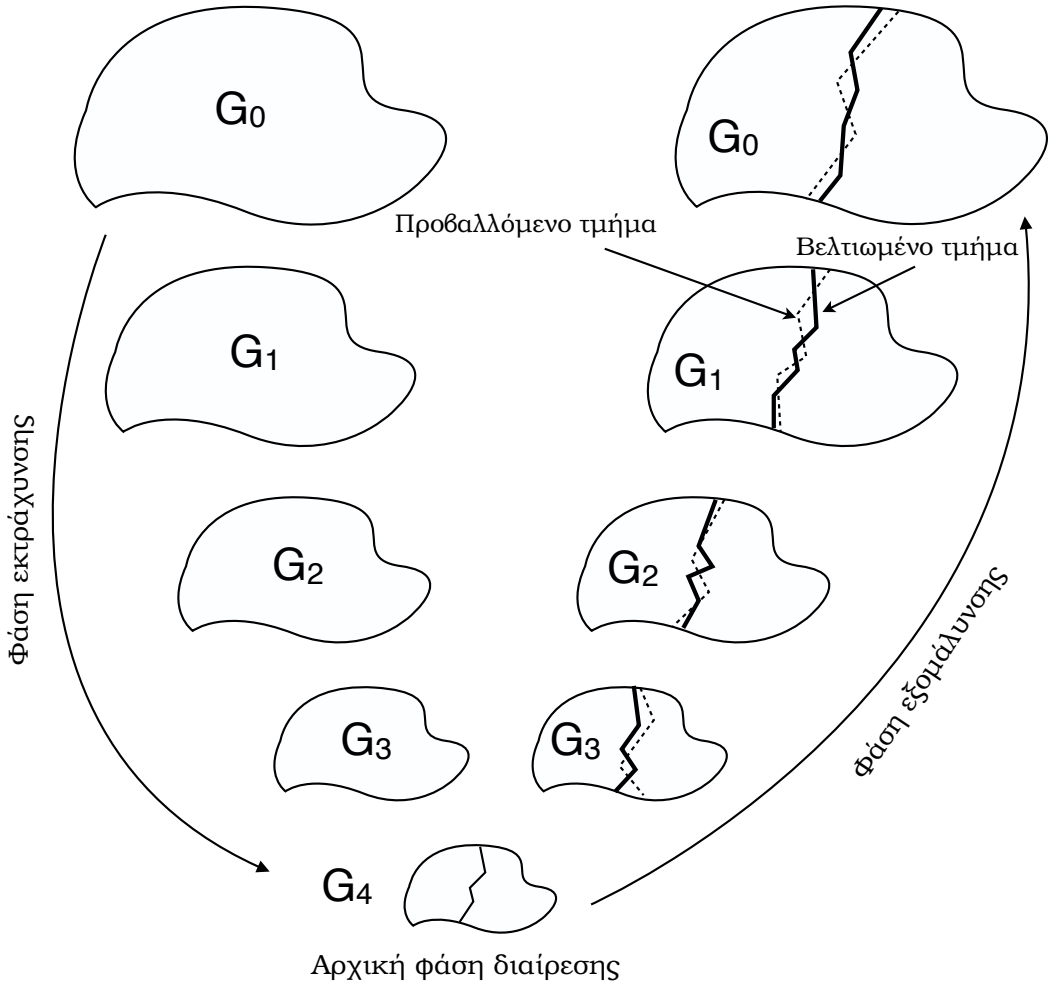
Πειράματα σε μεγάλο αριθμό γράφων που προκύπτουν από διαφορετικούς κλάδους όπως μεθόδων πεπερασμένων στοιχείων, γραμμικού προγραμματισμού, VLSI και μεταφοράς δείχνουν ότι το METIS παράγει τμήματα (partitions) που είναι συνεχώς καλύτερα από αυτά που παράγονται από άλλους ευρέως χρησιμοποιούμενους αλγορίθμους. Τα τμήματα που παράγει το METIS είναι 10% με 50% καλύτερα από τα αντίστοιχα που παράγουν φασματική αλγόριθμοι διαίρεσης (spectral partitioning algorithms). Παράλληλα πειράματα σε ένα ευρύ φάσμα γράφων έχουν δείξει ότι το METIS είναι μία με δύο τάξης μεγέθους γρηγορότερο από τους υπόλοιπους αλγορίθμους. Γράφοι με περισσότερους από 1.000.000 κόμβους μπορούν να διαιρεθούν σε 256 τμήματα σε μερικά δευτερόλεπτα με τη χρήση προσωπικών υπολογιστών.

2.1.3 Πολυεπίπεδη διαίρεση γράφου

Η βασική ιδέα των αλγορίθμων πολυεπίπεδης διαίρεσης γράφου που χρησιμοποιεί το METIS είναι πολύ απλή. Ο γράφος G αρχικά εκτραχύνεται (coarsened) σε μερικές εκατοντάδες κόμβους, υπολογίζεται μία διχοτόμηση σε αυτό τον κατά πολύ μικρότερο γράφο, και στη συνέχεια αυτό το τμήμα προβάλλεται πίσω στον αρχικό γράφο (λεπτότερος γράφος), με περιοδικές βελτιώσεις στο τμήμα. Αφού ο λεπτότερος γράφος έχει περισσότερους βαθμούς ελευθερίας, τέτοιες βελτιώσεις συνήθως μειώνουν τα κοψίματα ακμών (edge-cut). Τη διαδικασία αυτή μπορούμε να δούμε γραφικά στο Σχήμα 2.2. Στη συνέχεια θα παρουσιάσουμε αναλυτικά τη διαδικασία του αλγορίθμου.

Φάση Εκτράχυνσης (Coarsening Phase)

Κατά τη διαδικασία αυτή, μία ακολουθία μικρότερων γράφων $G_l = (V_l, E_l)$, δημιουργείται από τον αρχικό γράφο $G_0 = (V_0, E_0)$ τέτοιοι ώστε $|V_l| > |V_{l+1}|$. Ο γράφος $G_{l+1}|$ κατασκευάζεται από τον G_l βρίσκοντας ένα μέγιστο ταίριασμα (maximal matching) $M_l \subseteq E_l$ του G_l και ενώνοντας τους κόμβους που είναι γειτονικοί σε κάθε ακμή του ταιριάσματος. Στη διαδικασία αυτή δεν ενώνονται περισσότεροι από 2 κόμβοι αφού το ταίριασμα ενός γράφου είναι ένα σύνολο από ακμές εκ των οποίων



Σχήμα 2.2: Αναπαράσταση πολυεπίπεδης διαίρεσης γράφου

καμία δεν είναι γειτονική της άλλης. Οι κόμβοι που δεν είναι γειτονικοί σε καμία από τις ακμές του ταιριάσματος, απλώς αντιγράφονται στον $|G_{l+1}|$.

Όταν οι κόμβοι $v, u \in V_l$ ενώνονται για να δημιουργήσουν τον κόμβο $w \in V_{l+1}$, το βάρος του κόμβου w γίνεται ίσο με το άθροισμα των βαρών των κόμβων u και w, ενώ οι γειτονικές ακμές στο w γίνονται ίσες με την ένωση των ακμών που ήταν γειτονικές στο u και στο v χωρίς την ακμή (u, v). Αν υπήρχαν ακμές που ήταν γειτονικές τόσο για το u όσο και για το v $(u, p), (v, p)$, τότε το βάρος της ακμής γίνεται ίσο με το άθροισμα των βαρών των ακμών αυτών. Έτσι κατά τη διάρκεια μιας επιτυχημένης εκτράχυνσης, το βάρος τόσο των κόμβων όσο και των ακμών αυξάνεται.

Το μέγιστο ταίριασμα μπορεί να υπολογιστεί με διάφορους τρόπους [19]. Η μέθοδος που χρησιμοποιείται για τον υπολογισμού του ταιριάσματος επηρεάζει σημαντικά την ποιότητα της διχοτόμησης, καθώς και του χρόνου που απαιτείται κατά

τη διάρκεια της εξομάλυνσης. Το METIS υλοποιεί τέσσερις διαφορετικούς τρόπους ταιριάσματος. Θα περιγράψουμε σύντομα τους δύο από αυτούς.

Ο πρώτος τρόπος που ονομάζεται *random matching* (RM), υπολογίζει το μέγιστο ταίριασμα χρησιμοποιώντας έναν αλγόριθμο τυχαίων εκλογών. Ο αλγόριθμος επισκέπτεται τους κόμβους του γράφου με τυχαία σειρά. Αν ο κόμβος u δεν έχει ταιριαστεί ακόμα, τότε ένας αταίριαστος κόμβος v επιλέγεται τυχαία και η ακμή (u, v) προστίθεται στο ταίριασμα. Αν δεν υπάρχει αταίριαστος γειτονικός κόμβος v, τότε ο κόμβος u παραμένει αταίριαστος.

Ο δεύτερος τρόπος, ο οποίος ονομάζεται *heavy-edge matching* (HEM), υπολογίζει ένα ταίριασμα M_l, τέτοιο ώστε το βάρος των ακμών του M_l να είναι μεγάλο. Το *heavy-edge matching* υπολογίζεται χρησιμοποιώντας τυχαίο αλγόριθμο παρόμοιο με αυτόν που χρησιμοποιεί ο RM. Ο αλγόριθμος επισκέπτεται ξανά τους κόμβους με τυχαίο τρόπο. Όμως αντί να ταιριάζει τυχαία έναν κόμβο με κάποιον γειτονικό αταίριαστο, ο HEM ταιριάζει με τον κόμβο που είναι συνδεδεμένος με την πιο βαριά ακμή. Ως αποτέλεσμα, ο HEM μειώνει το άθροισμα των βαρών των ακμών κατά τη διαδικασία της εκτράχυνσης σε μεγαλύτερο βαθμό από τον RM. Πειραματικά έχει αποδειχθεί ότι ο αλγόριθμος HEM παράγει καλύτερα αποτελέσματα από τον RM, ενώ ο χρόνος που καταναλώνεται για βελτιώσεις είναι μικρότερος από αυτόν του RM.

Φάση Διαίρεσης (Partitioning Phase)

Η δεύτερη φάση ενός πολυεπίπεδου αλγορίθμου είναι να υπολογίσει μία διχοτόμηση ελάχιστων edge-cut του γράφου $G_k = (V_k, E_k)$ ώστε κάθε τμήμα να περιέχει περίπου το μισό βάρος κορυφών από τον αρχικό γράφο. Εφόσον κατά τη φάση της εκτράχυνσης, τα βάρη των ακμών και των κόμβων του εκτραχυμένου γράφου ορίστηκαν ώστε να αντικατοπτρίζουν τα βάρη των ακμών και κόμβων του λεπτότερου γράφου (finer graph), ο G_k περιέχει αρκετή πληροφορία ώστε έξυπνα να επιβάλει τα ισορροπημένα τμήματα και τις απαιτήσεις ελαχιστοποίησης των edge-cut.

Ένα τμήμα του G_k μπορεί να υπολογισθεί με διάφορους αλγορίθμους όπως (α) φασματική διχοτόμηση, (β) γεωμετρική διχοτόμηση (αν υπάρχουν συντεταγμένες), και (γ) συνδυαστικές μέθοδοι. Εφόσον το μέγεθος του γράφου G_k είναι μικρό ($|V_k| < 100$), αυτό το βήμα απαιτεί λίγο χρόνο. Το METIS χρησιμοποιεί τρεις διαφορετικούς αλγορίθμους graph growing heuristics και ακόμη έναν που στηρίζεται στη φασματική διχοτόμηση. Όλοι αυτοί οι αλγόριθμοι παράγουν περίπου παρόμοια τμηματοποίηση, με τους ευρετικούς να τα πηγαίνουν συνήθως λίγο καλύτερα.

Φάση Εξομάλυνσης (Uncoarsening Phase)

Κατά τη φάση της εξομάλυνσης, το τμήμα του εκτραχυμένου γράφου G_k προβάλλεται στον αρχικό γράφο περνώντας από τους γράφους $G_{k-1}, ..., G_1$. Αφού κάθε κόμβος $u \in V_l$ περιέχει ένα ξεχωριστό υποσύνολο U από κόμβους του V_{l-1}, η προβολή του τμήματος G_l στο G_{l-1} γίνεται απλώς αναθέτοντας τους κόμβους του U στο ίδιο τμήμα όπου ανήκει ο κόμβος u.

Επιπλέον, ακόμη και αν το τμήμα G_l είναι σε τοπικό ελάχιστο, η προβολή του τμήματος G_{l-1} μπορεί να μην είναι σε τοπικό ελάχιστο. Αφού ο G_{l-1} είναι λεπτότερος (finer), έχει περισσότερους βαθμούς ελευθερίας που μπορούν να χρησιμοποιηθούν για την περαιτέρω βελτίωση του τμήματος και επομένως τη μείωση των edge-cut. Συνεπώς μπορεί να είναι ακόμη δυνατή η βελτίωση της προβολής του τμήματος G_{l-1} από τοπικές ευρετικές βελτιώσεις. Γι' αυτό το λόγο, μετά την προβολή του τμήματος, εκτελείται ένας αλγόριθμος βελτίωσης τμήματος. Ο βασικός σκοπός ενός τέτοιου αλγορίθμου είναι να επιλέξει δύο υποσύνολα κόμβων, ένα από κάθε τμήμα που αν τα ανταλλάξουμε μεταξύ των τμημάτων, τα τμήματα που προκύπτουν έχουν μικρότερο edge-cut. Συγκεκριμένα, αν A και B είναι τα δύο τμήματα της διχοτόμησης, ένα αλγόριθμος βελτίωσης επιλέγει $A' \subset A$ και $B' \subset B$ τέτοια ώστε $A \setminus A' \cup B'$ και $B \setminus B' \cup A'$ να είναι μία διχοτόμηση με λιγότερα edge-cut.

Μία κλάση αλγορίθμων που τείνουν να παράγουν πολύ καλά αποτελέσματα είναι αυτοί που στηρίζονται στον Kernighan-Lin (KL) αλγόριθμο διαίρεσης. Ο αλγόριθμος KL είναι επαναληπτικός από τη φύση του. Ξεκινάει από ένα αρχικό τμήμα και σε κάθε επανάληψη βρίσκει υποσύνολα A' και B' με τις παραπάνω ιδιότητες. Αν υπάρχουν τέτοια υποσύνολα, τότε τα μεταφέρει στο άλλο μέρος και αυτό γίνεται το τμήμα για την επόμενη επανάληψη. Ο αλγόριθμος συνεχίζει επαναλαμβάνοντας την όλη διαδικασία. Αν δε μπορεί να βρει δύο τέτοια υποσύνολα, τότε ο αλγόριθμος τερματίζει, εφόσον το τμήμα βρίσκεται σε τοπικό ελάχιστο και δε μπορεί να γίνουν περαιτέρω βελτιώσεις από τον αλγόριθμο KL. Ο αλγόριθμος KL έχει βρεθεί ότι είναι αποτελεσματικός στην εύρεση τοπικά βέλτιστων τμημάτων όταν ξεκινάει από ένα σχετικά καλό αρχικό τμήμα. Δεδομένου ότι τα προβαλλόμενα τμήματα είναι ήδη αρκετά καλά, ο KL ουσιωδώς μειώνει τα edge-cut μέσα σε ένα μικρό αριθμό επαναλήψεων.

2.2 Δεδομένα - Οδικό δίκτυο Δυτικής Ευρώπης

Μία αναπαράσταση του οδικού δικτύου της Δυτικής Ευρώπης σε γράφο χρησιμοποιήθηκε για όλα τα πειράματα. Πρόκειται για δεδομένα που παρέχονται για επιστημονική χρήση από την PTV AG, Karlsruher και τα διανέμει το Ινστιτούτο Τεχνολογίας του Karlsruher [12]. Προς χάρη συντομίας παρακάτω θα αναφερόμαστε στο γράφο αυτό ως *ptv-europe*.Τα δεδομένα έχουν τα παρακάτω χαρακτηριστικά:

- Οι γράφοι είναι κατευθυνόμενοι. Συγκεκριμένα, οι μονόδρομοι αναπαριστώνται από μία κατευθυνόμενη ακμή και κάθε δρόμος διπλής κατεύθυνσης από δύο ξεχωριστές ακμές με αντίθετες κατευθύνσεις. Στα αρχικά δεδομένα κάποιοι δρόμοι ήταν καταγεγραμμένοι ως ¨κλειστοί για την γενική κυκλοφορία¨. Στα δεδομένα αυτά έχουν αρθεί αυτοί οι περιορισμοί ώστε να προκύψει ένας μεγαλύτερος γράφος. Οι ακμές αναπαριστούν είτε κομμάτια δρόμου είτε συνδέσεις πλοίων. Για κάθε κομμάτι δρόμου δίνεται η αντίστοιχη απόσταση και κατηγορία δρόμου (πχ. αυτοκινητόδρομος, εθνικός δρόμος, τοπικός δρόμος).

Για ακμές που αναπαριστούν διαδρομές πλοίων δίνονται οι αντίστοιχοι χρόνοι που ταξιδεύει το πλοίο.

- Οι γράφοι περιλαμβάνουν τις περισσότερες χώρες της Δυτικής Ευρώπης (εκτός από τις Τσεχία, Φινλανδία και Ιρλανδία), ενωμένες μαζί (να σημειωθεί ότι ο συνολικός γράφος δεν είναι συνδεδεμένος). Χρησιμοποιούνται δύο διαφορετικοί τύποι υπολογισμού του βάρους της ακμής:

  Time metric - Κάθε μονάδα αναπαριστά το $1/10$ του δευτερολέπτου: για τα πλοία υιοθετήθηκε ο χρόνος ταξιδιού, για τους δρόμους έχει θεωρηθεί μία μέση ταχύτητα 130 χμ/ώρα, 120 χμ/ώρα, .., 10 χμ/ώρα για τις 13 διαφορετικές κατηγορίες δρόμων (αυτή η μονάδα χρησιμοποιήθηκε στην παρούσα εργασία).

  Distance metric - Κάθε μονάδα αυτού του μέτρου αναπαριστά 1 μέτρο: για την περίπτωση των πλοίων υπολογίζεται η απόσταση μεταξύ των κόμβων έναρξης και τερματισμού.

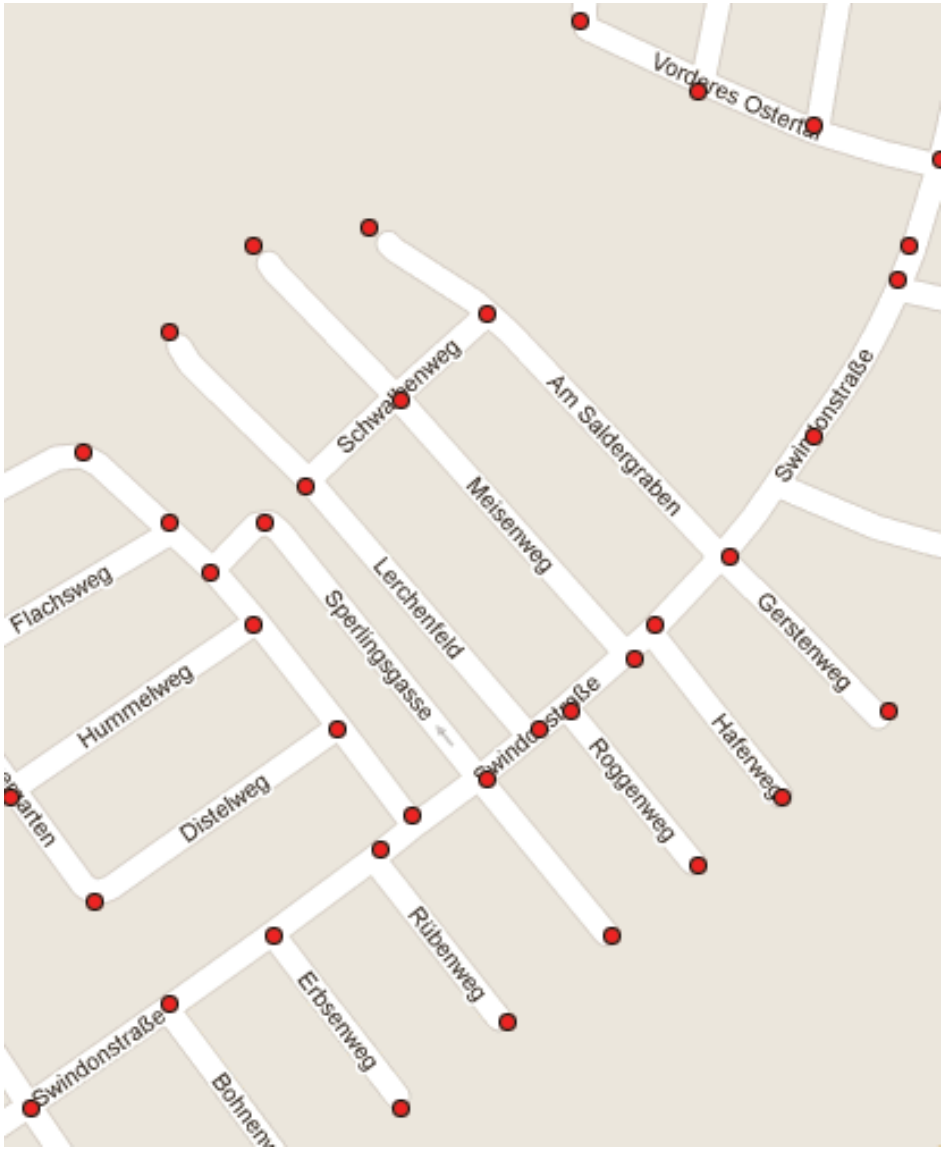
- Η συντεταγμένες αναπαριστούν το γεωγραφικό πλάτος και μήκος. Μία μονάδα αντιστοιχεί στο $1/100000$ της μοίρας ενώ η θετικοί και αρνητικοί αριθμοί αντικατοπτρίζουν αντίστοιχα το Ανατολικό και Δυτικό ημισφαίριο.

Στατιστικά γράφου

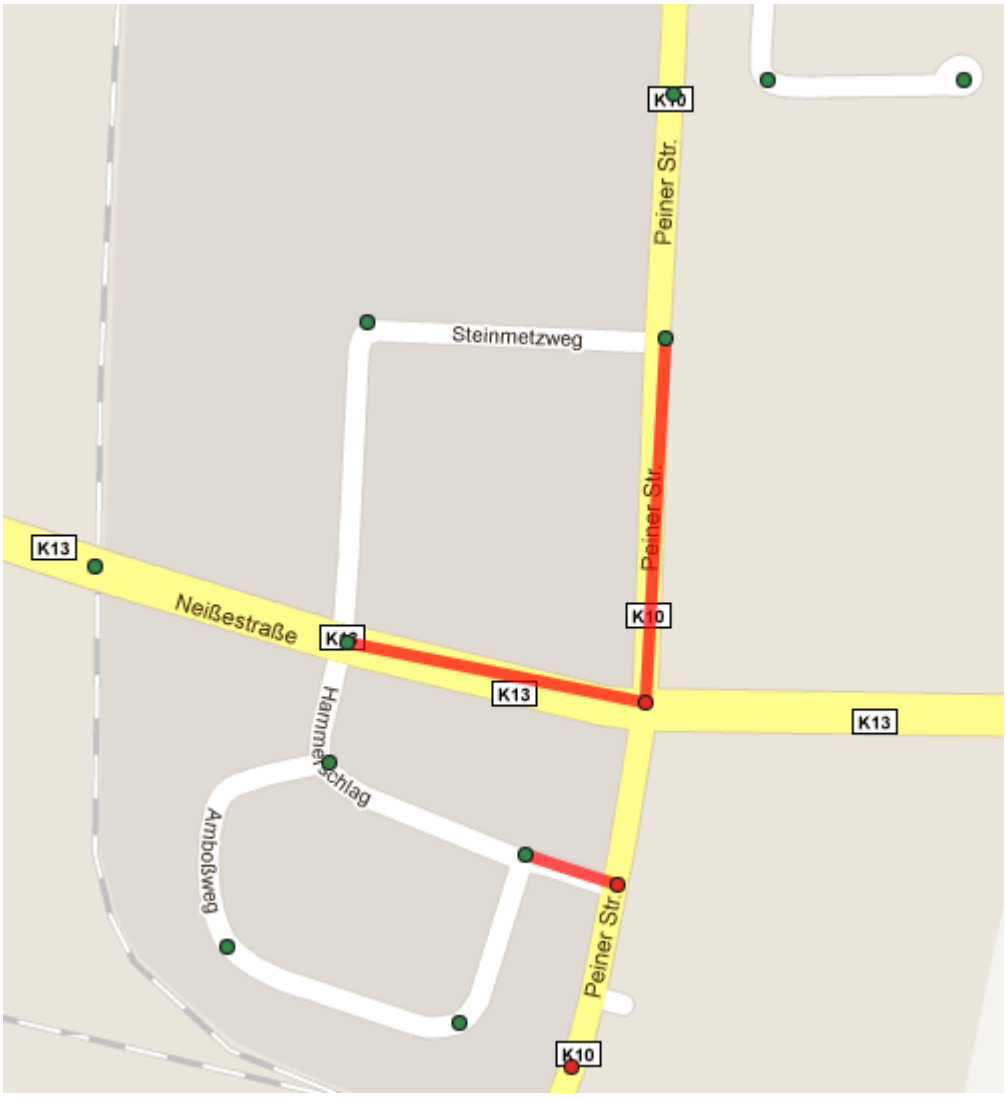
Ο γράφος αποτελείται από 18.010.173 κόμβους και 42.188.664 ακμές. Έχει βαθμό 2.33.

2.3 Εικονική αναπαράσταση

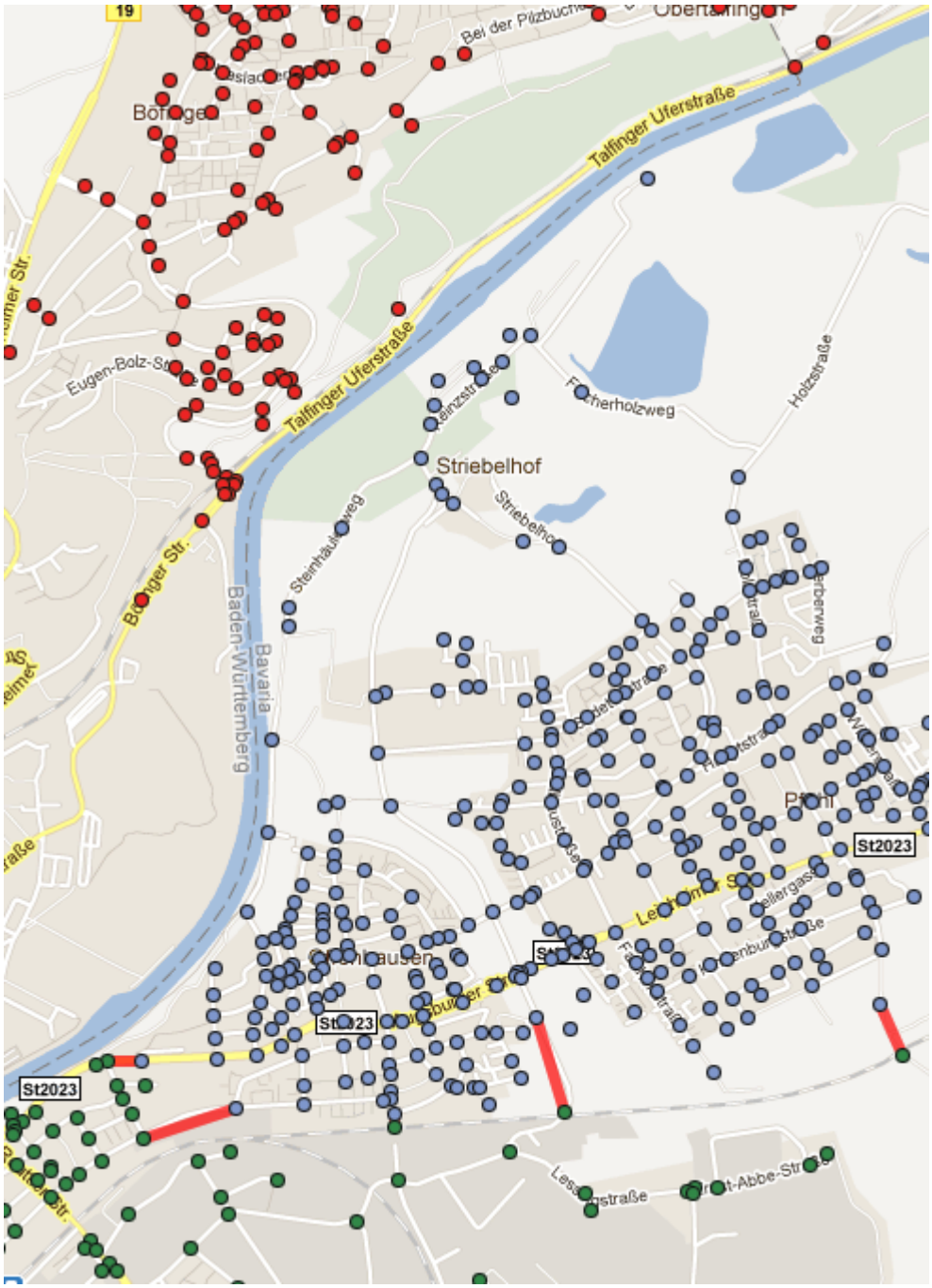
Για την καλύτερη εικονική αντίληψη της διαίρεσης του γράφου, κατασκευάστηκε ένα εργαλείο που τοποθετεί τους κόμβους κάθε κελιού στο χάρτη (με διαφορετικό χρώμα) μαζί με τις συνοριακές ακμές (κόκκινες γραμμές). Στο σχήμα 2.3 φαίνεται ξεκάθαρα η προβολή των κόμβων του γράφου στο οδικό δίκτυο. Στην ουσία πρόκειται για διασταυρώσεις ή τερματισμό του δρόμου. Κάθε κομμάτι δρόμου που φαίνεται στο χάρτη αποτελεί και μία ακμή στον αντίστοιχο γράφο. Στο σχήμα 2.5 είναι εντυπωσιακό ότι το METIS θα προτιμήσει να τοποθετήσει διαφορετικά κελιά εκατέρωθεν του ποταμού αλλά και της σιδηροδρομικής γραμμής. Είναι πολύ πιο σπάνιο δρόμοι (που θα δημιουργούσαν συνοριακούς κόμβους) να κόβουν αυτούς τους μεταφορικούς άξονες.



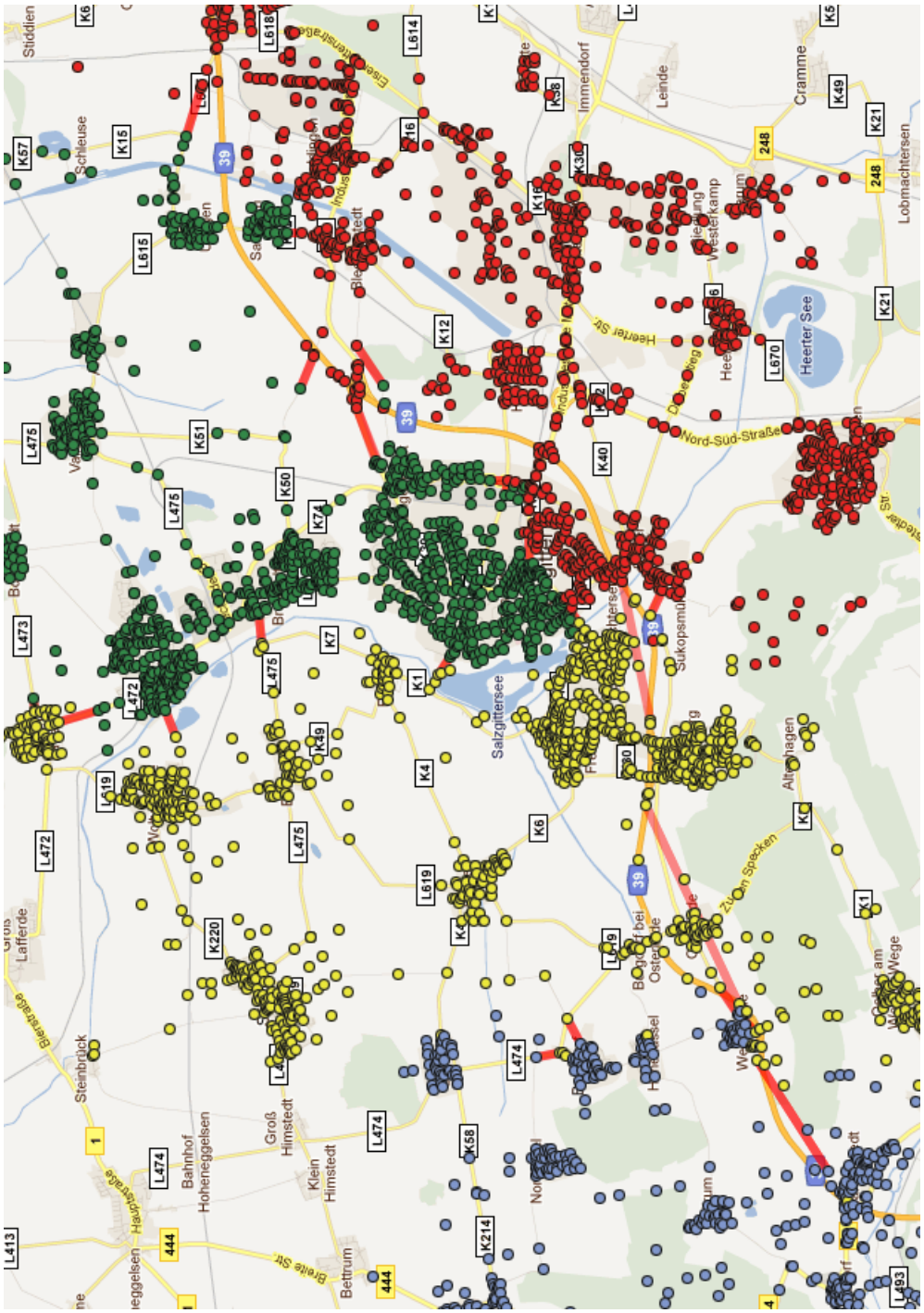
Σχήμα 2.3: Προβολή κόμβων επάνω στο χάρτη. Φαίνεται καθαρά ότι οι κόμβοι του γράφου τοποθετούνται σε διασταυρώσεις, στο τέλος του δρόμου ή σε αλλαγή του δρόμου.



Σχήμα 2.4: Μεγέθυνση σε περιοχή με τρεις συνοριακές ακμές πέντε συνοριακούς κόμβους. Φαίνεται ξεκάθαρα ότι το METIS δημιουργεί γειτονιές.



Σχήμα 2.5: Ο αλγόριθμος θα προτιμήσει να τοποθετήσει διαφορετικά κελιά εκατέρωθεν του ποταμού αλλά και της σιδηροδρομικής γραμμής. Είναι πολύ πιο σπάνιο δρόμοι να κόβουν αυτούς τους μεταφορικούς άξονες και αυτό επιβεβαιώνει τη σωστή επιλογή του METIS.



Σχήμα 2.6: Με τέσσερα διαφορετικά χρώματα φαίνονται τέσσερα μεγάλα γειτονικά κελιά.

2.4 Στατιστικά διαίρεσης γράφου

Ο συνολικός γράφος χωρίστηκε σε πλέγματα των 128, 256, 512, 1024, 2048, 4096, 8192, 16384 και 32768 κελιών για να μπορούν να συγκριθούν μεταξύ τους. Όπως έχουμε προαναφέρει ο αρχικός γράφος αποτελείται από 18.010.173 κόμβους και 42.188.664 ακμές. Επειδή είναι κατευθυνόμενος έχει βαθμό 2.33. Στον πίνακα 2.1 μπορεί κανείς να δει τα συνολικά αποτελέσματα για τους συνοριακούς κόμβους και τις συνοριακές ακμές που δημιουργήθηκαν.

Κελιά	Συνοριακοί κόμβοι	Συνοριακές ακμές
128	38155	36901
256	51170	49091
512	71814	68729
1024	101890	96882
2048	143553	136447
4096	206115	195999
8192	299978	285883
16384	445728	426791
32768	673066	648891

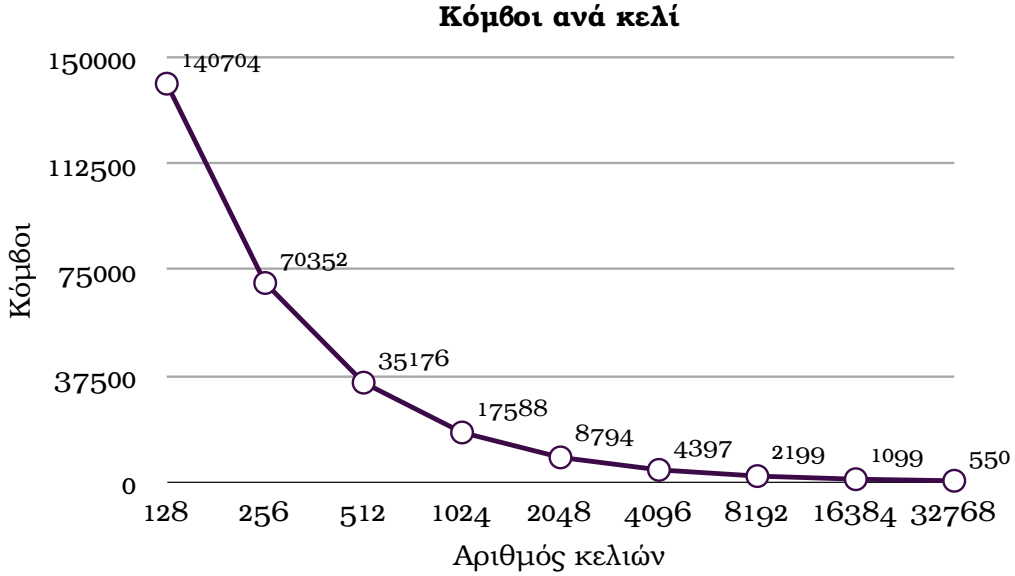
Πίνακας 2.1: Στατιστικά για τους κόμβους και τις ακμές συναρτήσει του αριθμού των κελιών.

	Μέση τιμή ανά κελί			
Κελιά	Κόμβοι	Συνοριακοί κόμβοι	Εσωτερικές ακμές	Συνοριακές ακμές
128	140704	298	329311	577
256	70352	200	164608	384
512	35176	140	82265	268
1024	17588	100	41105	189
2048	8794	70	20533	133
4096	4397	50	10252	96
8192	2199	37	5115	70
16384	1099	27	2549	52
32768	550	21	1268	40

Πίνακας 2.2: Στατιστικά ανά κελί για τους κόμβους και τις ακμές συναρτήσει του αριθμού των κελιών.

Ο αριθμός των συνοριακών κόμβων συνδέεται άμεσα με την ταχύτητα εκτέλεσης του αλγορίθμου σε ένα ολόκληρο κελί, αφού είναι ανάλογος με τον αριθμό των Dijkstra αλγορίθμων που θα εκτελεστούν. Ο γράφος ξεκινάει με 298 συνοριακούς

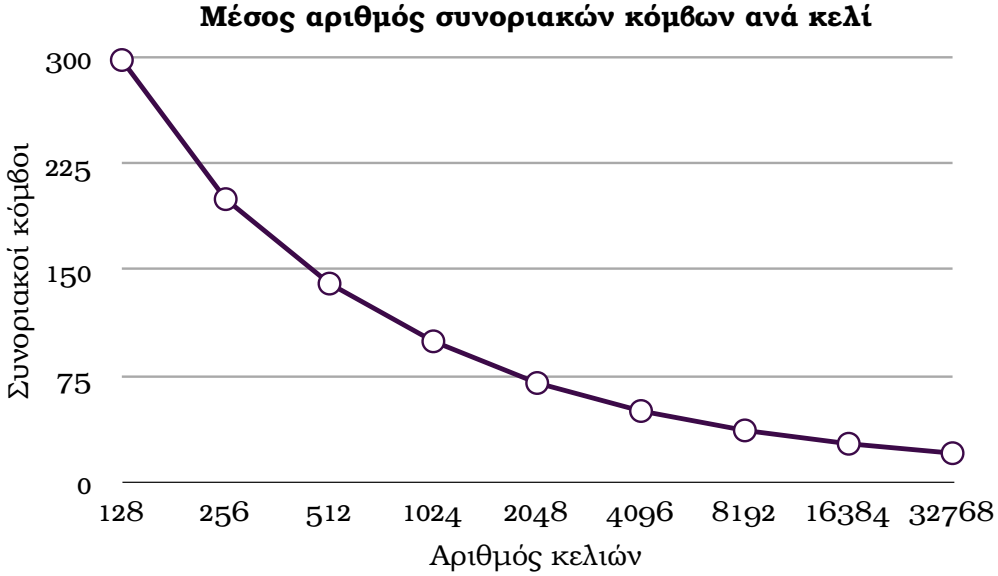
κόμβους ανά κελί για το μικρότερο κόψιμο και φτάνει μέχρι 21 συνοριακούς κόμβους ανά κελί για τα 32768 κελιά. Συνοπτικά μπορεί κανείς να δει τα στατιστικά του γράφου στον πίνακα 2.2 ενώ στο σχήμα 2.8 φαίνεται αναλυτικά η εξέλιξη των συνοριακών κόμβων συναρτήσει του αριθμού των κελιών. Αντίστοιχα στο σχήμα 2.9 φαίνεται το ποσοστό που καταλαμβάνουν οι συνοριακοί κόμβοι ανά κελί. Είναι φανερό ότι όσο αυξάνονται τα κελιά τόσο το ποσοστό των συνοριακών κόμβων αυξάνεται. Αυτό είναι απόλυτα φυσιολογικό, τα «κοψίματα» αυξάνονται και κατά συνέπεια αυξάνονται οι κόμβοι που βρίσκονται στο σύνορο αυτών των «κοψιμάτων». Αντίστοιχα στο πίνακα 2.2 φαίνεται ότι ο αριθμός των εσωτερικών ακμών ανά κελί μειώνεται όσο αυξάνεται ο αριθμός των κελιών. Αρχικά στα 128 κελιά, κάθε κελί έχει περίπου 330000 ακμές ενώ στα 32768 κελιά έχει μόλις 1268 ακμές. Το πρόβλημα του αλγορίθμου Dijkstra απλοποιείται σημαντικά αν θυμηθεί κανείς ότι η πολυπλοκότητα του είναι ανάλογη με τον αριθμό των ακμών. Η πολυπλοκότητα του συνολικού προβλήματος αναλύεται με λεπτομέρεια στα επόμενα κεφάλαια.



Σχήμα 2.7: Κόμβοι ανά κελί συναρτήσει του αριθμού των κελιών.

2.5 Ο αλγόριθμος Dijkstra

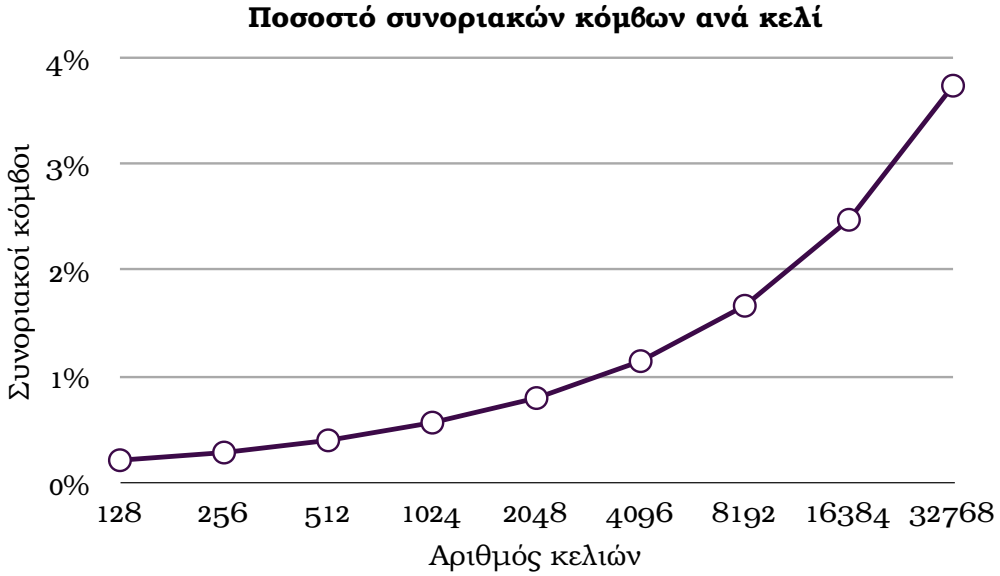
Ο γράφος επικάλυψης H (σχήμα 2.10) θα περιέχει όλους τους συνοριακούς κόμβους και τις συνοριακές ακμές του αρχικού γράφου G [15], [31]. Επιπροσθέτως θα περιέχει μια μορφή κλίκας για κάθε κελί C: Για κάθε ζευγάρι (u, w) συνοριακών κόμβων του C, κατασκευάζουμε μια ακμή (u, w) της οποίας το κόστος είναι το ίδιο με το κόστος του συντομότερου μονοπατιού (περιορισμένο στο C) μεταξύ των u και



Σχήμα 2.8: Συνοριακοί κόμβοι ανά κελί συναρτήσει του αριθμού των κελιών.

w. Οι συγκεκριμένες ακμές θα αναφέρονται στην παρούσα εργασία ως *ακμές επικάλυψης (overlay arcs)*. Προκειμένου να υπολογίσουμε τις συντομότερες αποστάσεις μεταξύ όλων των ζευγαριών συνοριακών κόμβων ενός κελιού, αρκεί να εκτελέσουμε έναν αλγόριθμο Dijkstra συντομότερης διαδρομής από κάθε συνοριακό κόμβο στο κελί C. Συνεπώς, εάν V_b ο αριθμός των συνοριακών κόμβων ενός κελιού, απαιτούνται συνολικά V_b αλγόριθμοι Dijkstra συντομότερης διαδρομής για κάθε κελί (συνάρτηση map του αλγορίθμου). Ο γράφος H που θα περιέχει α) όλους τους συνοριακούς κόμβους β) τις επιπλέον ακμές που θα προκύψουν από τους παραπάνω υπολογισμούς (ακμές επικάλυψης) και γ) τις συνοριακές ακμές θα αποτελεί γράφο επικάλυψης του G, δηλαδή η απόσταση μεταξύ δύο οποιωνδήποτε κόμβων στο H είναι ίδια με αυτή του G [18] (θυμίζω πως οι κόμβοι του H αποτελούν υποσύνολο αυτών του G). Ο γράφος επικάλυψης μπορεί να ανανεώνεται τμηματικά. Αν για κάποιο λόγο αυξηθεί η κυκλοφορία σε ένα δρόμο ή γίνονται έργα σε κάποιον άλλο, αρκεί να αναλυθεί ξανά ο υπογράφος όπου βρίσκεται ο συγκεκριμένος δρόμος. Αφού γίνει ο υπολογισμός το σύστημα έχει προσαρμοστεί αυτόματα στα νέα δεδομένα χωρίς να απαιτείται η ανάλυση ολόκληρου του γράφου.

Αν και ο αλγόριθμος Dijkstra δε χρησιμοποιείται για τον υπολογισμό του συντομότερου μονοπατιού στον αρχικό γράφο, χρησιμοποιείται για την κατασκευή του γράφου επικάλυψης H όπως αναφέραμε παραπάνω. Έτσι στις επόμενες παραγράφους θα αναφέρουμε κάποιες σημαντικές λεπτομέρειες για τον αλγόριθμο Dijkstra συντομότερης διαδρομής και τον τρόπο που υλοποιήθηκε [12].



Σχήμα 2.9: Ποσοστό συνοριακών κόμβων (σε σχέση με όλους τους κόμβους του κελιού) ανά κελί συναρτήσει του αριθμού των κελιών.

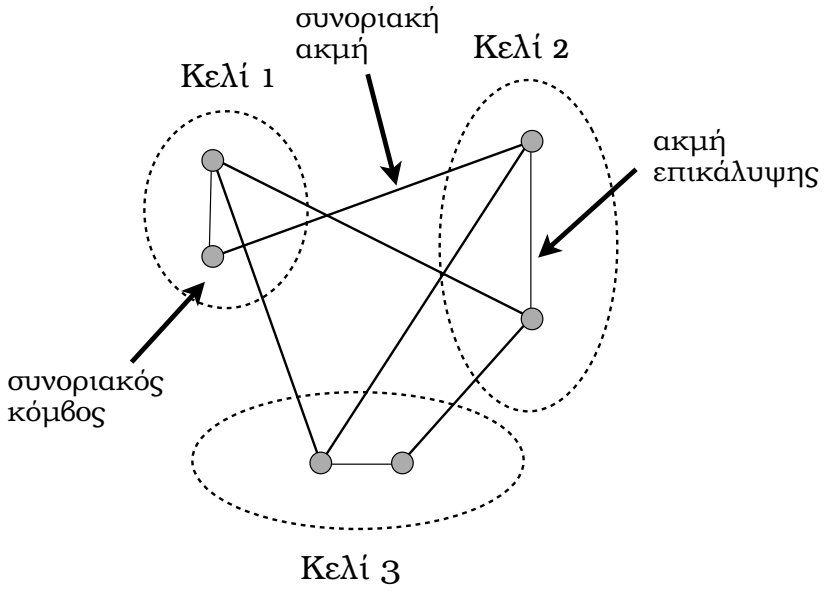
2.5.1 Περιγραφή

Ο αλγόριθμος του Dijkstra επιλύει το πρόβλημα των ομοαφετηριακών ελαφρύτατων διαδρομών για ένα κατευθυνόμενο γράφο με βάρη $G = (V, E)$ στην περίπτωση όπου όλα τα βάρη ακμών είναι μη αρνητικά [17]. Ως εκ τούτου, στην ενότητα αυτή υποθέτουμε ότι $w(u, v) \geq 0$ για όλες τις ακμές $(u, v) \in E$. Όπως θα δούμε, με καλή υλοποίηση ο χρόνος εκτέλεσης του αλγορίθμου του Dijkstra είναι μικρότερος από εκείνον του αλγορίθμου των Bellman-Ford.

Ο αλγόριθμος του Dijkstra τηρεί ένα σύνολο S από κόμβους των οποίων τα τελικά βάρη ελαφρύτατης διαδρομής από την αφετηρία s έχουν ήδη προσδιοριστεί. Ο αλγόριθμος επιλέγει επαναληπτικά τον κόμβο $u \in V - S$ με την ελάχιστη προεκτίμηση ελαφρύτατης διαδρομής, τον προσθέτει στο σύνολο S, και χαλαρώνει όλες τις ακμές που εκκινούν από αυτόν. Στην υλοποίηση που ακολουθεί, χρησιμοποιούμε μια ουρά προτεραιότητας ελαχίστου Q για τους κόμβους, με κλειδιά τις τιμές των πεδίων d.

2.5.2 Η διαδικασία χαλάρωσης

Η διαδικασία χαλάρωσης των ακμών στον αλγόριθμο του Dijkstra φαίνεται στο Σχήμα 2.11. Στη γραμμή 1 εκτελείται η συνήθης απόδοση αρχικών τιμών στα πεδία d και π, ενώ στη γραμμή 2 ορίζεται ως αρχικό σύνολο S το κενό σύνολο. Ο αλγόριθμος τηρεί την αναλλοίωτη συνθήκη ότι στην αρχή της κάθε επανάληψης του βρόχου $Q \neq 0$ ενόσω στις γραμμές 4-8 ισχύει ότι $Q = V - S$. Στη γραμμή 3 ορίζεται ως



Σχήμα 2.10: Εικονική αναπαράσταση δημιουργίας γράφου επικάλυψης. Οι ακμές επικάλυψης έχουν προκύψει από την ανάλυση κάθε υπογράφου.

αρχικό περιεχόμενο της ουράς προτεραιότητας ελαχίστου Q το σύνολο των κόμβων του V δεδομένου ότι τη συγκεκριμένη χρονική στιγμή έχουμε $S = 0$, η αναλλοίωτη συνθήκη ισχύει μετά τη γραμμή 3. Κάθε φορά που διατρέχεται ο βρόχος ενόσω στις γραμμές 4-8, αφαιρείται από την ουρά $Q = V - S$ ένας κόμβος u ο οποίος και προστίθεται στο σύνολο S. Με τον τρόπο αυτό, εξασφαλίζεται η τήρηση της αναλλοίωτης συνθήκης. (Την πρώτη φορά που εκτελείται ο βρόχος, έχουμε $u = s$.) Επομένως, ο κόμβος u έχει τη μικρότερη προεκτίμηση ελαφρύτατης διαδρομής από όλους τους κόμβους στο σύνολο $V - S$. Στη συνέχεια, ο αλγόριθμος χαλαρώνει στις γραμμές 7-8 όλες τις ακμές (u, v) που εκκινούν από τον u, ενημερώνοντας την προ-εκτίμηση $d[u]$ και το πεδίο προκατόχου $\pi[u]$ στην περίπτωση που η ελαφρύτατη

Algorithm 1 Dijkstra

```
S ← 0
Q ← V|G|
while Q ≠ 0 do
  u ← Pop(Q)
  S ← S ∪ {u}
  for all v ∈ Adj[u] do
    Relax(u, v, w)
  end for
end while
```

διαδρομή μέχρι τον v μπορεί να βελτιωθεί διερχόμενη από τον u. Σημειώστε ότι μετά τη γραμμή 3 δεν προστίθεται στην Q κανένας κόμβος, και ότι κάθε κόμβος αφαιρείται από την Q και προστίθεται στο S ακριβώς μία φορά. Κατά συνέπεια, ο βρόχος ενόσω στις γραμμές 4-8 επαναλαμβάνεται ακριβώς $|V|$ φορές.

Επειδή ο αλγόριθμος του Dijkstra προσθέτει πάντοτε στο σύνολο S τον «ελαφρύτερο» ή τον «πλησιέστερο» κόμβο από τον σύνολο $V - S$, λέμε ότι ακολουθεί άπληστη στρατηγική. Αν και οι άπληστες στρατηγικές εν γένει δεν δίνουν πάντοτε βέλτιστα αποτελέσματα, ωστόσο, αποδεικνύεται ότι ο αλγόριθμος του Dijkstra υπολογίζει όντως τις ελαφρύτατες διαδρομές. Αυτό που θα πρέπει ουσιαστικά να αποδειχθεί είναι ότι κάθε φορά που προστίθεται στο σύνολο S κάποιος κόμβος u, ισχύει ότι $d[u] = \delta(s, u)$ [32].

2.5.3 Κλειστή λίστα (Close list)

Η μέθοδος μπορεί να γίνει περισσότερο κατανοητή αν θεωρήσουμε το σύνολο W των κόμβων που έχουν ήδη περάσει από την ανοιχτή λίστα Q αλλά αυτή τη στιγμή δε βρίσκονται σε αυτήν [16]:

$$W = \{i | d_i < \infty, i \notin V\} \tag{2.2}$$

Μπορεί να αποδειχθεί ότι ως αποτέλεσμα της πολιτικής να αφαιρούνται οι κόμβοι με τη μικρότερη απόσταση από το Q, το W περιέχει κόμβους με «μικρές» τιμές καθ' όλη την εκτέλεση του αλγορίθμου, υπό την έννοια ότι

$$d_j \leq d_i, \epsilon\alpha\nu\ j \in W \kappa\alpha\iota\ i \notin W \tag{2.3}$$

Χρησιμοποιώντας αυτή την ιδιότητα και την υπόθεση ότι $a_{ij} \geq 0$, προκύπτει ότι όταν ένας κόμβος i αφαιρείται από το V, έχουμε, για όλα τα $j \in W$ για τα οποία (i, j) είναι ακμή,

$$d_j \leq d_i + a_{ij} \tag{2.4}$$

Συνεπώς, όταν ένας κόμβος μπαίνει στο W, παραμένει εκεί και το κόστος του δεν αλλάζει άλλο. Έτσι το W μπορεί να θεωρηθεί ως ένα σύνολο των μόνιμα υπολογισμένων κόμβων. Τα κόστος τους δεν πρόκειται να αλλάξει και μάλιστα είναι ίσο με το κόστος του συντομότερου μονοπατιού.

2.5.4 Χρόνος εκτέλεσης

Η χρονική αποδοτικότητα του αλγορίθμου Dijkstra εξαρτάται από τις δομές δεδομένων, οι οποίες χρησιμοποιούνται για την υλοποίηση της ουράς προτεραιότητας και την αναπαράσταση του γράφου εισόδου [9]. Η αποδοτικότητα αυτή είναι κλάσης $\Theta(|V^2|)$ για γράφους που αναπαριστώνται από τον πίνακα βαρών τους και για ουρές προτεραιότητας που υλοποιούνται ως μη διατεταγμένοι πίνακες. Για γράφους που αναπαριστώνται από τις λίστες γειτνίασης τους και για ουρές προτεραιότητας που

υλοποιούνται ως *min-heaps*, η αποδοτικότητα είναι κλάσης $O(|E|log(|V|))$. Ένα ακόμη πιο ικανοποιητικό άνω φράγμα μπορεί να επιτευχθεί για τον αλγόριθμο Dijkstra αν η ουρά προτεραιότητας υλοποιηθεί με τη χρήση μιας εξεζητημένης δομής δεδομένων η οποία ονομάζεται σωρός Fibonacci (*Fibonacci Heap*). Όμως η πολυπλοκότητα της δομής αυτής καθώς και μια σημαντική αρχική επιβάρυνση (overhead) καθιστούν αυτή τη βελτίωση ως έχουσα θεωρητική και μόνο αξία [3].

Στην υλοποίηση που έγινε σε Javascript για τον Dijkstra συντομότερης διαδρομής, κατασκευάστηκε ένα binary heap με τη βοήθεια πινάκων. Οι πίνακες είναι το δυνατό σημείο της Javascript και γι' αυτό προτιμήθηκε για την υλοποίηση της ανοιχτής λίστας. Από την άλλη πλευρά οι κόμβοι κάθε κελιού μειώνονται σημαντικά όσο μεγαλώνει ο αριθμός των κελιών. Για παράδειγμα στο «κόψιμο» των 32768 κελιών, καθένα από αυτά περιέχει μόλις 550 κόμβους κατά μέσο όρο. Έτσι η απόδοση της ανοιχτής λίστας δεν έχει και τόσο καθοριστική σημασία.

2.5.5 Ακμές επικάλυψης (Overlay arcs)

Στο Σχήμα 2.12 μπορούμε να δούμε ένα παράδειγμα δημιουργίας μιας ακμής επικάλυψης. Όπως αναφέρθηκε και παραπάνω το κόστος κάθε ακμής επικάλυψης (u, w) ισούται με την απόσταση που ο αλγόριθμος Dijkstra υπολογίζει *εντός του κελιού* όπου ανήκουν οι κόμβοι u και w. Συνεπώς η ακμή (u, w) δεν είναι πάντα η βέλτιστη διαδρομή από το u στο w. Αυτό συμβαίνει διότι οι ακμές επικάλυψης προκύπτουν μόνο από πληροφορία εντός του κελιού. Υπάρχει δηλαδή η πιθανότητα τα u και w να έχουν βέλτιστο μονοπάτι που περιέχει κόμβους που βρίσκονται έξω από το συγκεκριμένο κελί. Ένα τέτοιο παράδειγμα φαίνεται στο Σχήμα 2.13. Η διακεκομμένη διαδρομή που προκύπτει από κόμβους εντός του κελιού Υ δεν είναι η βέλτιστη. Αντίθετα υπάρχει ένας κόμβος στο κελί Χ μέσω του οποίου δημιουργείται το βέλτιστο μονοπάτι στο συνολικό γράφο για τους κόμβους Α και Β. Έτσι οι ακμές επικάλυψης είναι τα βέλτιστα μονοπάτια δεδομένου του κελιού και όχι ολόκληρου του γράφου. Η ύπαρξη αυτών των μη βέλτιστων ακμών δε δημιουργεί πρόβλημα αφού όταν επιλύσει κάποιος το γράφο επικάλυψης (που περιέχει και τις συνοριακές ακμές) θα βρει τη σωστή διαδρομή. Τέλος κατά την κατασκευή των ακμών επικάλυψης εφαρμόζεται παράλληλα μια σημαντική βελτιστοποίηση του αλγορίθμου που μας προσφέρει μείωση ακμών (edge reduction). Οι ακμές επικάλυψης που θα προέκυπταν από διαδρομές που περνούν μέσα από συνοριακούς κόμβους αγνοούνται. Αυτές οι ακμές αναπαριστώνται από δύο ή περισσότερες ακμές (που είναι ακμές επικάλυψης) στον επικαλυπτόμενο γράφο, συνεπώς η ύπαρξη τους θα ήταν περιττή. Μια τέτοια περίπτωση απεικονίζεται στο Σχήμα 2.14.

2.6 Map-Reduce

Έχουμε χωρίσει τον ptv-europe σε κελιά, έχουμε περιγράψει τη διαδικασία για τον υπολογισμό των ακμών επικάλυψης, δεν έχουμε περιγράψει όμως ποια είναι η ιδέα πίσω από το map-reduce. Σκοπός αυτής της ενότητας είναι να μυήσει τον αναγνώστη στο γενικότερο προγραμματιστικό μοντέλο του map-reduce αλλά και να τον εξειδικεύσει στην υλοποίηση που έγινε για τον υπολογισμό των ακμών επικάλυψης.

2.6.1 Περιγραφή προβλήματος

Το map-reduce είναι ένα προγραμματιστικό μοντέλο που σκοπό έχει την παραλληλοποίηση διαδικασιών. Στηρίζεται όπως και το όνομα του στο συνδυασμό των συναρτήσεων *map* και *reduce* όπως σε μία συναρτησιακή γλώσσα σαν τη Lisp. Στο βήμα map δέχεται ως είσοδο μία συνάρτηση και μια ακολουθία δεδομένων. Στη συνέχεια εφαρμόζει τη συνάρτηση σε κάθε δεδομένο της ακολουθίας. Στο reduce βήμα συνδυάζονται όλα τα στοιχεία της ακολουθίας χρησιμοποιώντας δυαδικούς τελεστές. Για παράδειγμα, μπορεί να χρησιμοποιηθεί ο τελεστής ¨+¨ για να προσθέσει όλα τα στοιχεία της ακολουθίας. Αναπτύχθηκε από εταιρίες όπως η Google [22] για την επεξεργασία μεγάλου όγκου δεδομένων, όπως έγγραφα που η μηχανή αναζήτησης βρίσκει ή αρχεία ιστορικού. Αυτά τα δεδομένα είναι τόσο μεγάλα, που απαιτείται ο διαμοιρασμός τους μεταξύ χιλιάδων μηχανημάτων ώστε να μπορεί να γίνει η επεξεργασία σε λογικό χρόνο. Αυτό απαιτεί παράλληλες αρχιτεκτονικές αφού οι ίδιοι υπολογισμοί εκτελούνται σε κάθε επεξεργαστή, αλλά με διαφορετικά δεδομένα. Όσον αφορά τη Google, το map-reduce που έχει αναπτύξει αποτελεί μια αφηρημένη διεπαφή που επιτρέπει στους μηχανικούς να εκτελούν απλούς υπολογισμούς, ενώ παράλληλα υποκρύπτουν τις λεπτομέρειες της παραλληλοποίησης, του διαμοιρασμού των δεδομένων, της εξισορρόπησης φόρτου καθώς και της ανοχής σε σφάλματα. Στο Σχήμα 2.16 μπορούμε να δούμε μια σχηματική αναπαράσταση της map-reduce αρχιτεκτονικής στη γενική της μορφή.

2.6.2 Παραδείγματα

Παρακάτω μπορούμε να δούμε διάφορα ενδιαφέροντα προγράμματα που μπορούν να εκφραστούν εύκολα ως υπολογισμοί map-reduce:

- *Κατανεμημένο grep (Distributed grep):* Η συνάρτηση map επιστρέφει μια γραμμή εάν ταιριάζει σε κάποιο συγκεκριμένο πρότυπο. Η συνάρτηση reduce είναι μια συνάρτηση αναγνώρισης που απλώς αντιγράφει τα ενδιάμεσα δεδομένα στην έξοδο.

- *Καταμέτρηση της συχνότητας των προσβάσεων σε κάποια URL διεύθυνση (Count of URL Access Frequency):* Η συνάρτηση map επεξεργάζεται αρχεία ιστορικού

αιτήσεων για ιστοσελίδες και επιτρέφει (URL, N) για N αιτήσεις που θα ανιχνεύσει στο αρχείο που εξετάζει. Η συνάρτηση reduce προσθέτει όλα μαζί τα αποτελέσματα για το ίδιο URL δίνοντας έτσι ζευγάρια (URL, συνολικές αιτήσιες).

- *Αντίστροφος γράφος web συνδέσμων (Reverse Web-Link Graph):* Η συνάρτηση map επιστρέφει ζευγάρια (στόχος, αφετηρία) για κάθε σύνδεσμο προς κάποια διεύθυνση «στόχο», που βρίσκεται σε μία σελίδα ονόματι «αφετηρία». Η συνάρτηση reduce ενώνει τη λίστα όλων των «αφετηριών» προς ένα συγκεκριμένο «στόχο» επιστρέφοντας τα ζευγάρια (στόχος, λίστα αφετηριών).

- *Διάνυσμα όρων ανά ιστοσελίδα (Term-Vector per Host):* Ένα διάνυσμα όρων συνοψίζει τις πιο σημαντικές λέξεις που εμφανίζονται σε ένα έγγραφο ή ένα σύνολο εγγράφων, σαν μία λίστα από ζευγάρια (λέξη, συχνότητα). Η συνάρτηση map επιστρέφει ένα ζευγάρι (ιστοσελίδα, διάνυσμα όρων) για κάθε έγγραφο εισόδου (όπου η ιστοσελίδα προκύπτει από τη διεύθυνση του εγγράφου). Τέλος η συνάρτηση reduce συγκεντρώνει για μια συγκεκριμένη ιστοσελίδα όλα τα διανύσματα όρων, τα προσθέτει μεταξύ τους, πετώντας τους μη συχνά εμφανιζόμενους όρους και επιστρέφοντας ένα τελικό ζευγάρι (ιστοσελίδα, διάνυσμα όρων).

- *Αντίστροφο ευρετήριο (Inverted Index):* Η συνάρτηση map διαβάζει κάθε έγγραφο και επιστρέφει μια ακολουθία ζευγαριών (λέξη, έγγραφο). Η συνάρτηση reduce δέχεται όλα τα ζευγάρια για μια συγκεκριμένη λέξη, ταξινομεί τα έγγραφα και επιστρέφει ένα ζευγάρι (λέξη, λίστα εγγράφων). Το σύνολο όλων αυτών των ζευγαριών αποτελεί ένα απλό αντίστροφο ευρετήριο. Εύκολα κανείς μπορεί να επεκτείνει τους υπολογισμούς, αποθηκεύοντας παράλληλα και τη θέση της λέξης μέσα στο κείμενο.

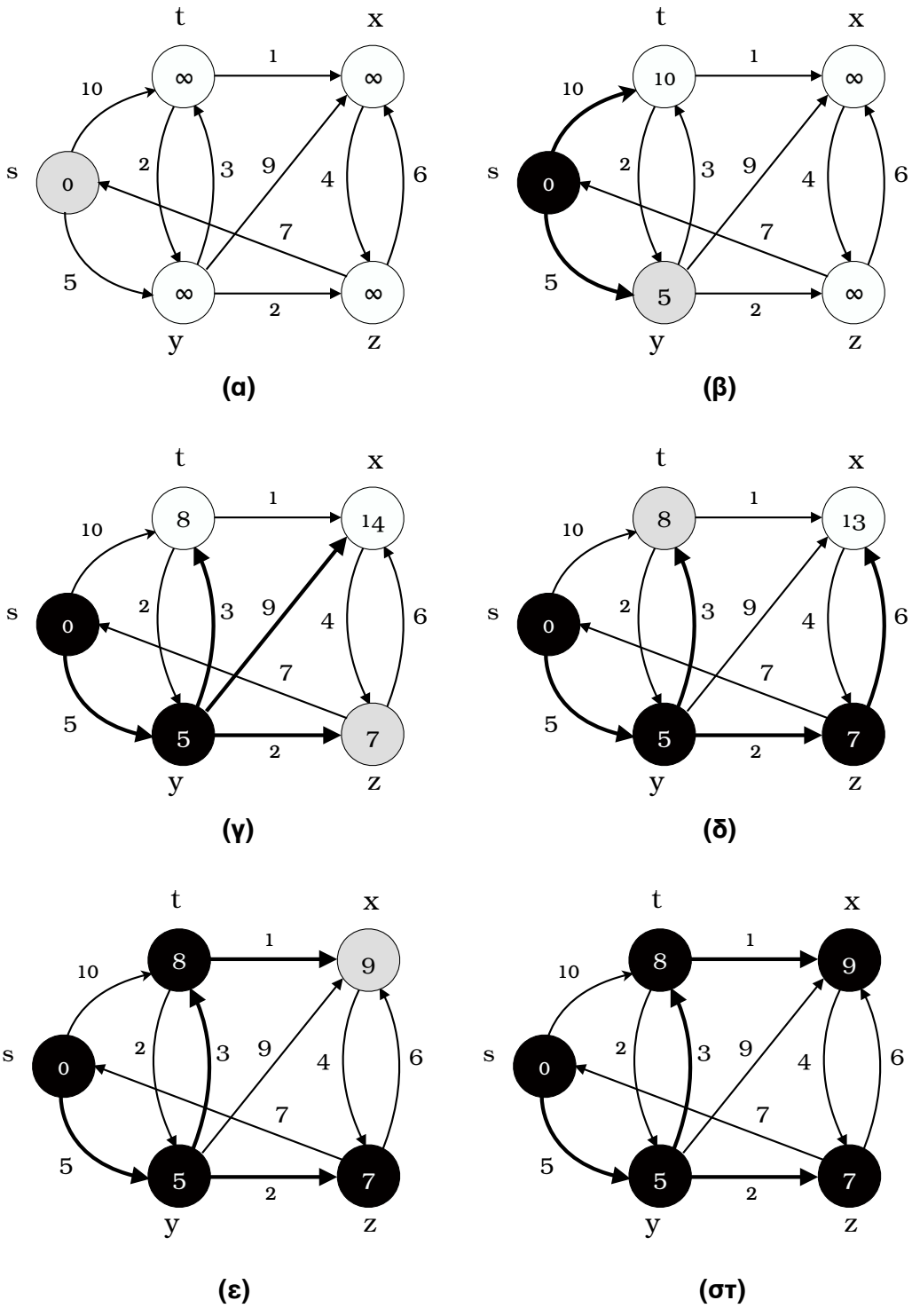
2.6.3 Υλοποίηση

Η εφαρμογή του map-reduce στην παρούσα έρευνα διαφέρει αρκετά από τη γενικευμένη παρουσίαση που έγινε παραπάνω. Το μεγαλύτερο σημείο διαφοροποίησης έγκειται στη φυσική υπόσταση των workers. Δεν πρόκειται για μηχανήματα ενός cluster ή κάποιας φάρμας διακομιστών, αλλά για απλούς χρήστες μιας ιστοσελίδας που ίσως ποτέ να μην καταλάβουν ότι ο επεξεργαστής τους, έστω και για ελάχιστα εκατοστά του δευτερολέπτου, αξιοποιήθηκε ως ένας εργάτης (worker) για την επίλυση ενός μεγάλου map-reduce προβλήματος. Η επιλογή αυτή έχει τα θετικά αλλά και τα αρνητικά της όπως θα δούμε παρακάτω:

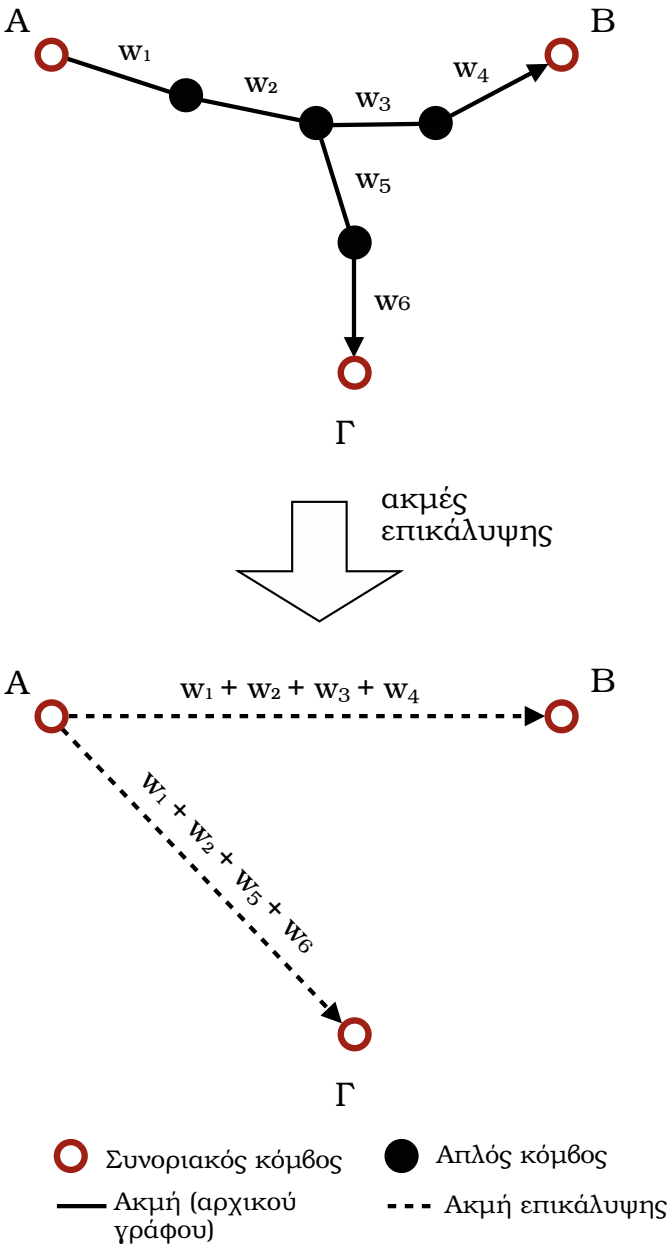
+ Δεν απαιτείται η ύπαρξη μεγάλου υπολογιστικού συστήματος το οποίο συνεπάγεται εντυπωσιακά μειωμένο κόστος.

+ Μπορεί να κάνει scale όσο αυξάνονται οι επισκέπτες ή τα websites όπου είναι εγκατεστημένο το σύστημα.
+ Αξιοποιούνται οι επεξεργαστές χιλιάδων χρηστών που διαφορετικά θα έμεναν αδρανείς.
– Υπάρχει καθυστέρηση δικτύου που σε ένα cluster περιβάλλον είναι πολύ μικρότερη.
– Μοναδικό προγραμματιστικό εργαλείο είναι η Javascript που δε φημίζεται για την ταχύτητα της.
– Ο αλγόριθμος πρέπει να τρέξει τόσο γρήγορα ώστε να περάσει απαρατήρητος από τον χρήστη.
– Τίθενται αρκετά ζητήματα προστασίας της πληροφορίας.

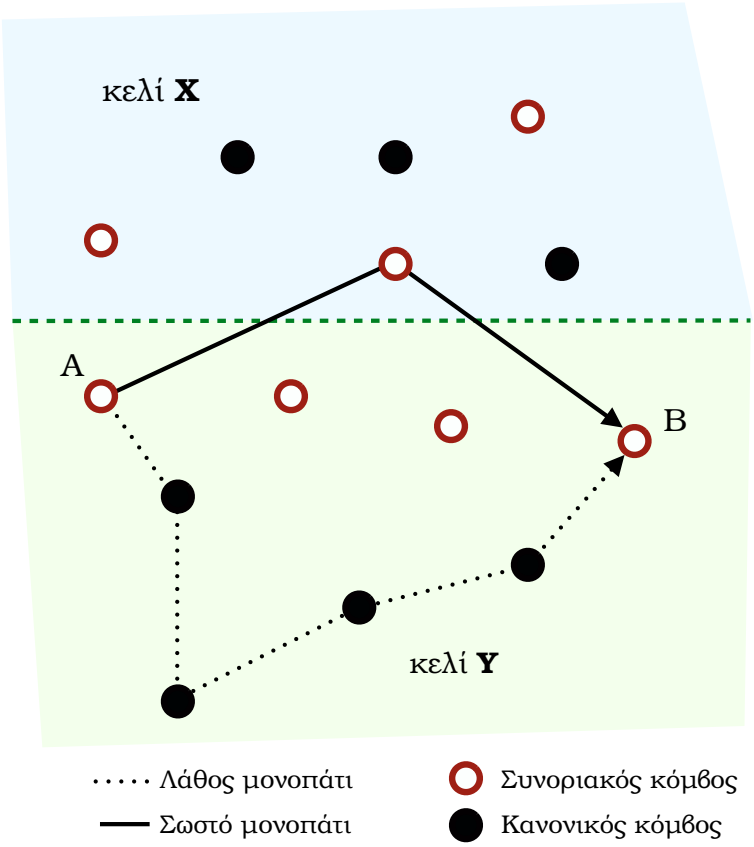
Παράλληλα ο κεντρικός εργάτης (master worker) είναι ένα κεντρικός διακομιστής όπως και στο Σχήμα 2.16 ο οποίος αναλαμβάνει τη διανομή των partitions στους εργάτες - επισκέπτες. Η map συνάρτηση υπολογίζει το κόστος των συντομότερων μονοπατιών από όλα τα border nodes προς όλα τα άλλα border nodes όπως έχουμε αναφέρει στο Κεφάλαιο 3 με τη χρήση του αλγορίθμου Dijkstra. Η συνάρτηση reduce δεν καλύπτεται στην παρούσα έρευνα και αποτελεί πολύ ενδιαφέρον κομμάτι μελλοντικής ανάλυσης. Στο σχήμα 2.17 μπορούμε να δούμε την αρχιτεκτονική που υλοποιήθηκε.



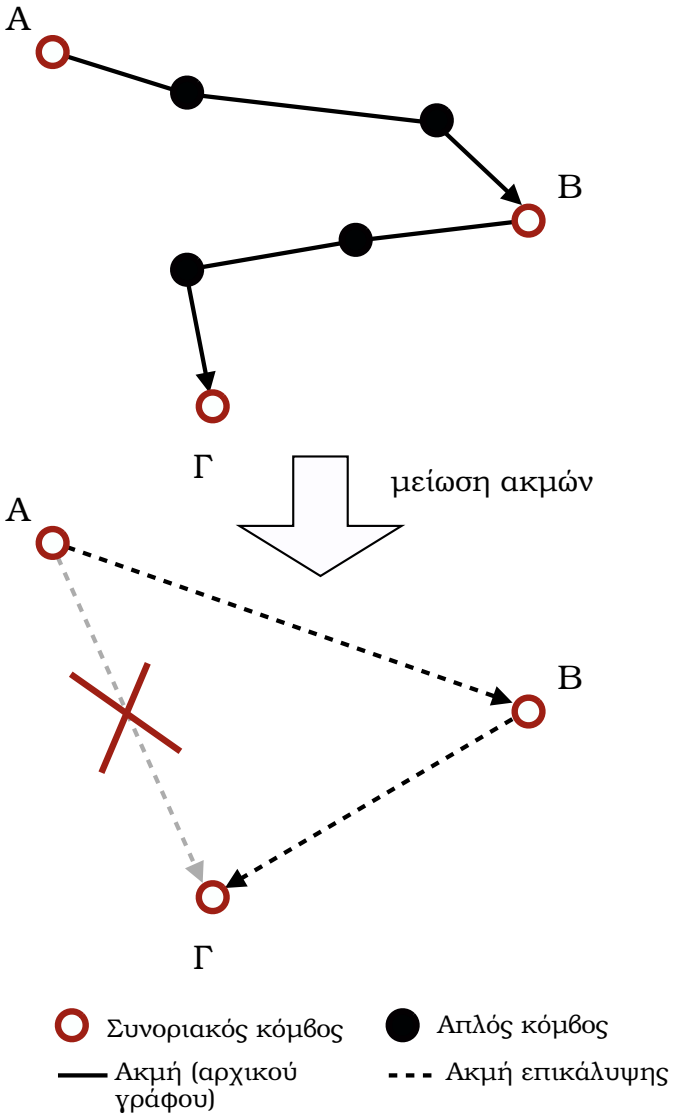
Σχήμα 2.11: Η λειτουργία του αλγορίθμου Dijkstra



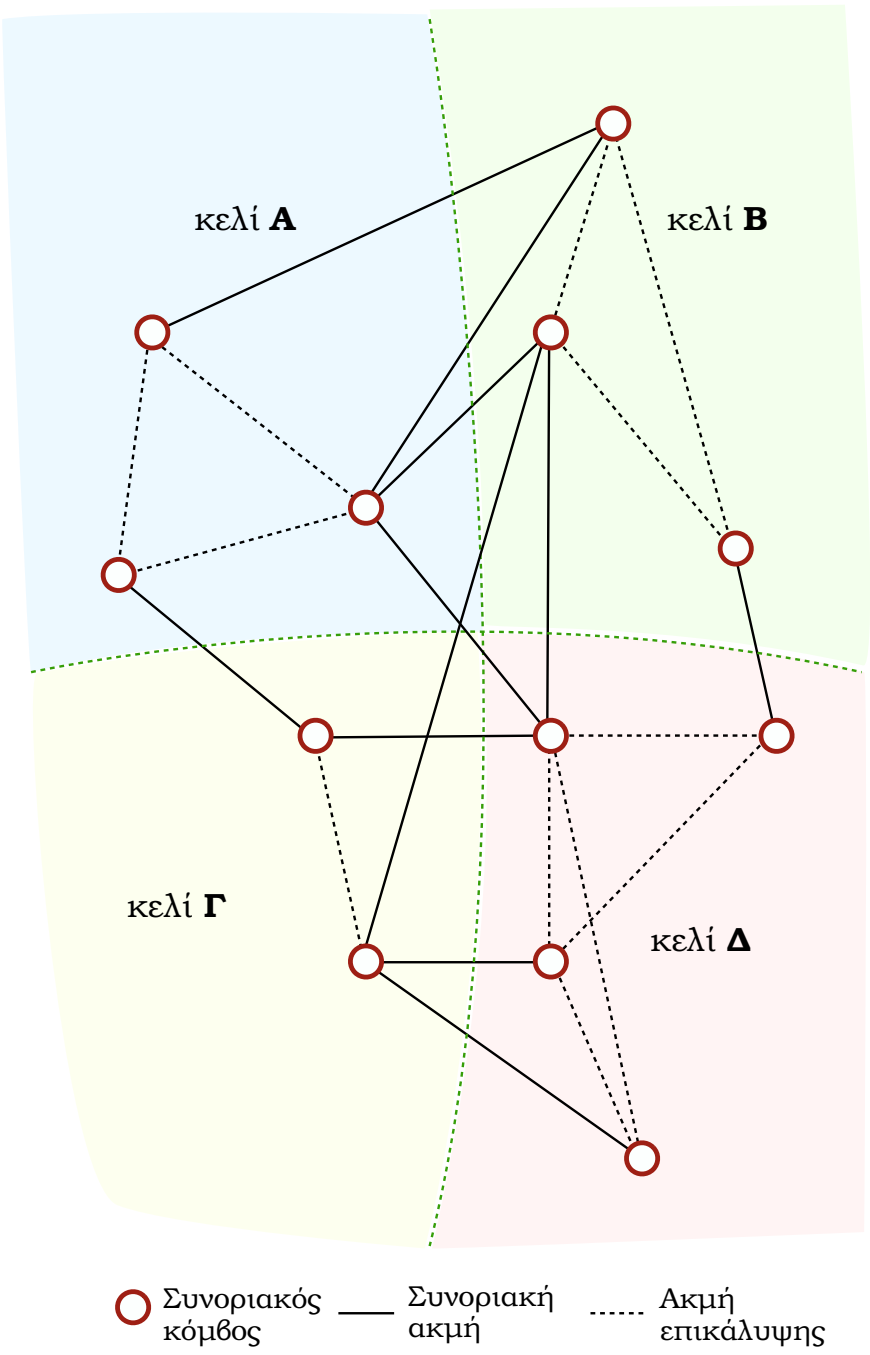
Σχήμα 2.12: Δημιουργία των ακμών επικάλυψης από τις ακμές του αρχικού γράφου. Κάθε ακμή επικάλυψης (u, w) έχει ως κόστος την απόσταση του συντομότερου μονοπατιού που υπολογίζει ο αλγόριθμος Dijkstra από το u στο w.



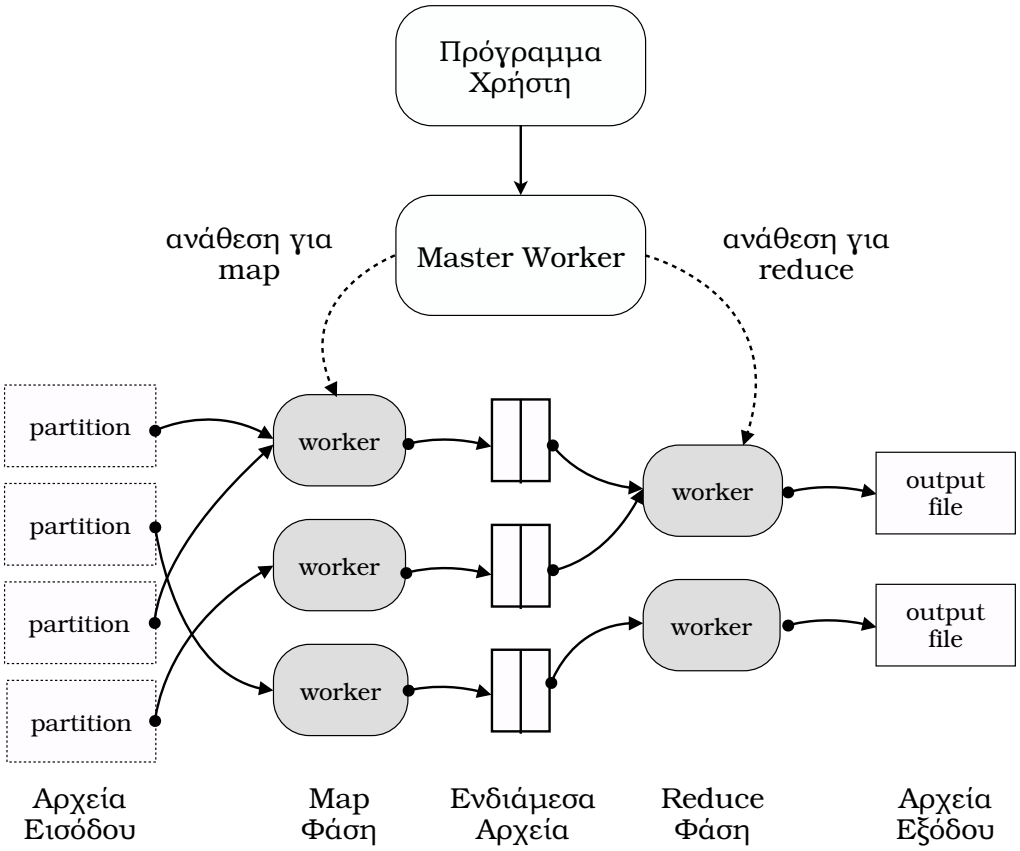
Σχήμα 2.13: Η ακμή επικάλυψης που προκύπτει από τη διακεκομμένη διαδρομή αντιπροσωπεύει λάθος μονοπάτι (δεν είναι το βέλτιστο δεδομένου του συνολικού γράφου). Αντίθετα το μονοπάτι που προκύπτει με τη χρήση του συνοριακού κόμβου από το κελί X είναι το βέλτιστο δεδομένου του συνολικού γράφου.



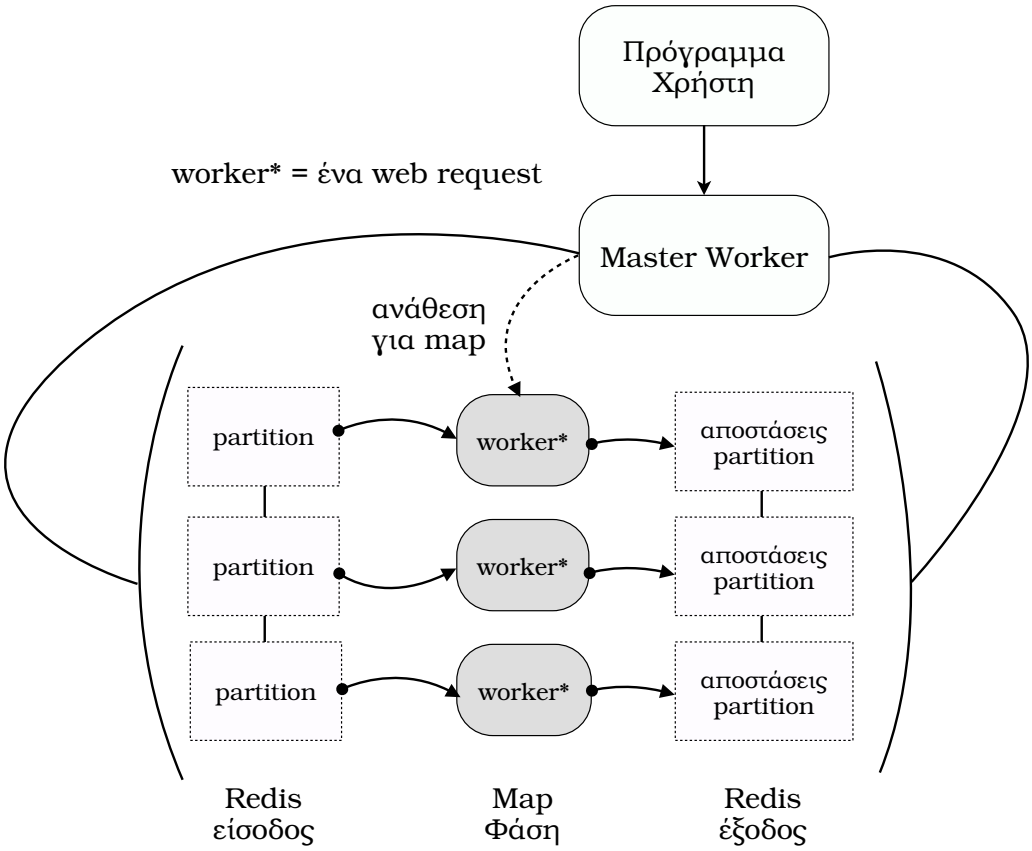
Σχήμα 2.14: Αγνοώντας τις περιττές ακμές επικάλυψης επιτυγχάνεται μείωση ακμών. Η ακμή ΑΓ αγνοείται αφού το βάρος της θα ήταν ίσο με ΑΒ+ΒΓ και συνεπώς είναι περιττή.



Σχήμα 2.15: Ο γράφος επικάλυψης



Σχήμα 2.16: Γενικευμένη αναπαράσταση μιας map-reduce αρχιτεκτονικής



Σχήμα 2.17: Αναπαράσταση map-reduce αρχιτεκτονικής που υλοποιήθηκε

Κεφάλαιο 3

Αρχιτεκτονική Συστήματος

Σε αυτό το κεφάλαιο θα παρουσιαστεί η αρχιτεκτονική του συστήματος που υλοποιήθηκε. Στις πρώτες ενότητες φαίνεται το σχήμα της βάσης δεδομένων και τα ερωτήματα που καλείται να απαντήσει. Στη συνέχεια περιγράφεται η δομή του διακομιστή και τα υποσυστήματα που χρησιμοποιήθηκαν για τη λειτουργία του.

3.1 Βάση Δεδομένων

3.1.1 Οργάνωση

Τα δεδομένα που πρέπει να αποθηκευτούν στη βάση δεδομένων αναπαριστούν τον κατευθυνόμενο γράφο, αφού πρώτα έχει διαιρεθεί σε κελιά. Η οργάνωση του έγινε σε δύο πίνακες με σκοπό να εξυπηρετούν τις ανάγκες των ερωτημάτων που θα γίνουν κατά την εκτέλεση του map-reduce. Κάθε εγγραφή του πρώτου αναπαριστά μία ακμή του γράφου με το αντίστοιχο βάρος, τους κόμβους που ξεκινάει και τερματίζει, καθώς και το κελί στο οποίο ανήκουν αυτοί οι κόμβοι ανάλογα με το «κόψιμο». Τα ερωτήματα μας γίνονται με βάση το κελί επομένως αποδεχόμαστε την επανάληψη δεδομένων και την έλλειψη απόλυτης κανονικοποίησης των πινάκων προκειμένου να κερδίσουμε σε ταχύτητα. Ένα συνδυαστικό B-tree κλειδί έχει τοποθετηθεί στα δύο πεδία που αφορούν τα κελιά βελτιώνοντας πολύ την απόδοση των ερωτημάτων.

Ο δεύτερος πίνακας αφορά τους συνοριακούς κόμβους. Για κάθε κελί μας δίνει όλους αυτούς τους κόμβους οι οποίοι είναι συνοριακοί κόμβοι. Στην ουσία αποτελεί προ-υπολογισμένη πληροφορία η οποία μπορεί να προκύψει με το κατάλληλο ερώτημα από τον πρώτο πίνακα. Θα μειωνόταν σημαντικά η απόδοση του συστήματος αν αυτή η πληροφορία δεν ήταν προ-υπολογισμένη μιας και απαιτούνται αρκετά δευτερόλεπτα για τον υπολογισμό της. Σε αυτό τον πίνακα ο αύξων αριθμός του κελιού αποτελεί primary key του πίνακα επομένως δε χρειάστηκε να προστεθεί κάποιο ειδικό κλειδί αφού όλα τα ερωτήματα γίνονται με βάση αυτό το πεδίο [8, 24].

3.1.2 Σχήμα Βάσης Δεδομένων

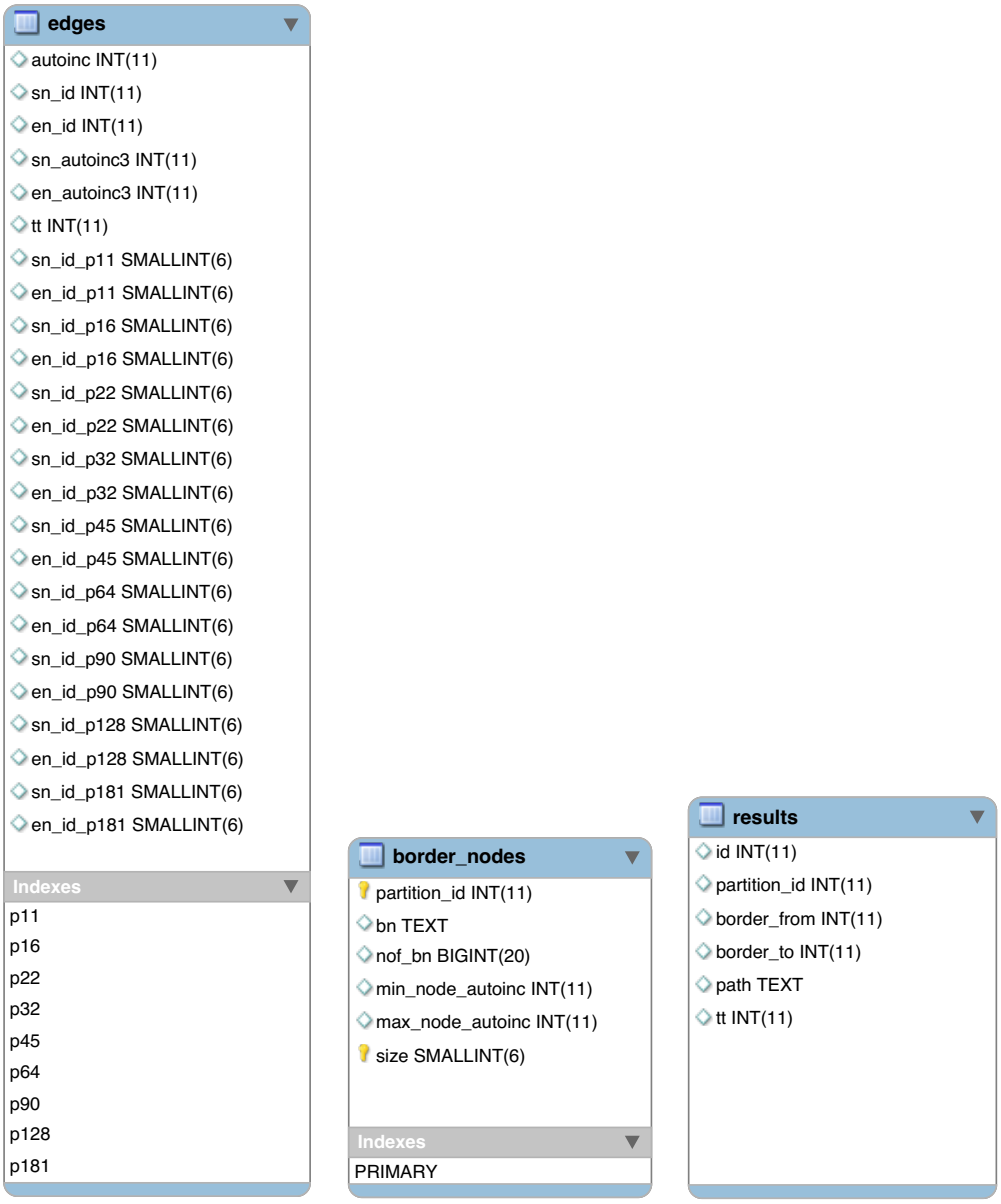
Όπως μπορεί κανείς να δεις στο σχήμα 3.1, η δομή της βάσης δεδομένων είναι αρκετά απλοϊκή. Ο πίνακας 3.1(α') περιέχει τις ακμές του κατευθυνόμενου γράφου. Συγκεκριμένα τα πεδία *sn_id_pN, en_id_pN* υποδηλώνουν τους κόμβους που ξεκινάει και καταλήγει η ακμή (start node id και end node id) και είναι τύπου INT(11). Εδώ να σημειωθεί ότι για κάθε ξεχωριστό «κόψιμο» η ακμή ξεκινάει και τερματίζει σε διαφορετικό κελί. Έτσι υπάρχουν ξεχωριστά πεδία για κάθε μέγεθος κελιού στον πίνακα ακμών. Αν για παράδειγμα θέλει κάποιος να βρει για το «κόψιμο» των 32768 κελιών από ποιο κελί ξεκινάει και τερματίζει μία ακμή, θα πρέπει να κοιτάξει τα πεδία *sn_id_p181, en_id_p181*. Τα πεδία αυτά είναι τύπου SMALLINT(8) αφού το κελί με το μεγαλύτερο αύξων αριθμό είναι το 32768. Τέλος το πεδίο *tt* υποδηλώνει το βάρος της συγκεκριμένης ακμής και είναι τύπου INT(11).

Τα ευρετήρια *pN* είναι συνδυαστικά ευρετήρια στα πεδία *sn_id_pN, en_id_pN* που σκοπό έχουν να αυξήσουν την απόδοση των ερωτημάτων κατά τη διαδικασία του ¨ζεστάματος¨ των cache (redis) [2]. Το βασικό ερώτημα που εκτελείται για αυτή τη διαδικασία φαίνεται παρακάτω:

Ερώτημα SQL

SELECT *sn_id, en_id* FROM *edges* WHERE *sn_id_pN = partition* AND *en_id_pN = partition*

Όπως μπορούμε εύκολα να διαπιστώσουμε το *WHERE* του ερωτήματος στηρίζεται στο συνδυασμό των πεδίων *sn_id_pN, en_id_pN* με τον τελεστή *AND* επομένως η χρήση ενός ευρετηρίου Hash ή B-tree και στα δύο πεδία ενδείκνυται. Στην ουσία αυτό το ερώτημα επιστρέφει τις εσωτερικές ακμές του κελιού με αύξων αριθμό partition.



(α') Ακμές με βάρη (β') Border nodes (γ') Αποτελέσματα

Σχήμα 3.1: Σχήματα πινάκων βάσης δεδομένων

Αντίστοιχα ο πίνακας 3.1(β') έχει ως *primary key* το *partition_id*. Το πεδίο *bn* είναι τύπου *TEXT* και περιέχει τους αύξοντες αριθμού των συνοριακών κόμβων του συγκεκριμένου κελιού διαχωρισμένους με κόμμα. Αυτή είναι προ-υπολογισμένη πληροφορία όπως έχουμε αναφέρει και παραπάνω. Το πεδίο *nof_bn* δείχνει το συνολικό αριθμό των συνοριακών κόμβων που έχει το συγκεκριμένο κελί. Επιπροσθέτως το πεδίο *size* υποδεικνύει το μέγεθος του «κοψίματος» που αντιστοιχεί το συγκεκριμένο κελί. Για παράδειγμα υπάρχει κελί 300 για τα 32768 κελιά αλλά και για τα 512 κελιά. Τέλος τα πεδία *min_node_autoinc, max_node_autoinc* δείχνουν το ελάχιστο και το μέγιστο αύξων αριθμό κόμβου που βρίσκεται εντός του συγκεκριμένου partition. Η χρήση αυτού του πεδίου είναι καθαρά τεχνική, βοηθάει στην κανονικοποίηση του αύξοντος αριθμού για κάθε κελί ώστε όλοι οι κόμβοι ενός κελιού να ξεκινούν από τον αριθμό ένα και να είναι συνεχόμενοι αριθμοί. Με αυτό τον τρόπο μπορούμε να «συμπιέσουμε» το κελί κατά την αποστολή του όπως θα δούμε στο κεφάλαιο 4.

Τέλος ο πίνακας *results* αναλαμβάνει να αποθηκεύσει τα δεδομένα που κάθε χρήστης υπολόγισε. Ουσιαστικά ο κάθε χρήστης υπολογίζει τα μονοπάτια από κάθε συνοριακό κόμβο προς κάθε άλλο συνοριακό κόμβο για το κελί που του έχει αποδοθεί. Έτσι κάθε εγγραφή του πίνακα διατηρεί το κόστος (*tt* INT(11)) για κάποιο *partition_id* INT(11) από κάποιο *border_from* προς κάποιο άλλο *border_to*. Με άλλα λόγια οι εγγραφές του πίνακα αυτού αντιστοιχούν στις ακμές επικάλυψης του γράφου επικάλυψης που προκύπτει.

3.1.3 Μηχανές αποθήκευσης

Storage Engine (ή Database Engine) είναι το λογισμικό που χρησιμοποιεί ένα σύστημα Βάσης Δεδομένων (DBMS) για να δημιουργεί, να διαβάζει, να ανανεώνει και να διαγράφει (CRUD) δεδομένα από τη βάση δεδομένων. Τα περισσότερα συστήματα βάσεων δεδομένων υποστηρίζουν διεπαφές (API) που επιτρέπουν στο χρήστη να αλληλεπιδρούν απευθείας με τη μηχανή αποθήκευσης χωρίς να περνούν μέσα από το περιβάλλον του DBMS. Οι περισσότερες μοντέρνες βάσης δεδομένων υποστηρίζουν πολλαπλές μηχανές αποθήκευσης. Συγκεκριμένα η MySQL υποστηρίζει τα δημοφιλή MyISAM και InnoDB καθώς και λιγότερα γνωστά, εξειδικευμένα storage engines, όπως τα Memory, Merge, Archive, Federated, NDB, CSV, Blackhole και Example.

MyISAM

Το προεπιλεγμένο storage engine της MySQL 5.1 και ένα από τα περισσότερο χρησιμοποιούμενα στο διαδίκτυο, καθώς και σε άλλα περιβάλλοντα εφαρμογών. Η έλλειψη συναλλαγών είναι το μεγαλύτερο μειονέκτημα αυτού του μηχανισμού αποθήκευσης. Κάθε MyISAM πίνακας αποθηκεύεται στο δίσκο σε τρία διαφορετικά αρχεία. Τα αρχεία έχουν ονόματα που ξεκινούν με το όνομα του πίνακα και κατά-

ληξη που υποδεικνύει τον τύπο του αρχείου. Η MySQL χρησιμοποιεί το .frm αρχείο για να αποθηκεύει τον ορισμό του πίνακα. Αυτό δεν είναι κομμάτι της μηχανής αποθήκευσης αλλά της βάσης δεδομένων. Το αρχείο .MYD (MYData) διατηρεί τα δεδομένα του πίνακα ενώ το .MYI (MYIndex) διατηρεί τα ευρετήρια του πίνακα.

- *Απλότητα*: Ένα από τα μεγαλύτερα πλεονεκτήματα της μηχανής αυτής είναι η απλότητα της. Μπορεί πολύ εύκολα να αντιληφθεί κανείς τον τρόπο λειτουργίας της και να γράψει ακόμη εφαρμογές που αλληλεπιδρούν άμεσα με αυτήν. Υπάρχουν πολύ σημαντικά εργαλεία όπως το mysqlhotcopy διαθέσιμα για MyISAM ενώ είναι πολύ πιο δύσκολο

- *Χρήση πόρων*: Είναι γενικώς αποδεκτό στην επιστήμη των υπολογιστών ότι συχνά υπάρχει ένα ζύγισμα μεταξύ ταχύτητας και αποτυπώματος μνήμης. Η InnoDB για παράδειγμα έχει κάποιους πολύ γρήγορους αλγορίθμους, όμως έχουν το αντίστοιχο κόστος. Όχι απλώς η InnoDB χρησιμοποιεί περισσότερη μνήμη, αλλά ακόμη και τα αρχεία δεδομένων είναι συχνά κάπως μεγαλύτερα. Επιπροσθέτως η InnoDB έχει ένα αρκετά μεγάλο αρχείο ιστορικού το οποίο αυξάνει σημαντικά τις απαιτήσεις σε πόρους. Αυτά κάνουν την MyISAM μία καλή λύση για διακομιστές περιορισμένους από την πλευρά των πόρων.

- *Βελτιστοποίηση*: Ένα άλλο μεγάλο πλεονέκτημα του MyISAM είναι η μεγάλη ιστορία του. Ως αποτέλεσμα, υπάρχουν πολλά συστήματα όπως το Drupal για παράδειγμα που είναι πολύ βελτιστοποιημένα στη συγκεκριμένη μηχανή.

InnoDB

Από την έκδοση της MySQL 5.5 και μετά γίνεται ο προεπιλεγμένος μηχανισμός αποθήκευσης. Είναι ACID-συμβατή με υποστήριξη συναλλαγών, καθώς και την υποστήριξη ξένων κλειδιών. Η InnoDB έγινε προϊόν της Oracle μετά την εξαγορά της Innobase Oy τον Οκτώβριο του 2005 και προσφέρεται με διπλή άδεια. Είτε μέσω του GNU General Public License είτε σε τρίτους (3rd-parties) που θέλουν να το συνδυάσουν με ιδιόκτητο λογισμικό. Ξεχωρίζει για:

- *Διατήρηση δεδομένων*: Είναι σχεδιασμένη για να εξασφαλίζει τη συνέπεια και διάρκεια των δεδομένων. Το καταφέρνει μέσα από τη διατήρηση ενός ιστορικού συναλλαγών (με την επιλογή για commit σε δύο βήματα αν το δυαδικό ιστορικό είναι ενεργοποιημένο), διπλής προσωρινής μνήμης εγγραφής και αυτόματων ελέγχων αθροίσματος (checksum) και επιβεβαίωσης των σελίδων της βάσης δεδομένων. Τα μέτρα αυτά όχι μόνο αντιμετωπίζουν περιπτώσεις βίαιου κλεισίματος του διακομιστή αλλά μπορούν να εντοπίσουν ακόμη και περιπτώσεις κατεστραμμένου υλισμικού.

- *Ταχύτητα στις εγγραφές και ανανεώσεις*: Εξαιτίας της χρήσης κλειδωμάτων σε επίπεδο γραμμής, μπορεί να συγκρατεί μονάχα τις γραμμές που αλλάζουν,

επιτρέποντας στον υπόλοιπο πίνακα να είναι διαθέσιμος προς παράλληλη χρήση. Αντίθετα το MyISAM δεν έχει αυτή την ικανότητα μιας και τα κλειδώματα του είναι σε επίπεδο πίνακα.

Σύγκριση και επιλογή

Συνοπτικά θα μπορούσε κανείς να πει ότι:

- Η InnoDB είναι *νεότερη* ενώ η MyISAM παλαιότερη.
- Η InnoDB είναι περισσότερο *πολύπλοκη* ενώ η MyISAM πιο απλή.
- Η InnoDB είναι πιο αυστηρή για τη *διατήρηση των δεδομένων* ενώ η MyISAM είναι λιγότερο.
- Η InnoDB υλοποιεί *κλειδώματα σε επίπεδο γραμμής* για προσθήκες και ανανεώσεις ενώ η MyISAM υλοποιεί κλειδώματα σε επίπεδο πίνακα.
- Η InnoDB έχει *συναλλαγές* ενώ η MyISAM δεν έχει.
- Η InnoDB διαθέτει *ξένα κλειδιά* και περιορισμούς συσχετίσεων σε αντίθεση με την MyISAM.
- Η InnoDB έχει καλύτερη *ανάκτηση από καταστροφή* ενώ η MyISAM είναι ευάλωτη σε αντίστοιχες περιπτώσεις.
- Η MyISAM κάνει *αναζήτηση εντός κειμένου* (full-text) ενώ η InnoDB δεν κάνει.

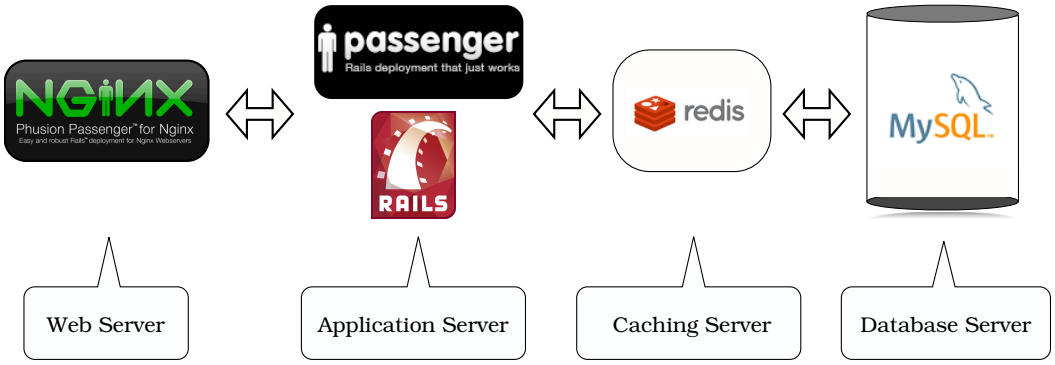
Στην εφαρμογή του map-reduce όπως αναφέρθηκε και παραπάνω υπάρχουν τρεις πίνακες στο σχήμα της βάσης δεδομένων. Οι πίνακες border_nodes και edges χρησιμοποιούνται μονάχα για ανάγνωση. Σε αυτή την περίπτωση προτιμήθηκε η χρήση MyISAM ως μηχανή αποθήκευσης γι' αυτούς τους δύο πίνακες χάρη στην απλότητα και την ταχύτητα σε αναγνώσεις που προσφέρει η συγκεκριμένη μηχανή. Αντίθετα ο τρίτος πίνακας results περιέχει δεδομένα που δε διαβάζονται στο map βήμα, μονάχα αποθηκεύονται σε αυτόν συνεχώς καινούργια δεδομένα (πάρα πολλά inserts), των οποίων η διάρκεια και η συνέπεια έχει πολύ μεγάλη σημασία για την εφαρμογή. Δεδομένου των παραπάνω είναι προφανής η επιλογή του InnoDB ως συστήματος αποθήκευσης γι' αυτόν τον πίνακα.

Έτσι εκμεταλλεύται κανείς την ευελιξία που προσφέρουν τα σύγχρονα συστήματα βάσεων δεδομένων, όπου ο σχεδιαστής τους μπορεί να επιλέξει διαφορετικά storage engines ανάλογα με τις ανάγκες και τις απαιτήσεις κάθε πίνακα.

3.2 Αρχιτεκτονική διακομιστή

3.2.1 Εισαγωγή

Η βάση της υλοποίησης στηρίζεται στο συνδυασμό μοντέρνων τεχνολογιών όπως είναι η Ruby on Rails και το Redis καθώς και σε παραδοσιακά εργαλεία όπως η MySQL για τη βάση δεδομένων. Όπως απεικονίζεται στο σχήμα 3.2 ένας Caching



Σχήμα 3.2: Αρχιτεκτονική συστήματος

Server δημιουργεί ένα δεύτερο επίπεδο προσωρινής μνήμης, μειώνοντας τις αναγνώσεις από τη βάση δεδομένων, αυξάνοντας εντυπωσιακά την απόδοση του συνολικού συστήματος [4]. Στη συνέχεια ο συνδυασμός NGINX / Passenger αναγνωρίζεται ως η καλύτερη λύση (από την οπτική της απόδοσης) για τη λειτουργία μιας Ruby on Rails εφαρμογής. Οι εκδόσεις που χρησιμοποιήθηκαν για τον κάθε διακομιστή φαίνονται στον πίνακα 3.1. Παρακάτω παρουσιάζονται αναλυτικά όλοι οι διακομιστές και η

Διακομιστής	Έκδοση	Διεύθυνση
MySQL	5.1.41	`http://www.mysql.com/`
NGINX	1.0.0	`http://nginx.net/`
Passenger	3.0.7	`http://www.modrails.com/`
Redis	2.2.2	`http://redis.io/`
Ruby on Rails	3.0.4	`http://rubyonrails.org/`

Πίνακας 3.1: Εκδόσεις διακομιστών

λειτουργία τους στο συνολικό σύστημα.

3.2.2 MySQL - *Database server*

Η MySQL είναι μία από τις δημοφιλέστερες σχεσιακές Βάσεις Δεδομένων ανοιχτού κώδικα (GNU General Public License). Πρωτοεμφανίστηκε το 1995 ενώ το 2010 εξαγοράστηκε από την Oracle. Σχεδιάστηκε για να προσφέρει εφαρμογές σε

βάσεις δεδομένων χαμηλού κόστους, αξιόπιστες, υψηλής απόδοσης - επέκτασης. Είναι ACID-compliant με full-commit, rollback, crash recovery και κλειδώματα σε επίπεδο εγγραφής. Συνοδεύεται από ένα σύνολο οδηγών και γραφικών εργαλείων που βοηθούν τους προγραμματιστές για να κτίσουν τις δικές τους MySQL εφαρμογές [23].

Η MySQL υποστηρίζει τις παρακάτω λειτουργίες:

- *Replication:* για να βελτιώσει την επεκτασιμότητα και την απόδοση του συστήματος.
- *Partitioning:* για να βελτιώσει την απόδοση και να απλοποιήσει την διαχείριση πολύ μεγάλων ΒΔ.
- *Stored procedures:* για να ενισχύσει το προγραμματιστικό περιβάλλον.
- *Triggers:* για να μεταφέρει βασικούς κανόνες σε επίπεδο ΒΔ.
- *Views:* για να μειώσει την πολυπλοκότητα των δεδομένων και των ερωτήσεων.
- *Information schema:* για να προσφέρει πρόσβαση σε μετα-δεδομένα.
- *Pluggable storage engine:* για να προσφέρει τη μέγιστη ευελιξία.

3.2.3 Redis - *Caching server*

Πρόκειται για ένα προηγμένο key-value store [7]. Η πρώτη έκδοση ανακοινώθηκε το 2009 από τον Salvatore Sanfilippo ενώ σήμερα η ανάπτυξη του χρηματοδοτείται από την VMware. Έχουν αναπτυχθεί οδηγοί για όλες τις γνωστές Γλώσσες Προγραμματισμού συμπεριλαμβανομένης και της Ruby. Πολλές φορές αναφέρεται ως *διακομιστής δομών δεδομένων* μιας και υποστηρίζει κλειδιά που μπορεί να είναι αλφαριθμικά, hashes, λίστες, σύνολα και ταξινομημένα σύνολα. Το Redis ξεχωρίζει για την ταχύτητα του τόσο στις αναγνώσεις όσο και στις εγγραφές. Στα πλαίσια του συστήματος που αναπτύχθηκε, μεγάλη σημασία έχει το γεγονός ότι το Redis έχει Persistence. Ουσιαστικά ανά τακτά χρονικά διαστήματα τα δεδομένα αποθηκεύονται στον σκληρό δίσκο διατηρώντας την ακεραιότητα τους σε αντίθεση με αντίστοιχους διακομιστές (όπως είναι το memcache) που σε περίπτωση αποτυχίας του συστήματος χάνουν τα δεδομένα. Αυτό προϋποθέτει ότι τα δεδομένα βρίσκονται κάπου αποθηκευμένα και ο caching διακομιστής λειτουργεί μονάχα ως ένα παραπάνω επίπεδο caching. Η χρήση του Redis σε αυτή την εφαρμογή αποσκοπεί κυρίως στη γρήγορη ανάκτηση των υπογράφων που θα σταλούν στο χρήστη και στη γρήγορη αποθήκευση των αντίστοιχων αποτελεσμάτων. Οι υπογράφοι (κελιά) αποθηκεύονται σε αλφαριθμητική μορφή (string, JSON encoded) και το μέγεθος τους μπορεί να φτάσει τα 2MB. Συνεπώς το όριο των 512MB του Redis για κάθε κλειδί υπερκαλύπτει τις ανάγκες του συστήματος. Έχει αρκετό ενδιαφέρον να δει κανείς μερικά benchmarks

του διακομιστή όπως φαίνονται στον πίνακα 3.2. Είναι ξεκάθαρο πως η απόδοση του συστήματος είναι πολύ υψηλή και σίγουρα δε θα αποτελέσει bottleneck της υλοποίησης.

Διαδικασία	Απαντήσεις ανά δευτερόλεπτο
SET	114.293
GET	81.234
INCR	68.458
LPUSH	88.109
LPOP	71.994

Πίνακας 3.2: Τυπικό benchmark του Redis

3.2.4 Ruby on Rails - *Web dev framework*

Εισαγωγή

Η Ruby on Rails είναι ένα Web Development Framework. Αναπτύχθηκε από τον David Heinemeier Hansson και άνοιξε ως open source project στην κοινότητα ανοιχτού λογισμικού τον Φεβρουάριο του 2005. Στηρίζεται στην αρχιτεκτονική του Model View Controller (MVC) για να οργανώσει τη δομή της εφαρμογής. Διαθέτει εργαλεία που κάνουν τον προγραμματισμό για το διαδίκτυο ευκολότερο υποστηρίζοντας "out of the box" συνήθεις προγραμματιστικές διεργασίες. Παράδειγμα τέτοιου εργαλείου είναι το scaffolding που αυτόματα κατασκευάζει τα μοντέλα, τους controllers και τα views που χρειάζεται μία απλή web εφαρμογή. Παράλληλα συνοδεύεται από χιλιάδες gems που στην ουσία είναι Rails plugins ανοιχτού κώδικα που διευκολύνουν ακόμη περισσότερο τον προγραμματισμό. Χρησιμοποιεί τη Ruby ως γλώσσα προγραμματισμού που πρωτοεμφανίστηκε το 1995 σχεδιασμένη από τον Yukihiro Matsumoto. Η Ruby προσφέρει στη Rails την προγραμματιστική ευελιξία μιας δυναμικής, reflective, αντικειμενοστραφούς γλώσσας προγραμματισμού ενώ παράλληλα εντυπωσιάζει με το συντακτικό της εμπνευσμένο από την Perl και τη Smalltalk που θυμίζει φυσική γλώσσα.

Φιλοσοφία

Δύο βασικά φιλοσοφικά θεμέλια κάνουν τον κώδικα της Rails μικρό και ευανάγνωστο [6]:

DRY and convention over configuration

Το *DRY* σημαίνει *Don't Repeat Yourself*: Κάθε κομμάτι γνώσης σε ένα σύστημα πρέπει να εκφράζεται μονάχα σε ένα σημείο. Η Rails χρησιμοποιεί τη δύναμη της Ruby για να το καταφέρει. Θα βρει κανείς πολύ μικρή επανάληψη σε μία εφαρμογή Rails. Ο προγραμματιστής γράφει αυτό που θέλει σε ένα σημείο, το οποίο μάλιστα του υποδεικνύει η MVC αρχιτεκτονική. Για τους προγραμματιστές που ήταν συνηθισμένοι στο να κάνουν δεκάδες αλλαγές όταν έπρεπε να γίνει μια μικρή τροποποίηση στη λογική ενός συστήματος, αυτό ήταν επανάσταση.

Το *Convention over configuration* είναι επίσης σημαντικό. Στην ουσία η Rails έχει προκαθορισμένες ρυθμίσεις για κάθε κομμάτι της εφαρμογής. Ακολουθώντας αυτές τις συμβάσεις μπορεί κάποιος να γράφει λιγότερο κώδικα απ' ότι θα έγραφε σε μία τυπική web εφαρμογή. Εάν ο προγραμματιστής θέλει να παρακάμψει αυτές τις συμβάσεις, μπορεί απλώς να το κάνει.

Η Rails χρησιμοποιεί από κατασκευής τα web standards με αποτέλεσμα να είναι εύκολο για τους προγραμματιστές να υλοποιήσουν προηγμένες τεχνολογίες όπως είναι η Ajax και RESTful διεπαφές.

Τέλος η Rails είναι ευέλικτη (Agile). Όπως αναφέρεται στο Agile Manifesto [11] με τέσσερις κανόνες:

- Άνθρωποι και αλληλεπίδραση, **όχι** διαδικασίες και εργαλεία.
- Λογισμικό που δουλεύει, **όχι** περιληπτικά εγγράφα.
- Συνεργασία με τους πελάτες, **όχι** διαπραγματεύσεις επί του συμβολαίου.
- Προσαρμογή στην αλλαγή, **όχι** να ακολουθείς στενά το πλάνο.

3.2.5 Passenger - *Application server*

Πρόκειται για έναν διακομιστή εφαρμογών όπως είναι η Ruby on Rails. Ουσιαστικά βρίσκεται ένα επίπεδο πάνω από τον web server. Γνωστός και ως *mod_rails*, *mod_rack* κάνει πανεύκολη τη λειτουργία μιας Rails εφαρμογής σε περιβάλλον παραγωγής. Χρησιμοποιεί τις γνωστές συμβάσεις της Ruby on Rails, επομένως:

- Η λειτουργία της εφαρμογής σε περιβάλλον παραγωγής απαιτεί μονάχα το ¨ανέβασμα¨ των αρχείων της εφαρμογής στο διακομιστή. Καμία ειδική ρύθμιση για τη Rails δεν απαιτείται.
- Υποστηρίζει τον εμπορικά καθιερωμένο Apache web server και τον γρήγορο - ελαφρύ NGINX web server.
- Δεν απαιτεί συντήρηση. Δε χρειάζεται η παρακολούθηση των διεργασιών του διακομιστή ή ο καθαρισμός των άχρηστων αρχείων. Τα σφάλματα ανακτώνται αυτόματα όποτε αυτό είναι δυνατό.

- Σχεδιασμένος για υψηλή απόδοση, σταθερότητα και ασφάλεια. Ο Passenger ποτέ δεν επηρεάζει τον web server ακόμα και στην περίπτωση που καταρρεύσει η Rails εφαρμογή.

- Πολύ καλή βιβλιογραφία τόσο για τους διαχειριστές όσο και για τους προγραμματιστές.

3.2.6 NGINX - *Web server*

Ο NGINX™ είναι ένα εξειδικευμένο λογισμικό διαδικτυακής υποδομής [5]. Είναι ένας υψηλής απόδοσης web διακομιστής με το λιγότερο δυνατό αποτύπωμα μνήμης ενώ παρέχει τον πληρέστερο συνδυασμό από τα πιο ουσιώδη χαρακτηριστικά που απαιτούνται για την κατασκευή μιας μοντέρνας και αποδοτικής διαδικτυακής υποδομής.

Σήμερα ο NGINX είναι ο δεύτερος πιο δημοφιλής web server ανοιχτού λογισμικού στο διαδίκτυο.

Η δυνατότητες του NGINX περιλαμβάνουν HTTP web server, HTTP και mail reverse proxy, caching περιεχομένου, κατανομή φόρτου, συμπίεση, bandwidth policing, πολύπλεξη και επαναχρησιμοποίηση σύνδεσης, μείωση φόρτου SSL και media streaming.

3.2.7 V8 Javascript Engine

Σε όλα τα πειράματα του συστήματος χρησιμοποιήθηκε ως πρόγραμμα περιήγησης το Google Chrome που με τη σειρά του χρησιμοποιεί τη μηχανή ανοιχτού κώδικα V8 για την εκτέλεση της Javascript [27]. Το πιο βαρύ κομμάτι της υλοποίησης εκτελείται εκεί και έχει ενδιαφέρον να αναφερθούν μερικές από τις καινοτομίες αυτής της μηχανής.

Αναπτύχθηκε από τη Google, είναι γραμμένη σε C++ και είναι μηχανή ανοιχτού κώδικα. Η V8 επιτυγχάνει υψηλή απόδοση μεταγλωττίζοντας τον πηγαίο κώδικα σε κώδικα μηχανής πριν την εκτέλεση του, αντί να εκτελεί το bytecode ή να τον εκτελεί με διερμηνέα. Η απόδοση αυξάνεται ακόμα περισσότερο με τη χρήση τεχνικών βελτιστοποίησης όπως είναι το *inline caching*. Με αυτές τις δυνατότητες, οι εφαρμογές Javascript που τρέχουν με τη V8 λέγεται ότι έχουν επιδόσεις συγκρίσιμες με ένα μεταγλωττισμένο δυαδικό πρόγραμμα.

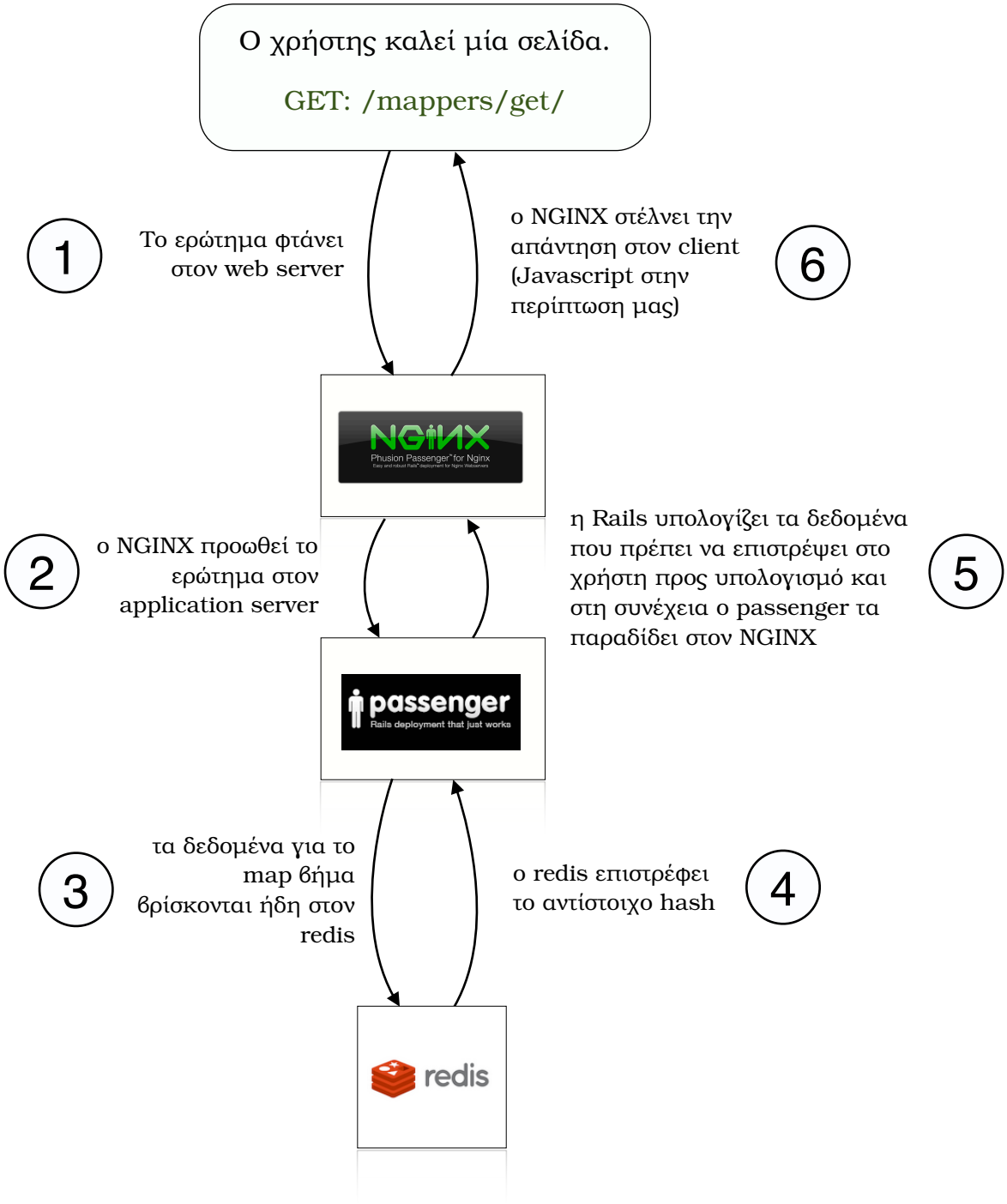
3.2.8 Η πορεία ενός request

Παραπάνω περιγράψαμε αναλυτικά τη λειτουργία κάθε διακομιστή χωρίς όμως να έχουμε αναφέρει τη λειτουργία τους στο συνολικό σύστημα. Όπως μπορούμε να δούμε και στο σχήμα 3.3 κάθε φορά που ένας χρήστης συνδέεται στην εφαρμογή

καλεί την "/mappers/get" (στην ουσία αυτή η κλήση γίνεται από τον Javascript client) που του αποδίδει ένα υποπρόβλημα προς επίλυση. Τι γίνεται όμως από τη στιγμή της κλήσης μέχρι την απάντηση· Το ερώτημα σε πρώτη φάση φτάνει στον NGINX ο οποίος στη συνέχεια το προωθεί στον Application Server, τον Passenger. Πλέον εκτελείται ο Rails κώδικας που με τη σειρά του θα κάνει μία κλήση στον Redis για την ανάκτηση του κατάλληλου υπο-γράφου που θα κληθεί να επιλύσει ο συγκεκριμένος χρήστης. Αφού η Rails έχει κατασκευάσει κατάλληλα το μήνυμα που περιέχει τον υπο-γράφο, θα ακολουθήσει το ανάποδο δρόμο για να φτάσει στον NGINX και στη συνέχεια πίσω στο χρήστη με τη μορφή απάντησης (reponse).

Όπως εύκολα μπορούμε να παρατηρήσουμε, σε όλη αυτή τη διαδικασία δεν εμφανίστηκε καθόλου η Βάση Δεδομένων. Όπως έχει αναφερθεί και στην αρχή του κεφαλαίου ο Redis έχει προ-φορτώσει τα δεδομένα που απαιτούνται για το mapping των χρηστών, αυξάνοντας σε πολύ μεγάλο βαθμό την απόδοση του συστήματος.

Αφού υπολογίσει το αποτέλεσμα ο χρήστης θα εκτελέσει μία νέα κλήση. Αυτή τη φορά στην κλήση θα περιέχονται τα δεδομένα που υπολόγισε ο χρήστης (απάντηση), τα οποία ο Application Server με τη σειρά του θα στείλει στο Redis για προσωρινή αποθήκευση. Η απάντηση πίσω στον χρήστη θα είναι ένα απλό OK. Ούτε σε αυτή την κλήση συμμετέχει καθόλου η βάση δεδομένων.



Σχήμα 3.3: Η πορεία ενός request

Κεφάλαιο 4

Υλοποίηση

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα των πειραμάτων που έγιναν για τον υπολογισμό των ακμών επικάλυψης με τη χρήση map-reduce από επισκέπτες ιστοσελίδων. Τα πειράματα έγιναν για μεταβλητό αριθμό κελιών αλλά και μηχανημάτων που επιστρατεύτηκαν για τα πειράματα. Μέσα από τα πειράματα αναλύεται η δυνατότητα παραλληλοποίησης της εφαρμογής καθώς και οι περιορισμοί (bottlenecks) της στην πράξη. Για τα πειράματα χρησιμοποιήθηκε ένα 3COM Gigabit switch, τέσσερις MacBook Pro 2.2GHz Intel Core 2 Duo 4GB RAM και ένας iMac 2.5GHz Intel Core 2 Duo 4GB RAM. Τέλος κάθε υπολογιστής χρησιμοποιούσε τον Google Chrome σε έκδοση 15. Στα πειράματα που δεν έγινε παραλληλοποίηση (single browser tests), ο διακομιστής (NGINX web server) βρισκόταν στο ίδιο μηχάνημα με τον web browser, χάρη απλότητας. Στα πειράματα που έγιναν με περισσότερα του ενός μηχανήματα, ένας από τους υπολογιστές είχε το ρόλο του διακομιστή ενώ οι υπόλοιποι είχαν το ρόλο επισκεπτών στην ιστοσελίδα.

4.2 Πειράματα

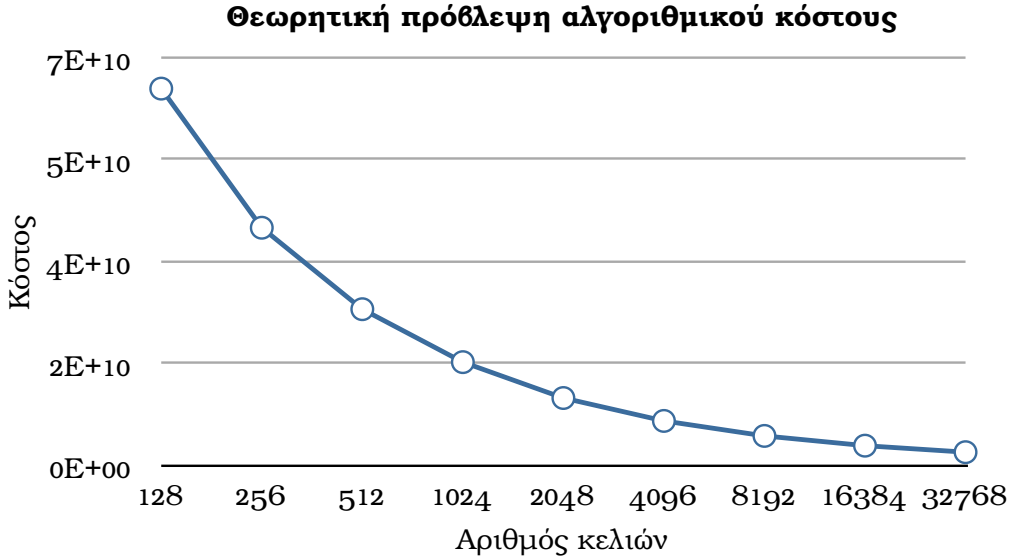
Ένα πείραμα περιλαμβάνει την *ανάλυση του συνολικού γράφου* (υπολογισμό όλων των ακμών επικάλυψης) ptv-europe για το συγκεκριμένο μέγεθος πλέγματος. Όπως έχουμε αναφέρει και σε προηγούμενο κεφάλαιο, η ανάλυση ενός κελιού αφορά την εκτέλεση V_b^i φορές τον αλγόριθμο Dijkstra από κάθε συνοριακό κόμβο του κελιού, όπου V_b^i ο αριθμός των συνοριακών κόμβων του κελιού i. Παράλληλα σε θεωρητικό επίπεδο έχουμε αναφέρει ότι η πολυπλοκότητα του αλγορίθμου Dijkstra με χρήση ουράς προτεραιότητας είναι $O(|E|log(|V|))$. Δεδομένου ότι έχουμε n_s αριθμό κελιών, η συνολική πολυπλοκότητα του προβλήματος σε θεωρητική βάση μπορεί να εκφραστεί ως:

$$O(n_s * V_b * |E| * log(|V|)) \tag{4.1}$$

όπου V_b, E και V μπορούμε να θεωρήσουμε τις μέσες τιμές των αντίστοιχων μεγεθών (V_b: μέσος αριθμός συνοριακών κόμβων ανά κελί, E: μέσος αριθμός ακμών ανά κελί, V: μέσος αριθμός κόμβων ανά κελί).

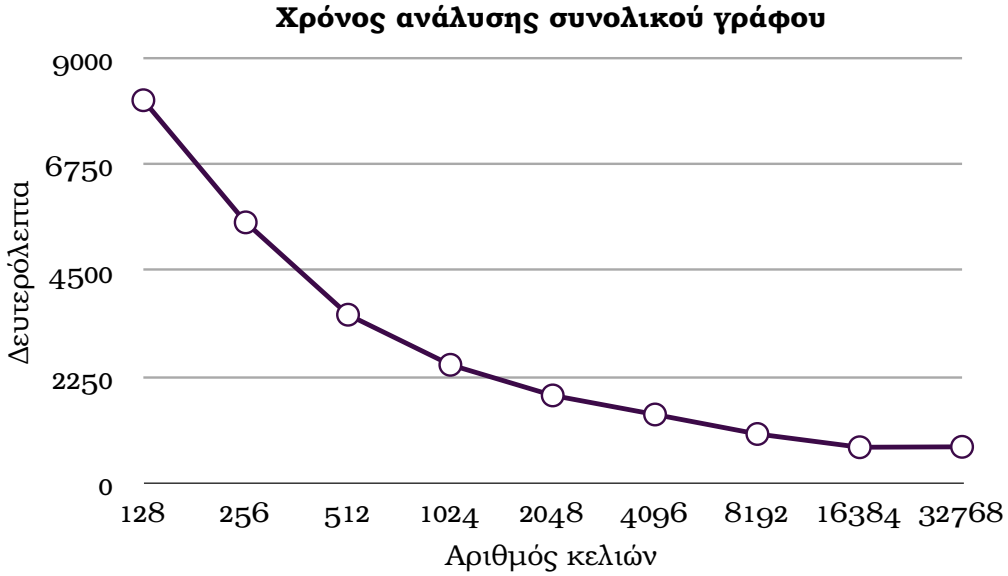
4.3 Χρόνος εκτέλεσης

Σύμφωνα με τον τύπο 4.1 αλλά και τα στατιστικά του γράφο, το κόστος για το συνολικό γράφο θα πρέπει να έχει τη μορφή του Σχήματος 4.1 συναρτήσει του μεγέθους του κελιού. Φαίνεται ότι όσο αυξάνουμε τον αριθμό των κελιών τόσο μειώνεται



Σχήμα 4.1: Θεωρητικό κόστος εκτέλεσης συνολικού αλγορίθμου σε συνάρτηση με το μέγεθος του κελιού. Το κόστος εκφράζει ποιοτικά τις επαναλήψεις που θα εκτελεστούν.

και το κόστος του αλγορίθμου [15]. Αν εξετάσει κανείς τον τύπο 4.1 θα διαπιστώσει ότι όσο αυξάνεται ο αριθμός των κελιών τα μεγέθη V_b, E και V μειώνονται αναλογικά ενώ μόνο το n_s αυξάνεται. Είναι προφανές ότι το συνολικό κόστος μειώνεται. Πράγματι τα θεωρητικά αποτελέσματα επιβεβαιώνονται από τα πειράματα. Στο Σχήμα 4.2 φαίνονται οι χρόνοι εκτέλεσης του συνολικού αλγορίθμου για ένα μονάχα υπολογιστή (στον οποίο έτρεχε ο διακομιστής και ο browser). Όπως φαίνεται στον πίνακα 4.1 η ταχύτερη εκτέλεση του αλγορίθμου έγινε με μέγεθος πλέγματος 16384 κελιών όπου χρειάστηκαν μόλις 12.6 λεπτά για τον υπολογισμό του συνολικού γράφου (κατασκευή γράφου επικάλυψης). Η πειραματική καμπύλη φαίνεται να ακολουθεί σε μεγάλο βαθμό τη θεωρητική, με το χρόνο για τα 128 κελιά να εκτοξεύεται περίπου στις 2.6 ώρες. Όπως θα δούμε και παρακάτω τα κελιά αρχίζουν και γίνονται πάρα πολύ μεγάλα όσο μειώνεται ο αριθμός τους, ενώ παράλληλα αυξάνεται ο αριθμός των

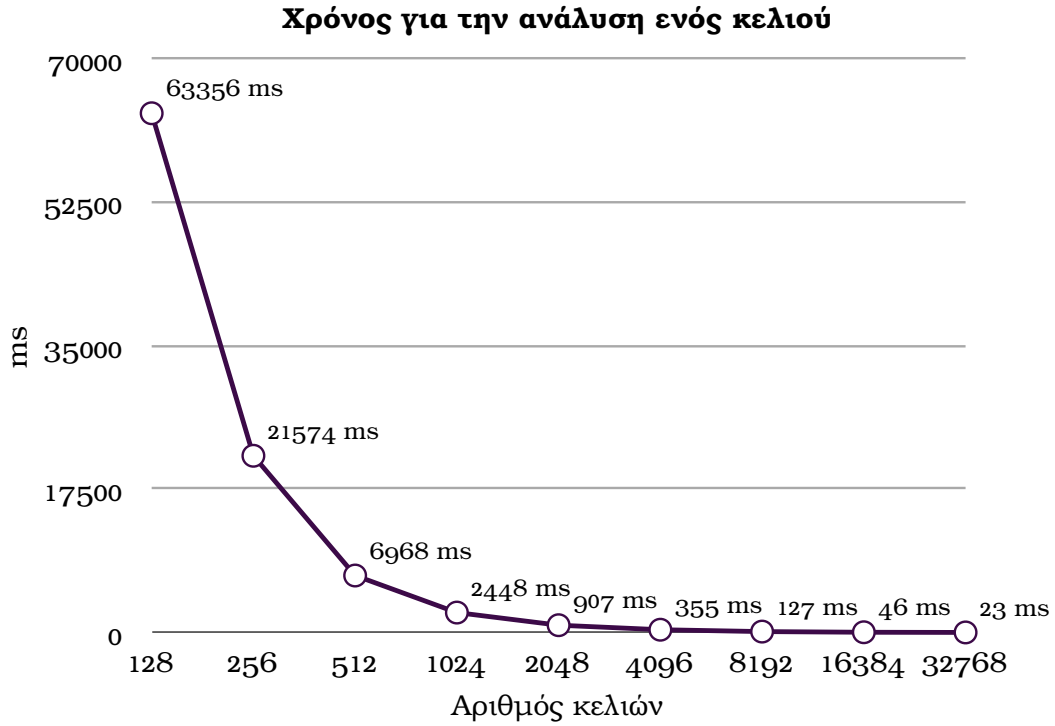


Σχήμα 4.2: Πειραματικοί χρόνοι εκτέλεσης συνολικού αλγορίθμου σε ένα υπολογιστή.

συνοριακών κόμβων δυσχεραίνοντας πολύ την εκτέλεση του αλγορίθμου. Αντίστοιχα μπορούμε να δούμε στο Σχήμα 4.3 το χρόνο που απαιτείται για την ανάλυση ενός κελιού. Ο χρόνος αυτός περιλαμβάνει την εκτέλεση V_b Dijkstra από κάθε συνοριακό κόμβο. Τέλος είναι σημαντικό να αναφέρουμε ότι οι πειραματικοί χρόνοι περιλαμβάνουν και όλες τις λοιπές διεργασίες (εκτός από το κόστους του αλγορίθμου) που δεν υπολογίζονται στη θεωρητική πρόβλεψη όπως θα δούμε στην επόμενη παράγραφο.

4.4 Κατανομή χρόνου

Έχει μεγάλο ενδιαφέρον να εξετάσει κανείς τον τρόπο με τον οποίο διαμοιράζεται ο χρόνος μεταξύ των διαφόρων αυτών διεργασιών. Στις διεργασίες περιλαμβάνονται η εκτέλεση του αλγορίθμου Dijkstra, το κόστος δικτύου και το parsing των δεδομένων. Όπως είναι αναμενόμενο όσο μεγαλώνει ο αριθμός των κελιών, τόσο μεγαλώνει και το κόστος μεταφοράς στο δίκτυο αφού πολλαπλασιάζονται τα ερωτήματα που γίνονται στον διακομιστή. Μπορεί τα συνολικά δεδομένα που μεταφέρονται να είναι περίπου τα ίδια, παρ' όλα αυτά τα ερωτήματα είναι πολλαπλάσια. Αρκεί να σκεφτεί κανείς ότι για κάθε κελί γίνονται δύο ερωτήματα στον διακομιστή. Το πρώτο φέρνει το γράφο και το δεύτερο επιστρέφει την απάντηση. Συνεπώς για το πλέγμα των 32768 κελιών θα γίνουν περίπου 65.000 ερωτήματα στον διακομιστή ενώ για τα 4096 κελιά θα γίνουν μόλις 8.000. Είναι προφανές πως το κόστος στην πρώτη περίπτωση είναι κατά πολύ μεγαλύτερο. Αντίθετα το υπολογιστικό κόστος του αλγορίθμου Dijkstra όπως είδαμε και στην προηγούμενη παράγραφο συνεχώς μειώνεται. Συγκεκριμένα όπως



Σχήμα 4.3: Χρόνος ανάλυσης ενός κελιού συναρτήσει του συνολικού αριθμού κελιών.

μπορεί να παρατηρηθεί στον πίνακα 4.1, το κόστος του δικτύου ξεκινάει από τα 76 δευτερόλεπτα (για τα 128 κελιά) όπου η συνολική διαδικασία απαιτεί 8109 δευτερόλεπτα. Αποτελεί δηλαδή το 0.9% του συνολικού κόστους. Σταδιακά αυτή η τιμή αυξάνεται καταλήγοντας στα 377 δευτερόλεπτα (για τα 32768 κελιά) με συνολικό χρόνο 769 δευτερόλεπτα που μεταφράζεται στο 49% του κόστους όπως μπορούμε να δούμε και εικονικά στο σχήμα 4.4. Αυτή η δραματική απόδοση στο δίκτυο είναι που δε μας επιτρέπει τη συνεχή βελτίωση του αποτελέσματος μέσα από την αύξηση των κελιών. Αντίθετα όπως βλέπουμε στον πίνακα 4.1 και στο σχήμα 4.4, το ποσοστό του υπολογιστικού κόστους συνεχώς μειώνεται όπως και ο απόλυτος χρόνος (σχήμα 4.5(α')). Συγκεκριμένα για τα 128 κελιά ο υπολογιστικός χρόνος ανέρχεται στα 7989 δευτερόλεπτα που μεταφράζεται στο 99% του συνολικού κόστους καταλήγοντας στα 352 δευτερόλεπτα για τα 32768 κελιά και το 46% του συνολικού χρόνου. Αξίζει να σημειωθεί ότι ο χρόνος του parsing των δεδομένων (από και προς JSON μορφή) ενώ αρχικά είναι αμελητέος, στα 32768 κελιά απαιτεί το 5% του συνολικού χρόνου ενώ σε απόλυτα νούμερα διατηρείται σχετικά σταθερός.

Τελικώς διαπιστώνουμε ότι η μέθοδος της συνεχόμενης αύξησης των κελιών βελτιώνει τη συνολική απόδοση του συστήματος μέχρι να φτάσει το ολικό ελάχιστο όπου το κόστος δικτύου (network overhead) είναι πλέον τόσο μεγάλο που τελικώς ο αλγό-

ριθμος δε βελτιώνεται. Το φαινόμενο αυτό αναφέρεται στον κλάδο της Παράλληλης Επεξεργασίας (Parallel Computing) ως *communication bottleneck* [1]. Έτσι η βελτίωση του συστήματος επιτυγχάνεται μέχρι τα 16384 κελιά, ενώ από τα 32768 κελιά αρχίζει να χειροτερεύει.

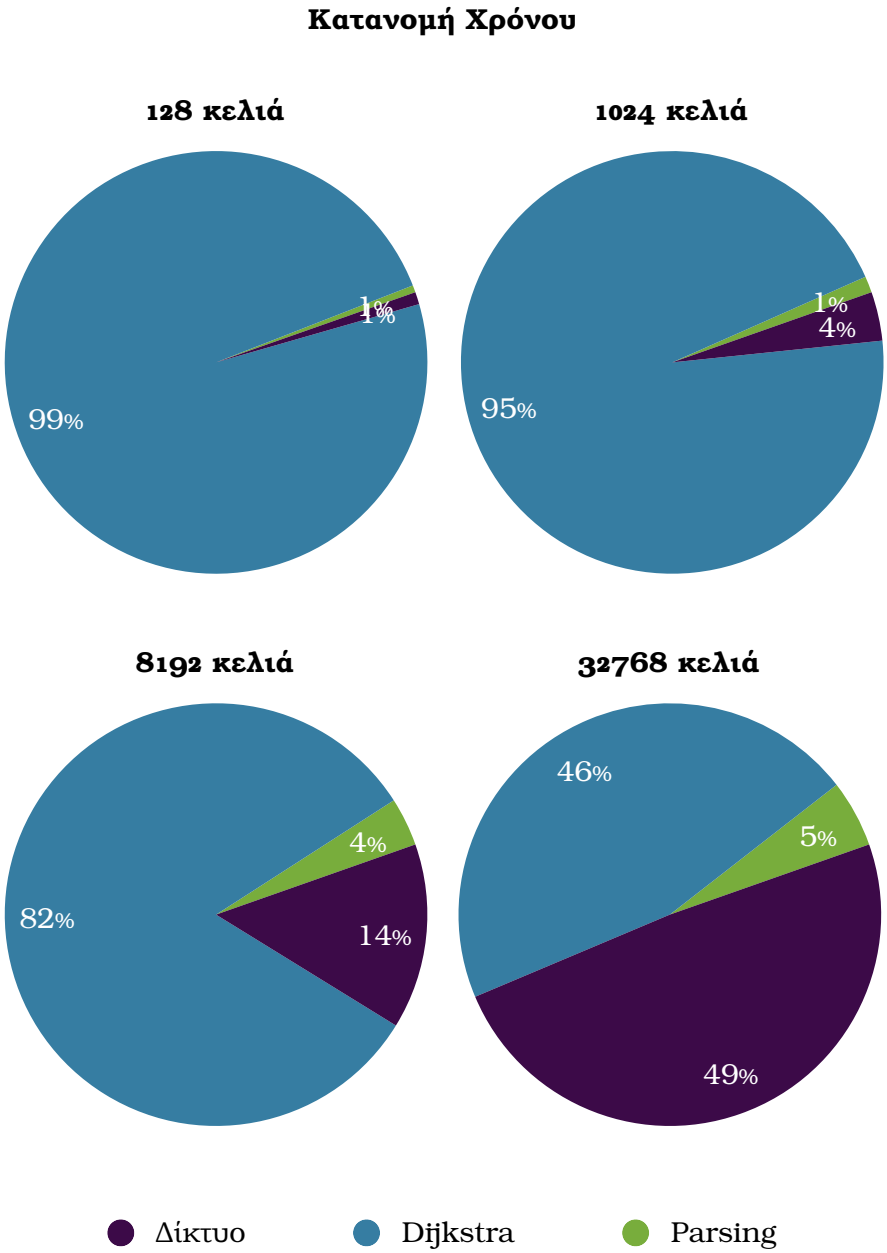
Κελιά	Δίκτυο	Dijkstra	Parsing	Συνολικά	Χρόνος / κελί
128	76 s	7989 s	44 s	**8109 s**	63356 ms
256	70 s	5410 s	42 s	**5522 s**	21574 ms
512	98 s	3423 s	45 s	**3567 s**	6967 ms
1024	93 s	2381 s	31 s	**2506 s**	2447 ms
2048	76 s	1750 s	30 s	**1858 s**	907 ms
4096	99 s	1320 s	33 s	**1454 s**	355 ms
8192	147 s	855 s	38 s	**1041 s**	127 ms
16384	227 s	495 s	37 s	**760 s**	46 ms
32768	377 s	352 s	39 s	**769 s**	23 ms

Πίνακας 4.1: Πειραματικοί χρόνοι εκτέλεσης συνολικού αλγορίθμου συναρτήσει του αριθμού των κελιών.

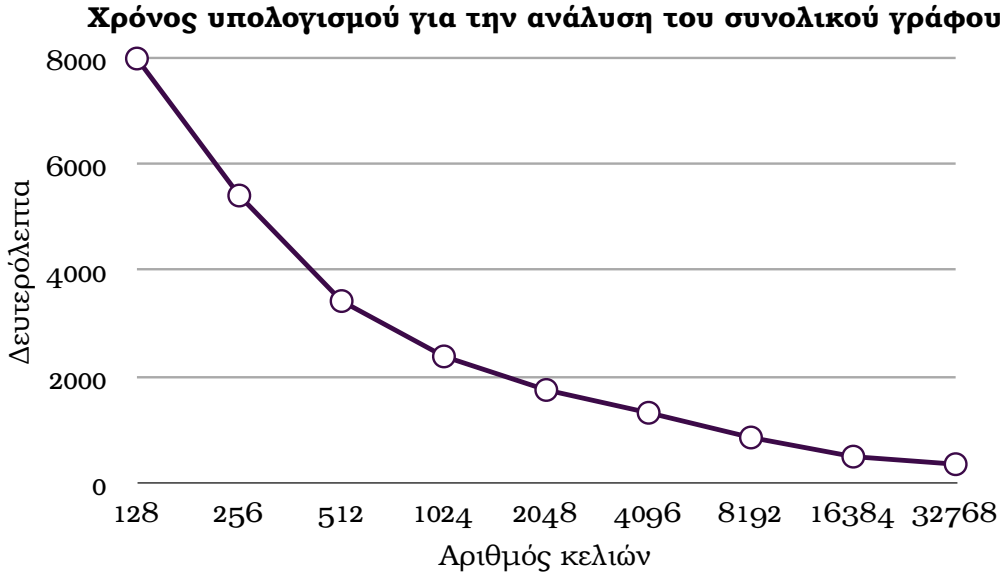
4.5 Παραλληλοποίηση

Ένα από τα μεγαλύτερα πλεονεκτήματα της υλοποίησης που έχει ακολουθηθεί σε αυτή την έρευνα είναι η δυνατότητα παραλληλοποίησης της όλης διαδικασίας. Έτσι ένα από τα πειράματα που έγιναν ήταν η ανάλυση του συνολικού γράφου με τη χρήση πολλών ταυτόχρονων επισκεπτών (concurrent users) στην υπηρεσία. Όπως φαίνεται στο σχήμα 4.6(α') ο χρόνος για τα 32768 κελιά ξεκινάει από τα 665 δευτερόλεπτα με ένα επισκέπτη. Στον πίνακα 4.1 φαίνεται ότι γι' αυτό τον αριθμό κελιών απαιτούνται 769 δευτερόλεπτα. Στην περίπτωση όμως του σχήματος 4.6(α') έχουμε διαχωρίσει τον επισκέπτη από το διακομιστή σε δύο διαφορετικά μηχανήματα όπως αναφέραμε και στην αρχή του κεφαλαίου. Έτσι είναι προφανές ότι η απόδοση του συστήματος είναι αρκετά καλύτερη. Στη συνέχεια ο δεύτερος επισκέπτης θα μειώσει το χρόνο στα 327 δευτερόλεπτα δημιουργώντας speedup περίπου δύο (σχήμα 4.6(β')). Στη συνέχεια ο τρίτος και τέταρτος επισκέπτης μειώνουν τους χρόνους σε 273 και 218 δευτερόλεπτα αντίστοιχα. Φαίνεται ότι το speedup μειώνεται για τους δύο τελευταίους επισκέπτες μιας και αναμενόταν γραμμική αύξηση. Αντίστοιχα για τα 4096 κελιά ο χρόνος θα ξεκινήσει από 1228 δευτερόλεπτα για ένα επισκέπτη ενώ θα μειωθεί στα 716 δευτερόλεπτα, 471 δευτερόλεπτα και 368 δευτερόλεπτα για τους υπόλοιπους τρεις επισκέπτες. Μπορεί να παρατηρηθεί ότι το speedup για τα λιγότερα κελιά είναι κάπως καλύτερο. Φαίνεται ότι στην παραλληλοποίηση το communication overhead «χαλάει» τη γραμμική αύξηση του speedup. Ο διακομιστής

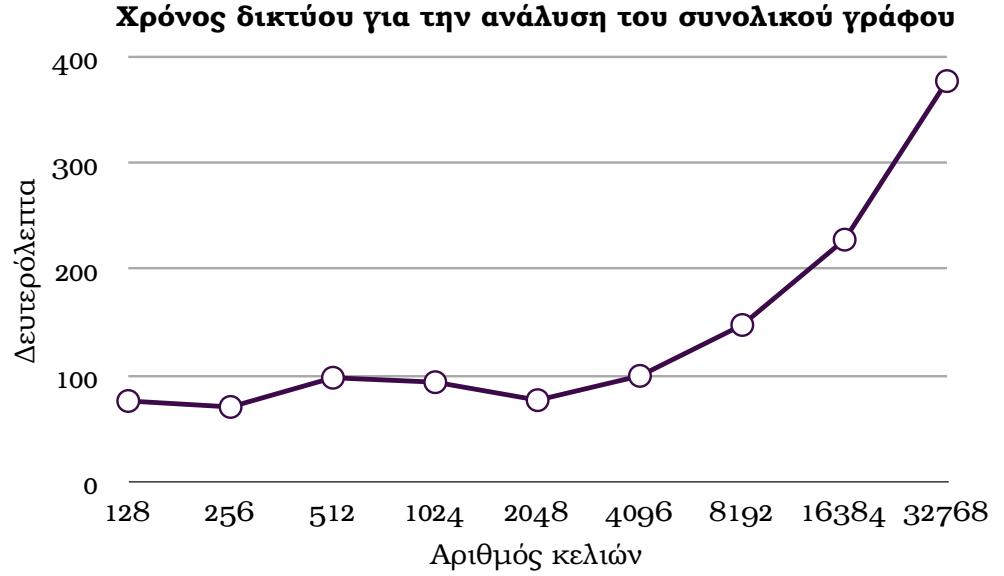
δέχεται τόσα πολλά παράλληλα ερωτήματα που καταλήγει να αποτελεί το bottleneck του συστήματος. Συγκεκριμένα στην πρώτη περίπτωση το σύστημα κλήθηκε να αναλύσει 32768 κελιά σε 218 δευτερόλεπτα. Τα 32768 κελιά μεταφράζονται σε 65536 ερωτήματα στον διακομιστή (αποστολή γράφου, επιστροφή αποτελέσματος). Για να εξυπηρετήσει ο διακομιστής αυτό τον τρομερά υψηλό αριθμό ερωτημάτων θα πρέπει να απαντάει σε 300 ερωτήματα το δευτερόλεπτο με μέσο χρόνο απάντησης τα 3.3 milliseconds. Φαίνεται πως κάπου εκεί είναι τα όρια του διακομιστή. Μία αρχιτεκτονική με περισσότερους διακομιστές και κάποιου είδους load balancing πιθανότατα θα βελτίωνε πολύ το speedup της παραλληλοποίησης.



Σχήμα 4.4: Ποσοστό κατανάλωσης χρόνου από τις διάφορες διεργασίες για μεταβλητό αριθμό κελιών. Βλέπουμε ότι όσο αυξάνονται τα κελιά τόσο αυξάνεται το ποσοστό του δικτύου ενώ μειώνεται αυτό του υπολογισμού Dijkstra.

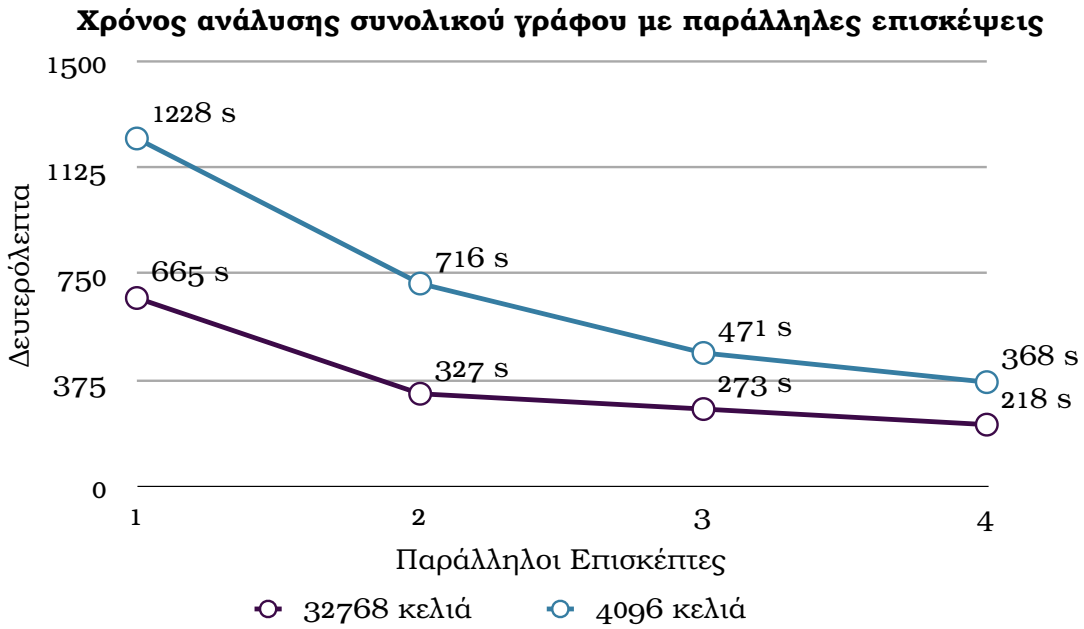


(α') Υπολογιστικός χρόνος που καταναλώνει ο αλγόριθμος Dijkstra συναρτήσει του αριθμού κελιών. Όσο αυξάνονται τα κελιά τόσο μειώνεται ο απαιτούμενος χρόνος.

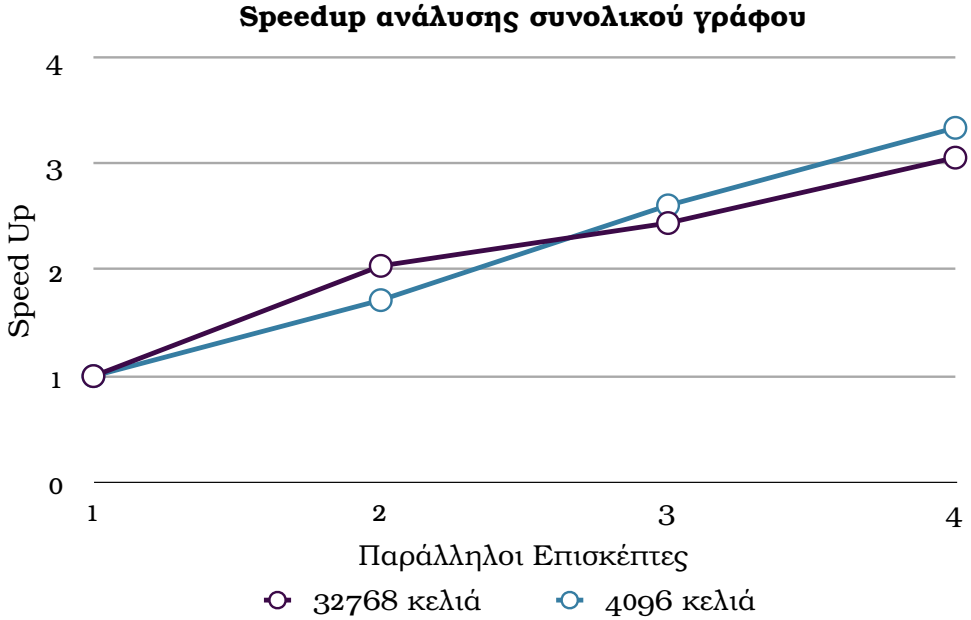


(β') Χρόνος δικτύου (communication time) που απαιτείται για την ανάλυση ολόκληρου του γράφου συναρτήσει του αριθμού κελιών. Εδώ ο χρόνος αυξάνεται όσο πληθαίνουν τα κελιά. Στο χρόνο περιλαμβάνεται η αποστολή του κελιού στον χρήστη καθώς και η ασύγχρονη επιστροφή των αποτελεσμάτων πίσω στο διακομιστή.

Σχήμα 4.5: Κατανομή χρόνου για ανάλυση συνολικού γράφου



(α') Ο χρόνος που απαιτήθηκε για την εκτέλεση του συνολικού αλγορίθμου παραλληλοποιώντας τη διαδικασία με ένα, δύο, τρεις και τέσσερις παράλληλους επισκέπτες στην υπηρεσία.

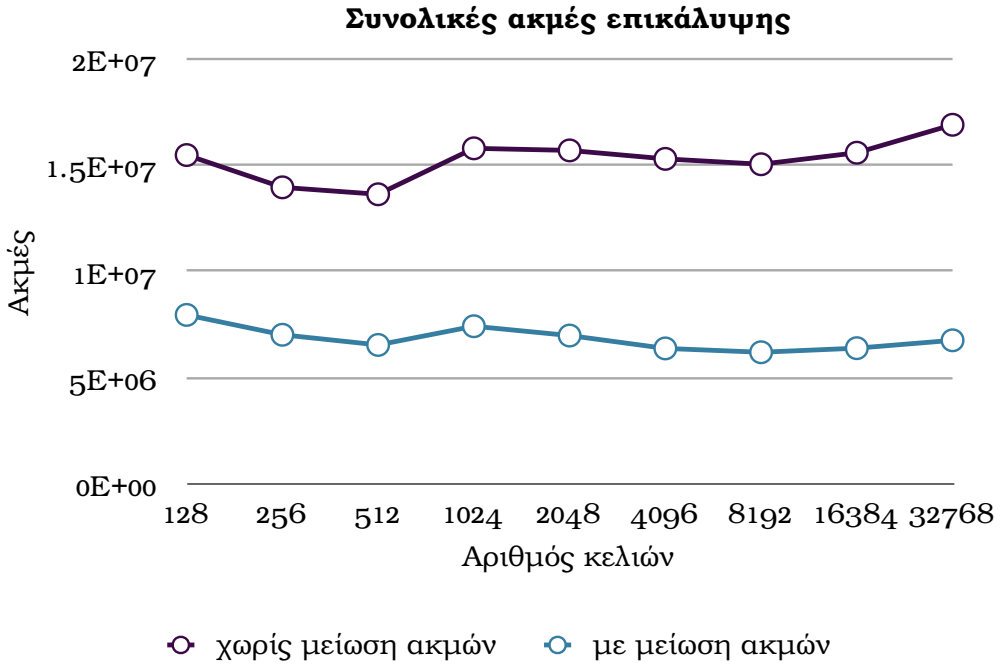


(β') Το speedup που που προκύπτει από την παραλληλοποίηση της διαδικασίας.

Σχήμα 4.6: Παραλληλοποίηση συστήματος

4.6 Μείωση ακμών (Edge reduction)

Όπως έχει αναφερθεί και σε προηγούμενο κεφάλαιο, παράλληλα με την εκτέλεση του αλγορίθμου Dijkstra υπολογίζεται με μία απλή φόρμουλα εάν το μονοπάτι που βρέθηκε θα πρέπει να συμπεριληφθεί στο γράφο επικάλυψης. Για να δει παραστατικά τον τρόπο απόφασης του συστήματος, ο αναγνώστης θα πρέπει να ανατρέξει στο σχήμα 2.14. Με τη βελτιστοποίηση αυτή μειώνονται οι ακμές που τελικά διατηρούνται στον γράφο επικάλυψης στο 40% με 50% (ανάλογα με τον αριθμό κελιών) των ακμών που θα κατασκευάζαμε χωρίς μείωση ακμών. Στο σχήμα 4.7 φαίνεται η περίπτωση χωρίς και με μείωση ακμών.



Σχήμα 4.7: Ακμές επικάλυψης για το συνολικό γράφο επικάλυψης με και χωρίς την εφαρμογή μείωσης ακμών.

4.7 Μέγεθος ερωτημάτων

Είναι επίσης αρκετά ενδιαφέρον να εξετασθεί το μέγεθος των ερωτημάτων και οι απαντήσεις τους. Για την ανάλυση κάθε ξεχωριστού κελιού γίνονται δύο ερωτήματα. Στο πρώτο ερώτημα ο επισκέπτης ζητάει για ένα κελί και ο διακομιστής του το στέλνει ενώ στο δεύτερο ερώτημα ο επισκέπτης επιστρέφει το αποτέλεσμα πίσω στο διακομιστή.

4.7.1 Συμπίεση περιεχομένου

Το περιεχόμενο του πρώτου ερωτήματος είναι σαφέστατα πολύ μεγαλύτερο αλλά έχει το πλεονέκτημα ότι μπορεί να συμπιεστεί με τη χρήση gzip από το διακομιστή. Το πρωτόκολλο HTTP στηρίζεται στη λογική της ερώτησης και της απάντησης. Η αρχιτεκτονική του επιτρέπει μονάχα τις απαντήσεις να είναι συμπιεσμένες, τα ερωτήματα δε μπορούν να συμπιεστούν. Όταν ο επισκέπτης ζητάει το γράφο από το διακομιστή, στις επικεφαλίδες δηλώνει ότι δέχεται συμπιεσμένο περιεχόμενο. Με αυτό τον τρόπο ο διακομιστής επιστρέφει την απάντηση συμπιεσμένη. Στην ανάποδη περίπτωση το πρωτόκολλο που θέλει να στείλει συμπιεσμένο περιεχόμενο στην ερώτηση, θα έπρεπε να «μαντέψει» ότι ο διακομιστής υποστηρίζει συμπίεση. Έτσι στην εφαρμογή αυτή συμπιέζεται μονάχα η αποστολή του γράφου που είναι και το μεγάλο πρόβλημα. Ο λόγος συμπίεσης φτάνει το 2.5 που σημαίνει ότι γλιτώνουμε αρκετά σε αποστολή δεδομένων.

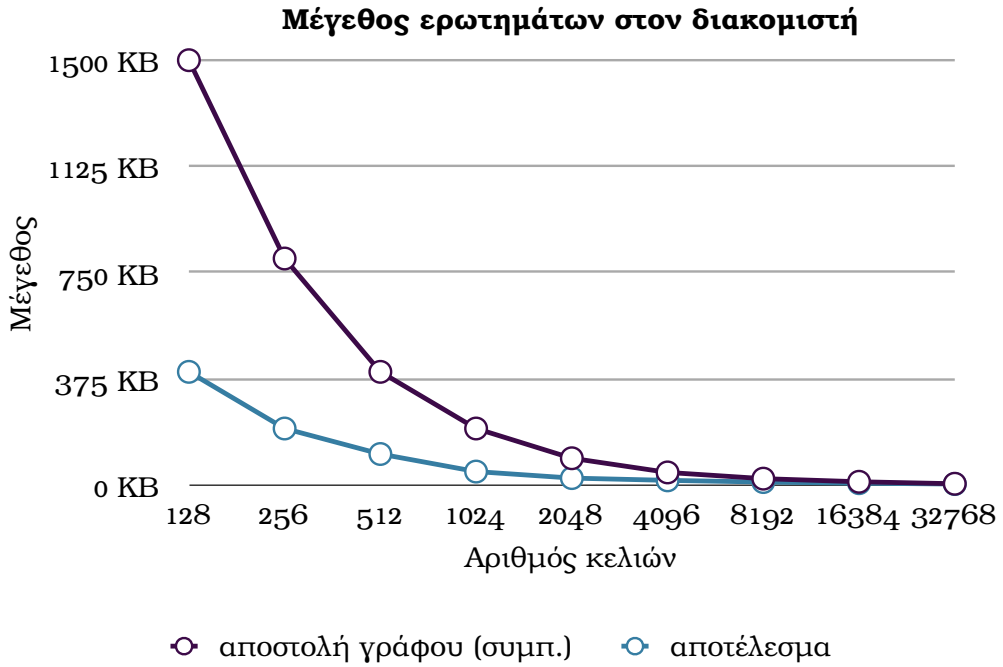
4.7.2 Μορφή πληροφορίας

Για την εξοικονόμηση όσο δυνατόν περισσότερης πληροφορίας ο γράφος αποθηκεύεται σε αρχείο JSON με φωλιασμένους πίνακες. Η αρίθμηση των κόμβων σε κάθε κελί ξεκινάει από το ένα και φτάνει μέχρι το τελευταίο κελί. Με αυτό τον τρόπο τοποθετούνται οι ακμές κάθε κόμβου σε πίνακες μέσα σε έναν αρχικό πίνακα ανάλογα με την αρίθμηση του κελιού. Ένα ημιτελές παράδειγμα γράφου φαίνεται παρακάτω:

```
[[[3, 10], [4, 50], [10, 1]], [[1, 3], [20, 10]]]
```

Η πρώτη θέση του πρώτου πίνακα περιέχει ένα πίνακα. Αυτός ο πίνακας έχει ως περιεχόμενο τις ακμές του πρώτου κόμβου. Έτσι ο πρώτος κόμβος συνδέεται με τον τρίτο με βάρος 10, με τον τέταρτο με βάρος 50 και με τον δέκατο με βάρος 1 κ.ο.κ Με αυτό τον τρόπο γλιτώνουμε σημαντικό χώρο «συμπιέζοντας» με ένα έμμεσο τρόπο τα δεδομένα. Παρόμοια τακτική ακολουθείται και για την επιστροφή των δεδομένων.

Στο σχήμα 4.8 φαίνεται το μέγεθος τόσο για την αποστολή ενός κελιού, όσο και για την επιστροφή του αποτελέσματος πίσω στον διακομιστή. Είναι προφανές ότι για τα 128 κελιά το μέγεθος κάθε κελιού είναι πολύ μεγάλο. Αν και συμπιεσμένο δυόμιση φορές φτάνει το 1.5MB καθένα από αυτά, ενώ η απάντηση για το ίδιο αριθμό κελιών είναι περίπου στα 400KB χωρίς συμπίεση. Στη συνέχεια τα μεγέθη μειώνονται αισθητά με τα 32768 κελιά να στέλνουν μόλις 7KB για κάθε κελί και 4KB για την απάντηση του.

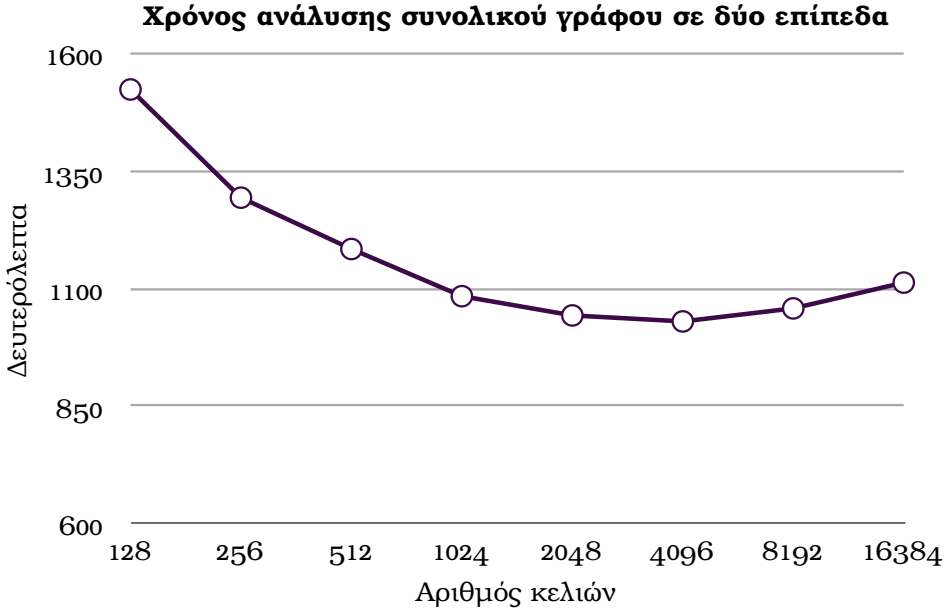


Σχήμα 4.8: Μέγεθος ερωτημάτων για την αποστολή του κελιού και την επιστροφή του αποτελέσματος, συναρτήσει του αριθμού των κελιών.

4.8 Ανάλυση σε δύο επίπεδα

Μία ακόμα ιδέα που έχει αρκετό ενδιαφέρον να εξεταστεί είναι η δημιουργία του γράφου επικάλυψης σε περισσότερα του ενός επίπεδα. Με λίγα λόγια μπορεί να κατασκευαστεί σε πρώτη φάση ο γράφος επικάλυψης για μεγάλο αριθμό κελιών (όπου ο χρόνος εκτέλεσης είναι ικανοποιητικός) και αυτός ο γράφος επικάλυψης να χρησιμοποιηθεί ως είσοδος για τη δημιουργία του γράφου επικάλυψης με λιγότερα κελιά. Αυτό τεχνικά προϋποθέτει ότι το «κόψιμο» έχει γίνει φωλιασμένα. Δηλαδή ένα κελί στο «κόψιμο» των 128 κελιών θα πρέπει να περιέχει δύο κελιά του «κοψίματος» των 256 κελιών, όπως φαίνεται στο σχήμα 4.11. Το εργαλείο METIS λόγω της διχοτόμησης (βλ. κεφάλαιο 2) μας προσφέρει ακριβώς αυτή τη δυνατότητα. Ο υπολογισμός του γράφου επικάλυψης σε αυτή την περίπτωση γίνεται εντυπωσιακά πιο γρήγορα. Ο συνολικός χρόνος υπολογίζεται ως το άθροισμα του χρόνου που απαιτείται για τον υπολογισμός του γράφου επικάλυψης σε κάθε επίπεδο. Πειράματα έγιναν σε δύο επίπεδα και με βάση τον υπολογισμό του γράφου επικάλυψης για τα 32768 κελιά. Δεδομένου αυτού του γράφου επικάλυψης υπολογίστηκαν όλοι οι χρόνοι για το μικρότερο αριθμό κελιών και φαίνονται στον πίνακα 4.2. Να σημειωθεί ότι στο συνολικό χρόνο δεν περιέχονται τα 770 δευτερόλεπτα που απαιτούνται για τη δημιουργία του γράφου επικάλυψης των 32768 κελιών. Αντίστοιχα στο σχήμα 4.9 φαίνεται γραφικά η ο συνολικός χρόνος που απαιτήθηκε για κάθε μέγεθος κελιού

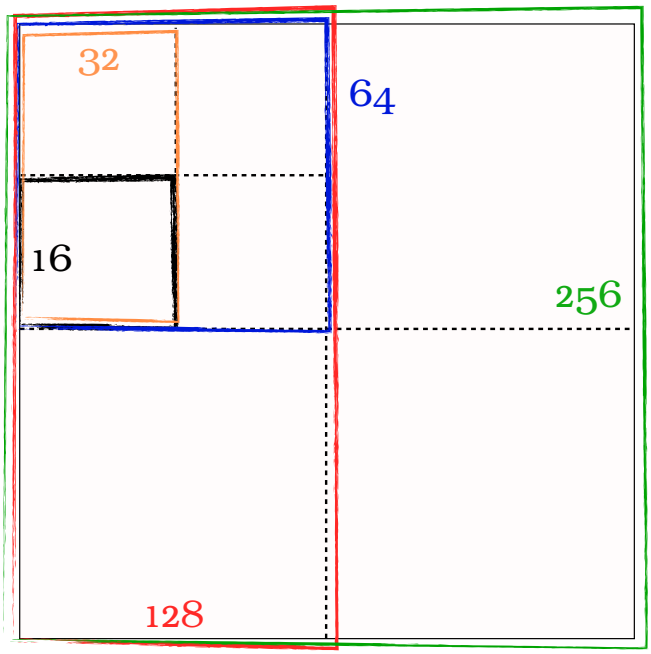
συμπεριλαμβανομένου του χρόνου για τη δημιουργία του γράφου επικάλυψης στο πρώτο επίπεδο. Ο γράφος επικάλυψης με βάση τον οποίο έγιναν οι υπολογισμοί έχει μόλις 7 εκατομμύρια σε σχέση με τα 40 εκατομμύρια του αρχικού γράφου. Γι' αυτό το λόγο είναι πολύ πιο ξεκάθαρη το ελάχιστο που δημιουργείται για το συνολικό χρόνο ανάλυσης του γράφου. Φαίνεται ότι για τα 4096 κελιά η εξισορρόπηση αλγοριθμικού κόστους και κόστους επικοινωνίας (communication overhead) μας δίνει τα καλύτερα αποτελέσματα [13].



Σχήμα 4.9: Μέγεθος ερωτημάτων για την αποστολή του κελιού και την επιστροφή του αποτελέσματος, συναρτήσει του αριθμού των κελιών.

Κελιά	Δίκτυο	Dijkstra	Parsing	Συνολικά	Χρόνος / κελί
128	31 s	713 s	9 s	**755 s**	5898 ms
256	28 s	485 s	10 s	**524 s**	2049 ms
512	30 s	373 s	11 s	**415 s**	810 ms
1024	41 s	261 s	12 s	**314 s**	307 ms
2048	50 s	209 s	13 s	**273 s**	133 ms
4096	167 s	75 s	17 s	**260 s**	63 ms
8192	167 s	100 s	20 s	**288 s**	35 ms
16384	264 s	57 s	21 s	**343 s**	20 ms

Πίνακας 4.2: Πειραματικοί χρόνοι ανάλυσης συνολικού γράφου χρησιμοποιώντας το γράφο επικάλυψης των 32768 κελιών.



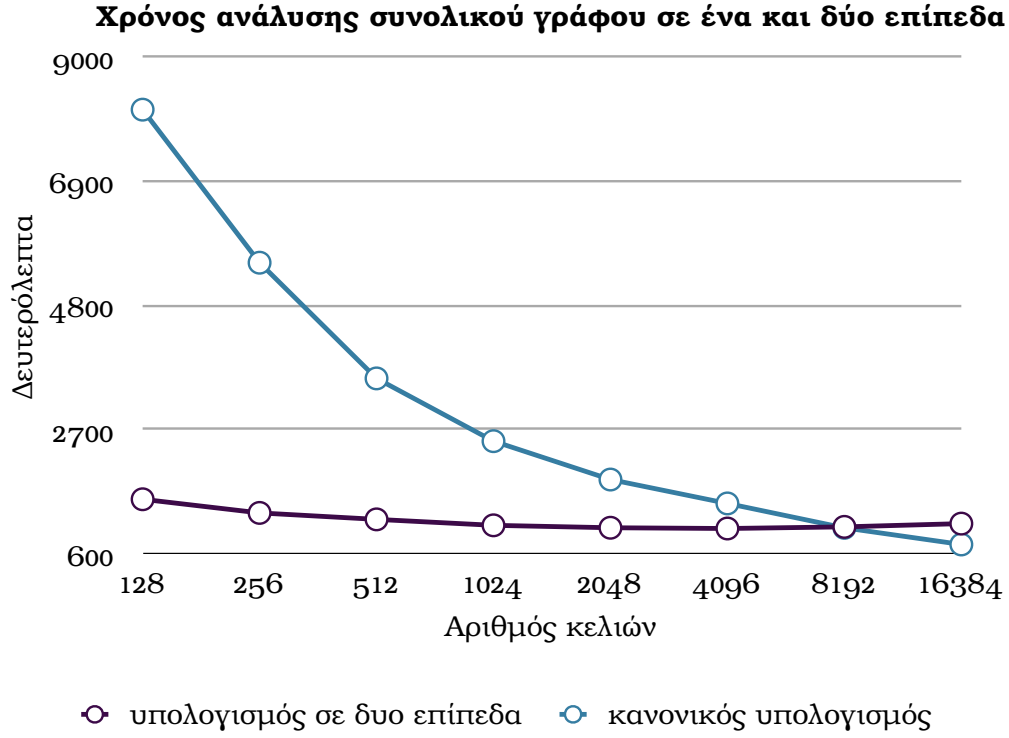
Σχήμα 4.10: Τα κελιά είναι φωλιασμένα λόγω της διχοτόμησης. Ένα κελί σε «κόψιμο» 256 περιέχει δύο κελιά των 128, τέσσερα των 64 κ.ο.κ

4.9 Σύνοψη

Με λίγα λόγια τα πειραματικά αποτελέσματα ήταν αρκετά εντυπωσιακά. Καταφέραμε να υπολογίσουμε όλες τις ακμές επικάλυψης ενός γράφου με 18 εκατομμύρια κόμβους χρησιμοποιώντας ένα μονάχα web browser μέσα σε 12.6 λεπτά, ενώ παραλληλοποιώντας τη διαδικασία ο χρόνος έπεσε σε λιγότερο από 4 λεπτά για τέσσερις επισκέπτες. Από την άλλη πλευρά τα bottlenecks της υλοποίησης έρχονται από δύο αντιδιαμετρικές πλευρές:

- Όσο διατηρήθηκε μικρός ο αριθμός των συνολικών κελιών ο γράφος ήταν τεράστιος. Φανταστείτε ότι κάθε κελί για το «κόψιμο» των 128 κελιών περιέχει 140704 κόμβους σε αντίθεση με τους μόλις 550 που περιέχει κάθε «κελί» στο κόψιμο των 32768. Έτσι ο υπολογισμός των Dijkstra σε τόσο μεγάλα κελιά ήταν πραγματικά δαπανηρός. Η ανάλυση ενός κελιού για το «κόψιμο» των 128 απαιτεί 633 δευτερόλεπτα (εκτέλεση 298 Dijkstra από κάθε συνοριακό κόμβο) ανεβάζοντας το συνολικό υπολογιστικό κόστος στα 7989 δευτερόλεπτα, περισσότερες από δύο ώρες.

- Από την άλλη πλευρά όσο τεμαχίζει κανείς τον αρχικό γράφο δημιουργώντας περισσότερα κελιά, αυξάνεται το κόστος επικοινωνίας (communication overhead). Στα 32768 κελιά ο υπολογιστικός χρόνος μειώθηκε εντυπωσιακά μόλις στα 352 δευτερόλεπτα. Όμως ο αντίστοιχος χρόνος δικτύου έφτασε τα 377



Σχήμα 4.11: Σύγκριση χρόνου ανάλυσης σε ένα και δύο επίπεδα.

δευτερόλεπτα σε σύγκριση με τα 77 δευτερόλεπτα που κόστισε για το «κόψιμο» των 128. Έτσι όσο περισσότερο κόβει κανείς το γράφο, τόσο θα αυξάνεται το κόστος δικτύου.

Η χρυσή τομή βρέθηκε στο κόψιμο των 16384 κελιών όπου συνολικά απαιτήθηκαν μόλις 760 δευτερόλεπτα για την ανάλυση του συνολικού γράφου. Μία πολύ ενδιαφέρουσα ιδέα που μπορεί να επιλύσει το πρόβλημα των μεγάλων κελιών, είναι η πολυεπίπεδη ανάλυση του γράφου. Αυτή η μέθοδος είχε εντυπωσιακά αποτελέσματα μειώνοντας το χρόνο για τα 128 κελιά σε μόλις 1524 δευτερόλεπτα από 8109 δευτερόλεπτα που αρχικά χρειάστηκαν. Έτσι με τη χρήση ακόμη περισσότερων επιπέδων θα μπορούσε κανείς να υπολογίσει κελιά που οι χρόνοι δεν το επιτρέπουν σε ένα επίπεδο. Τέλος έχει αρκετό ενδιαφέρον να αναφερθεί συνοπτικά η πληροφορία που αποστέλλεται. Με τις μεθόδους συμπίεσης που έχουν εφαρμοστεί, κάθε κελί για το «κόψιμο» των 128, έχει μέγεθος περίπου 1.5MB ενώ το αντίστοιχο για τα 32768 κελιά έχει μέγεθος μόλις 7KB. Συνεπώς μία ενδιάμεση λύση (100-200KB) είναι απόλυτα φυσιολογική σε ένα πρακτικό περιβάλλον και με τις ευρυζωνικές συνδέσεις που διαθέτει πλέον κάθε σπίτι.

Κεφάλαιο 5

Επίλογος

5.1 Συμπεράσματα

Σε ένα τεχνολογικό περιβάλλον όπου οι υπολογιστές των χρηστών και τα προγράμματα περιήγησης που χρησιμοποιούν εξελίσσονται καθημερινά, φαίνεται πως η δυνατότητα αξιοποίησης του χρήστη για υπολογισμούς είναι ολοένα και πιο εφικτή. Τα αποτελέσματα αυτής της εργασίας δείχνουν να είναι πολύ ενθαρρυντικά. Μέσα σε λιγότερο από 4 λεπτά, επισκέπτες ενός ιστοτόπου κατάφεραν να δημιουργήσουν ένα γράφο επικάλυψης για ολόκληρη τη Δυτική Ευρώπη. Παράλληλα οι χρήστες αυτοί μπορούν να τον κρατούν ενημερωμένο, απλούστατα ανανεώνοντας τμηματικά τις αλλαγές στα κελιά που έχουν αλλάξει, μέσα σε εκατοστά του δευτερολέπτου. Όλα αυτά γίνονται χωρίς να το συνειδητοποιήσει καθόλου ο επισκέπτης. Από την άλλη πλευρά αναγνωρίζεται ένα σημαντικό κόστος δικτύου που αντιμετωπίζει μία τέτοια εφαρμογή. Παρατηρήθηκε σε πολλές περιπτώσεις το bottleneck του συστήματος να οφείλεται σε αυτό το κόστος. Η αποστολή λιγότερης και συγκεντρωτικής πληροφορίας είναι μία καλή λύση για την αντιμετώπιση τέτοιων προβλημάτων. Παράλληλα η ιδέα μία πολυεπίπεδης ιεραρχίας που παρουσιάστηκε στις τελευταίες ενότητες είναι μία πολύ καλή λύση για τον υπολογισμό γράφων επικάλυψης σε πολύ λίγα κελιά. Φαίνεται ξεκάθαρα ότι ο υπολογισμός με ενδιάμεσα - κρυφά επίπεδα είναι συνολικά πολύ πιο γρήγορος. Έτσι η δημιουργία ενός συστήματος καθοδήγησης με χάρτη που θα στηρίζει τους υπολογισμούς του μόνο στους υπολογιστές των χρηστών φαίνεται να είναι περισσότερο από εφικτή.

5.2 Μελλοντικές επεκτάσεις

Μελλοντικές επεκτάσεις αυτής της εργασίας θα ήταν καταρχάς η δημιουργία ενός συστήματος που θα εκτελεί ερωτήματα στο γράφο επικάλυψης χρησιμοποιώντας ξανά τον επισκέπτη της ιστοσελίδας. Το πρόβλημα μιας τέτοιας υλοποίησης είναι ότι θα πρέπει να αποσταλεί στο χρήστη ολόκληρος ο γράφος επικάλυψης αυτή τη

φορά. Αν και το μέγεθος κάθε κελιού έχει μειωθεί αισθητά σε σύγκριση με τον αρχικό γράφο, ο αριθμός των κελιών που θα αποσταλούν στο χρήστη δε μειώνεται. Ο συνδυασμός της ιεραρχίας που δημιουργήθηκε με κάποια κατεύθυνση στόχου (goal direction [21, 28]) θα ήταν μία πολύ ενδιαφέρουσα μελλοντική εργασία. Με λίγα λόγια η κατεύθυνση στόχου θα αποκλείει κελιά που φαίνεται να μη χρειάζονται για τον υπολογισμό του μονοπατιού. Τελικά η πληροφορία που θα αποστέλλεται στο χρήστη θα μειωθεί σημαντικά. Άλλες μελλοντικές επεκτάσεις θα μπορούσαν να περιλαμβάνουν τη δημιουργία κάποιας γενικευμένης μηχανής map-reduce σε επισκέπτες ιστοσελίδων που δε θα περιορίζεται μονάχα στον υπολογισμό ακμών επικάλυψης.

Βιβλιογραφία

[1] *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1998.

[2] *Συστήματα Βάσεων Δεδομένων*. Εκδόσεις Μ. Γκιούρδας, 2002.

[3] *Αλγόριθμοι: Μέθοδοι Σχεδίασης και Ανάλυσης Πολυπλοκότητας*. Εκδόσεις Συμμετρία, 2006.

[4] *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly Media, 2009.

[5] *Nginx HTTP Server*. PACKT PUBLISHING, 2010.

[6] *Agile Web Development with Rails*. The Pragmatic Programmers, 2011.

[7] *Redis: The Definitive Guide: Data modeling, caching, and messaging*. O'Reilly Media, 2011.

[8] Alexandros Efentakis, Dieter Pfoser and Agnès Voisard. Efficient Data Management in Support of Shortest-Path Computation. *Fourth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, 2011.

[9] Anany Levitin. *Ανάλυση και Σχεδίαση Αλγορίθμων*. Εκδόσεις Τζιόλα, 2007.

[10] Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D. In Transit to Constant Shortest-Path Queries in Road Networks. *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 46-59. SIAM, Philadelphia*, 2007.

[11] Beck, Kent; et al. Manifesto for Agile Software Development. 2001.

[12] C. Demetrescu, A. V. Goldberg, and D. S. Johnson. The Shortest Path Problem. *Ninth DIMACS Implementation Challenge. DIMACS Book 74. AMS*, 2009.

[13] D. Delling, M. Holzer, K. Müller, F. Schulz, and D. Wagner. High-Performance Multi-Level Routing. *In Demetrescu et al. [9], pp. 73-92*.

[14] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. *Algorithmics of Large and Complex Networks, LNCS 5515*, 2009.

[15] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning. *SEA'11 Proceedings of the 10th international conference on Experimental algorithms*, 2011.

[16] Dimitri P. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[17] EW Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959.

[18] Frank Schulz, Dorothea Wagner and Karsten Weihe. Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. *Journal of Experimental Algorithmics, Volume 5*, 2000.

[19] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *International Conference on Parallel Processing*, 1995.

[20] George Karypis, Vipin Kumar. METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System. 1995.

[21] J. Maue, P. Sanders, and D. Matijevic. Goal directed shortest path queries using precomputed cluster distances. *5th Workshop on Experimental Algorithms (WEA), Number 4007 IN LNCS, pages 316–328. Springer*, 2006.

[22] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, 2008.

[23] Jeremy D. Zawodny, Derek J. Balling. *High Performance MySQL*. O'Reilly Media, 2008.

[24] M. Rice and V. J. Tsotras. Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions. *Proc. VLDB Endowment, vol. 4, no. 2*, 2010.

[25] Mohring, R.H., Schilling, H., Schutz, B., Wagner, D., Willhalm, T. Partitioning Graphs to Speed Up Dijkstra's Algorithm. *Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 189–202. Springer, Heidelberg*, 2005.

[26] Muller, K. Design and Implementation of an Efficient Hierarchical Speed-up Technique for Computation of Exact Shortest Paths in Graphs. *Master's thesis, Universitat Karlsruhe (TH), Fakultat fur Informatik*, 2006.

[27] Nicholas C. Zakas. *High Performance Javascript*. O'Reilly, 2010.

[28] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM JEA 15(2.3):1-31*, 2010.

[29] R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning graphs to speedup Dijkstra's. *algorithm. J. Exp. Algorithmics, 11*, February 2007.

[30] Sanders, P., Schultes, D. Highway Hierarchies Hasten Exact Shortest Path Queries. *Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568-579. Springer, Heidelberg*, 2005.

[31] Sungwon Jung, and Sakti Pramanik. An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps. *IEEE Transactions on Knowledge and Data Engineering, Volume 14 Issue 5*, 2002.

[32] Thomas H. Cormen, Chrles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Εισαγωγή στους Αλγορίθμους*. Πανεπιστημιακές Εκδόσεις Κρήτης, 2006.

[33] U.S. Census Bureau, Washington, DC. UA Census 2000 TIGER/Line Files, 2002.