ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Διπλωματική Εργασία

Αναζήτηση Κορυφογραμμής

Νικόλαος Γ. Παπαβασιλείου

Επιβλέποντες: Παναγιώτης Μποζάνης, Επίκουρος Καθηγητής

Δημήτριος Κατσαρός, Συμβασιούχος ΠΔ407/80

Βόλος

Σεπτέμβριος 2009

Περιεχόμενα:

6. υπολογισμός κορυφογραμμής σε δίκτυο αισθητήρων

Πρόλογος

Η παρούσα διπλωματική εργασία είναι μια λεπτομερής εισαγωγή και επεξήγηση του υπολογισμού μιας κορυφογραμμής. Το πρόβλημα του υπολογισμού αυτού είναι ιδιαίτερο και υπάρχουν αρκετοί αλγόριθμοι που εφαρμόζονται για την επίλυση του. Στην προσπάθεια να γίνει πιο κατανοητό το πρόβλημα αυτό καθώς και στην ανάδειξη του σε διάφορα πεδία της πληροφορικής και όχι μόνο, επιχειρήθηκε η προσέγγιση των αλγορίθμων με την χρήση ψευδοκώδικα. Σε κάθε περίπτωση που χρησιμοποιήθηκε ψευδοκώδικας υπήρξε ανάλυση και επεξήγηση του. Για την καλύτερη κατανόηση χρησιμοποιήθηκαν λεπτομερέστατες εικόνες, καθώς και παραδείγματα ώστε ο αναγνώστης να μην περιοριστεί μόνο σε μια επιφανειακή αντίληψη του θέματος.

Όσον αφορά την διάρθρωση της ύλης, στο κεφάλαιο 1 παρουσιάζεται γενικά το πρόβλημα της αναζήτησης κορυφογραμμής, χρησιμοποιείται ένα παράδειγμα προς επίδειξη και παρουσιάζονται ένα μέρος των προβλημάτων που συσχετίζονται με την αναζήτηση. Στο κεφάλαιο 2, παρουσιάζονται με λεπτομέρεια οι βασικοί αλγόριθμοι που χρησιμοποιούνται για τον υπολογισμό της κορυφογραμμής. Σε κάθε αλγόριθμο παρουσιάζεται και ένα παράδειγμα. Στο κεφάλαιο 3, περιγράφεται ο υπολογισμός της κορυφογραμμής σε ροές δεδομένων και στο κεφάλαιο 4 οι συνεχόμενες αναζητήσεις κορυφογραμμής. Στα κεφάλαια 5 και 6, περιγράφεται η εφαρμογή της κορυφογραμμής στα ομότιμα (p2p) δίκτυα καθώς και στα δίκτυα αισθητήρων αντίστοιχα. Στο κεφάλαιο 6, περιγράφεται και ένα υποτιθέμενο σενάριο για την καλύτερη κατανόηση.

Κεφάλαιο 1

Εισαγωγικά-Βασικές έννοιες

Περιεχόμενα

1.1 Εισαγωγή

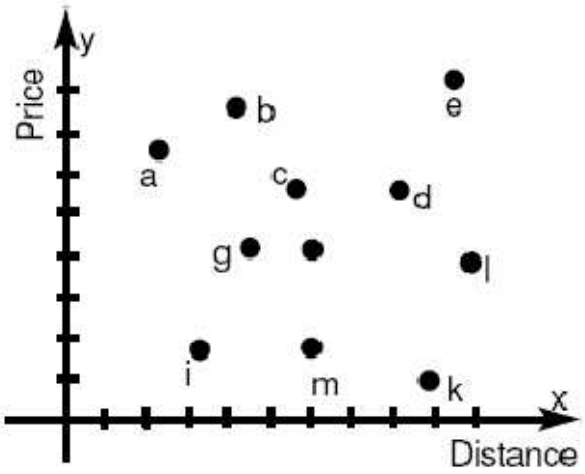
Η κορυφογραμμή ενός συνόλου από d-διαστάσεων σημείων περιέχει τα σημεία που δεν κυριαρχούνται από κανένα άλλο σημείο σε καμία διάσταση. Ο υπολογισμός της κορυφογραμμής έχει αποκτήσει ιδιαίτερο ενδιαφέρον στις βάσεις δεδομένων, ιδιαίτερα στους προοδευτικούς αλγόριθμους που μπορούν να επιστρέψουν το πρώτο σημείο της κορυφογραμμής χωρίς να χρειάζεται να διαβάσουν όλα τα δεδομένα. Υπάρχουν πολλοί αλγόριθμοι που εφαρμόζονται για τον υπολογισμό της κορυφογραμμής καθένας από τους οποίους εξυπηρετεί κάποιες λειτουργίες. Στο κεφάλαιο 2 αναφερόμαστε λεπτομερέστερα στους κυριότερους αλγόριθμος και συγκρίνουμε τις επιδόσεις του.

Το πρόβλημα υπολογισμού της κορυφογραμμής εντοπίζεται σε πολλά τομείς της πληροφορικής και όχι μόνο. Όπως θα δούμε παρακάτω, το πρόβλημα του υπολογισμού εμφανίζεται στις βάσεις δεδομένων, όταν θα πρέπει να παρθεί κάποια απόφαση με βάση πολλά κριτήρια, στα ομότιμα (p2p) δίκτυα καθώς και στα δίκτυα αισθητήρων.

1.2 Παράδειγμα εκτέλεσης αναζήτησης

Ο χειριστής του υπολογισμού της κορυφογραμμής είναι πολύ σημαντικός για αρκετές εφαρμογές όπου θα πρέπει να λάβουμε υπόψη πολλαπλά κριτήρια για την απάντηση του προβλήματος. Δεδομένου ενός συνόλου αντικειμένων p1, p2, ….. , pn, ο χειριστής επιστρέφει όλα τα αντικείμενα pi που έχουν την ιδιότητα να μην κυριαρχούνται από κανένα άλλο αντικείμενο pj.

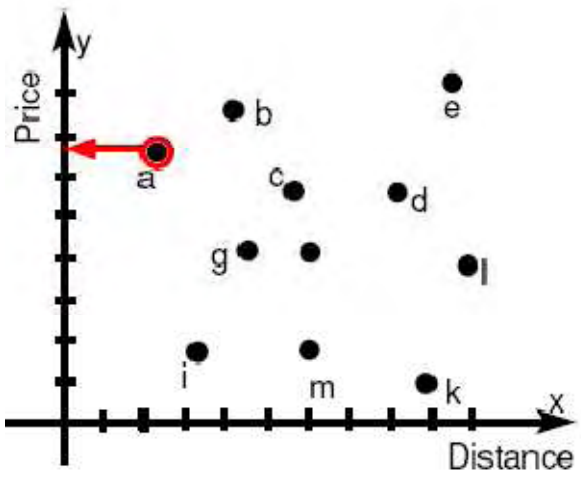
Στην προσπάθεια μας να κάνουμε όσο το δυνατόν πιο κατανοητή την αναζήτηση κορυφογραμμής θα χρησιμοποιήσουμε ένα παράδειγμα. Ένα σύνολο δεδομένων περιέχει πληροφορίες για ξενοδοχεία. Τα δεδομένα που καταγράφονται για κάθε ξενοδοχείο είναι η απόσταση του από την παραλία και η τιμή του δωματίου. Θα θεωρήσουμε ένα δισδιάστατο σχήμα όπου η απόσταση θα ανατίθεται στον άξονα X και η τιμή του δωματίου στον άξονα Y. Το σχήμα έχει την παρακάτω απεικόνιση, όπου κάθε σημείο αντιπροσωπεύει και ένα ξενοδοχείο.



Εικόνα 1.1

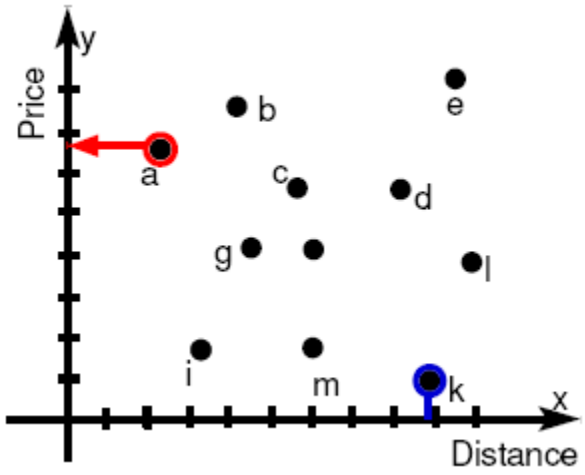
Στόχος της αναζήτησης μας είναι να εντοπιστεί το ξενοδοχείου του οποίου η απόσταση από την παραλία και η τιμή του είναι και οι ελάχιστες δυνατές. Φυσικά θα μπορούσαμε να χρησιμοποιήσουμε και άλλες συναρτήσεις όπως την max, δηλαδή την μεγιστή δυνατή τιμή που μπορούν να πάρουν ταυτόχρονα και οι δυο παράμετροι. Στην περίπτωση μας η συνάρτηση που θα χρησιμοποιήσουμε για την συλλογή των επιθυμητών αποτελεσμάτων είναι «ελάχιστη τιμή και ελάχιστη απόσταση». Από το σύνολο των δεδομένων που έχουμε στην διάθεση μας δεν είναι απαραίτητο ότι θα προκύψει μόνο ένα

σημείο δεδομένων που θα ικανοποιεί της απαιτούμενες ιδιότητες. Το αποτέλεσμα της αναζήτησης θα είναι η εμφάνιση ενός συνόλου από ικανοποιητικά σημεία που θα ικανοποιούν εν μέρει τους περιορισμούς που θέσαμε. Αν μελετήσουμε προσεκτικά το σχήμα θα δούμε ότι το σημείο a έχει την μικρότερη δυνατή απόσταση από την παραλία και σημειώνεται στο παρακάτω σχήμα.



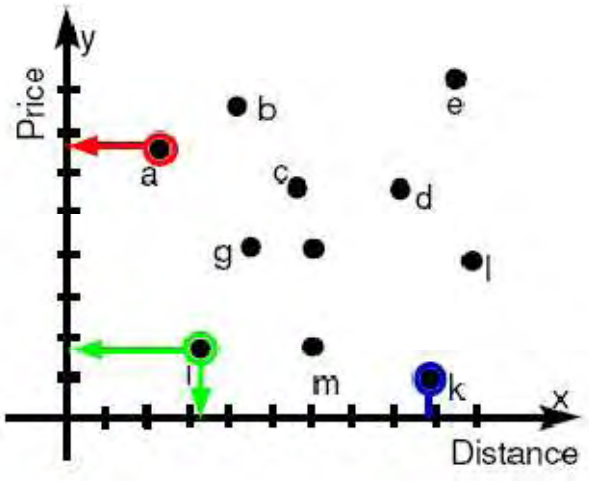
Εικόνα 1.2

Το επόμενο ενδιαφέρον σημείο που παρατηρούμε είναι το k που εμφανίζει την μικρότερη δυνατή τιμή δωματίου και σημειώνεται στο παρακάτω σχήμα.



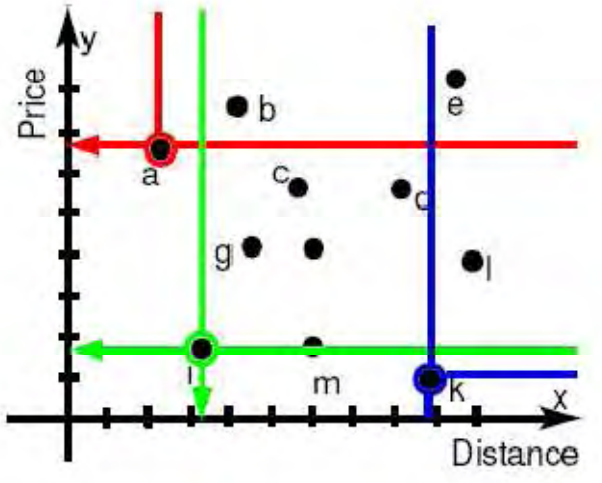
Εικόνα 1.3

Τέλος, αξιοσημείωτο θεωρείτε το σημείο i που δεν έχει ούτε την μικρότερη απόσταση αλλά ούτε και την μικρότερη τιμή και σημειώνεται στο παρακάτω σχήμα.



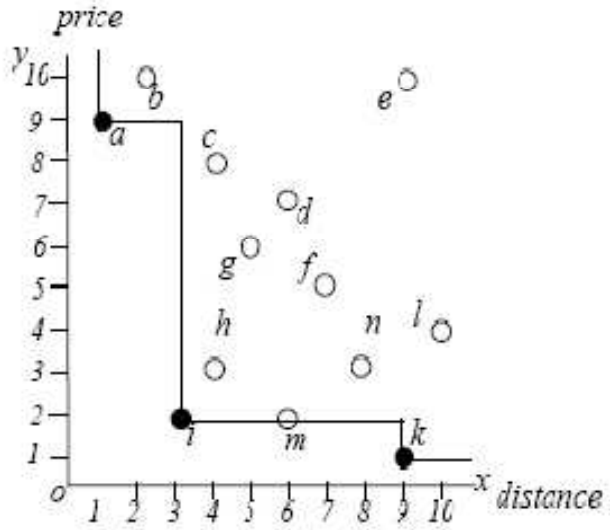
Εικόνα 1.4

Από την παραπάνω ανάλυση το ενδιαφέρον σύνολο σημείων που προκύπτει είναι το {a, i, k}. Όπως παρατηρούμε, το σημείο i έχει μικρότερη απόσταση από το σημείο k και μικρότερη τιμή δωματίου από το σημείο a. Επομένως, με βάση τους ορισμούς που δώσαμε, το σημείο i δεν κυριαρχείται από κανένα από τα δυο σημεία.



Εικόνα 1.5

Όλα τα υπόλοιπα σημεία του σχήματος κυριαρχούνται από το σύνολο των σημείων {a, i, k}, δηλαδή και οι δυο τιμές των παραμέτρων θα είναι μεγαλύτερες από ένα ή περισσότερα σημεία της κορυφογραμμής. Η κορυφογραμμή που προκύπτει από την αναζήτηση είναι η παρακάτω.



Εικόνα 1.6

Οι κορυφογραμμές σχετίζονται άμεσα με αρκετά άλλα προβλήματα όπως τα convex hulls, top-K queries και την αναζήτηση των πλησιέστερων γειτόνων. Πιο συγκεκριμένα, το convex hull περιέχει ένα υποσύνολο σημείων κορυφογραμμής που μπορεί να είναι βέλτιστο μόνο για γραμμικές εξισώσεις. Οι B□hm και Kriegel [2001] πρότειναν έναν αλγόριθμο για convex hulls, που εφαρμόζει την αναζήτηση branch-and-bound σε δεδομένα που καταρτίζονταν σε R-δέντρα.

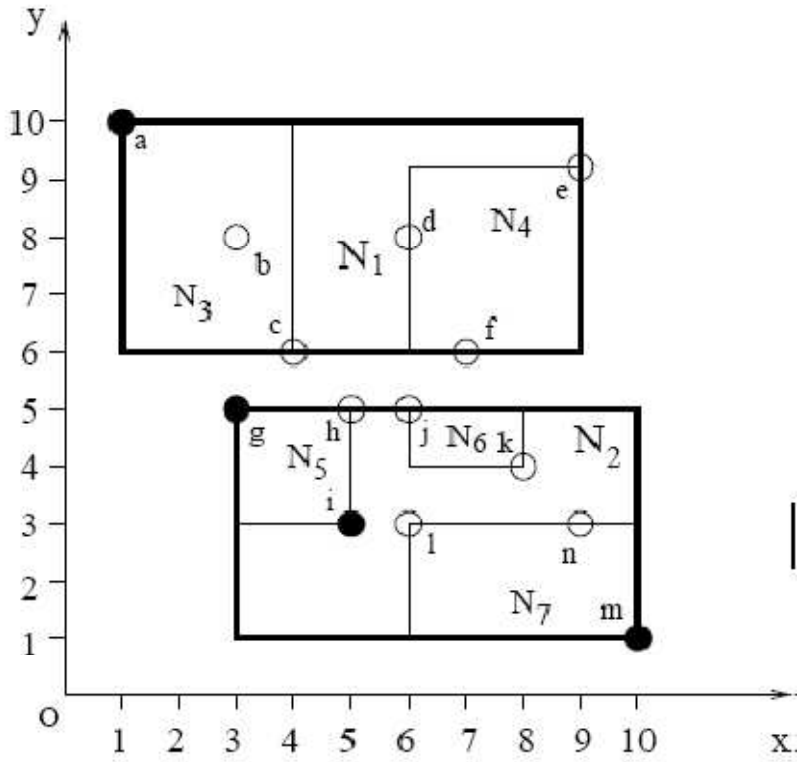
Οι top-K αναζητήσεις επιστρέφουν σαν αποτέλεσμα τις καλύτερες K επιλογές που ελαχιστοποιούν μια συγκεκριμένη συνάρτηση. Για παράδειγμα, θα χρησιμοποιήσουμε την συνάρτηση $f(x,y) = x + y$, που θα μας επιστρέψει τις 3 καλύτερες επιλογές (3-top query), στα δεδομένα του παραδείγματος που χρησιμοποιήσαμε παραπάνω. Επιστρέφονται με την σειρά τα εξής αποτελέσματα: <i,5>, <h,7>,<m,8>, όπου ο αριθμός που συμπεριλαμβάνεται σε κάθε γράμμα είναι το αποτέλεσμα της εφαρμογής της συνάρτησης. Οι διαφορά μεταξύ των αναζητήσεων κορυφογραμμής είναι το αποτέλεσμα που επιστρέφεται από την συνάρτηση, το οποίο διαφέρει ανάλογα με τη συνάρτηση που χρησιμοποιούμε, και από το γεγονός ότι τα σημεία που επιστρέφονται δεν είναι εγγυημένα τμήμα της κορυφογραμμής. Στο παράδειγμα που χρησιμοποιήσαμε τα σημεία h και m που μας επιστράφηκαν κυριαρχούνται από το σημείο i.

Οι αναζητήσεις του πλησιέστερου γείτονα ορίζουν ένα σημείο αναζήτησης q και δίνουν ως αποτέλεσμα τα πλησιέστερα σ' αυτό το σημείο αντικείμενα σε αύξουσα κατά απόσταση σειρά. Αλγόριθμοι που χρησιμοποιούνται σε βάσεις δεδομένων θεωρούν ότι τα αντικείμενα είναι διατεταγμένα σε ένα R-δένδρο και εφαρμόζεται αναζήτηση branch-and-bound. Πιο συγκεκριμένα, ο αλγόριθμος «κατά βάθος πρώτα» ξεκινάει πρώτα από την ρίζα του R-δένδρου και αναδρομικά επισκέπτεται την πλησιέστερη στο σημείο αναζήτησης εγγραφή. Οι εγγραφές, που είναι μακρύτερα από τον πλησιέστερο γείτονα που εντοπίσαμε, αποκόπτονται από το δένδρο. Ο best-first αλγόριθμος εισάγει τις εγγραφές από τους κόμβους που έχουμε επισκεφθεί σε έναν σωρό, και αναζητά τον κόμβο που είναι πλησιέστερος σο σημείο αναζήτησης.

1.3 Προβλήματα συσχέτισης με την αναζήτηση κορυφογραμμής

1.3.1 Top-K

Δεδομένου ενός συνόλου αντικειμένων και μιας συνάρτησης, η αναζήτηση top-K επιστρέφει τα καλύτερα αντικείμενα. Στην εικόνα 1.7, εάν η συνάρτηση που χρησιμοποιούμε είναι η x+y, η αναζήτηση top-3 πρέπει να επιστρέψει τα g, i και l.



Εικόνα 1.7

Υπάρχουν πολλοί αλγόριθμοι που αναπτύχθηκαν και χρησιμοποιήθηκαν για την επίλυση αυτού του προβλήματος.

1.3.2 K-Nearest-Neighbours

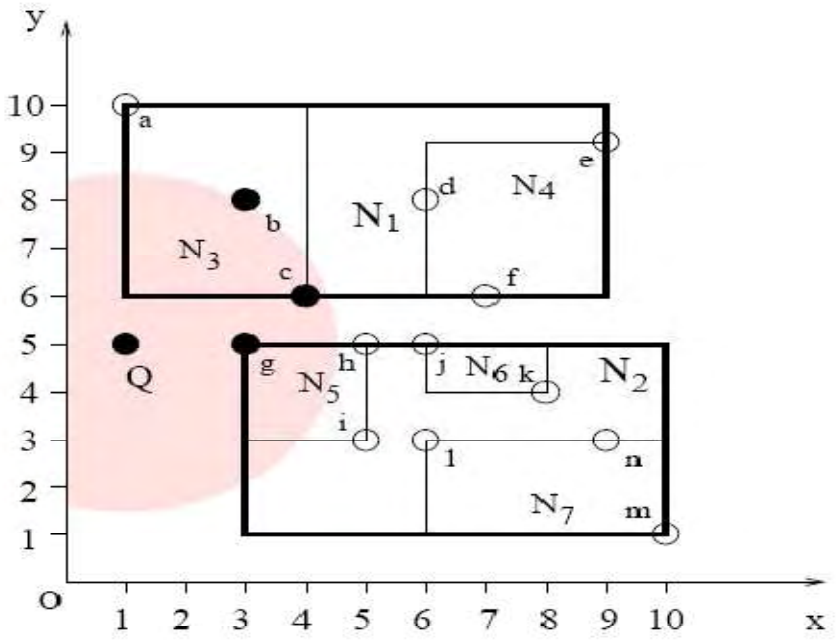
Εάν η συνάρτηση που χρησιμοποιούμε είναι η συνάρτηση υπολογισμού της απόστασης από ένα σημείο p, τα top-K σημεία αποκαλούνται επίσης και k-πλησιέστεροι-γείτονες του p.

Η αναζήτηση των k-πλησιέστερων- γειτόνων είναι ευρέως εφαρμόσιμη. Κάποιες δομές αναπαράστασης δεδομένων δημιουργήθηκαν ειδικά για να απαντούν σ' αυτές τις αναζητήσεις.

Εφόσον, τα R-δένδρα είναι η πιο σημαντική μορφή αναπαράστασης, θα δώσουμε λεπτομέρειες για την διαδικασία υπολογισμού των k-πλησιέστερων – γειτόνων όταν χρησιμοποιούνται R-δένδρα. Με τη χρήση των R-δένδρων, η αναζήτηση μπορεί να πραγματοποιηθεί χωρίς να είναι απαραίτητη η εξέταση όλων των αντικειμένων. Υπάρχουν δύο τρόποι για να διατρέξεις ένα δένδρο, η αναζήτηση κατά βάθος-πρώτα και η καλύτερη-πρώτη αναζήτηση. Στην ακόλουθη περιγραφή, η απόσταση των k-πλησιέστερων-γειτόνων είναι η απόσταση μεταξύ του k υποψήφιου πλησιέστερου-γείτονα και του σημείο αναζήτησης.

Στην αναζήτηση κατά βάθος-πρώτα, ξεκινώντας από την ρίζα του δένδρου, επισκέπτεται πρώτα η εγγραφή με την μικρότερη απόσταση. Η διαδικασία επαναλαμβάνεται αναδρομικά μέχρι να φτάσουμε σε ένα κόμβο-φύλλο που θα είναι υποψήφιος πλησιέστερος-γείτονας. Η απόσταση του k πλησιέστερου γείτονα ενημερώνεται ώστε να απομακρυνθούν άλλοι κόμβοι καθώς οπισθοχωρούμε στα άλλα κλαδιά του R-δένδρου.

Για παράδειγμα, όταν το σημείο αναζήτησης είναι Q στην εικόνα 1.8 και k = 3, η σειρά αναζήτησης με την αναζήτηση κατά βάθος-πρώτα στο R-δένδρο φαίνεται στην εικόνα 1.9.



Εικόνα 1.8

Entry in Process	3NN distance	3NN Candidates
$root$	∞	$\emptyset$
e_1	∞	$\emptyset$
e_3	5	c, b, a
e_2	5	c, b, a
e_5	$\sqrt{13}$	g, c, b

Εικόνα 1.9

Στην καλύτερη-πρώτη αναζήτηση, μια λίστα από τους κόμβους που πρέπει να επισκεφθούμε διατηρείται στην μνήμη. Οι κόμβοι αυτοί είναι ταξινομημένοι κατά αύξουσα σειρά με βάση την απόσταση τους. Επίσης ξεκινώντας από την ρίζα του δένδρου, αναδρομικά επισκεπτόμαστε τον κόμβο με την μικρότερη απόσταση. Ο αλγόριθμος τερματίζει όταν η πρώτη εγγραφή στην λίστα έχει μεγαλύτερη απόσταση από τους υποψήφιους k-πλησιέστερους-γείτονες.

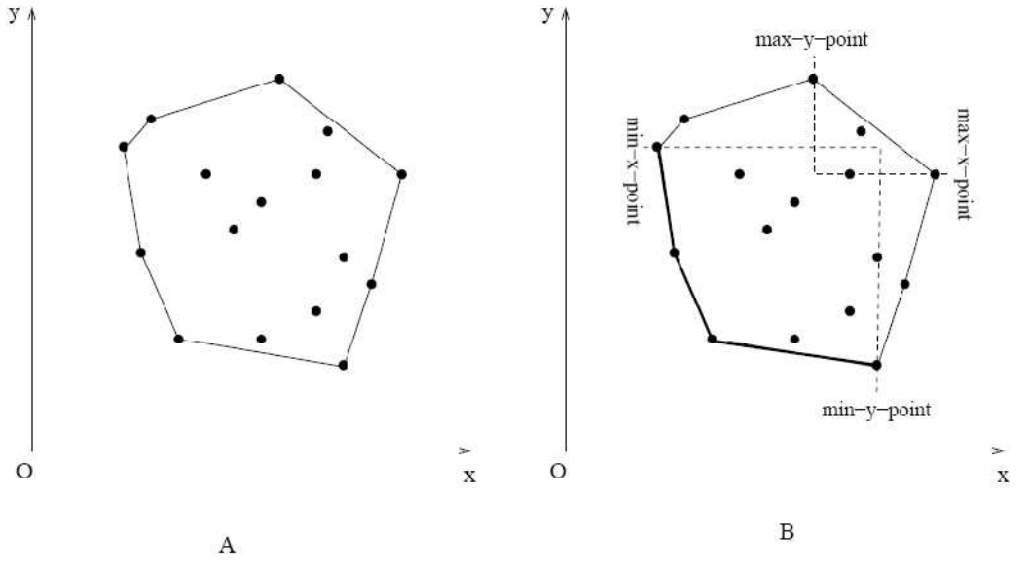
Αυτό το σχήμα αναζήτησης βελτιστοποιεί το I/O κόστος, αλλά έχει μεγάλο κόστος σε μνήμη. Η εικόνα 1.10 παρουσιάζει την εκτέλεση του παραδείγματος της εικόνας 1.8.

Entries in Mem	3NN distance	3NN Candidates
$root$	∞	$\emptyset$
e_1, e_2	∞	$\emptyset$
e_3, e_2, e_4	∞	$\emptyset$
e_2, c, b, a, e_4	∞	$\emptyset$
$e_5, c, b, a, e_6, e_4, e_7$	∞	$\emptyset$
$g, c, b, h, i, a, e_6, e_4, e_7$	$\sqrt{13}$	g, c, b

Εικόνα 1.10

1.3.3 Convex Hull

Το convex hull ενός συνόλου σημείων είναι το μικρότερο κυρτό πολύγωνο που περιέχει όλα τα σημεία. Βλέπουμε ως παράδειγμα την εικόνα 1.11.

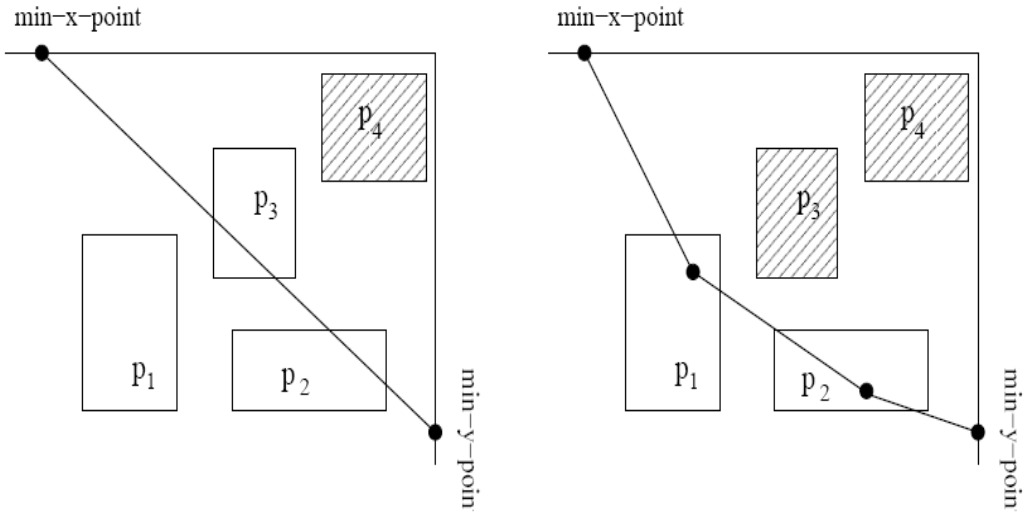


εικόνα 1.11

Ο υπολογισμός του convex hull είναι ένα παραδοσιακό γεωμετρικό πρόβλημα. Το πρόβλημα μελετήθηκε πρώτα σε βάσεις δεδομένων και δυο αλγόριθμοι αναπτύχθηκαν βασιζόμενοι σε δομές πολλαπλών-κλειδιών όπως τα R-δένδρα.

Πριν ξεκινήσουμε την αναζήτηση, 4 ακραία σημεία απεικονίζονται στην εικόνα 1.11 (B). Και οι δυο αλγόριθμοι προσανατολίζονται ώστε να βρουν το τμήμα του convex hull ανάμεσα στον min-x-point και το min-y-point, ενώ άλλα τμήματα μπορούν να βρεθούν με τον ίδιο τρόπο.

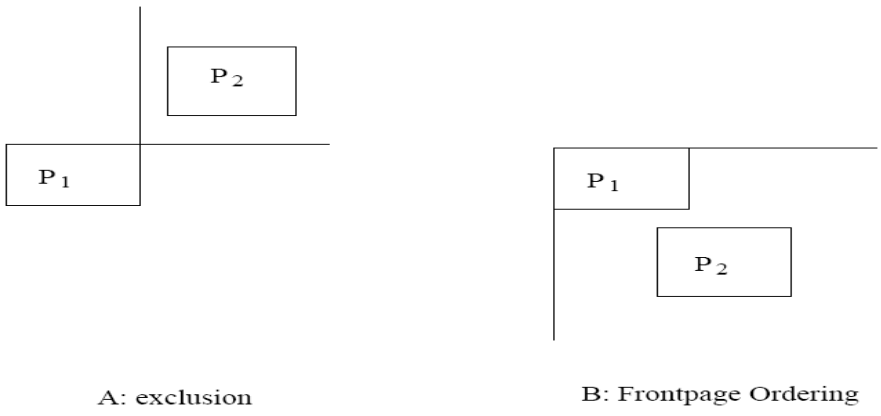
Ο πρώτος αλγόριθμος Προτεραιότητα Απόστασης βασίζεται στην αναζήτηση καλύτερη-πρώτη σε ένα R-δένδρο. Το πρόβλημα είναι πώς θα απομακρύνουμε τα κλαδιά του δένδρου όσο το δυνατόν γρηγορότερα. Ένας σωστός convex hull αλγόριθμος πρέπει να έχει πρόσβαση τουλάχιστον στις σελίδες που τοπολογικά δεν περιέχονται εξολοκλήρου στο πολύγωνο convex hull, ενώ αυτές που περιέχονται ολόκληρες δεν είναι απαραίτητο να τις επισκεφθεί. Η εικόνα 1.12 δείχνει δυο ενδιάμεσα βήματα κατά την αναζήτηση του convex hull. Στην πρώτη εικόνα, η σελίδα p4 μπορεί να απομακρυνθεί, ενώ στην δεύτερη εικόνα μετά την εύρεση δυο σημείων, η p3 μπορεί επίσης να απομακρυνθεί.



εικόνα 1.12

Στην μνήμη διατηρούνται μια λίστα με τις εγγραφές του R-δένδρου apl και το προσωρινό convex hull tch. Αρχικά, το apl περιέχει την ρίζα του δένδρου και το tch το σημείο min-x-point και το min-y-point. Αναδρομικά, η εγγραφή με την μεγαλύτερη απόσταση από το προσωρινό convex-hull επισκέπτεται. Εάν είναι μια ενδιάμεση σελίδα, τα παιδιά της σελίδας αυτής προστίθενται στην apl λίστα. Διαφορετικά, τα παιδία είναι σημεία δεδομένων, και το convex hull ενημερώνεται. Σε κάθε επανάληψη, οι εγγραφές του apl ελέγχονται για απομάκρυνση.

Ο δεύτερος αλγόριθμος κατά βάθος-πρώτα βασίζεται στην αναζήτηση καλύτερη-πρώτη. Βασικά, ο αλγόριθμος καθορίζει δύο σχέσεις: «frontpage διάταξη» και «αποκλεισμός», βασιζόμενος στην σχετική θέση δυο μη-επικαλυπτόμενων περιοχών (εικόνα 1.13).



εικόνα 1.13

Εάν μια περιοχή p2 είναι ολόκληρη στον τομέα πιο πάνω και δεξιά από μια άλλη περιοχή p1, το p2 αποκλείεται από το p1 (εικόνα 1.13(α)). Εάν μια περιοχή p2 είναι ολόκληρη στα δεξιά του p1 ή κάτω του p1, το p1 είναι frontpage ordered πριν από το p2 (εικόνα 1.13(β)).

Στην αναζήτηση κατά βάθος-πρώτα, όταν μια ενδιάμεση σελίδα επισκέπτεται, οι σελίδες παιδιά της ελέγχονται ώστε να απομακρύνουν αυτές που έχουν αποκλειστεί. Οι σελίδες που είναι δεξιά επισκέπτονται κατά την frontpage ordering. Ο αλγόριθμος παρουσιάζεται παρακάτω.

Algorithm distance-priority-ch

Input:

R-tree *R*.

Output:

convex hull between min-x-point and min-y-point.

Description:

```
1: apl := {root};
2: search database for minxpoint and minypoint;
3: while apl ≠ ∅ do
4:    dequeue the element e of apl with maximal distance to tch;
5:    if is-data-page(e) then
6:       for each v of e on the left of tch do
7:          tch := tch ∪ v;
8:          while NOT convex(tch) do
9:             remove next(v) from tch;
10:         end while
11:      end for
12:   else
13:      apl := apl ∪ {child pages of e};
14:   end if
15:   delete all pages on the right of tch from apl;
16: end while
```

οι δυο αλγόριθμοι είναι και οι δυο βέλτιστοι κατά I/O, αλλά η πολυπλοκότητα του κάτω ορίου της CPU είναι διαφορετική. Ο αλγόριθμος Προτεραιότητα Απόστασης τρέχει με $O(nlogn)$, ενώ ο Βάθος-Πρώτα μειώνει την πολυπλοκότητα χειρότερης περίπτωσης σε $O(n)$, εάν το σύνολο των δεδομένων αποθηκεύεται χωρίς επικάλυψη σε ένα πολυδιάστατο τρόπο απεικόνισης δεδομένων.

Βιβλιογραφία

- **An Optimal and Progressive Algorithm for Skyline Queries**
- **The skyline operator**

Κεφάλαιο 2

Αλγόριθμοι για τον υπολογισμό αναζήτησης κορυφογραμμής

Περιεχόμενα

2.1 Εισαγωγή

Το πρόβλημα του υπολογισμού της κορυφογραμμής επιχειρείται να αντιμετωπιστεί από αρκετούς αλγόριθμους. Στο κεφάλαιο αυτό παρουσιάζονται αρκετοί αλγόριθμοι που χρησιμοποιούνται σ αυτό το πρόβλημα. Για την καλύτερη κατανόηση των αλγορίθμων χρησιμοποιείται ψευδοκώδικας αλλά και παραδείγματα ώστε να μην μείνει ο αναγνώστης σε μια επιφανειακή και θεωρητική αντίληψη του θέματος. Οι αλγόριθμοι που περιγράφονται με την σειρά είναι οι παρακάτω:

1. **Διαίρει και βασίλευε**
2. **Block Nested Loop**
3. **Bitmap**
4. **Index**
5. **Nearest Neighbours**
6. **Branch and Bound Skyline**
7. **Sort First Skyline Algorithm.**

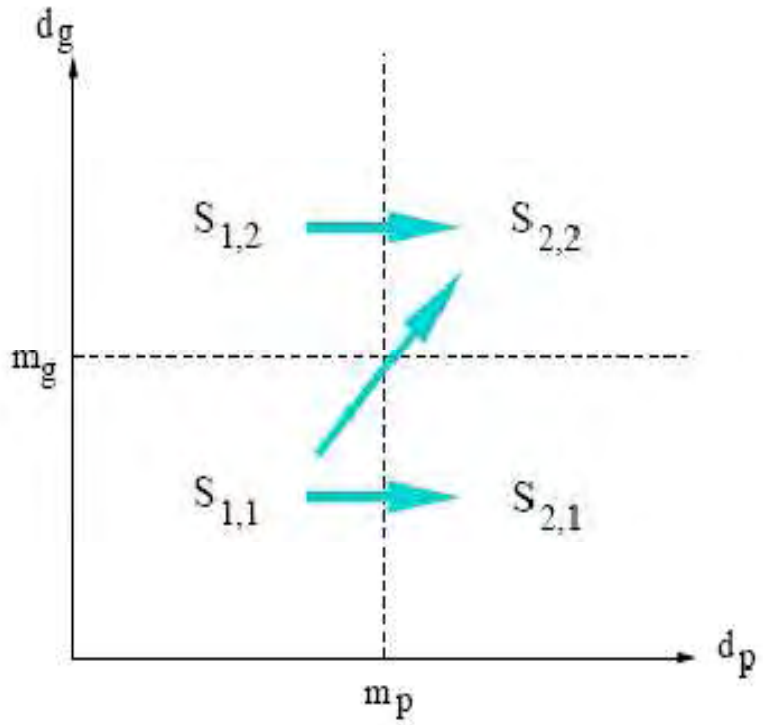
2.2 Divide-and-Conquer

Ο αλγόριθμος divide-and-conquer χωρίζει τα δεδομένα σε ορισμένα τμήματα έτσι ώστε κάθε τμήμα να χωράει στην μνήμη. Ο αλγόριθμος αυτός είναι θεωρητικά ο καλύτερος αλγόριθμος για την χειρότερη περίπτωση. Στην χειρότερη περίπτωση, η πολυπλοκότητα του αλγόριθμου είναι της τάξης του $O\left(n * (logn)^{d-2}\right) + O(n * logn)$; όπου n είναι ο αριθμός των εγγραφών και d ο αριθμός των διαστάσεων.

Ο βασικός divide-and-conquer αλγόριθμος λειτουργεί ως εξής:

1. *Υπολογίζω το μέσο mp (ή κάποιο κατά προσέγγιση μέσο) της εισόδου για κάποια διάσταση dp. Διαιρούμε την είσοδο σε δυο μέρη, το P1 και P2. Το P1 περιλαμβάνει τα σημεία, οι τιμές των οποίων είναι καλύτερες από την mp. Το P2 περιλαμβάνει όλα τα άλλα σημεία.*
2. *Υπολογίζουμε το σημεία της κορυφογραμμής S1 του P1 και S2 του P2. Αυτό επιτυγχάνεται με την αναδρομική εφαρμογή όλου του αλγόριθμου στο P1 και το P2; Το P1 και το P2 διαιρούνται ξανά. Η αναδρομική εκτέλεση σταματά εάν καταλήξει σε ένα τμήμα που περιέχει μόνο ένα σημείο.*
3. *Υπολογίζουμε την συνολική κορυφογραμμή ως το αποτέλεσμα της συγχώνευσης του S1 και S2. Εξαλείφουμε τα σημεία του S2 που κυριαρχούνται από τα σημεία του S1. Κανένα από τα σημεία του S1 δεν είναι δυνατόν να κυριαρχείται από κάποιο σημείο του S2 γιατί ένα σημείο του S1 είναι καλύτερο σε διάσταση dp από κάθε άλλο σημείο του S2.*

Το πιο προκλητικό σημείο του παραπάνω αλγόριθμου είναι το βήμα 3. Το βασικό τέχνασμα που χρησιμοποιούμε σ' αυτό το βήμα φαίνεται στην εικόνα 1. Η βασική ιδέα είναι η διαίρεση και του S1 αλλά και του S2 χρησιμοποιώντας ένα μέσο (κατά προσέγγιση) mg για μια άλλη διάσταση dg, όπου dg≠dp. Η παραπάνω κίνηση έχει ως αποτέλεσμα την δημιουργία τεσσάρων τμημάτων: S1,1 , S1,2 , S2,1 , S2,2. Το τμήμα S1,i είναι καλύτερο από το S2,i στην διάσταση dp και το Si,1 είναι καλύτερο από το Si,2 στην διάσταση dg (i = 1,2). Στην συνέχεια χρειάζεται να συγχωνεύσουμε το S1,1 με S2,1 , S1,1 με S2,2 , και S1,2 με S2,2. Το σημαντικό της υπόθεσης είναι ότι δεν χρειάζεται να συγχωνεύσουμε τα S1,2 και S2,1, καθώς οι εγγραφές και των δυο αυτών συνόλων δεν είναι δυνατόν να συγκριθούν. Η συγχώνευση των S1,1 και S2,1 πραγματοποιείται αναδρομικά εφαρμόζοντας την συγχώνευση. Το S1,1 και το S2,2 διαιρούνται ξανά. Η αναδρομική συγχώνευση τερματίζεται εάν έχουν ελεγχθεί όλες οι διαστάσεις ή εάν κάποιο τμήμα είναι άδειο ή περιέχει μία εγγραφή.



Εικόνα 1

Παρακάτω περιγράφω αναλυτικά σε ψευδοκώδικα τον βασικό αλγόριθμο του Divide-and-Conquer.

B Basic Divide-and-Conquer Algorithm

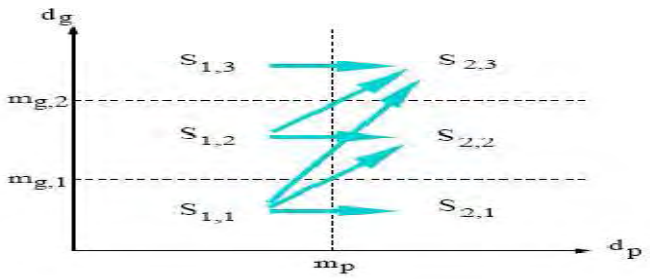
```
function SkylineBasic(M,Dimension)
begin
if |M| = 1 then return M                                   // terminate the recursion
Pivot := Median{m_Dimension|m ∈ M}                         // equi-partition the set
(P1,P2) := Partition(M,Dimension,Pivot)
S1 := SkylineBasic(P1,Dimension)                           // compute skyline recursively
S2 := SkylineBasic(P2,Dimension)
return S1 ∪̇ MergeBasic(S1,S2,Dimension)
end

function MergeBasic(S1,S2,Dimension)
begin
if S1 = {p} then R := {q ∈ S2|p ⊀ q}                       // trivial cases
else if S2 = {q} then begin
   R := S2
   foreach p ∈ S1 do if p ≺ q then R := ∅
end else if Dimension = 2 then begin                       // dimension is low
   Min := Minimum{p1|p ∈ S1}
   R := {q ∈ S2|q1 < Min}
end else begin                                             // general case
   Pivot := Median{p_Dimension−1|p ∈ S1} // partition both sets
   (S1,1,S1,2) := Partition(S1,Dimension − 1,Pivot)
   (S2,1,S2,2) := Partition(S2,Dimension − 1,Pivot)
   R1 := MergeBasic(S1,1,S2,1,Dimension) // compare adjacent parts
   R2 := MergeBasic(S1,2,S2,2,Dimension)
   R3 := MergeBasic(S1,1,R2,Dimension − 1) // compare "diagonally"
   R := R1 ∪̇ R3
end
return R
end
```

2.2.1 Επέκταση του βασικού αλγορίθμου

M-way τμηματοποίηση Αν η είσοδος δεν ταιριάζει στην κυρίως μνήμη, ο βασικός αλγόριθμος που περιγράψαμε παραπάνω έχει πολύ κακή απόδοση. Ο λόγος στον οποίο οφείλεται αυτή η απόδοση είναι το γεγονός ότι τα δεδομένα που δίνονται ως είσοδο διαβάζονται, τμηματοποιούνται, γράφονται στον δίσκο, διαβάζονται ξανά ώστε να τμηματοποιηθούν ξανά, και η διαδικασία επαναλαμβάνεται έως ότου ένα τμήμα να ταιριάζει στην κύρια μνήμη. Κάποιος θα μπορούσε να διαφωνήσει λέγοντας ότι οι κυρίως μνήμες επεκτείνονται όλο και περισσότερο κάτι όμως που εμφανίζεται ταυτόχρονα και στις βάσεις δεδομένων. Αυτό έχεις ο αποτέλεσμα να απαιτούνται περισσότερες ταυτόχρονες αναζητήσεις, κάτι που περιορίζει την κυρίως μνήμη. Γι ‘ αυτό το λόγο οι σχεδιαστές βάσεων δεδομένων δεν επιθυμούν να υλοποιήσουν έναν αλγόριθμο που στηρίζεται στο γεγονός ότι όλα τα δεδομένα χωράνε στην κυρίως μνήμη.

Η I/O συμπεριφορά του αλγορίθμου μπορεί να βελτιωθεί αρκετά εύκολα. Η ιδέα είναι να διαιρέσουμε την είσοδο σε m τμήματα τέτοια ώστε κάθε τμήμα να χωράει στην μνήμη. Αντί του μέσου που χρησιμοποιούσαμε για να πραγματοποιήσουμε την τμηματοποίηση, υπολογίζονται α-σημεία με σκοπό να καθορίσουν τα όρια των τμημάτων. Εάν κάποιο τμήμα δεν χωράει στην μνήμη, πρέπει να χωριστεί ξανά.

Ο m-way χωρισμός μπορεί να χρησιμοποιηθεί τόσο στο πρώτο αλλά όσο και στο τρίτο τμήμα βήμα του βασικού αλγόριθμου. Στο πρώτο βήμα, η m-way τμηματοποίηση χρησιμοποιείται για να δημιουργήσει m τμήματα P1,…., Pm τέτοια ώστε κάθε Pi να χωράει στην μνήμη και κάθε Si, κορυφογραμμή του Pi, να υπολογίζεται στην μνήμη χρησιμοποιώντας τον βασικό αλγόριθμο. Η τελική απάντηση υλοποιείται στο τρίτο βήμα με την συγχώνευση των Si. Με την συνάρτηση συγχώνευσης, εφαρμόζονται και m-way τμηματοποίηση έτσι ώστε όλα τα υπό-τμήματα να μπορούν να συγχωνευτούν στην κυρίως μνήμη. Στην εικόνα 2 φαίνονται πια υπό-τμήματα χρειάζεται να συγχωνευτούν εάν εφαρμόσουμε μια 3-way υπό-τμηματοποίηση στην συνάρτηση τμηματοποίησης.



Εικόνα 2

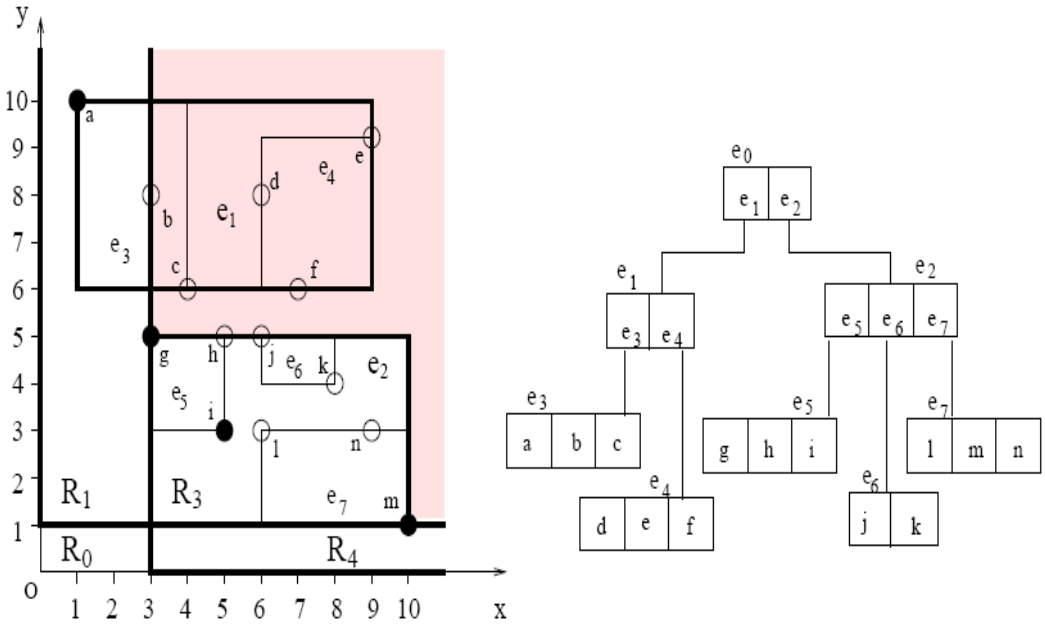
```
function SkylineMway(M,Dimension)
begin
Partitions := Minimum{2^n | n ∈ N ∧ |S| · 2^n > |M|}                    // partition the set
Quantiles := α-Quantiles(M,Dimension,Partitions)
(P_1, ..., P_Partitions) := Partition(M,Dimension,Quantiles)
for i := 1 to Partitions do                                              // compute skyline recursively
   if |P_i| < |S| then S_i := SkylineBasic(P_i,Dimension)
      else S_i := SkylineMway(P_i,Dimension)
while Partitions > 1 do begin
   for i := 1 to Partitions/2 do S_i := S_i ∪̇ MergeMway(S_2i-1,S_2i,Dimension)
   Partitions := Partitions/2
end
return S_1
end

function MergeMway(S_1,S_2,Dimension)
begin
if S_1 = {p} then R := {q ∈ S_2 | p ⊀ q}                                 // trivial cases
else if S_2 = {q} then begin
   R := S_2
   foreach p ∈ S_1 do if p ≺ q then R := ∅
end else if |S_1| + |S_2| < |S| then R := MergeBasic(S_1,S_2,Dimension)
else if Dimension = 2 then begin                                         // dimension is low
   Min := Minimum{p_1 | p ∈ S_1}
   R := {q ∈ S_2 | q_1 < Min}
end else begin                                                           // general case
   Partitions := Maximum{⌈|S_1|/|S|/2⌉, ⌈|S_2|/|S|/2⌉} // partition both sets
   Quantiles := α-Quantiles(S_1,Dimension − 1,Partitions)
   (S_1,1, ..., S_1,Partitions) := Partition(S_1,Dimension − 1,Quantiles)
   (S_2,1, ..., S_2,Partitions) := Partition(S_2,Dimension − 1,Quantiles)
   R := ∅
   for j := 1 to Partitions do begin // compare all parts
      for i := 1 to j do
         if i < j then S_2,j := MergeMway(S_1,i,S_2,j,Dimension − 1)
            else S_2,j := MergeMway(S_1,i,S_2,j,Dimension)
      append(R,S_2,j)
   end
end
return R
end
```

Early skyline Σε περίπτωση που η διαθέσιμη κυρίως μνήμη είναι περιορισμένη, υπάρχει μια εύκολη επέκταση του βασικού αλγορίθμου. Η επέκταση αυτή αναφέρεται στο πρώτο βήμα του αλγόριθμου κατά το οποίο τα δεδομένα χωρίζονται σε m τμήματα. Τα βήματα που ακολουθούνται είναι τα εξής:

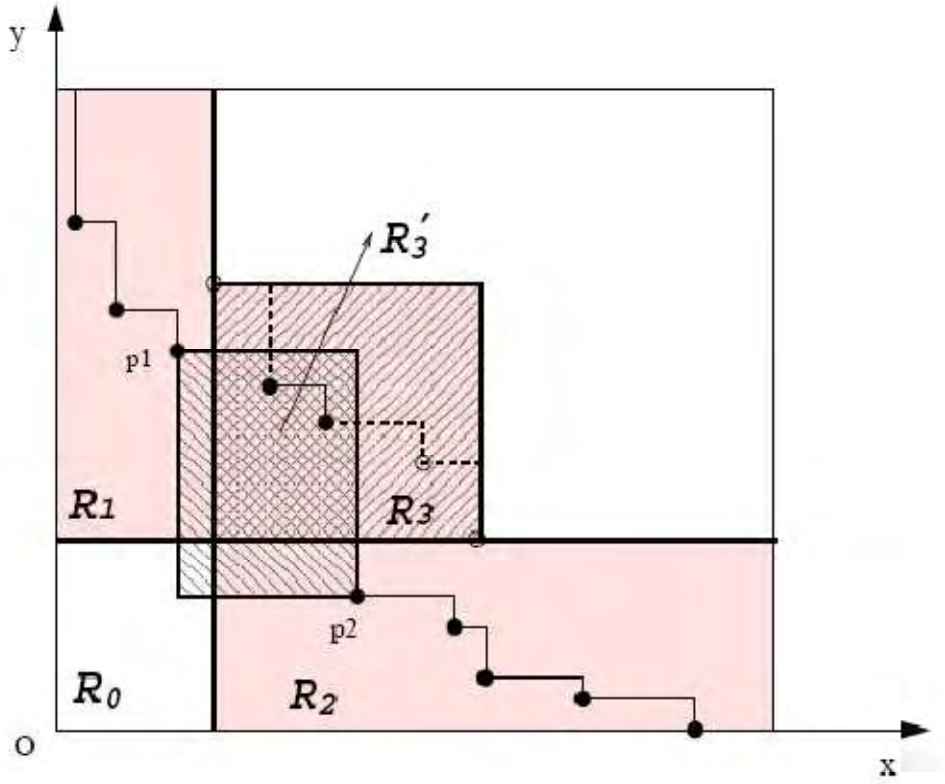
1. Φορτώνουμε όσες περισσότερες πλειάδες μπορούμε (block) από τα δεδομένα εισόδου ώστε να χωράνε στους ενταμιευτές της κυρίως μνήμης.
2. Εφαρμόζουμε τον βασικό αλγόριθμο στο block των πλειάδων με σκοπό να εξαλείψουμε τις πλειάδες που κυριαρχούνται από άλλες.
3. Τμηματοποιούμε τις υπόλοιπες πλειάδες σε m τμήματα.

Είναι αρκετά ξεκάθαρο ότι η εφαρμογή της παραπάνω επέκτασης επιφέρει μεγαλύτερο CPU κόστος, αλλά σώζει I/O καθώς λιγότερες πλειάδες χρειάζεται να γραφτούν και να διαβαστούν ξανά κατά την διάρκεια της τμηματοποίησης.

Στην συνέχεια θα παρουσιάσουμε ένα παράδειγμα εκτέλεσης του αλγορίθμου. Όπως φαίνεται στην παρακάτω εικόνα, υποθέτουμε ότι στην ορθογώνια περιοχή R, το σημείο e2 έχει την ελάχιστη απόσταση στην περιοχή. Στο σημείο όπου γίνεται ο χωρισμός, το R χωρίζεται σε 4 ορθογώνιες περιοχές: R_0, R_1, R_2 *και* R_3.



Εικόνα 3



Εικόνα 4

Όπως φαίνεται στην εικόνα 4, ο αλγόριθμος εφαρμόζει αναδρομικά τα παρακάτω βήματα:

1. Βρίσκει το σημείο κορυφογραμμής στην περιοχή R1, και επιστέφει το χαμηλότερο σημείο p1.
2. Βρίσκει το σημείο κορυφογραμμής στην περιοχή R2, και επιστρέφει το δεξιά πιο σημαντικό σημείο p2.
3. Χρησιμοποιεί τα p1 και p2 για να πάρει το R3', και βρίσκει το σημείο κορυφογραμμής R3'.

Αυτή διαδικασία χωρισμού συνεχίζεται μέχρι να συμβεί μια από τις παρακάτω περιπτώσεις σε μια υπο-περιοχή:

1. Περιέχει μόνο μια εγγραφή, η οποία έχει εξεταστεί, όπως και τα παιδιά της.
2. Περιέχει μόνο ένα σημείο δεδομένων, και το σημείο αυτό επιστρέφεται ως σημείο κορυφογραμμής.
3. Είναι άδεια, οπότε και απομακρύνεται αυτή η περιοχή.

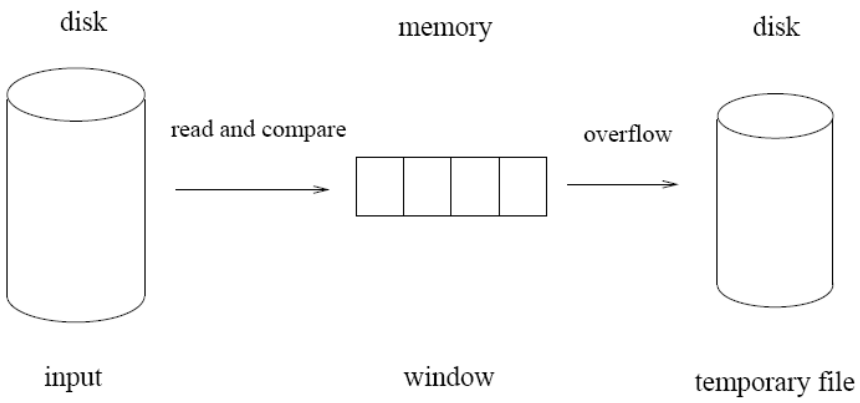
Στον παρακάτω πίνακα, φαίνονται οι είσοδοι σε κάθε υπό-περιοχή βήμα προς βήμα. Κάθε περιοχή περιέχει την δική της λίστα εισόδου. Το δεύτερο πεδίο σε κάθε είσοδο είναι η τιμή της ελάχιστης απόστασης, που υπολογίζεται χρησιμοποιώντας την συνάρτηση $x + y$.

Action	Region list Contents	Skyline Points	Region+ Entry
initial state	$[R, \{(e_0, 2)\}]$	$\emptyset$	$1 + 1$
expand e_0 & divide R	$[R_1, \{(e_1, 7)\}]$,$[R_3, \{(e_2, 4)\}]$	$\emptyset$	$2 + 2$
expand e_1	$[R_1, \{(e_3, 7)\}]$,$[R_3, \{(e_2, 4)\}]$	$\emptyset$	$2 + 2$
expand e_3	~~$[R_1, \{(a, 11)\}]$~~,$[R_3, \{(e_2, 4)\}]$	{a}	$2 + 2$
expand e_2 & divide R_3	$[R_{23}, \{(e_7, 7)\}]$,$[R_{33}, \{(e_5, 6)\}]$	{a}	$2 + 2$
expand e_7	~~$[R_{23}, \{(m, 11)\}]$~~,$[R_{33}, \{(e_5, 6)\}]$	{a,m}	$2 + 2$
expand e_5 & divide R_{33}	~~$[R_{233}, \{(i, 8)\}]$~~,$[R_{333}, \{(g, 8)\}]$	{a,m,i}	$2 + 2$
divide R_{333}	~~$[R_{333}, \{(g, 8)\}]$~~	{a,m,i,g}	$1 + 1$

Εικόνα 5

2.3 Block Nested Loop

Μια απλή προσέγγιση για τον υπολογισμό της κορυφογραμμής είναι να συγκρίνουμε κάθε σημείο p με όλα τα υπόλοιπα σημεία, και να αναφέρουμε το p σαν μέρος της κορυφογραμμής εάν δεν κυριαρχείται από κάποιο σημείο. Ο αλγόριθμος Block Nested Loop δημιουργήθηκε πάνω σ' αυτήν την ιδέα σαρώνοντας τα δεδομένα και κρατώντας σε μια λίστα όλα τα υποψήφια σημεία στην κυρίως μνήμη.



Εικόνα 6

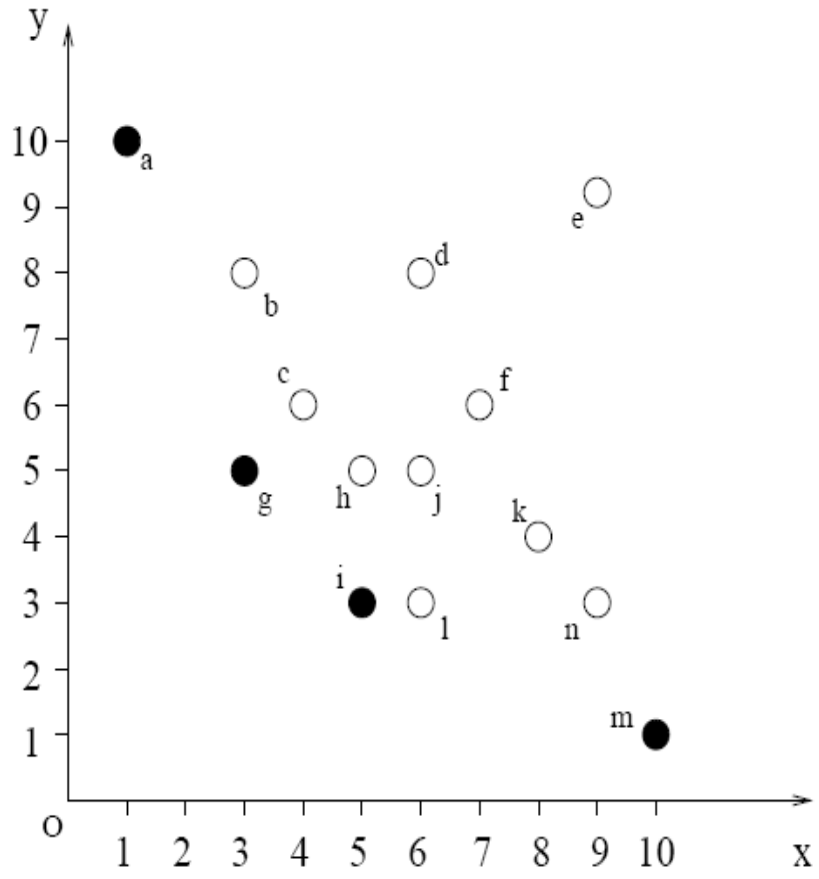
Αρχικά, η λίστα περιέχει το πρώτο σημείο, ενώ για κάθε επόμενο σημείο p, υπάρχουν τρεις υποθέσεις:

1. Εάν το p κυριαρχείται από κάποιο σημείο στη λίστα, απορρίπτεται καθώς δεν μπορεί να είναι μέρος της κορυφογραμμής. Φυσικά, δεν χρειάζεται να συγκριθεί με τα υπόλοιπα σημεία της λίστας.
2. Εάν το p κυριαρχεί έναντι κάποιου σημείου στην λίστα, εισάγεται και όλα τα σημεία που κυριαρχούνται από το p απομακρύνονται.
3. Εάν το p δεν κυριαρχείται αλλά ούτε κυριαρχεί έναντι κάποιου σημείου στη λίστα, απλά εισάγεται χωρίς να απαιτείται η απομάκρυνση κάποιου σημείου.

Η λίστα οργανώνεται αυτόματα , καθώς κάθε σημείο που κυριαρχεί έναντι κάποιων άλλων μετακινείται στην κορυφή της λίστας. Η λειτουργία αυτή μειώνει τον αριθμό των συγκρίσεων γιατί σημεία που κυριαρχούν έναντι πολλαπλών σημείων έχουν μεγαλύτερη πιθανότητα να ελεγχθούν πρώτα. Ένα πρόβλημα που εντοπίζεται σ' αυτόν τον αλγόριθμο είναι η περίπτωση η λίστα να γίνει μεγαλύτερη από την διαθέσιμη κυρίως μνήμη. Όταν

συμβαίνει η παραπάνω περίπτωση, όλα τα σημεία που εντάσσονται στη τρίτη περίπτωση του αλγορίθμου (η πρώτη και η δεύτερη περίπτωση δεν οδηγούν σε αύξηση του μεγέθους της λίστας) αποθηκεύονται σε προσωρινό αρχείο. Το γεγονός αυτό απαιτεί πολλαπλά περάσματα του αλγορίθμου. Πιο συγκεκριμένα, μετά την ολοκλήρωση του σαρώματος των δεδομένων του αρχείου, μόνο τα σημεία που είχαν εισαχθεί στην λίστα πριν την δημιουργία του προσωρινού αρχείου είναι εγγυημένα μέρη της κορυφογραμμής. Τα υπόλοιπα σημεία πρέπει να συγκριθούν με αυτά του προσωρινού αρχείου. Επομένως, ο BNL πρέπει να εκτελεστεί ξανά, χρησιμοποιώντας όμως αυτή την φορά το προσωρινό αρχείο σαν είσοδο.

Υποθέτοντας ότι το μέγεθος της λίστας είναι δυο, παρακάτω εκτελούμε τον αλγόριθμο BNL.



Εικόνα 7

Στον παρακάτω πίνακα παρουσιάζουμε συνοπτικά αλλά και οργανωμένα όλα τα βήματα του αλγορίθμου στα δεδομένα του διαγράμματος.

Action	Window Contents	Temporary File	Skyline
Loop 1:			
read a	a	$\emptyset$	$\emptyset$
read b	a,b	$\emptyset$	$\emptyset$
read c	a,b,c	$\emptyset$	$\emptyset$
read d	a,b,c	$\emptyset$	$\emptyset$
read e,f	a,b,c	$\emptyset$	$\emptyset$
read g	a,g	$\emptyset$	$\emptyset$
read h	a,g	$\emptyset$	$\emptyset$
read i	a,g,i	$\emptyset$	$\emptyset$
read j,k,l	a,g,i	$\emptyset$	$\emptyset$
read m	a,g,i	m	$\emptyset$
read n	a,g,i	m,n	$\emptyset$
Loop 2:			
read m	m	$\emptyset$	a,g,i
read n	m	$\emptyset$	a,g,i
end	$\emptyset$	$\emptyset$	a,g,i,m

Εικόνα 8

Το πλεονέκτημα του BNL είναι ευρέως εφαρμόσιμο, από την στιγμή που μπορεί να χρησιμοποιηθεί σε οποιαδήποτε διάσταση χωρίς να δεικτοδότηση ή ταξινόμηση των δεδομένων. Τα κυριότερα προβλήματα του αλγόριθμου είναι η εξάρτηση του από την κυρίως μνήμη (μικρή μνήμη μπορεί να οδηγήσει σε πολλαπλές επαναλήψεις) και η ανεπάρκεια του για προοδευτική επεξεργασία (πρέπει να διαβάσει ολόκληρο το αρχείο δεδομένων για να επιστρέψει το πρώτο σημείο της κορυφογραμμής).

2.4 Bitmap

Ο bitmap αλγόριθμος κωδικοποιεί κάθε σημείο δεδομένων με μια χαρτογράφηση σύμφωνα με την θέση της τιμή του σε κάθε διάσταση. Όλες οι χαρτογραφήσεις επιτρέπουν στον αλγόριθμο να αποφασίσει αποτελεσματικά εάν ένα σημείο είναι σημείο κορυφογραμμής σύμφωνα με κάποια λειτουργία. Θεωρούμε ένα σημείο p = (p[1],p[2],…,p[d]), όπου d είναι η διάσταση. Κάθε συντεταγμένη p[i] (1≤i≤d) μετατρέπεται σε ένα διάνυσμα μήκους mi, όπου mi είναι ο αριθμός των διακριτών τιμών στην i διάσταση, στην οποία τα (mi – θέση(pi) + 1) πιο σημαντικά bit είναι 1 και τα υπόλοιπα είναι 0. Μετά την μετατροπή, κάθε σημείο δεδομένων χαρτογραφείται από ένα m-bit διάνυσμα, όπου $m = \sum_{i=1}^{d} m_i$. Για παράδειγμα, στην προηγούμενη εικόνα m1=10 και m2=10. Το σημείο c(4,6) σημειώνεται ως (1111111000,1111100000). Για να ελέγξουμε ως προς την κυριαρχία τα σημεία, συγκρίνονται οι χαρτογραφημένες απεικονίσεις τους.

Η παρακάτω εικόνα δείχνει την χαρτογραφημένη απεικόνιση του παραπάνω διαγράμματος. Για να ελέγξουμε αν το σημείο c ανήκει στην κορυφογραμμή, το 7^{ο} και 5^{ο} bit από τις απεικονίσεις του επιλέγονται από τον πρώτο και τον δεύτερο άξονα αντίστοιχα, τα οποία τονίζονται και στην εικόνα.

Point	Bitmap Representation
$a(1,10)$	$(1111111111, 1000000000)$
$b(3,8)$	$(1111111100, 1110000000)$
$\mathbf{c(4,6)}$	$(1111111000, 1111100000)$
$d(6,8)$	$(1111100000, 1110000000)$
$e(9,9)$	$(1100000000, 1100000000)$
$f(7,6)$	$(1111000000, 1111100000)$
$g(3,5)$	$(1111111100, 1111110000)$
$h(5,5)$	$(1111110000, 1111110000)$
$i(5,3)$	$(1111110000, 1111111100)$
$j(6,5)$	$(1111100000, 1111110000)$
$k(8,4)$	$(1110000000, 1111111000)$
$l(6,3)$	$(1111100000, 1111111100)$
$m(10,1)$	$(1000000000, 1111111111)$
$n(9,3)$	$(1100000000, 1111111100)$

Εικόνα 9

Και στις δυο θέσεις αυτές το bit είναι 1. Εάν το σημείο c κυριαρχείται από κάποιο άλλο σημείο, για παράδειγμα το g, θα πρέπει στις αντίστοιχες θέσεις να υπάρχουν 1. Ελέγχοντας όλα τα σημεία μ' αυτόν τον τρόπο, ο αλγόριθμος bitmap βρίσκει όλη την κορυφογραμμή.

2.5 Index

Αρχικά, τα d-διαστάσεων σημεία εκχωρούνται σε d λίστες. Ένα σημείο p κατηγοριοποιείται στην i λίστα αν και μόνο αν η συντεταγμένη του στο i είναι η μικρότερη από όλες τις συντεταγμένες του. Στην συνέχεια, οι λίστες ταξινομούνται με βάση την μικρότερη συντεταγμένη (minC) τους σε αύξουσα σειρά και τοποθετούνται σε ένα B-δένδρο. Για παράδειγμα, οι λίστες που αντιπροσωπεύουν το διάγραμμα που χρησιμοποιήσαμε φαίνονται στην παρακάτω εικόνα. Σε κάθε λίστα, τα σημεία με τα ίδια minC τοποθετούνται στην γραμμή.

List 1		List 2	
$a(1,10)$	$minC$=1	$m(10,1)$	$minC$=1
$g(3,5)$,$b(3,8)$	$minC$=3	$i(5,3)$,$l(6,3)$,$n(9,3)$	$minC$=3
$c(4,6)$	$minC$=4	$k(8,4)$	$minC$=4
$h(5,5)$	$minC$=5	$j(6,5)$	$minC$=5
$d(6,8)$	$minC$=6	$f(7,6)$	$minC$=6
$e(9,9)$	$minC$=9		

Εικόνα 10

Ο αλγόριθμος ξεκινά από την γραμμή με την μικρότερη minC σε όλες τις λίστες. Σ' αυτό το παράδειγμα είναι το {α} από την λίστα 1 ή το {m} από την λίστα 2. Μετά την επεξεργασία της γραμμής, τα σημεία της κορυφογραμμής που περιέχονται σ' αυτήν επιστρέφονται στον χρήστη και εισάγονται στην λίστα της κορυφογραμμής. Στην συνέχεια ο αλγόριθμος επιλέγει από τις μη-επεξεργασμένες γραμμές την μικρότερη minC. Η επεξεργασία μιας γραμμής περιλαμβάνει δυο βήματα:

1. εντοπισμό του σημείο κορυφογραμμής μέσα στην γραμμή
2. και συγκρίνουμε τα σημεία αυτά με τα σημεία κορυφογραμμής που έχουμε ήδη εντοπίσει.

Συνεχίζοντας το παράδειγμα, από την στιγμή που το {α} περιέχει μόνο ένα σημείο, και το σύνολο των σημείων κορυφογραμμής είναι κενό, το α επιστρέφεται στον χρήστη και εισάγεται στο σύνολο. Η δεύτερη γραμμή είναι η {m}, με minC=1. Το m είναι σημείο της κορυφογραμμής. Η επόμενη γραμμή που εξετάζουμε είναι η {g,b}. Από την στιγμή που το b κυριαρχείται από το g, απορρίπτεται. Στην συνέχεια το g συγκρίνεται με το σύνολο των σημείων κορυφογραμμής {α}, και δίνεται ως έξοδο. Ο αλγόριθμος τερματίζει όταν το τρέχων minC είναι μεγαλύτερο από την μέγιστη συντεταγμένη του προηγούμενου σημείο

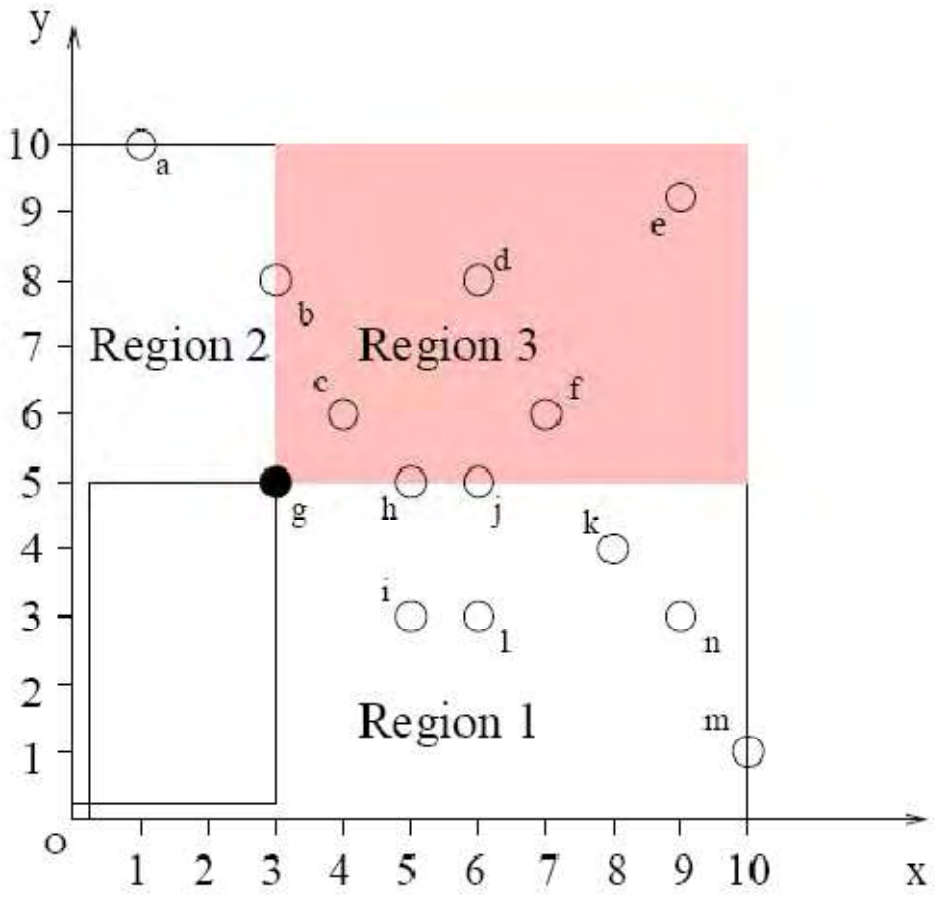
κορυφογραμμής που εντοπίσαμε. Στο παράδειγμα αυτό, η συνθήκη τερματισμού είναι minC<5.

Οι αλγόριθμοι Bitmap και Index είναι προοδευτική αλγόριθμοι. Ένας αλγόριθμος κορυφογραμμής θεωρείται προοδευτικός εάν το πρώτο αποτέλεσμα επιστρέφεται στον χρήστη γρήγορα, και όλο και περισσότερα αποτελέσματα επιστρέφονται όσο μεγαλώνει και ο χρόνος εκτέλεσης του. Σε καταστάσεις πραγματικού χρόνου, οι προοδευτικοί αλγόριθμοι επιτρέπουν στους χρήστες να σταματούν την διαδικασία όποτε εντοπίσουν τα δεδομένα που τους ενδιαφέρουν.

2.6 Nearest neighbours

Ο Kossmann πρότεινε έναν διαίρει-βασίλευε αλγόριθμο NN που βασίζεται σε πολλαπλές κλήσεις του αλγόριθμου αναζήτησης πλησιέστερου γείτονα. Από την στιγμή που η αναζήτηση του πλησιέστερου γείτονα μπορεί να εκτελεστεί γρήγορα σε ένα R-δένδρο, ο αλγόριθμος αυτός δίνει την δυνατότητα να επιταχυνθεί εξίσου και η διαδικασία αναζήτησης των σημείων κορυφογραμμής.

Ο αλγόριθμος δημιουργεί επαναληπτικά ένα σημείο πλησιέστερου γείτονα στην αρχή μιας δεδομένης περιοχής, και είναι εφαρμόσιμος σε κάθε μονοτονική συνάρτηση υπολογισμού απόστασης. Η βασική ιδέα είναι να ενισχύσουμε τα σημεία πλησιέστερου γείτονα που δημιουργούνται επαναληπτικά για να αποτελέσουν μέρος της κορυφογραμμής.

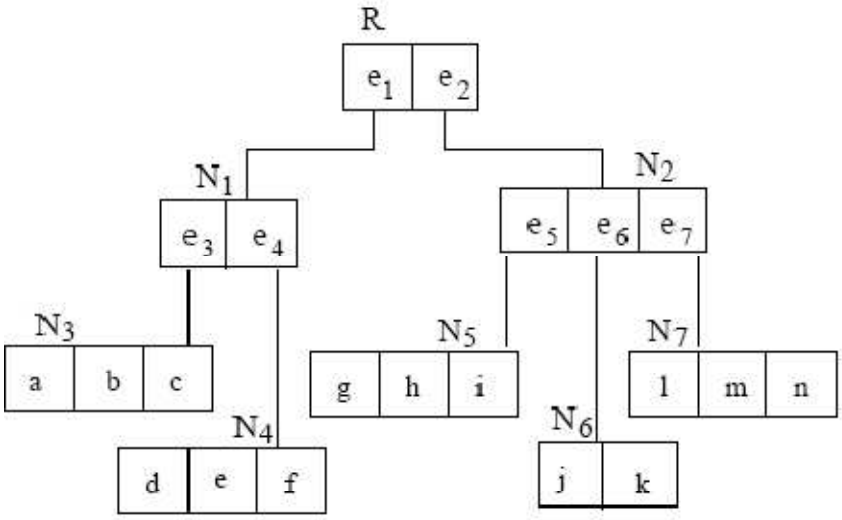


Εικόνα 11

Όπως φαίνεται στην εικόνα 11, το σημείο g εντοπίστηκε ως ο πλησιέστερος γείτονας, και θεωρείται σημείο της οριζοντιογραμμής. Στην συνέχεια, οι περιοχές 1 και 2 προστίθενται σε μία λίστα στον αλγόριθμο για περαιτέρω τμηματοποίηση του χώρου τους. Από την στιγμή

που όλα τα σημεία της περιοχής 3 κυριαρχούνται από το g, η περιοχή μπορεί να απορριφθεί από τον αλγόριθμο. Ο αλγόριθμος τερματίζεται όταν η λίστα είναι κενή.

Παρακάτω δίνουμε το R-δένδρο του διαγράμματος και τις λίστες του αλγόριθμου του πλησιέστερου γείτονα αυτού του R-δένδρου.



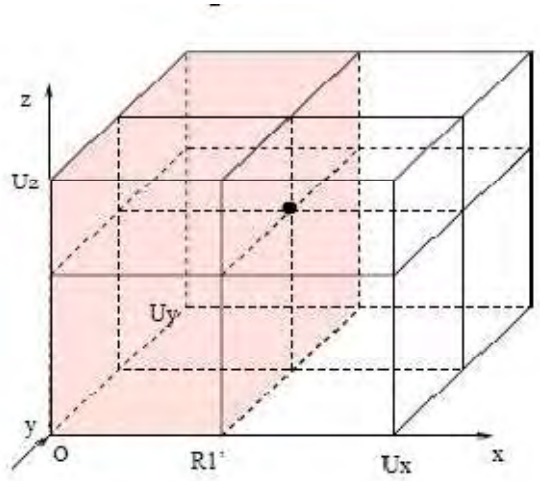
Εικόνα 12

Action	To-do List Contents	Skyline Points
initial state	$(10, 10)$	$\emptyset$
NN query in $(10, 10)$	$(10, 4)$,$(2, 10)$	$\{g\}$
NN query in $(10, 4)$	$(10, 2)$,$(4, 4)$,$(2, 10)$	$\{g, i\}$
NN query in $(10, 2)$	$(9, 2)$,$(4, 4)$,$(2, 10)$	$\{g, i, n\}$
NN query in $(9, 2)$	$(4, 4)$,$(2, 10)$	$\{g, i, n\}$
NN query in $(4, 4)$	$(2, 10)$	$\{g, i, n\}$
NN query in $(2, 10)$	$(2, 9)$	$\{g, i, n, a\}$
NN query in $(2, 9)$		$\{g, i, n, a\}$

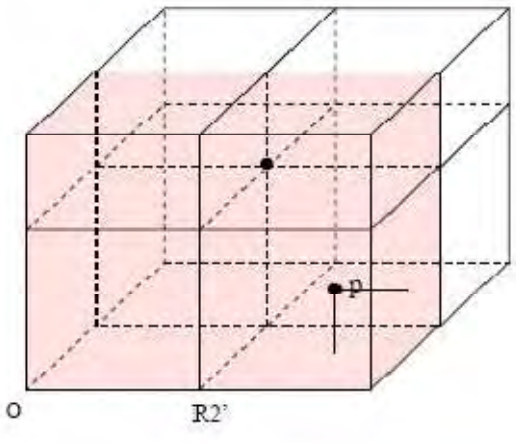
Εικόνα 13

Σ’ αυτό το παράδειγμα, α, g, i και m σχηματίζουν την κορυφογραμμή. Για να ελαττώσουμε τον αποθηκευτικό χώρο στην λίστα, κάθε περιοχή μπορεί εύκολα να καταγραφεί από την πάνω-δεξιά γωνία.

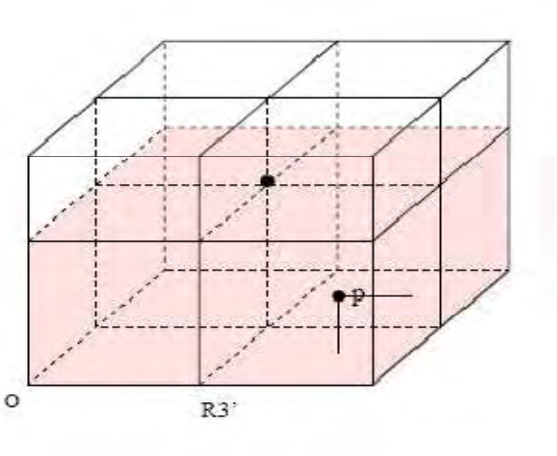
Τα σημεία κορυφογραμμής εντοπίζονται προοδευτικά. Όταν η διάσταση είναι μεγαλύτερη από 2, το τμήμα στο βήμα διαίρεσης είναι διαφορετικό από το συνηθισμένο. Θεωρούμε μια 3-διάστατη περιοχή $[0, U_x)\big[0, U_y\big)[0, U_z)$ που διαιρείται από τα σημεία κορυφογραμμής (α, b, c) όπως φαίνεται στην παρακάτω εικόνα. Οι τρεις υποπεριοχές πρέπει να είναι: $[0, \alpha)\big[0, U_y\big)[0, U_z)$, $[0, U_x)[0, b)[0, U_z)$*και* $[0, U_x)\big[0, U_y\big)[0, c)$.



Εικόνα 14



Εικόνα 15



Εικόνα 16

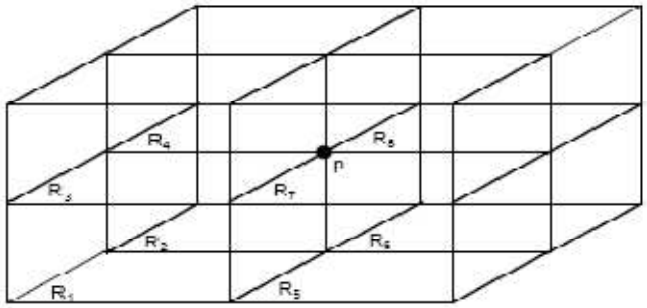
Ένα σημείο p μπορεί να χρησιμοποιηθεί περισσότερο από μία φορά. Παρακάτω παρουσιάζονται τέσσερις μέθοδοι που αντιμετωπίζουν την επανάληψη:

- **laisser - faire** Ο αλγόριθμος πλησιέστερου γείτονα διατηρεί έναν καθολικό πίνακα κατακερματισμού που αποθηκεύει τα σημεία της κορυφογραμμής που έχουν υπολογιστεί μέχρι στιγμής. Όταν ένα σημείο κορυφογραμμής έχει βρεθεί, ελέγχεται αρχικά με τον πίνακα κατακερματισμού. Εάν δεν υπάρχει στον πίνακα κατακερματισμού, εισάγεται στον πίνακα. Διαφορετικά αγνοείται ως ήδη ανακαλυφθέν. Αυτή η μέθοδος είναι απλή και αποτελεσματική από πλευράς χρόνου. Ωστόσο, δεν αποκλείει την επανάληψη κάποιας αναζήτησης.

- **Propagate** Από την στιγμή που ένα σημείο p της οριζοντιογραμής έχει βρεθεί, ο αλγόριθμος του πλησιέστερου γείτονα εξετάζει όλες τις περιοχές που πρέπει να ελεγχθούν και τμηματοποιεί ξανά τις περιοχές που περιέχουν το p με το p με τον ίδιο τρόπο. Αν και αυτή η μέθοδος δεν επιστρέφει το ίδιο σημείο κορυφογραμμής περισσότερες από δυο φορές, απαιτεί πιο ακριβό υπολογισμό καθώς κάθε σημείο κορυφογραμμής που βρίσκεται πρέπει να τμηματοποιεί ξανά όλες τις περιοχές που περιέχουν το p.

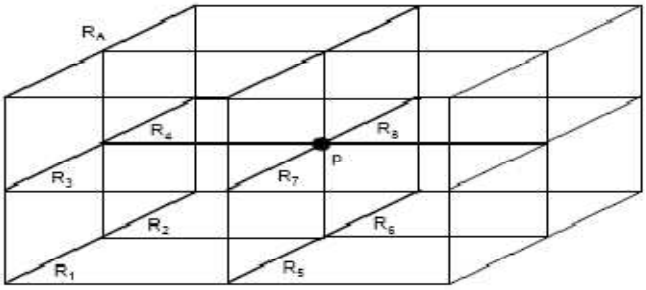
- **Merge** Η βασική ιδέα αυτής της μεθόδου είναι η συγχώνευση κάποιων περιοχών που πρέπει να ελεγχθούν έτσι ώστε να ο συνολικός αριθμός των αναζητήσεων του πλησιέστερου γείτονα να μειωθεί. Ομοίως με την αμέσως προηγούμενη μέθοδο, απαιτεί υψηλό υπολογιστικό κόστος καθώς είναι ακριβότερο να βρεις «καλές» περιοχές για συγχώνευση.

- **Fine-grained Partitioning** Ο βασικός αλγόριθμος του πλησιέστερου γείτονα χωρίζει τον χώρο σε d περιοχές όταν εντοπιστεί ένα σημείο κορυφογραμμής. Με σκοπό να εξαλείψει τις επικαλυπτόμενες περιοχές, η μέθοδος αυτή δημιουργεί 2^d μη-επικαλυπτόμενες περιοχές κάθε φορά. Ένα παράδειγμα φαίνεται στις παρακάτω εικόνες όπου ο χώρος χωρίζεται σε 8 περιοχές. Είναι φανερό ότι οι περιοχές R2-R7 ανήκουν στο τμήμα 3. Αυτό έχει ως αποτέλεσμα, ο αλγόριθμος του πλησιέστερου γείτονα εξετάζει επανειλημμένα αυτές τις περιοχές. Αν και αυτή η μέθοδος αποφεύγει όλες τις επαναλήψεις των αναζητήσεων και των αποτελεσμάτων, μπορεί να οδηγήσει σε λάθος αποτέλεσμα, γιατί ένα σημείο κορυφογραμμής που εντοπίζεται σε μια περιοχή (π.χ. στην περιοχή R7) μπορεί να κυριαρχείται από κάποια σημεία σε μία άλλη περιοχή (π.χ. στην περιοχή R3). Επομένως, όταν ένα σημείο κορυφογραμμής εντοπίζεται, ελέγχεται με όλα τα σημεία κορυφογραμμής που έχουν εντοπιστεί μέχρι στιγμής για να αντιμετωπιστεί πιθανό λάθος απάντηση.



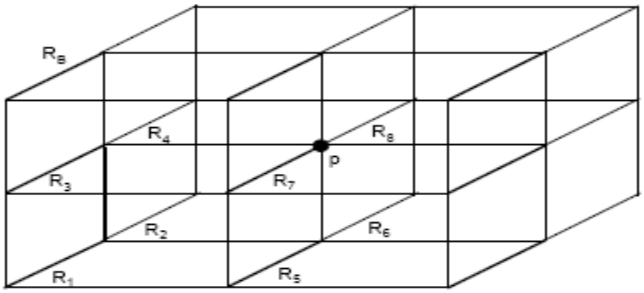
(a) The first skyline point.

Εικόνα 17



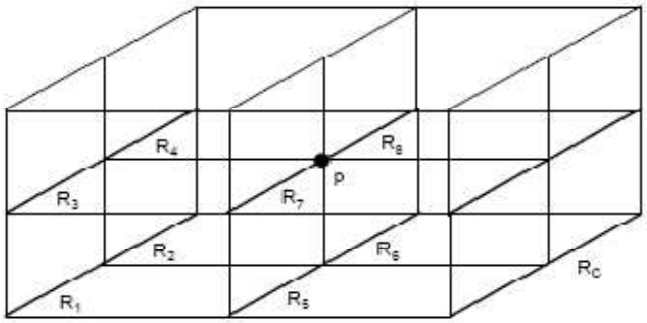
(b) The first region in part 3.

Εικόνα 18



(c) The second region in part 3.

Εικόνα 19



(d) The third region in part 3.

Εικόνα 20

2.7 Branch and Bound Skyline

Ο BBS αλγόριθμος εφαρμόζει και αυτός τεχνικές που βασίζονται στην αναζήτηση του πλησιέστερου γείτονα με βάση R-δένδρα. Υποθέτουμε ότι τα δεδομένα είναι οργανωμένα σε R-δένδρο, ο αλγόριθμος BBS είναι I/O βέλτιστος-δηλαδή ο αριθμός των κόμβων που επισκεπτόμαστε ελαχιστοποιείται. Για να βρει το σημείο κορυφογραμμής, ο αλγόριθμος αυτός διατρέχει το R-δένδρο με τον *καλύτερο-πρώτο (best-first)* τρόπο: αξιολογεί και χρησιμοποιεί τον κόμβο που βρίσκεται πλησιέστερα στην περιοχή όπου βρίσκονται όλοι οι κόμβοι που δεν έχουμε επισκεφτεί. Για το πετύχει αυτό, ο αλγόριθμος χρησιμοποιεί έναν σωρό στον οποίο το κλειδί για κάθε είσοδο είναι η ελάχιστη απόστασή του από την περιοχή. Εδώ, η ελάχιστη απόσταση ενός κόμβου του R-δένδρου από την περιοχή είναι η άθροιση των συντεταγμένων της κάτω-αριστερά γωνίας. Αρχικά, όλα τα παιδιά είσοδοι του κόμβου-ρίζας του R-δένδρου εισάγονται στον σωρό. Σε κάθε επανάληψη, η είσοδος e που βρίσκεται στην κορυφή του σωρού απομακρύνεται και ελέγχεται με τα σημεία κορυφογραμμής που έχουν υπολογιστεί μέχρι στιγμής. Εάν η e κυριαρχείται από κάποιο σημείο κορυφογραμμής, απορρίπτεται. Διαφορετικά, το e επιστρέφεται ως έξοδος σαν σημείο κορυφογραμμής ή συνεχίζεται η ανάλυση του. Εάν το e είναι κόμβος ενός R-δένδρου, τότε εισάγονται στον σωρό όλα τα παιδιά του που δεν κυριαρχούνται από κάποιο σημείο κορυφογραμμής. Εάν το e είναι σημείο δεδομένων, τότε είναι σημείο κορυφογραμμής. Ο αλγόριθμος τερματίζεται όταν ο σωρός είναι άδειος. Για να επιταχύνουμε την εξέταση εάν το e κυριαρχείται από κάποιο σημείο κορυφογραμμής, η τρέχουσα κορυφογραμμή διατηρείται στην μνήμη σε ένα R-δένδρο. Η παρακάτω εικόνα δείχνει τα περιεχόμενα του σωρού για το παράδειγμα μας, όταν εφαρμόζουμε τον αλγόριθμο BBS. Το δεύτερο πεδίο σε κάθε στοιχείο είναι η τιμή της ελάχιστης απόστασης όπου για τον υπολογισμό της χρησιμοποιούμε την συνάρτηση υπολογισμού απόστασης x+y.

Algorithm BBS
Input:
R-tree R.
Output:
Skyline points in R.
Description:

```
1: S := ∅; Insert the root entry of R into the heap H.
2: while H ≠ ∅ do
3:    delete the first element e of H;
4:    if e is dominated by a point in S then
5:       discard e;
6:    else
7:       if e is a data point then
8:          S := S ∪ {e}; //output e as a skyline point
9:       else
10:         add to H the children entries of e, which are not dominated by a point
            in S
11:      end if
12:   end if
13: end while
```

Action	Heap Contents	Skyline Points	Heap Size
initial state	$(e_0, 2)$	$\emptyset$	1
expand e_0	$(e_2, 4)$,$(e_1, 8)$,$(e_3, 13)$	$\emptyset$	3
expand e_2	$(e_5, 6)$,$(e_7, 7)$,$(e_1, 7)$,$(e_6, 10)$	$\emptyset$	4
expand e_5	$(e_7, 7)$,$(e_1, 7)$,$(g, 8)$, $(i, 8)$,$(h, 10)$,$(e_6, 10)$	$\emptyset$	6
expand e_7	$(e_1, 7)$,$(g, 8)$,$(i, 8)$,$(l, 9)$, $(h, 10)$,$(e_6, 10)$,$(m, 11)$,$(n, 12)$	$\emptyset$	8
expand e_1	$(e_3, 7)$,$(g, 8)$,$(i, 8)$, $(l, 9)$,$(h, 10)$,$(e_6, 10)$, $(m, 11)$,$(e_4, 12)$,$(n, 12)$	$\emptyset$	9
expand e_3	$(g, 8)$,$(i, 8)$,$(l, 9)$, $(c, 10)$,$(h, 10)$,$(e_6, 10)$,$(a, 11)$, $(b, 11)$,$(m, 11)$,$(e_4, 12)$,$(n, 12)$	$\{g, i,$ $a, m\}$	11

Εικόνα 21

Έχει αποδειχθεί ότι ο αλγόριθμος BBS είναι βέλτιστος από πλευράς I/O κόστους. Αυτό ισχύει, γιατί μόνο οι απαραίτητοι κόμβοι ενός R-δένδρου επισκέπτονται και επισκέπτονται μόνο για μια φορά.

2.8 Sort First Skyline Algorithm

Ο Sort First Skyline Algorithm είναι μια παραλλαγή του BNL. Με σκοπό να βελτιώσει τον BNL, ο SFS υπολογίζει την εντροπία E(p) για κάθε σημείο δεδομένων p:

$$E(p) = \sum_{i=1}^{d} ln(p' + 1)$$

όπου p' είναι η κανονικοποιημένη τιμή του p. Είναι φανερό ότι αν έχω δυο σημεία p και q, το p δεν μπορεί να κυριαρχεί έναντι του q εάν E(p) είναι μεγαλύτερη ή ίση με E(q). Βασιζόμενοι στην παραπάνω διαπίστωση, SFS αρχικά ταξινομεί όλα τα σημεία δεδομένων σε μη-ελάττωσα σειρά με βάση την τιμή της εντροπίας. Στην συνέχεια, ο SFS επεξεργάζεται την ταξινομημένη λίστα όπως και ο BNL.

Συγκρινόμενος με τον BNL, ο SFS έχει τα ακόλουθα πλεονεκτήματα:

1. Ο αριθμός των συγκρίσεων ανάμεσα στα σημεία δεδομένων μειώνεται. Εφόσον πρώτα εκτιμάται το σημείο με την μικρότερη εντροπία, ένα σημείο κορυφογραμμής μπορεί να βρεθεί νωρίτερα. Επομένως, ο αριθμός των συγκρίσεων μεταξύ των σημείων και των σημείων που δεν ανήκουν στην κορυφογραμμή, που είναι άχρηστες, μειώνεται.
2. Ο SFS είναι προοδευτικός αλγόριθμος, ενώ ο BNL δεν είναι. Όταν ένα σημείο p εισάγεται στην λίστα, είναι εγγυημένα σημείο κορυφογραμμής. Αυτό συμβαίνει, γιατί τα μη επεξεργασμένα σημεία δεν κυριαρχούν έναντι του p εφόσον έχουν μεγαλύτερη εντροπία από την E(p).

Βιβλιογραφία

- **Progressive Skyline Computation in Database Systems**
- **An Optimal and Progressive Algorithm for Skyline Queries**
- **The Skyline Operator**

Κεφάλαιο 3

Υπολογισμός Κορυφογραμμής σε Ροές Δεδομένων

Περιεχόμενα

3.1 Εισαγωγή

Σε πολλές εφαρμογές, η ροή των δεδομένων μπορεί να είναι append-only. Αυτό σημαίνει ότι δεν υπάρχει διαγραφή των στοιχείων δεδομένων που περιλαμβάνονται. Στην συνέχεια, θα αναλύσουμε το πρόβλημα υπολογισμού της κορυφογραμμής σε append-only ροές δεδομένων. Σε μια ροή δεδομένων, τα στοιχεία τοποθετούνται σύμφωνα με την σειρά εμφάνισης τους και χαρακτηρίζονται από έναν ακέραιο. Να διευκρινίσουμε ότι η θέση k(e) σημαίνει ότι το στοιχείο e έφτασε κ-στο στην ροή των δεδομένων.

Θεωρούμε ότι ο υπολογισμός της κορυφογραμμής απαιτεί κύρια μνήμη. Από την άλλα, μπορεί να μην είναι δυνατή η αποθήκευση μιας ολόκληρής ροής δεδομένων στην κύρια μνήμη από την στιγμή που δεν υπάρχει κάποιο όριο που να καθορίζει το μέγεθος μιας ροής δεδομένων. Η αναφορά στο πρόβλημα του υπολογισμού της κορυφογραμμής θα περιοριστεί για τα N πιο πρόσφατα στοιχεία, που χωράνε στην κύρια μνήμη. Πιο συγκεκριμένα, θα αναφερθούμε στον on-line υπολογισμό της κορυφογραμμής με βάση τα παρακάτω μοντέλα.

1. $n - of - N$ *μοντέλο*. Θα αναλύσουμε το πρόβλημα της αποτελεσματικής οργάνωσης των N πιο πρόσφατων στοιχείων που έχουμε εντοπίσει σε μια ροή δεδομένων, έτσι ώστε ο υπολογισμός της κορυφογραμμής των n πιο πρόσφατων στοιχείων (n≤N) να μπορεί να πραγματοποιηθεί αποτελεσματικά. Το παράθυρο ολίσθησης είναι μια ειδική περίπτωση του μοντέλου αυτού όταν n=N.
2. $(n_1, n_2) - of - N$ *μοντέλο*. Είναι μια γενίκευση του παραπάνω μοντέλου. Σ΄ αυτήν την περίπτωση θέλουμε να υπολογίσουμε την κορυφογραμμή ανάμεσα στα n_2 πιο πρόσφατα στοιχεία και στα n_1 πιο πρόσφατα στοιχεία ($για\ κάθε\ n_1 \leq n_2 \leq N$).

Το $n - of - N$ *μοντέλο* επιστρέφει την κορυφογραμμή βασιζόμενο στις πιο πρόσφατες πληροφορίες, ενώ το $(n_1, n_2) - of - N$ *μοντέλο* παρέχει πρόσφατες «ιστορικές» πληροφορίες. Ο συνδυασμός των δύο μοντέλων μπορεί να επιδείξει μια εναλλαγή από τα n_2 πιο πρόσφατα στοιχεία στα n_1 πιο πρόσφατα στοιχεία.

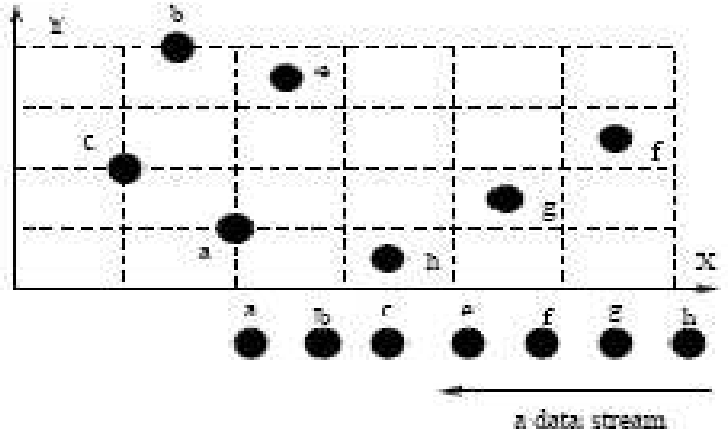
Στον παρακάτω πίνακα επεξηγούνται τα σύμβολα που χρησιμοποιούνται σ' αυτό το κεφάλαιο.

Notation	Definition
N, n	number of the most recent elements
M	number of the elements seen so far
P_N	the most recent N elements
R_N	non-redundant elements in P_N
S_N	skyline points in P_N
$\mathcal{N}$	$\|R_N\|$
e	an element in a data stream
$\kappa(e)$	position (integer) of e in a data stream
s	number of skyline points
d	space dimension

Πίνακας 1.1

3.2 On-line υπολογισμός

Θεωρούμε το παράδειγμα της εικόνας 3.2 όπου τα στοιχεία έχουν αφιχθεί με αλφαβητική σειρά. Υποθέτουμε ότι η ροή των δεδομένων μέχρι στιγμής αποτελείται από 6 στοιχεία α, b, c, e, f, και g. Η κορυφογραμμή S_6 έναντι αυτών των 6 πιο πρόσφατων στοιχείων αποτελείται από τα α και c, ενώ η κορυφογραμμή S_4 έναντι των 4 πιο πρόσφατων στοιχείων (π.χ. c, e, f και g) αποτελείται από τα c και g. Στην συνέχεια, όταν φτάνει το καινούργιο στοιχείο h, οι οριζοντιογραμμές S_6 και S_4 ενημερώνονται και αποτελούνται από τα εξής στοιχεία αντίστοιχα: {c, h} και {e, h}.



Εικόνα 3.2

Το παράδειγμα αυτό δείχνει ότι η κορυφογραμμή S_n των n πιο πρόσφατων στοιχείων του P_nδεν είναι υποσύνολο της κορυφογραμμής S_N των N πιο πρόσφατων στοιχείων όταν n<N, ωστόσο το P_n είναι υποσύνολο του P_N. Επομένως, το πρόβλημα της αποτελεσματικής επεξεργασίας μιας αναζήτησης $n-of-N$ $(\forall\, n \leq N)$ είναι πιο προκλητικό από τον υπολογισμό της κορυφογραμμής των N πιο πρόσφατων στοιχείων με δεδομένο το N. Επιπλέον, το S_n (S_N) μπορεί να αλλάξει όταν γίνει ενημέρωση για τα n (N) πιο πρόσφατα στοιχεία κατά την άφιξη νέων στοιχείων. Αυτό έχει σαν συνέπεια, σε ένα on-line περιβάλλον να απαιτούνται αποδοτικές τεχνικές για τον υπολογισμό των ενημερωμένων οριζοντιογραμμών. Επιπρόσθετα, οι τεχνικές αυτές θα πρέπει να αντιμετωπίσουν και την περίπτωση της διαγραφής των στοιχείων δεδομένων. Και αυτό, γιατί η διαγραφή ενός παλιού στοιχείου λόγω της λήξης του από τα πιο πρόσφατα N στοιχεία πραγματοποιείται κάθε φορά που έχουμε την άφιξη ενός νέου.

Παρακάτω, έχουμε την περιγραφή των on-line τεχνικών για την πραγματοποίηση μιας αναζήτησης $n-of-N$.

- Καθορίζουμε την σχέση κυριαρχίας ανάμεσα στα στοιχεία που αποτελούν τα N πιο πρόσφατα στοιχεία και τα μοντελοποιούμε σε ένα γράφημα που ονομάζεται δάσος. Η παραπάνω διαδικασία ελαχιστοποιεί τον αριθμό των στοιχείων που πρέπει να κρατιούνται από το P_N.
- Κωδικοποιούμε αυτό το γράφημα με ένα σύνολο από διαστήματα σε μια διάσταση, και προσδιορίζω μια αναζήτηση $n-of-N$ σε μια stabbing αναζήτηση στα διαστήματα που έχω καθορίσει.
- Το γράφημα και το κωδικοποιημένο σχήμα του διατηρούνται δυναμικά έτσι ώστε να ενημερώνονται σε μια αλλαγή των N πιο πρόσφατων στοιχείων σε μια ροή δεδομένων.
- Ένας πρωτότυπος αλγόριθμος αναπτύχθηκε για την επεξεργασία μιας συνεχής μιας αναζήτησης $n-of-N$.

Οι παραπάνω τεχνικές επεκτείνονται επίσης ώστε να καλύψουν και τις $(n_1, n_2)-of-N$ αναζητήσεις.

3.3 Stabbing Queries

Με δεδομένο ένα σύνολο από διαστήματα και ένα stabbing σημείο p σε ένα μονοδιάστατο χώρο, η αναζήτηση stabbing έχει στόχο να βρει τα διαστήματα που περιέχουν το p. Μια stabbing αναζήτηση μπορεί να ολοκληρωθεί σε $O(logm + l)$ όπου l είναι ο αριθμός των διαστημάτων που επιστρέφονται ως αποτέλεσμα. Αποθηκεύοντας ένα διάστημα μόνο στον κόμβο που είναι ο χαμηλότερος κοινός απόγονος των δύο ακραίων σημείων του διαστήματος, η πολυπλοκότητα του διαστήματος στο δένδρο του διαστήματος είναι $O(m)$. Έχει επίσης αποδειχτεί ότι η χρονική πολυπλοκότητα μιας ενημέρωσης σε ένα δένδρο του διαστήματος είναι $O(logm)$ για κάθε διαγραφή ή εισαγωγή. Τα διαστήματα μπορεί να είναι κλειστά, μισό κλειστά και ανοιχτά.

3.4 Εκτέλεση μιας n-of-N αναζήτησης

Στην παρακάτω ενότητα παρουσιάζουμε τεχνικές για αποδοτική εκτέλεση μιας n-of-N αναζήτησης. Αρχικά, ελαχιστοποιούμε τον αριθμό των στοιχείων που πρέπει να κρατάμε για να εκτελεστούν όλες οι n-of-N αναζητήσεις. Στην συνέχεια δίνουμε ένα αποδοτικό κωδικοποιημένο σχήμα για τα αποθηκευμένα στοιχεία για να υποστηρίξουν την εκτέλεση μια n-of-N αναζήτησης. Τέλος, αναπτύσσεται και τεχνικές για την ενημέρωση των δομών δεδομένων που περιλαμβάνονται.

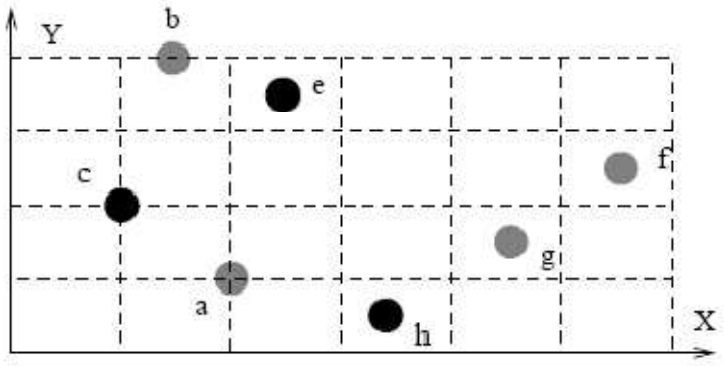
3.4.1 Ελαχιστοποίηση του αριθμού των στοιχείων

Υποθέτουμε ότι στα N πιο πρόσφατα στοιχεία P_N υπάρχουν δυο στοιχεία e και e' τέτοια ώστε το e'να κυριαρχεί το e και το e' φτάνει αργότερα από το e. Όπως καταλαβαίνουμε το στοιχείο e δεν μπορεί να είναι σημείο κορυφογραμμής. Ένα στοιχείο e είναι περιττό ανάμεσα στα υπόλοιπα πιο πρόσφατα στοιχεία είτε όταν έχει λήξει είτε όταν κυριαρχείται από κάποιο νεότερο στοιχείο (μετέπειτα άφιξη) e'.

Θεώρημα 3.1: *υποθέτουμε ότι το P_N είναι το σύνολο με τα N πιο πρόσφατα στοιχεία δεδομένων. Τότε:*

1. *Ένα σημείο κορυφογραμμής στα n πιο πρόσφατα στοιχεία* $(\forall\, n \leq N)$ *δεν πρέπει να είναι περιττό στοιχείο του υπάρχοντος συνόλου P_N.*
2. *Κάθε μη περιττό στοιχείο του P_N είναι σημείο κορυφογραμμής στα n πιο πρόσφατα στοιχεία.*
3. *Κάποιο περιττό στοιχείο e δεν θα συμπεριληφθεί στην κορυφογραμμή των n πιο πρόσφατων στοιχείων μέχρι να γίνει απαραίτητο.*

Έστω η ροή δεδομένων της εικόνας 3.2 με 7 στοιχεία και N=6. Τα μη-περιττά στοιχεία σημειώνονται στην εικόνα 3.3 με μαύρο χρώμα. Το στοιχείο α δεν συμπεριλαμβάνεται εφόσον έχει λήξει.



Εικόνα 3.3

Έστω το R_N είναι το σύνολο των μη-περιττών στοιχείων του P_N. Σύμφωνα με το θεώρημα 3.1, χρειαζόμαστε μόνο το σύνολο R_N έναντι του P_N για να εκτελέσουμε όλες τις n-of-N αναζητήσεις. Έτσι θα απομακρυνθούν όσα στοιχεία δεν είναι απαραίτητα. Επιπρόσθετα, σύμφωνα με το θεώρημα, το R_N μας δίνει τον ελάχιστο αριθμό στοιχείων που πρέπει να κρατήσουμε ώστε να επιτύχουμε ακριβείς υπολογισμούς των n-of-N αναζητήσεων.

Θεώρημα 3.2: *σε μια ροή δεδομένων σε ένα d-διάστατο χώρο, υποθέτουμε ότι η κατανομή των δεδομένων σε κάθε διάσταση, περιλαμβάνοντας και την σειρά άφιξης, είναι ανεξάρτητη. Τότε, η μέση τιμή του □ είναι* $O(log^D N)$ *όπου* □=$|R_N|$.

Απόδειξη: υποθέτουμε ότι έχουμε M στοιχεία. Αναλύω κάθε στοιχείο $e = (x_1, x_{2,\ldots,} x_d)$ σε ένα σημείο $p_e = (x_1, x_{2,\ldots,} x_d, M - k(e))$ στην (d+1)-διάσταση.

Σύμφωνα με τον ορισμό του R_N, το R_N αντιστοιχεί στο σύνολο των σημείων οριζοντιογραμμής, από την παραπάνω απεικόνιση: του $\{p_e : e \in P_N\}$ στην (d+1)-διάσταση. □

Σημείωση: όταν το d είναι μικρό, □ είναι αρκετά μικρότερο από το N εάν η ροή των δεδομένων ακολουθεί την κατανομή του θεωρήματος 3.2.

3.4.2 Κωδικοποίηση R_N για n-of-N αναζητήσεις

Σ' αυτήν την υποενότητα, παρουσιάζουμε ένα αποδοτικό σχήμα κωδικοποίησης του R_N τέτοιο ώστε κάθε n-of-N αναζήτηση να εκτελείται αποδοτικά.

Ένα στοιχείο e στο R_N μπορεί να κυριαρχείται από πολλά άλλα στοιχεία του R_N που εμφανίζονται αργότερα από το e. Είναι φανερό ότι ο αριθμός των σχέσεων κυριαρχίας ανάμεσα στα στοιχεία του R_N μπορεί να είναι $O(□^2)$. Κάποιος θα μπορούσε να σκεφτεί σαν παράδειγμα έναν πλήρως διασυνδεδεμένο γράφο. Επομένως, η αποθήκευση όλων των σχέσεων κυριαρχίας δεν είναι μόνο ακριβή σε αποθηκευτικό χώρο αλλά και ακριβή στην διατήρηση της. Το παρακάτω παράδειγμα αποδεικνύει ότι δεν χρειάζεται να κρατάμε όλες τις σχέσεις κυριαρχίας.

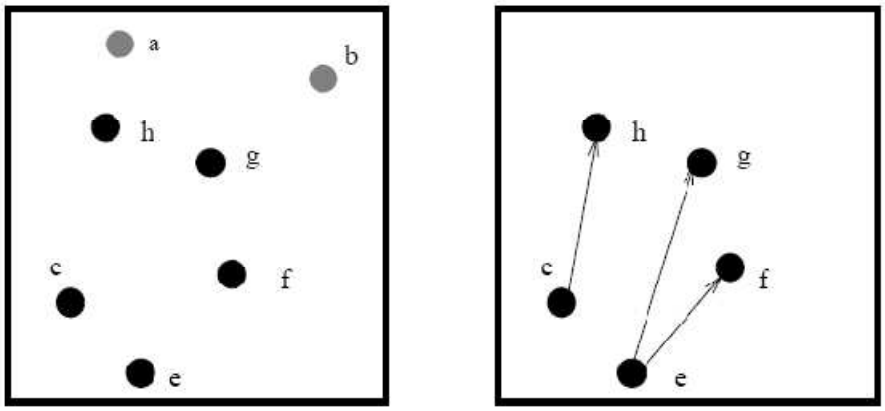
Παράδειγμα 3.1: *υποθέτουμε ότι έχουμε 3 στοιχεία δεδομένων a, b και c που φτάνουν με αλφαβητική σειρά, και το c κυριαρχείται και από το a αλλά και από το b.*

Είναι φανερό σ' αυτό το παράδειγμα αν η σχέση κυριαρχίας b→c αποδεσμευτεί λόγω της λήξης του b, τότε η σχέση κυριαρχίας a→c έχει ήδη αποδεσμευτεί, εφόσον το a έχει λήξει νωρίτερα από το b. Επομένως, χρειάζεται μόνο να κρατήσουμε την σχέση b→c.

Στο R_N, η σχέση κυριαρχίας e'→e είναι σημαντική αν και μόνο αν το e' είναι το νεότερο στοιχείο (αλλά μεγαλύτερο από το e) στο R_N, το οποίο κυριαρχεί έναντι του e. Έτσι, το κ(e') μεγιστοποιείται ανάμεσα στα στοιχεία που κυριαρχούν έναντι του e. Ισχύει ότι κ(e') < κ(e) από την στιγμή που το R_N δεν περιέχει κανένα περιττό στοιχείο στο P_N. Στο παράδειγμα 3.1, η σχέση b→c είναι σημαντική. Ο γράφος για το R_N είναι ένας γράφος κυριαρχίας εάν το σύνολο των ακμών από τις κρίσιμες κυριαρχικές σχέσεις; Σημειώνεται ως G_{R_N}. Χρησιμοποιούμε $e' \xrightarrow{c} e$ για να υποδηλώσουμε ότι « το e' κυριαρχεί σημαντικά του e».

Πρέπει να σημειώσουμε ότι ο γράφος κυριαρχίας είναι *δάσος* καθώς κάθε στοιχείο έχει τουλάχιστον ένα εισερχόμενο τόξο.

Παράδειγμα 3.2: *υποθέτουμε ότι μια ροή δεδομένων αποτελείται από τα στοιχεία a, b, c, e, f, g, και h που εμφανίζονται με αλφαβητική σειρά όπως φαίνεται στην εικόνα 3.5(a). Η εικόνα 3.5(b) απεικονίζει τον αντίστοιχο γράφο κυριαρχίας μετά την απομάκρυνση των περιττών στοιχείων.*



(a) Stream S (b) Dominance Graph

Εικόνα 3.5

Το παρακάτω θεώρημα είναι θεμελιώδες για την κωδικοποίηση του γράφου κυριαρχίας για την εκτέλεση μιας αναζήτησης n-of-N. Μπορεί εύκολα να αποδειχθεί σύμφωνα με τον ορισμό του γράφου κυριαρχίας.

Θεώρημα 3.3: για ένα δεδομένο n (n≤N), ένα στοιχείο e ∈ P_n είναι σημείο κορυφογραμμής της n-of-N αναζήτησης αν και μόνο αν είτε

- e είναι ρίζα του υπάρχοντος γράφου κυριαρχίας G_{R_N}, είε
- υπάρχει μια ακμή $e' \xrightarrow{c} e$ στο G_{R_N} τέτοια ώστε το e' φτάνει νωρίτερα από το n-οστό πιο πρόσφατο στοιχείο των N πιο πρόσφατων στοιχείων; Οπότε, κ(e') < M – n + 1 ≤ κ(e), όπου M είναι ο αριθμός των στοιχείων που έχουμε δει μέχρις στιγμής.

Το σχήμα κωδικοποίησης στο G_{R_N} είναι αρκετά σαφές: το G_{R_N} αποτελείται από τις ακμές του. Οπότε:

1. κάθε ακμή e'→e στο G_{R_N} αντιπροσωπεύεται από το διάστημα (κ(e'),κ(e)],
2. κάθε ρίζα του G_{R_N} αντιπροσωπεύεται από το διάστημα (0, κ(e)].

Σύμφωνα με το θεώρημα 3.3 ένα στοιχείο του G_{R_N} είναι στην απάντηση της n-of-N αναζήτησης (n ≤ N) εάν και μόνο αν κ(e) είναι το δεξί άκρο του διαστήματος (α, κ(e)] που περιέχει M-n+1. Βασισμένοι στο παραπάνω σχήμα κωδικοποίησης, το πρόβλημα του

υπολογισμού μιας n-of-N αναζήτησης μετατρέπεται στο πρόβλημα της stabbing αναζήτησης με stabbing σημείο το M-n+1 που αναπτύχθηκε στην υποενότητα 3.1.3.

Έστω I_{R_N} υποδηλώνει το ενδιάμεσο δένδρο των διαστημάτων που καθορίζονται από το σχήμα κωδικοποίησης του G_{R_N}. Μπορούμε να πραγματοποιήσουμε μια n-of-N αναζήτηση ως εξής:

Χωρίζω τα διαστήματα στο I_{R_N}κατά M-n+1,και επιστρέφω τα στοιχεία δεδομένων e τέτοια ώστε κ(e) είναι το δεξί άκρο του διαστήματος που χωρίσαμε.

Παράδειγμα 3.3: *σχετικά με το παράδειγμα της εικόνας 3.5, a, b, c, e, f, g, και h φτάνουν την στιγμή 1, 2, 3, 4, 5, 6, και 7, αντίστοιχα. Ο γράφος κυριαρχίας μπορεί να κωδικοποιηθεί από τα ακόλουθα διαστήματα: (0,3], (0,4], (3,7], (4,5] και (4,6]. Όταν το n = 6, M-n+1 = 2 καθώς M=7. Είναι φανερό ότι, τα διαστήματα (0,3] και (0,4] είναι τα αποτελέσματα της stabbing αναζήτησης. Κατά συνέπεια, τα c και e είναι σημεία κορυφογραμμής για τα 6 πιο πρόσφατα στοιχεία ανάμεσα στα 7 στοιχεία που έχουν εμφανιστεί.*

Εφόσον το G_{R_N} είναι δάσος, προκύπτει άμεσα ο αριθμός των διαστημάτων στο I_{R_N} είναι $O(|R_N|)$. Συνεπώς, ο αλγόριθμος της αναζήτησης τρέχει σε $O(log\square + s)$ όπου $\square = |R_N|$. Οπότε, ο χρόνος της εκτέλεσης της αναζήτησης είναι $O(logmin\{\square, log^d N\} + s)$, όπου η κατανομή των δεδομένων σε κάθε διάσταση είναι ανεξάρτητη, σύμφωνα με το θεώρημα 3.2.

3.4.3 Διατήρηση του R_N και το σχήμα κωδικοποίησης

Μετά την άφιξη ενός νέου στοιχείου e_{new} στην ροή δεδομένων, το e_{new} προστίθεται στο R_N και το παλαιότερο στοιχείο e_{old} στο R_N πρέπει να απομακρυνθεί αν έχει λήξει. Επομένως, το R_N μπορεί να χρειαστεί να ενημερωθεί, όπως και το εσωτερικό δένδρο I_{R_N}. Ο αλγόριθμος παρουσιάζεται παρακάτω όπως και η επεξήγηση του.

Algorithm 3.1 Maintaining R_N & its Encoding Scheme

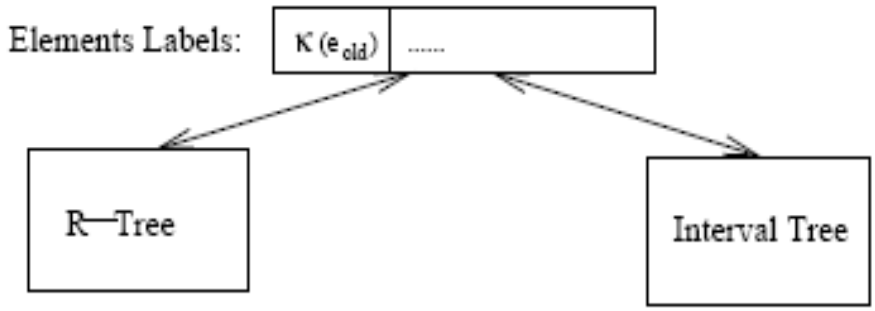
Description:

1: while new element e_{new} do
2: if the oldest e_{old} in R_N is expired then
3: $R_N := R_N - \{e_{old}\}$;
4: remove $(0, \kappa(e_{old})]$ from I_{R_N};
5: for $\forall (\kappa(e_{old}), \kappa(e)] \in I_{R_N}$ do
6: update $(\kappa(e_{old}), \kappa(e)]$ to $(0, \kappa(e)]$;
7: end for
8: end if
9: find $D_{e_{new}} \subseteq R_N$ dominated by e_{new};
10: for $\forall\, e \in D_{e_{new}}$ do
11: remove the intervals in I_{R_N} with $\kappa(e)$ as an end;
12: end for
13: $R_N := R_N - D_{e_{new}} + \{e_{new}\}$;
14: determine the critical relation $e \rightarrow e_{new}$;
15: add $(\kappa(e), \kappa(e_{new})]$ (or $(0, \kappa(e_{new})]$) to I_{R_N};
16: end while

Οι γραμμές 3-7 του παραπάνω αλγόριθμου περιγράφουν τις ενημερώσεις εάν το e_{old} έχει λήξει. Αν και το παλαιότερο στοιχείο P_N πάντα λήγει όταν έχουμε την εμφάνιση ενός νέου στοιχείου, το παλαιότερο στοιχείο e_{old} στο R_N δεν λήγει πάντα. Για παράδειγμα, σύμφωνα με το παράδειγμα της εικόνας 3.5 το παλαιότερο στοιχείο c στο R_N (N = 6) δεν πρέπει να λήξει εάν έχουμε την άφιξη ενός νέου στοιχείου. Να σημειώσουμε ότι το e_{old} είναι πάντα ρίζα του G_{R_N}.

Οι γραμμές 9-15 περιγράφουν τις ενημερώσεις του R_N και των ενδιάμεσων μέσων με την είσοδο του e_{new}. Η ενημέρωση του ενδιάμεσου δένδρου μπορεί να πραγματοποιηθεί σε $O(log□)$ χρόνο ανά ενημέρωση. Το σημαντικό θέμα είναι ο υπολογισμός του $D_{e_{new}}$ και να καθορίσουμε την σχέση κυριαρχίας για το e_{new} εάν υπάρχει.

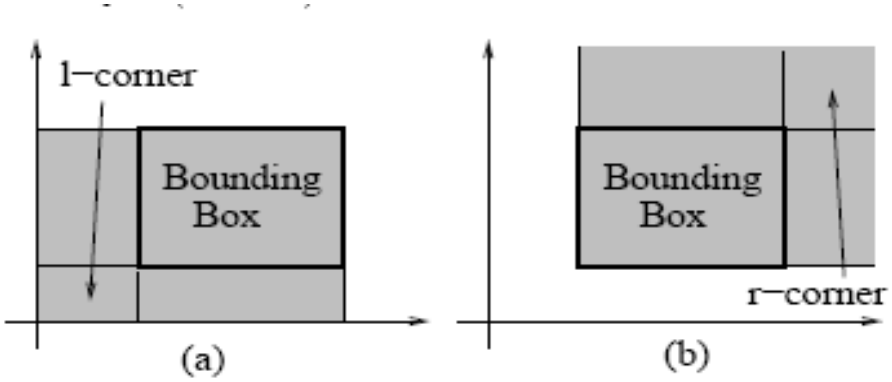
Είναι γνωστό ότι οι περισσότερες in-memory δομές δεδομένων για σημεία είναι δύσκολο να ισοσκελιστούν όταν έχουμε ενημέρωση των δεδομένων τους. Παρακάτω θα χρησιμοποιηθεί το in-memory δένδρο για να οργανώσουμε το R_N ώστε να υποστηρίξει τα δυο είδη των υπολογισμών που κάνουμε. Η μορφή των δομών δεδομένων που υιοθετούμε φαίνεται στην εικόνα 3.6.



Εικόνα 3.6

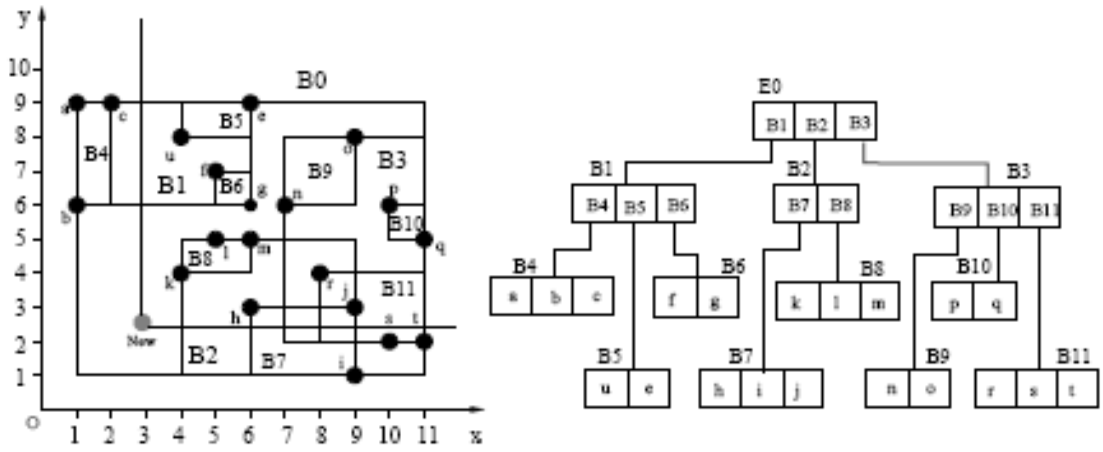
Το συνόλου {κ(e)} για τα στοιχεία στο R_N αποθηκεύονται κατά αύξουσα σειρά. Χρησιμοποιείται 1-1 αντιστοίχιση για να συνδέσουμε τα στοιχεία που αποθηκεύονται στο R-tree και στο σύνολο. Οι συνδέσεις ανάμεσα στο σύνολο και το δεξί άκρο των διαστημάτων επίσης διατηρούνται. Το σύνολο και αυτές οι συνδέσεις διατηρούνται συνέχεια για να υπολογίσουν την σχέση ανάμεσα στο άκρο ενός διαστήματος και το αντίστοιχο στοιχείο, όπως και για να καθορίσουν εάν το e_{old} έχει λήξει.

Θα χρησιμοποιήσουμε την αναζήτηση *κατά βάθος-πρώτα* στο R-δένδρο του παραπάνω αλγόριθμου. Ένας κόμβος στο R-δένδρο επεκτείνεται περισσότερο εάν το e_{new} πέσει στην «υποψήφια περιοχή» του bounding box. Για παράδειγμα, σε έναν δισδιάστατο χώρο, η σκιαγραφημένη περιοχή της εικόνας 3.7(a) είναι η υποψήφια περιοχή του bounding box. Εάν το e_{new} πέσει στην σκιαγραφημένη περιοχή, τότε ο κόμβος που αντιστοιχεί στο bounding box θα εξεταστεί περισσότερο. Διαφορετικά, το υποδένδρο που βρίσκεται στο bounding box θα απορριφθεί κατά την έρευνα μας. Επιπρόσθετα, όλο το υποδένδρο μπορεί αν απορριφθεί εάν το e_{new} πέσει στην αριστερή γωνία όπως σημειώνεται στην εικόνα 3.7. Σ' αυτή την περίπτωση όλα τα στοιχεία του υποδένδρου πρέπει να συμπεριληφθούν στο $D_{e_{new}}$. Ίδιες περιπτώσεις συναντάμε και χώρο d-διαστάσεων.



Εικόνα 3.7

Στην αναζήτηση κατά «βάθος – πρώτα», αμέσως αφαιρούμε ένα νέο στοιχείο του $D_{e_{new}}$ από το R-δένδρο αλλά δεν επιχειρούμε αμέσως να εξισορροπήσουμε το R-δένδρο για κάθε διαγραφή. Το R-δένδρο εξισορροπείται μετά την διαγραφή κάθε στοιχείου του $D_{e_{new}}$. Για αποδοτική εξισορρόπηση του R-δένδρου μετά την διαγραφή του $D_{e_{new}}$, υιοθετούμε ένα B+-δένδρο. Επιπλέον, για να κάνουμε πιο αποδοτική την αναζήτηση κατά «βάθος - πρώτα», τροποποιούμε το Bounding box ενός κόμβου όταν η αναζήτηση επιστρέφει πίσω σ' αυτόν τον κόμβο. Για παράδειγμα, σύμφωνα με το παράδειγμα της εικόνας 3.5 τροποποιούμε το bounding box B1 σε B4 μετά την διαγραφή των υποδένδρων που βρίσκονται στα B5 και B6, και επιστρέφουμε στο B1. Αυτό μας αποτρέπει από το να ερευνήσουμε χωρίς λόγο τα υπόλοιπα παιδιά του B1.



Εικόνα 3.8

Θα χρησιμοποιήσουμε την «καλύτερη πρώτη αναζήτηση» στο R-δένδρο του αλγορίθμου 3.1 για να καθορίσουμε την σχέση κυριαρχίας στο e_{new}. Για κάθε κόμβο u στο R-δένδρο, θα διατηρήσουμε την μέγιστη τιμή m_u ανάμεσα σε όλα τα στοιχεία του υποδένδρου που έχουν ρίζα το u. Για να επιτύχουμε την αποδοτικότερη αναζήτηση , διατηρούμε το max-heap του m_u ανάμεσα στους κόμβους που είναι υποψήφιοι ώστε να επεκταθούν, και ο heap top node επιλέγεται για να επεκταθεί. Τα κριτήρια επέκτασης του κόμβου είναι τα ίδια με αυτά που αναφέραμε και στην κυριαρχία. Όπως φαίνεται και στην εικόνα 3.7(β), επεκτείνουμε το u αν και μόνο αν το e_{new} βρίσκεται στην σκιαγραφημένη περιοχή του bounding box; Εάν το e_{new} βρίσκεται στην r-corner τότε ο αλγόριθμος τερματίζει και δίνει ως έξοδο το στοιχείο e' με k(e') = m_u στο υποδένδρο με ρίζα το u. Τερματίζουμε τον αλγόριθμο αν ο σωρός είναι άδειος ή αν ο κόμβος που εξερευνάτε είναι ένα στοιχείο e' που κυριαρχεί του e_{new}. Συνεπώς, το (k(e'), k(e_{new})] προστίθεται στο I_{R_N} εάν το e' υπάρχει. Διαφορετικά (0, k(e_{new})] προστίθεται.

Χρησιμοποιείται η κλασσική τεχνική εισαγωγής σε R-δένδρα για να εισάγουμε το e_{new} στο R-δένδρο.

3.5 Εκτέλεση μιας (n1,n2)-of-N αναζήτησης

Σ' αυτή την ενότητα, θα μελετήσουμε το πρόβλημα της εκτέλεσης μιας (n1,n2)-of-N αναζήτησης. Είναι ο υπολογισμός της κορυφογραμμής των στοιχείων που φτάνουν ανάμεσα στα n_2-οστά πιο πρόσφατα στοιχεία και στα n_1-οστά πιο πρόσφατα στοιχεία σε μια ροή δεδομένων, όπου $n_1 \leq n_2 \leq N$. Σε αντίθεση με την n-of-N αναζήτηση, όλα τα στοιχεία του P_N πρέπει να κρατούνται για την εκτέλεση όλων των (n1,n2)-of-N αναζητήσεων. Ο λόγος είναι σαφής: το n1 θα μπορούσε να ισούται με το n2. Ωστόσο, όμοια με την εκτέλεση της n-of-N αναζήτησης, οι δομές δεδομένων που πρέπει να διατηρηθούν είναι ένα R-δένδρο για το R_N και δυο ενδιάμεσα δένδρα. Παρακάτω, θα καθορίζουμε τις ιδιότητες που πρέπει να διέπουν ένα στοιχείο για να μπορεί να χαρακτηριστεί ως σημείο κορυφογραμμής για μια (n1,n2)-of-N αναζήτηση.

Ένα στοιχείο e από τα N πιο πρόσφατα στοιχεία του P_N μπορεί να κυριαρχείται από πολλά άλλα στοιχεία του P_N. Χρησιμοποιούμε το α_e για να σημειώνουμε το νεότερο στοιχείο e' που κυριαρχεί του e και έχει φτάσει πριν από το e; αυτό είναι,

$$k(\alpha_e) = \max\{k(e'): e' \kappa\upsilon\rho\iota\alpha\rho\chi\epsilon\acute{\iota}\ \tau o\upsilon\ e\ \&\ k(e') < k(e)\}.$$

Ομοίως, χρησιμοποιούμε το b_e για να σημειώνουμε το παλαιότερο στοιχείο e' που κυριαρχεί του e και φτάνει μετά το e; Αυτό είναι,

$$k(b_e) = \min\{k(e'): e' \kappa\upsilon\rho\iota\alpha\rho\chi\epsilon\acute{\iota}\ \tau o\upsilon\ e\ \&\ k(e') > k(e)\}.$$

Σε περίπτωση που το $\alpha_{e\,(b_e)}$ δεν υπάρχει, ένα dummy στοιχείο e_0 (e_∞) χρησιμοποιείται για να αναπαραστήσει το $\alpha_{e\,(b_e)}$ με $k(e_0) = 0$ $(k(e_\infty) = \infty)$. Να σημειώσουμε ότι $\alpha_e \rightarrow e$ έχει καθοριστεί ως η κρίσιμη σχέση κυριαρχίας . Θα ονομάζουμε $b_e \rightarrow e$ «οπισθοδρομική κρίσιμη σχέση κυριαρχίας», α_e κρίσιμο πρόγονο του e, και το b_e οπισθοδρομικό πρόγονο του e. Αμέσως μπορεί να γίνει αντιληπτό ότι ένα στοιχείο e μπορεί να είναι σημείο κορυφογραμμής μιας (n1,n2)-of-N αναζήτησης αν και μόνο αν ο κρίσιμος πρόγονος φτάσει νωρίτερα από το n2-οστό πιο πρόσφατο στοιχείο και ο οπισθοδρομικός πρόγονος να φτάσει αργότερα από το n1-οστό πιο πρόσφατο στοιχείο. Το παραπάνω μπορεί να τεθεί με το ακόλουθο θεώρημα.

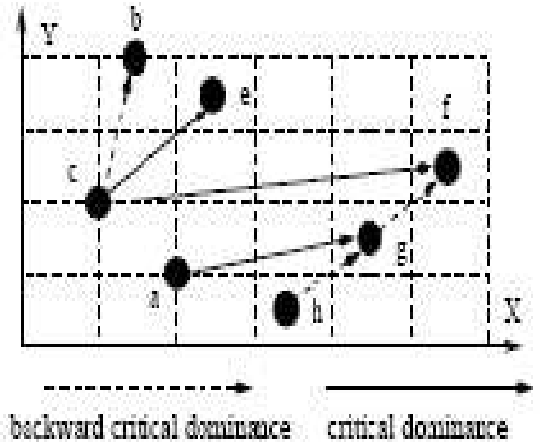
Θεώρημα 3.4: *Ένα στοιχείο e του P_N είναι στοιχείο κορυφογραμμής για μια (n1,n2)-of-N αναζήτηση αν και μόνο αν*

$$k(\alpha_{\square}) < M - n2 + 1 \leq k(e) \leq M - n1 + 1 < k(b_e),$$

όπου M είναι ο αριθμός των στοιχείων που έχουν εμφανιστεί μέχρι στιγμής.

Ένα γράφημα, με κορυφή το σύνολο $P_N \cup \{e_0, e_\infty\}$ και σύνολο ακμών που αποτελείται από κρίσιμες και οπισθοδρομικές κρίσιμες σχέσεις, καλείται CBC γράφημα κυριαρχίας του P_N. Και σημειώνεται ως CG_{P_N}. Είναι φανερό ότι ο αριθμός των ακμών είναι $O(|P_N|)$ από την στιγμή που κάθε στοιχείο έχει μόνο δυο εισερχόμενα τόξα.

Λαμβάνοντας υπόψη την ροή δεδομένων της εικόνας 3.2, στην εικόνα 3.9 απεικονίζεται το γράφημα κυριαρχίας ένα N = 7 αφού έχουμε παραλείψει τις dummy κορυφές e_0 *και* e_∞ και τις ακμές που τις αντιστοιχούσαν.



Εικόνα 3.9

3.5.1 Κωδικοποίηση, Εκτέλεση Αναζήτησης, & Διατήρηση

Το θεώρημα 3.4 είναι θεμελιώδες για τους αλγορίθμους εκτέλεσης μιας (n1,n2)-of-N αναζήτησης. Το CG_{P_N} μπορεί να απεικονιστεί από τις ακτές που κωδικοποιούνται σε ενδιάμεσα δέντρα.

Ένας γράφος CG_{P_N} κωδικοποιείται ως εξής: για κάθε στοιχείο e του P_N, χρησιμοποιούμε $\big((k(a_e), k(e)], k(b_e)\big)$ για να απεικονίσει τα δυο εισερχόμενα τόξα $a_e \to e$ και $b_e \to e$. Στην συνέχεια, κατασκευάζουμε το ενδιάμεσο δένδρο στα διαστήματα $\{(k(a_e), k(e)] : e \in P_N\}$. Για να εκτελέσουμε μια (n1,n2)-of-N αναζήτηση, εφαρμόζουμε τον αλγόριθμο τεμαχισμού χρησιμοποιώντας M - n2 + 1 για την διαίρεση των διαστημάτων ενώ ελέγχουμε την συνθήκη $k(e) \leq M - n1 - +1 < k(b_e)$. Παρακάτω είναι η περιγραφή του αλγόριθμου.

Algorithm 3.2 Processing (n_1, n_2)-of-N Query

Description:

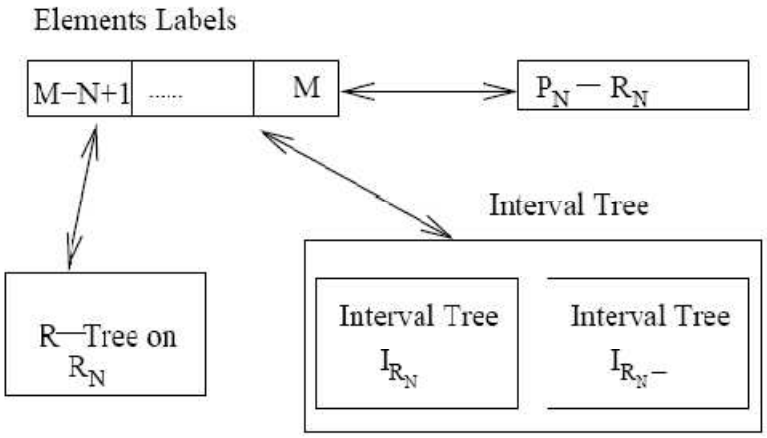
1: Stab the intervals by $M - n_2 + 1$;
2: for each element e in stabbing result **do**
3: if $\kappa(e) \leq M - n_1 + 1 < \kappa(b_\epsilon)$ then
4: return e;
5: end if
6: end for

Να σημειώσουμε ότι στην γραμμή 2 του παραπάνω αλγόριθμου «element e in stabbing result» σημαίνει ότι το διάστημα $(k(a_e), k(e)]$ έχει διαιρεθεί. Ο αλγόριθμος τρέχει σε $O(logN + l)$ όπου l είναι ο αριθμός των διαστημάτων που έχουν διαιρεθεί. Θεωρούμε ότι μια n-of-N αναζήτηση είναι ειδική περίπτωση μιας (n1,n2)-of-N αναζήτησης. Για να πετύχουμε την ίδια χρονική πολυπλοκότητα της εκτέλεσης μιας n-of-N αναζήτησης, διαιρούμε το ενδιάμεσο δένδρο του CG_{P_N} σε δυο ενδιάμεσα δένδρα. Η διαδικασία περιγράφεται στην συνέχεια.

I_{R_N}: *διαστήματα με το δεξί άκρο να αντιστοιχεί σε ένα στοιχείο του* R_N.

$I_{R_{N-}}$: *διαστήματα με το δεξί άκρο να αντιστοιχεί σε ένα στοιχείο στο* $P_N - R_{\square}$.

Μπορούμε εύκολα να αποδείξουμε ότι το I_{R_N} αποτελείται από τα διαστήματα που κωδικοποιήθηκαν από τον γράφο κυριαρχίας. Οι δομές δεδομένων που χρησιμοποιούνται φαίνονται στην εικόνα 3.10.



εικόνα 3.10

Είναι φανερό ότι οι οπισθοδρομικές κρίσιμες σχέσεις κυριαρχίας και οι κρίσιμες σχέσεις κυριαρχίας μπορούν να καθοριστούν αναδρομικά. Όταν ένα νέο στοιχείο e φτάνει, εάν ένα στοιχείο e' του R_N κυριαρχείται από το e, τότε το e είναι ο οπισθοδρομικός κρίσιμος πρόγονος του e'. Εάν ένα στοιχείο e'' του R_N κυριαρχεί του e, τότε το e''είναι ο κρίσιμος πρόγονος του e. Συνεπώς, ο αλγόριθμος διατήρησης του CG_{P_N} μέσω του I_{R_N} και του $I_{R_{N-}}$ είναι όμοιος με τον αλγόριθμο 3.1. Παρακάτω στον αλγόριθμο 3.3 παρουσιάζεται ο αλγόριθμος διατήρησης περιγράφοντας τις ομοιότητες και τις διαφορές του αλγόριθμου 3.1.

Algorithm 3.3 Data Structures Maintenance

- It always expires the oldest element in P_N; the subsequent deletion in R-trees and update of interval trees are the same as that in Algorithm 3.1. Inserting the new element e_{new} to R_N is also the same as Algorithm 3.1.

- The search paradigms on the R-tree of R_N are the same as those in Algorithm 3.1. That is, the depth-first search is used to determine the new redundant elements of which e_{new} is the backward critical ancestor, while the best-first search is used to determine the critical ancestor of e_{new}.

- The interval corresponding to the critical dominance will be added to I_{R_N} in the same way as in Algorithm 3.1. For a new redundant element e' caused by e, remove e' from the R-tree and place it in $P_N - R_N$. Then, remove $((\kappa(a_{e'}), \kappa(e')], \infty)$ from I_{R_N} and insert $((\kappa(a_{e'}), \kappa(e')], \kappa(e))$ to $I_{R_N -}$.

Το κόστος ενημέρωσης του R-δένδρου στο R_N είναι το ίδιο με αυτό του αλγόριθμου 3.1

Βιβλιογραφία

- **Constrained Skyline Computing over Data Streams**

Κεφάλαιο 4

Συνεχόμενες αναζητήσεις κορυφογραμμής

Περιεχόμενα

4.1 Εισαγωγή

Στο κεφάλαιο αυτό, θα απαντήσουμε ικανοποιητικά και αποδοτικά στις αναζητήσεις κορυφογραμμής που αφορούν τα N πιο πρόσφατα στοιχεία σε μια ροή δεδομένων και στις τεχνικές που έχουν αναπτυχθεί για αυτό το σκοπό. Σκοπός είναι να παρακολουθήσουμε συνεχόμενα τα αποτελέσματα για αναζητήσεις μεγάλης διάρκειας. Οι τεχνικές που έχουν αναπτυχθεί για ad-hoc δίκτυα κρίνονται μη-αποδοτικές ή άχρηστες, καθώς δεν λαμβάνουν υπόψη τα συγκεκριμένα χαρακτηριστικά των συνεχόμενων αναζητήσεων κορυφογραμμής.

Στην συνέχεια, θα παρουσιάσουμε μια αποδοτική παρακολούθηση μιας n-of-N αναζήτησης κορυφογραμμής σε μια ροή δεδομένων.

Η οργάνωση του κεφαλαίου είναι η εξής: στην ενότητα 4.2 θα παρουσιάσουμε την τεχνική για τις συνεχόμενες n-of-N αναζητήσεις κορυφογραμμής. Στην ενότητα 4.3 παρουσιάζουμε έναν αλγόριθμο για συνεχόμενες (n1,n2)-of-N αναζητήσεις.

4.2 συνεχόμενες n-of-N αναζητήσεις

Οι συνεχόμενες αναζητήσεις τίθενται μια φορά και τρέχουν συνεχόμενα για να παράγουν αποτελέσματα με τις ενημερώσεις των υπογραμμισμένων συνόλων δεδομένων. Με την άφιξη ενός στοιχείου, το αποτέλεσμα S_N μιας n-of-N αναζήτησης πρέπει να αλλάζουν. Ένας απλός τρόπος είναι να τρέξουμε ξανά τον αλγόριθμο αναζήτησης της ενότητας 3.4 για την άφιξη ενός νέου στοιχείου. Αυτό απαιτεί $O(logN + s)$ για κάθε νέο στοιχείο. Η ορθότητα του αλγόριθμου που θα παρουσιάσουμε παρακάτω βασίζεται στην παρακάτω πρόταση.

Πρόταση 4.1: *Μόλις ένα νέο στοιχείο e_{new} φτάνει, το προσωρινό αποτέλεσμα S_N μιας n-of-N αναζήτησης μπορεί να υποστεί τις παρακάτω αλλαγές:*

- ***Διαγραφή:*** *ένα στοιχείο $e \in S_N$ απομακρύνεται εάν το e_{new} κυριαρχεί το e ή το e έχει λήξει.*
- ***Εισαγωγή:*** *ένα στοιχείο $e \in S_N$ προστίθεται στο S_N εάν στο ενημερωμένο G_{R_N} μετά την εφαρμογή του αλγόριθμου 3.1 για την εισαγωγή του e_{new}, είτε:*
    - *e είναι το e_{new} και $\nexists e'$ τέτοιο ώστε $\kappa(e') \geq M - n + 1$ και $e' \xrightarrow{c} e_{new}$, είτε*
    - *e κυριαρχείται από το e'' στο S_N που μόλις έληξε και το e'' δεν κυριαρχείται από το e_{new}.*

Στον αλγόριθμο που θα δείξουμε αμέσως μετά, το M είναι ο συνολικός αριθμός των στοιχείων. Η κορυφή του σωρού που χρησιμοποιούμε περιέχει πάντα την μικρότερη τιμή. Στον ψευδοκώδικα, το e_{top} είναι το στοιχείο του S_N αντιστοιχεί στην κορυφή του σωρού του min-heap. Λήγει από τα n πιο πρόσφατα στοιχεία εάν $k(e_{top}) < M - n + 1$. οπότε, κάθε φορά χρειάζεται να ελέγξουμε μόνο την κορυφή του σωρού στην προσωρινή λύση. Να σημειώσουμε ότι το $D_{e_{new}}$ είναι το σύνολο των νέων πλεοναζόντων στοιχείων που κυριαρχούνται από το e_{new} που υπολογίστηκε από τον αλγόριθμο 3.1. Υποθέτουμε ότι η διαγραφή του στοιχείου στο $D_{e_{new}}$ στον αλγόριθμο 3.1 θα ειδοποιήσει την εκτέλεση μια συνεχόμενης αναζήτησης q. Αυτό μπορεί να επιτευχθεί με την ένωση ενός στοιχείου e στην συνεχόμενη αναζήτηση που χρησιμοποιεί το e σαν μέρος του αποτελέσματος.

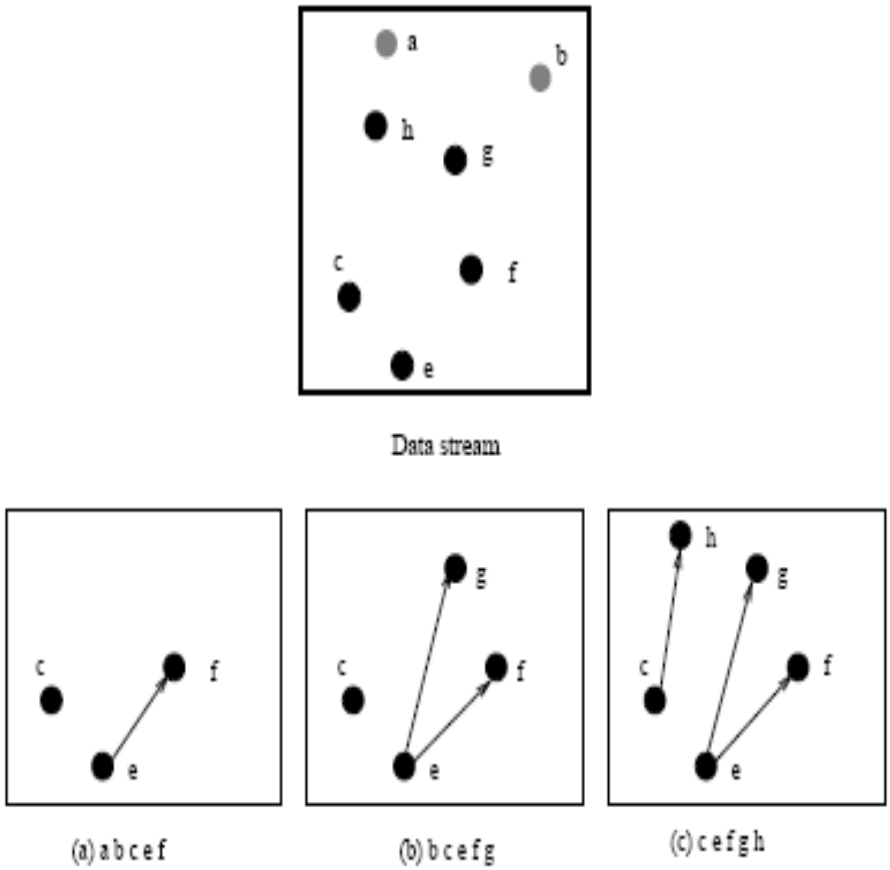
Στον αλγόριθμο 4.1, η $Removal(e, S_N)$ αφαιρεί το e από το S_N και επίσης αφαιρεί το k(e) από το min-heap. Η $Add(e, S_N)$ προσθέτει το e στο S_N και εισάγει το k(e) στον min-heap.

Algorithm 4.1 Processing Continuous n-of-N Queries

Description:

1: while new element e_{new} do
2: $M := M + 1$;
3: for $e \in S_n \cap D_{new}$ do
4: Removal (e, S_n);
5: end for
6: if $\nexists e' \xrightarrow{\varepsilon} e_{new}$ with $\kappa(e') \geq M - n + 1$ then
7: Add (e_{new}, S_n);
8: end if
9: while $\kappa(e_{top}) < M - n + 1$ do
10: Removal (e_{top}, S_n);
11: for $\forall e$ with $e_{top} \xrightarrow{\varepsilon} e$ in G_{RN} do
12: Add (e, S_n);
13: end for
14: end while
15: end while

Αλγόριθμος 4.1



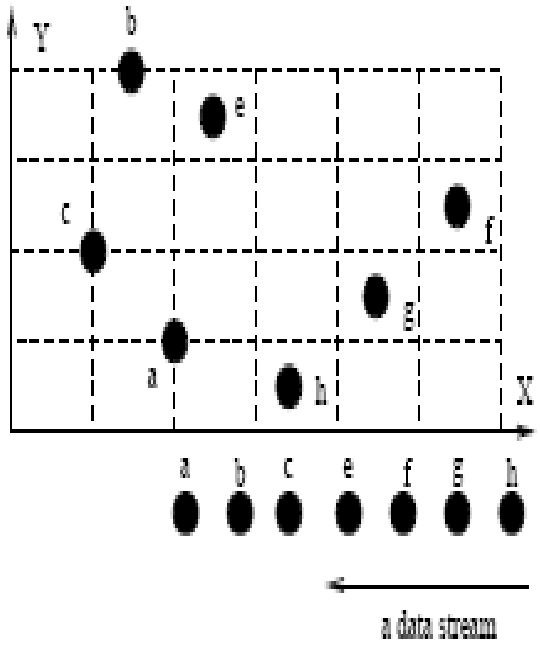
Εικόνα 4.1: Παράδειγμα

Παράδειγμα 4.1 *Θεωρώντας την ροή του παραδείγματος όπως φαίνεται στην εικόνα 4.1, υποθέτουμε ότι N = 5 και n = 4. Τα στοιχεία εμφανίζονται σύμφωνα με την αλφαβητική τους σειρά. Η ροή έχει αρχικά 5 στοιχεία {a, b, c, e, f}. Όταν ένα στοιχείο φτάνει, τα αποτελέσματα της αναζήτησης ενημερώνονται σύμφωνα με τον αλγόριθμο 4.1. Το αποτέλεσμα αποτελείται από {c, e} για την n-of-N αναζήτηση και παραμένουν αμετάβλητα όταν τα N πιο πρόσφατα στοιχεία αλλάζουν από {a, b, c, e, f} σε {b, c, e, f, g}. Οι εικόνες 4.1(α) και 4.1(b) δείχνουν τον αντίστοιχο γράφο κυριαρχίας. Μόλις το στοιχείο h φτάσει, ο νέος ενημερωμένος γράφος κυριαρχίας φαίνεται στην εικόνα 4.1(c). Από τη στιγμή που k(c) = 3 < 7-4+1, το c λήγει και επομένως το h προστίθεται στην λύση. Συνεπώς, το e και το h σχηματίζουν την απάντηση μόλις το h φτάσει.*

4.3 συνεχόμενες (n1,n2)-of-N αναζητήσεις

Μια (n1,n2)-of-N αναζήτηση ταυτίζεται με την n-of-N αναζήτηση όταν n1 = 1. Θα υποθέσουμε ότι n1 ≠ 1.

Σε αντίθεση με την συνεχόμενη n-of-N αναζήτηση, ένα σημείο της κορυφογραμμής για μια (n1,n2)-of-N αναζήτησης μπορεί να μην είναι σημείο κορυφογραμμής σε προηγούμενο αποτέλεσμα, ούτε να κυριαρχείται από κάποιο σημείο κορυφογραμμής ούτε από κάποιο νέο στοιχείο. Για παράδειγμα, θεωρούμε την ροή δεδομένων της εικόνας 4.2; υποθέτουμε ότι N = 6, n1 = 2 και n2 = 4 και τα στοιχεία φτάνουν με την σειρά a,b,….,g. Η κορυφογραμμή μιας τέτοιας (n1,n2)-of-N αναζήτησης αποτελείται μόνο από ένα σημείο το c. Μόλις το νέο στοιχείο h φτάσει, το αποτέλεσμα γίνεται e και g όπου το g δεν κυριαρχείται από το c.



Εικόνα 4.2

Υποθέτουμε ότι το $S_{n1,n2}$ είναι η κορυφογραμμή του προσωρινού αποτελέσματος της (n1,n2)-of-N αναζήτησης. Οπότε:

$$S_{n1,n2} = \{ \text{ e : e ικανοποιεί το (3.3) } \} \qquad (4.1)$$

Έστω $C_{n1,n2} = \{\ e : \kappa(\alpha_e) < M - n2 + 1 \leq M - n1 + 1 < k(e)\}$ (4.2)

Το παρακάτω θεώρημα αποδεικνύει ότι κάθε στοιχείο του $C_{n1,n2}$ τελικά θα είναι σημείο της κορυφογραμμής της (n1,n2)-of-N αναζήτησης.

Θεώρημα 4.1 *Υποθέτουμε ότι το e είναι στοιχείο του $C_{n1,n2}$. Τότε το e θα είναι σημείο της κορυφογραμμής της (n1,n2)-of-N αναζήτησης όταν k(e) – M + n1 -1 νέα στοιχεία φτάσουν όπου M είναι ο προσωρινός αριθμός των στοιχείων που βρίσκονται σε μια ροή δεδομένων.*

Απόδειξη: υποθέτουμε ότι ο a_e είναι ο κρίσιμος πρόγονος του e. Έστω M' = M + (κ(e) – M + n1 -1), a'_e είναι κρίσιμος πρόγονος του e και b'_e ο οπισθοδρομικός κρίσιμος πρόγονος του e μόλις τα κ(e) – M + n1 -1 νέα στοιχεία φτάσουν. Είναι φανερό ότι $\alpha'_e = a_e$ ή $\alpha'_e = e_0$. Οπότε, $\kappa(\alpha'_e) \leq k(a_e) < M - n2 + 1 \leq M' - n2 + 1$. Να σημειώσουμε ότι $\kappa(b'_e) > \kappa(e)$. Οπότε, $k(e) = M' - n1 + 1 < k(b'_e)$.

Στον αλγόριθμο αυτό, το $C_{n1,n2}$ χρησιμοποιείται ως το υποψήφιο σύνολο που διατηρείται και θα ενημερώνεται συνεχόμενα σε αντίθεση με το $S_{n1,n2}$. Η ορθότητα του αλγορίθμου βασίζεται στο παρακάτω θεώρημα.

Θεώρημα 4.2 *υποθέτουμε ότι το $S_{n1,n2}$ και το $C_{n1,n2}$ είναι τα προσωρινό σύνολο αποτελεσμάτων και το υποψήφιο σύνολο αντίστοιχα σε σχέση μια ροή δεδομένων με M στοιχεία που έχουν εμφανιστεί μέχρι τώρα. Επιπρόσθετα, υποθέτουμε ότι το $S'_{n1,n2}$ και $C'_{n1,n2}$ είναι το προσωρινό σύνολο αποτελεσμάτων και το υποψήφιο σύνολο μετά την άφιξη ενός στοιχείου. Οπότε, $\forall e \in S'_{n1,n2} \cup C'_{n1,n2}$ είτε*

- $e \in S_{n1,n2} \cup C_{n1,n2}$, *είτε*
- $\exists e' \in S_{n1,n2}$ *με* $k(e') = M - n2 + 1$, *με το e' να κυριαρχεί του e, είτε*
- *e είναι το νέο στοιχείο που πέφτει στο $C'_{n1,n2}$. Να σημειώσουμε αν $n1 \neq 1$, το νέο στοιχείο δεν μπορεί να πέσει στο $S'_{n1,n2}$.*

Ο αλγόριθμος για την συνεχόμενη εκτέλεση μιας (n1,n2)-of-N αναζήτησης είναι όμοιος με την μέθοδο που αναπτύξαμε και στον αλγόριθμο για μια n-of-N αναζήτηση. Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται παρακάτω.

Algorithm 4.2 Continuous (n_1, n_2)-of-N queries

Description:

1: Stab the intervals with the point $M - n_2 + 1$;

2: for $\forall$ e in the stabbing result do

3: if $\kappa(e) \leq M - n_1 + 1 < \kappa(b_e)$ then

4: add e into S_{n_1,n_2};

5: add the trigger T_e to $T_{S_{n_1,n_2}}$;

6: else if $\kappa(e) > M - n_1 + 1$ then

7: add e into C_{n_1,n_2};

8: add the trigger T_e into $T_{C_{n_1,n_2}}$

9: end if /* produce the initial result */

10: end for

11: while new element e do

12: Update $(R_{RN}, I_{RN}, I_{RN-}; e)$;

13: Incremental $(T_{S_{n_1,n_2}}, T_{C_{n_1,n_2}}, S_{n1,n_2}, C_{n1,n_2}; e)$

14: end while

- **Επεξεργασία ενός νέου στοιχείου e:** το M αυξάνεται κατά 1. Οι ενημερώσεις των R δένδρων και των ενδιάμεσων δένδρων παρουσιάστηκαν από τον αλγόριθμο 3.3, όπου ο κρίσιμος πρόγονος a_e του e έχει ήδη παραχθεί. Εάν $\alpha_e < M - n2 + 1$, τότε το e τοποθετείται στο $C_{n1,n2}$ και το $T_e = k(e)$ στο $T_{C_{n1,n2}}$. Πρέπει να ενημερώσουμε το $\beta_{e\prime}$ εάν το e' είναι στο $S_{n1,n2} \cup C_{n1,n2}$ και το e' τώρα κυριαρχείται οπισθοδρομικά από το e.
- **Επεξεργασία** $C_{n1,n2}$: μόλις ένα νέο στοιχείο φτάσει, ελέγχουμε η κορυφή του σωρού είναι ίση με M – n1 + 1 μετά την αύξηση κατά 1 του M. Εάν η κορυφή του σωρού ισούται με M – n1 + 1, τότε το αντίστοιχο στοιχείο e' προστίθεται στο $S_{n1,n2}$ και προσθέτουμε το $(\alpha_{e'}, \beta_{e\prime})$ στο $T_{S_{n1,n2}}$.
- **Επεξεργασία** $S_{n1,n2}$: να σημειώσουμε ότι οι δυο μίνι-σωροί T_α, T_β διατηρούνται για κάθε στοιχείο $e \in S_{n1,n2}$, αντίστοιχα. Μόλις ένα νέο στοιχείο e φτάσει και το M θα αυξηθεί κατά 1, ελέγχουμε τους δύο σωρούς. Πρώτα ελέγχουμε την κορυφή του σωρού T_β. Εάν η κορυφή του σωρού $T_\beta = M - n1 + 1$ τότε το αντίστοιχο στοιχείο αφαιρείται από το $S_{n1,n2}$. Εάν η κορυφή του σωρού $T_\alpha = M - n2$, τότε αφαιρούμε το αντίστοιχο στοιχείο από το $S_{n1,n2}$ αλλά και από T_β, και προσθέτουμε τα παιδιά του με βάση τις σχέσεις κυριαρχίας στο $S_{n1,n2}$ ή $C_{n1,n2}$.

Βιβλιογραφία

- **Efficient continuous skyline computation**

Κεφάλαιο 5

Υπολογισμός Κορυφογραμμής σε ομότιμα (p2p) δίκτυα

Περιεχόμενα

5.1 Εισαγωγή

Τα ομότιμα συστήματα (p2p), ως ένα δημοφιλές μέσω αναζήτησης και διαμοιρασμού κατανεμημένων πληροφοριών, έχει αποκτήσει ιδιαίτερο ενδιαφέρον στον τομέα της πληροφορικής. Για την παροχή αποδοτικής ανταλλαγής πληροφοριών και μεθόδων αναζήτησης στους χρήστες των συστημάτων αυτών, έρευνες έχουν επικεντρωθεί στην ακριβή αναζήτηση, την αναζήτηση κατά εύρος και την αναζήτηση με βάση κάποια λέξη κλειδί.

Οι αναζητήσεις οριζοντιογραμμης έχουν μελετηθεί στα κεντρικοποιημένα συστήματα. Όπως αναφέραμε και στην εισαγωγή, η αναζήτηση κορυφογραμμής επιστρέφει ένα σύνολο σημείων που δεν κυριαρχούνται από κανένα σημείο σε ένα δοθέν σύνολο δεδομένων.

Ωστόσο, δεν υπάρχει αρκετή δουλειά που να αφορά στην αποδοτική εκτέλεση αναζήτησης κορυφογραμμής σε p2p δίκτυα. Το πρόβλημα που υπάρχει είναι στην υιοθέτηση ενός κεντρικοποιημένου αλγορίθμου και στην δημιουργία ενός ευρετηρίου που θα υπολογίζει τις οριζοντιογραμμές. Από τις πρώτες προσπάθειες μελέτης για την προοδευτική εκτέλεση αναζητήσεων κορυφογραμμής είναι το [1] που αναφέρθηκε σε ένα p2p δίκτυο, το CAN [2]. Το μειονέκτημα ήταν ότι η μελέτη αυτή επικεντρώθηκε στις περιορισμένες αναζητήσεις κορυφογραμμής.

Παρακάτω, θα παρουσιάσουμε το Skyline Space Partitioning (SSP) σε ένα γνωστό p2p δίκτυο που ονομάζεται BATON για να μπορέσουμε να παρέχουμε μια επαρκή και αποτελεσματική εκτέλεση μιας αναζήτησης κορυφογραμμής που αναζητά σημεία κορυφογραμμής σε όλο το διάστημα των δεδομένων. Τα 3 παρακάτω κριτήρια είναι πολύ σημαντικά για να επιτύχουμε μια αποδοτική εκτέλεση:

1. Μικρός αριθμός εμπλεκόμενων κόμβων
2. Μικρός αριθμός μηνυμάτων προς αναζήτηση
3. Ισορροπία στον φόρτο αναζήτησης

Σε αντίθεση με τις περιορισμένες αναζητήσεις κορυφογραμμής, οι οποίες πρέπει να έχουν πρόσβαση σε οποιοδήποτε μέρος του διαστήματος των δεδομένων, η πρόσβαση σε δεδομένα των μη περιορισμένων αναζητήσεων είναι πιο πιθανό τα πέσει στο τμήμα του διαστήματος των δεδομένων που περιέχουν την κορυφογραμμή.

Η παρακάτω αναφορά ικανοποιεί τα παραπάνω κριτήρια σε 3 επίπεδα. Αρχικά, οργανώνουμε τις πολυδιάστατες περιοχές δεδομένων χρησιμοποιώντας ένα ισορροπημένο BATON δένδρο. Αναθέτουμε σε κάθε περιοχή έναν αριθμό περιοχής, στον οποίο κάθε bit κωδικοποιεί ένα τμήμα του διαστήματος δεδομένων, οι λεπτομέρειες του οποίου

αποθηκεύονται στο ιστορικό του τμήματος της περιοχής. Μ’ αυτόν τον τρόπο μπορούμε να δρομολογήσουμε μια αναζήτηση υπολογίζοντας τον επιθυμητό αριθμό περιοχής βασισμένοι στα ιστορικά των τμημάτων περιοχής. Δεύτερον, καθορίζουμε το χώρο αναζήτησης της κορυφογραμμής ώστε να περιορίσουμε τον αριθμό των εμπλεκόμενων κόμβων και να χωρίσουμε το διάστημα αναζήτησης σε υποδιαστήματα σε κάθε hop για να παραλληλοποιήσουμε την εκτέλεση και να ελέγξουμε τον αριθμό των μηνυμάτων αναζήτησης. Τρίτον, ισορροπούμε τον φόρτο αναζήτησης με την σωστή και ισορροπημένη διαμέριση του διαστήματος των δεδομένων κατά την είσοδο ή έξοδο των κόμβων στο δίκτυο.

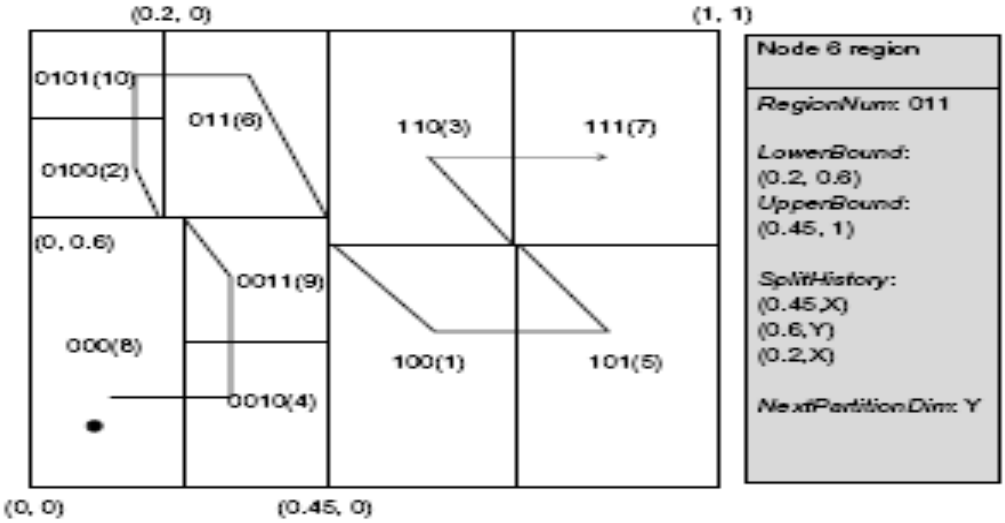
Στην αναφορά προτείνεται μια προσέγγιση που λέγεται SSP η οποία τμηματοποιεί και αριθμεί τα διαστήματα δεδομένων ανάμεσα στους κόμβους έτσι ώστε το επιθυμητό υποδιάστημα να μπορεί να υπολογιστεί με ακρίβεια για να μπορούν να ελεγχθούν με ακρίβεια οι κόμβοι που αποκτούν πρόσβαση αλλά και τα μηνύματα αναζήτησης κατά την εκτέλεση μιας αναζήτησης κορυφογραμμής.

Η ισορροπία του φόρτου αναζήτησης ανάμεσα στους κόμβους επιτυγχάνεται με την ισορροπημένη τμηματοποίηση του διαστήματος των δεδομένων και την δυναμική μετακίνηση του φόρτου. Ο μηχανισμός δειγματοληψίας που χρησιμοποιείται παρακολουθεί την ποιότητα της κατανομής του φόρτου αλλά και την αποδοτική εκτέλεση της δειγματοληψίας με τον συνδυασμό άμεσης και τυχαίας δειγματοληψίας.

5.2 Τμηματοποίηση χώρου

Θεωρούμε ότι οι τιμές των δεδομένων σε κάθε διάσταση ανήκουν στο εύρος [0,1]. Όλος ο d-διαστάσεων χώρος δεδομένων {[0,1],[0,1],…..,[0,1]} είναι κατανεμημένος σε ένα p2p δίκτυο με N κόμβους, ο καθένας από τους οποίους διατηρεί μια d-διαστάσεων περιοχή δεδομένων και τα σημεία των δεδομένων βρίσκονται μέσα σ' αυτήν την περιοχή. Η περιοχή δεδομένων κατασκευάζεται με ένα ή περισσότερα πολύ-ορθογώνια, καθένα από τα οποία έχει ένα κατώτατο όριο (αριστερό σημείο βάσης του ορθογωνίου) και ένα ανώτατο όριο (δεξί κορυφαίο σημείο του ορθογωνίου).

Εκτελούμε την αναζήτηση κορυφογραμμής στην SQ = (q1,q2,…,qd), όπου το $q_i \in \{Max, Min\}$ ($1 \leq i \leq d, Max, Min$ *η μεγαλύτερη και η μικρότερη προτίμηση*) για την διάσταση d. Όλα τα παραδείγματα που θα χρησιμοποιήσουμε βασίζονται στο δίκτυο που φαίνεται στην εικόνα 5.1, όπου κάθε κόμβος έχει ως αναγνωριστικό έναν αριθμό περιοχής και ένα id που βρίσκεται μέσα σε παρένθεση. Το SQ1 = {Min, Min} χρησιμοποιείται παρακάτω για την αναζήτηση κορυφογραμμής.



Εικόνα 5.1

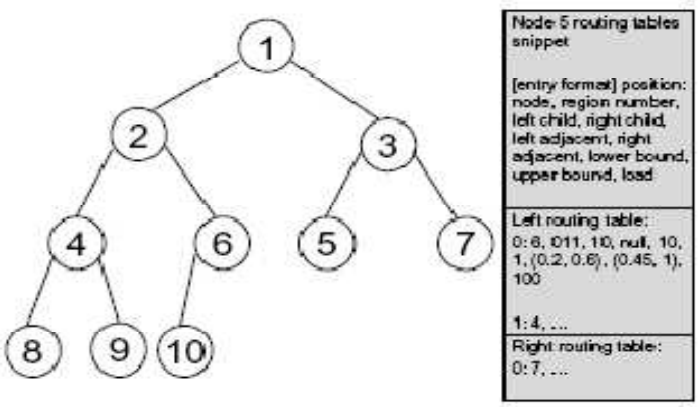
5.3 Πληροφορίες για το BATON

Το BATON (Balanced Tree Overlay Network) βασίζεται σε ένα δυαδικό ισορροπημένο δένδρο όπου κάθε φύλλο του δένδρου αντιστοιχεί σε έναν κόμβο του p2p δικτύου. Η θέση του κάθε κόμβου καθορίζεται από ένα ζεύγος τιμών (επίπεδο, αριθμός), όπου το επίπεδο ξεκινάει με το 0 από την ρίζα, και ο αριθμός ξεκινά με το 1 από τον αριστερότερο κόμβο του κάθε επιπέδου. Κάθε κόμβος του δένδρου αποθηκεύει τους συνδέσμους με τους γονείς του, τα παιδία του, τους παρακείμενους κόμβους αλλά και επιλεγμένους κόμβους με απόσταση που είναι ίση με δύναμη του 2 από την αριστερή/δεξιά πλευρά του ίδιου επιπέδου στον δεξί/αριστερό πίνακα δρομολόγησης. Το BATON διατηρεί ισορροπημένο το δένδρο αναγκάζοντας κάθε κόμβο να έχει συμπληρώσει και τους δυο πίνακες δρομολόγησης (δεξί και αριστερό) πριν έχει έναν κόμβο-παιδί, παράμετρος που είναι πολύ σημαντική για την αποτελεσματική δρομολόγηση. Το κόστος για την είσοδο/έξοδο ενός κόμβου και για την αναζήτηση είναι $O(logN)$.

5.4 Μηχανισμός αναζήτησης

Χρησιμοποιούμε την z-curve μέθοδο για να αποτυπώσουμε τον πολυδιάστατο χώρο δεδομένων στο μονοδιάστατο BATON. Αριθμούμε μια περιοχή σύμφωνα με την z-σειρά και την θέση της περιοχής στο διαιρεμένο χώρο των δεδομένων. Ο μηχανισμός αναζήτησης βασίζεται στην σχέση μεταξύ των περιοχών και των συναρτήσεων αυτών των περιοχών.

Στην εικόνα 5.2 περιγράφεται η αποτύπωση των περιοχών δεδομένων της εικόνας 5.1. Κάθε κόμβος του δένδρου διατηρεί μια περιοχή δεδομένων και οι σύνδεσμοι που φαίνονται αντιστοιχούν στην σειρά με την οποία έγιναν οι επισκέψεις στις περιοχές δεδομένων στην z-σειρά. Όπως φαίνεται στο δεξί μέρος της εικόνας 5.2, κρατάμε την δομή του πίνακα δρομολόγησης του BATON, και προσθέτουμε τους προσκείμενους κόμβους σε κάθε εγγραφή του πίνακα δρομολόγησης για να διευκολύνεται ο έλεγχος της συγχώνευσης του χώρου δεδομένων.



Εικόνα 5.2

Κάθε κόμβος διατηρεί τις ακόλουθες πληροφορίες για την περιοχή που του ανήκει:

1. **Αριθμό περιοχής:** Έναν 0-1 RNum που χαρακτηρίζει μια περιοχή και είναι σύμφωνος με την z-σειρά της περιοχής, στην οποία η τιμή του bit 0 ή 1 σε μια συγκεκριμένη τοποθεσία του bit υποδεικνύει αν η περιοχή είναι στο κατώτατο ή ανώτατο τμήμα του διαιρεμένου χώρου δεδομένων.
2. **Εύρος δεδομένων:** ζεύγη των κατώτατων ορίων LowerBound (αριστερό σημείο της βάσης) και ανώτατων ορίων UpperBound (δεξί κορυφαίο σημείο).
3. **Ιστορικό διαίρεσης:** μια λίστα εγγραφών με τις τιμές διαχωρισμού και τις διαστάσεις (SplitPos,SplitDim).
4. **Επόμενη διάσταση τμηματοποίησης:** ένα bit που υποδεικνύει την επόμενη διάσταση της διαίρεσης. Στην εικόνα 1δίνεται ένα παράδειγμα για τις πληροφορίες της περιοχής του κόμβου 6.

Να σημειωθεί δεν αντιπροσωπεύει μόνο ένα τμήμα δεδομένων που διατηρείται στον κόμβο, αλλά αναφέρεται επίσης σε ένα υπερσύνολο περιοχών στο οποίο μπορεί να καταλήξει το εύρος ή ο στόχος της αναζήτησης. Δεδομένου μια περιοχής m, θέτουμε RNum(m) ως αριθμό περιοχής και RNum(m).length ως τον αριθμό των bits του RNum(m). Καθορίζουμε τις παρακάτω σχέσεις των περιοχών σύμφωνα με τις λειτουργίες των αριθμών περιοχών.

Ορισμός 5.1: *(Περιοχή πετυχημένη, >) Η περιοχή m πετυχαίνει έναντι της n, ή m > n , εάν και μόνο αν RNum(m)[b] = 1, RNum(n)[b] = 0 και RNum(m)[i] = RNum(n)[i] για 1≤ β ≤ min(RNum(m).length, RNum(n).length), 1≤ i < b.*

Με τον ίδιο τρόπο εκφράζεται και η σχέση "<".

Ορισμός 5.2: *(Κάλυψη περιοχής ⊇) Η περιοχή m καλύπτει την n, ή η m είναι υπερσύνολο της n, ή m⊇n, εάν και μόνο αν RNum(m).length≤ RNum(n).length και RNum(m)[i] = RNum(n)[i] για 1≤ i ≤Rnum(m).length.*

Με τον ίδιο τρόπο εκφράζεται και η σχέση "⊆".

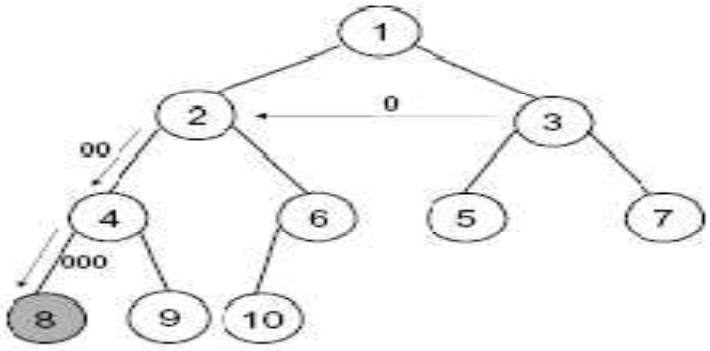
Για να δρομολογήσουμε μια αναζήτηση, ο κόμβος πρέπει πρώτα να διαχωρίσει την περιοχή του ζητούμενου στόχου υπολογίζοντας τον αριθμό περιοχής της βασιζόμενος στην ιστορία της διαίρεσης της περιοχής, και στην συνέχεια να περάσει την αναζήτηση σε έναν κόμβο η περιοχή του οποίου καλύπτεται ή βρίσκεται πιο κοντά στην διαχωρισμένη περιοχή. Ο αλγόριθμος 1 περιγράφει την διαδικασία υπολογισμού του αριθμού περιοχής του στόχου. Με την έκφραση «εκτίμηση», εννοούμε ότι μπορούμε μόνο να υπολογίσουμε τον αριθμό περιοχής ενός υπερσυνόλου του στόχου περιοχής. Η ακρίβεια βασίζεται στο πόσες φορές το ζητούμενο σημείο p πέφτει στο ίδιο εύρος με την τοπική περιοχή μετά τον χωρισμού της περιοχής δεδομένων (γραμμή 3-4). Ο υπολογισμός τερματίζει όταν το p πέσει έξω από το εύρος ιστορίας του τρέχοντος κόμβου (γραμμή 6-7).

Algorithm 1 estimate_num$(node\ n, target\ p, bit\ b)$

```
1: if (Region(p) ⊇ Region(n)) then
2:    for b to RNum(n).length do
3:       if (p[b%d] ∈ HistoryRange(n)[b]) then
4:          RNum(p)[b] ← RNum(n)[b]
5:       else
6:          RNum(p)[b] ← 1 − RNum(n)[b]
7:          break
```

αλγόριθμος 1

Υποθέτουμε ότι ο κόμβος 3 θέλει να εντοπίσει ένα σημείο στον κόμβο 8 όπως φαίνεται στην εικόνα 5.2, ο κόμβος 3 πρώτα υπολογίζει τους αριθμούς περιοχής του ως 0, όπως είναι στον κατώτερο τμήμα του πρώτου τμήματος στον άξονα x. Στην συνέχεια, ο κόμβος 3 δρομολογεί την αναζήτηση στον κοντινότερο από τα αριστερά κόμβο του, δηλαδή τον κόμβο 2, η περιοχή του οποίο διαχωρίζει την περιοχή που καλύπτει η περιοχή στόχος. Η ίδια διαδικασία εκτελείται στον κόμβο 2 και στον κόμβο 4 ως συνέπεια της διαδικασίας μέχρι η αναζήτηση να φτάσει στον κόμβο 8 όπως φαίνεται στην εικόνα 3. Ο μέγιστος αριθμός των hops δρομολόγησης είναι περίπου ο αριθμός των bit στον ακριβή αριθμό περιοχής στου στόχου που αναζητούμε. Για τον ομοιόμορφα κατανεμημένο φόρτο και σε ίσα τμήματα δεδομένων, το μέσο μήκος του μονοπατιού δρομολόγησης είναι O(logN).



εικόνα 5.3

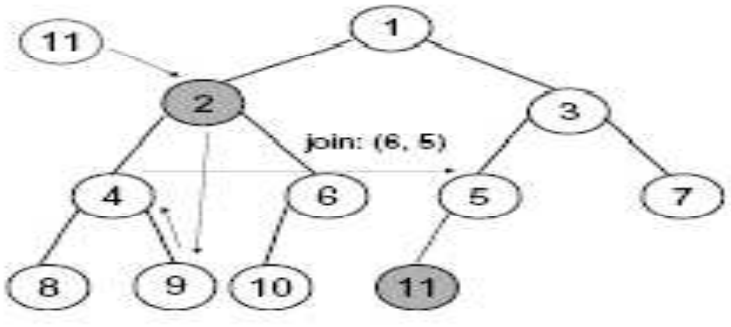
5.5 Query Load Balanced Partition

Χωρίζουμε το φόρτο του δικτύου σε ίσα τμήματα δεδομένων και επιλέγουμε καλύτερους υποψήφιους για να μοιράζονται τον φόρτο όταν συνδέονται/απομακρύνονται.

Αν εκχωρώ μικρότερο χώρο στις πιο "hot" περιοχές διευκολύνεται η ισορροπία του φόρτου του δικτύου. Έτσι, τμηματοποιούμε την περιοχή δεδομένων διαιρώντας μια πολύ-ορθογώνια περιοχή σε δυο ίσες πολύ-ορθογώνιες περιοχές με τον ίδιο φόρτο. Στην συνέχεια, συνδυάζω 2 κοντινές περιοχές (όπως τις περιοχές 2, 10 στην εικόνα 5.1) ή περιοχές παρακείμενων κόμβων στο διάστημα των δεδομένων. Ένα παράδειγμα τέτοιων τμηματοποιήσεων παρουσιάζεται στην εικόνα 5.1.

<u>Σύνδεση κόμβου:</u> αναζητούμε καλύτερη ισορροπία φόρτου αναζήτησης κατά την σύνδεση ενός κόμβου επιλέγοντας έναν κόμβο με μεγαλύτερο φόρτο ανάμεσα από κάποιους γνωστούς υποψήφιους κόμβους του ίδιου επιπέδου από το να επιλέξουμε άμεσα τον πρώτο υποψήφιο κόμβου που δεν έχει αρκετά παιδιά και έχει πλήρη τον πίνακα δρομολόγησης του. Οι υποψήφιοι επιλέγονται από τους γείτονες των πινάκων δρομολόγησης.

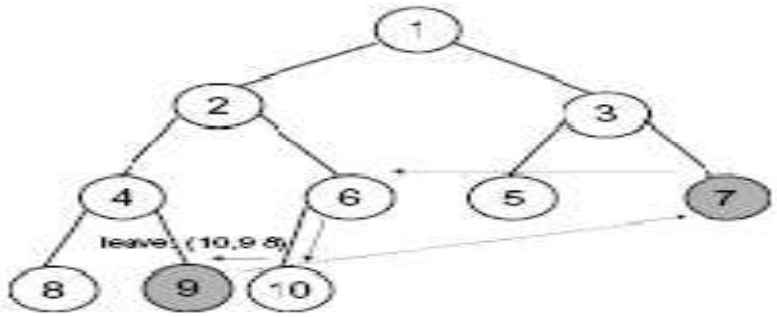
Για παράδειγμα, όταν ο κόμβος 4 δέχεται αίτημα σύνδεσης από τον κόμβο 11 όπως φαίνεται από τα βέλη της εικόνας 4, βρίσκει 2 υποψήφιους γείτονες: τον κόμβο 5 και τον κόμβο 6. Ας υποθέσουμε ότι ο κόμβος 5 έχει μεγαλύτερο φόρτο από τον φόρτο του κόμβου 6. Ο κόμβος 4 πρέπει να αποφασίσει και να περάσει τον νέο κόμβο στον κόμβο 5. Η διαδικασία επιλογής του υποψήφιου κόμβου προσθέτει σταθερό κόστος, κατά τέτοιο τρόπο ώστε το κόστος διατηρείται στο O(logN).



Εικόνα 5.4. Σύνδεση κόμβου

<u>Αποχώρηση κόμβου:</u> όταν ένας κόμβος αποχωρεί, αρχικά ζητά από κάποιο παρακείμενο κόμβο να καλύψει την περιοχή του. Εάν η κενή θέση προκλήθηκε από τον κόμβο που αποχώρησε δεν επισύρει αλλαγή στην ισορροπία του δένδρου και οι δύο περιοχές μπορούν να συνδυαστούν χωρίς να την είσοδο φόρτου, καμία επιπλέον ενέργεια δεν είναι απαραίτητη. Διαφορετικά, η διαδικασία εύρεσης του αντικαταστάτη στρέφεται σε έναν από τους παρακείμενους κόμβους του χαμηλότερου επιπέδου.

Για την αντικατάσταση, επιλέγουμε έναν κόμβο με ελαφρύτερο φόρτο η περιοχή του οποίου μπορεί να συνδυαστεί καλύτερα με την περιοχή κάποιο παρακείμενου κόμβου από την λίστα των γειτονικών κόμβων του ίδιου επιπέδου. Σαν παράδειγμα, όταν ο κόμβος 7 αποχωρεί όπως φαίνεται στην εικόνα 5, πρέπει να βρει αντικαταστάτη από το επίπεδο του κόμβου 10. Ανάμεσα στον κόμβο 10 και τους δυο γειτονικούς κόμβους 9 και 8, εάν ο κόμβος 9 έχει ελαφρύτερο φόρτο, θα είναι ο καλύτερος υποψήφιος για να αντικαταστήσει τον κόμβο, εκτός αυτού η περιοχή του μπορεί να συγχωνευτεί με την περιοχή του παρακείμενου κόμβου 4. Το κόστος αναζήτησης αντικατάστασης είναι O(logN) με σταθερό επιπλέον κόστος για την επιλογή του υποψήφιου.



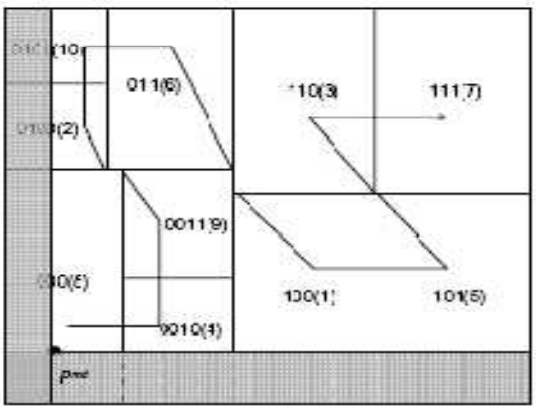
Εικόνα 5.5. Αποχώρηση κόμβου

5.6 Εκτέλεση αναζήτησης κορυφογραμμής:

Η αποδοτική αναζήτηση κορυφογραμμής στα p2p δίκτυα απαιτεί γρήγορη απάντηση και μικρό αριθμό εμπλεκόμενων κόμβων και μηνυμάτων αναζήτησης. Είναι φανερό ότι απλές προσεγγίσεις είτε με επακόλουθες αναζητήσεις στις περιοχές δεδομένων είτε με αναζήτηση πλημμύρας μέσω συνδεδεμένων κόμβων είναι ικανοποιητικές. Αποτελεσματικές στρατηγικές απαιτούνται για την αντιμετώπιση 3 σημαντικών προβλημάτων:

1. Παράλληλη αναζήτηση.
2. Απομάκρυνση άσχετων κόμβων.
3. Μείωση των ίδιων αναζητήσεων.

Χρησιμοποιούμε το SSP (Skyline Search Partitioning). Διαχωρίζει το διάστημα της αναζήτησης κορυφογραμμής για να λύσει το δεύτερο πρόβλημα, και στην συνέχεια τμηματοποιεί το χώρο αναζήτησης για να λύση το πρώτο και το τρίτο πρόβλημα.



Εικόνα 5.6. Διάστημα αναζήτησης κορυφογραμμής

5.6.1 ορισμός του χώρου αναζήτησης κορυφογραμμής:

Ξεκινάμε την αναζήτηση από τον κόμβο τα τοπικά αποτελέσματα του οποίου είναι εγγυημένα στην τελική κορυφογραμμή. Σημειώνουμε αυτόν τον κόμβο ως SQ-Starter. Ο κόμβος αυτός μπορεί να εντοπιστεί με την αναζήτηση του σημείου που κυριαρχεί όλων των υπολοίπων σημείων στο διάστημα των δεδομένων, όπως το σημείο (0,0) του κόμβου 8 της εικόνας 5.1.

Στην συνέχεια, σχηματίζουμε τον χώρο αναζήτησης της κορυφογραμμής. Πρακτικά, υπολογίζουμε τα τοπικά αποτελέσματα της κορυφογραμμής στον SQ-Starter και επιλέγουμε το πιο κυρίαρχο σημείο που έχει την μεγαλύτερη περιοχή κυριαρχίας. Θέτουμε αυτό το

σημείο να είναι το $p_{md}(a_1, a_{2,..,}a_d)$. Για να αποφύγουμε την απώλεια εύρους ζώνης του δικτύου, χρησιμοποιούμε μόνο το p_{md} για να απομακρύνουμε τους άσχετους κόμβους και να καθαρίσω ουσιαστικά το χώρο αναζήτησης . Έτσι καθορίζουμε το διάστημα αναζήτησης της κορυφογραμμής ως εξής.

Ορισμός 5.6.1 *Ο χώρος αναζήτησης της κορυφογραμμής είναι ένα σύνολο που αποτελείται από όλα τα σημεία που δεν κυριαρχούνται από το p_{md}. Είναι η ένωση d πολύ-ορθογωνίων εύρων αναζήτησης (SR), κάθε όριο του οποίου περιορίζεται μόνο από μια συντεταγμένη του p_{md} στον έναν άξονα, όπως {[0,1], …, [0,α_i),…,[0,1]}.*

Για το παράδειγμα αναζήτησης SQ1, το σκιαγραφημένο μέρος της εικόνας 5.6 δείχνει τον χώρο αναζήτησης κορυφογραμμής, που αποτελείται από τα SR = { [0,a_1), [0,1]} και SR = {[0,1], [0,a_2)}.

Ορισμός 5.6.2 *Ένας κόμβος κυριαρχείται και πρέπει να απομακρυνθεί, εάν και μόνο αν το μοναδικό ιδανικό σημείο με την καλύτερη τιμή σε κάθε διάσταση, το οποίο σημειώνεται ως p_{best}, κυριαρχείται από το p_{md}.*

Τώρα θα αποδείξουμε πως μπορούμε να βρούμε τις σωστές απαντήσεις με την επίσκεψη μας μόνο στους μη-κυρίαρχους κόμβους που βρίσκονται μέσα στον χώρο αναζήτησης κορυφογραμμής.

Λήμμα 5.6.1 *Όλοι οι κόμβοι που βρίσκονται έξω από χώρο αναζήτησης κορυφογραμμής είναι κόμβοι που κυριαρχούνται.*

Απόδειξη: Από την στιγμή που το διάστημα δεδομένων έξω από τον χώρο αναζήτησης της κορυφογραμμής όπως το $\{(\alpha_1, 1], (\alpha_2, 1], \ldots., (\alpha_d, 1]\}$ έχει τις χειρότερες τιμές σε όλες τις διαστάσεις σε σχέση με το p_{md} και το p_{best} οποιουδήποτε κόμβου του διαστήματος πρέπει να κυριαρχείται από το p_{md}. Επομένως, κάθε κόμβος σ' αυτό το διάστημα κυριαρχείται από το p_{md}.

Λήμμα 5.6.2 *Κάθε κόμβος που κυριαρχείται δεν μπορεί περιέχει σημεία κορυφογραμμής.*

Απόδειξη: Δοθέντος ενός κόμβου που κυριαρχείται, κάθε σημείο δεδομένων πρέπει να έχει τουλάχιστον μια κατώτερη τιμή σε σύγκριση με το p_{best}. Από την στιγμή που το p_{best} κυριαρχείται από το p_{md}, κάθε σημείο δεδομένων του κόμβου κυριαρχείται από το $p_{\square d}$, και δεν μπορεί να είναι σημείο κορυφογραμμής.

Λήμμα 5.6.3 *Οποιοδήποτε τελικό σημείο της κορυφογραμμής δεν μπορεί να κυριαρχείται από κανένα σημείο των κόμβων που κυριαρχούνται.*

Απόδειξη: Έστω SQ1 = { Min, Min }. Υποθέτουμε ένα σημείο κορυφογραμμής $p_s(s_1, s_2)$ κυριαρχείται από ένα σημείο $p_d(u_1, u_2)$ ενός συγκεκριμένου κόμβου που κυριαρχείται, οπότε έχουμε $s_1 \geq u_1, s_2 \geq u_2$. Σύμφωνα με τον ορισμό 5.6.2, το p_d κυριαρχείται από το p_{md}, $u_1 \geq a_1, u_2 \geq a_2$, οπότε έχουμε $s_1 \geq a_1, s_2 \geq a_2$, που σημαίνει ότι το p_s κυριαρχείται από το p_{md} και δεν μπορεί να εμφανίζεται σαν σημείο της τελικής κορυφογραμμής. Αυτό έρχεται σε αντίθεση με την υπόθεση, επομένως το p_sδεν μπορεί να κυριαρχείται από το p_d.

Θεώρημα 5.6.1 *Οι κόμβοι που δεν κυριαρχούνται στον χώρο δεδομένων της αναζήτησης κορυφογραμμής επιστρέφουν το ολοκληρωμένο σύνολο της κορυφογραμμής και μόνο το σύνολο αυτό.*

5.6.2 Βελτιστοποίηση χώρου αναζήτησης δεδομένων

Δοθέντος ενός χώρου αναζήτησης κορυφογραμμής που κατασκευάστηκε από πολύ-ορθογώνιες περιοχές αναζήτησης (SR), τον διαιρούμε σε ξεχωριστά υποδιαστήματα (subSR) αναζήτησης για παράλληλη αναζήτηση προώθησης μονοπατιού. Κόμβοι που βρίσκονται μέσα σε κάθε subSR προωθούν την αναζήτηση στους κόμβους οι περιοχές των οποίων καλύπτονται ή βρίσκονται κοντά στις άλυτες τμήμα του subSR. Επομένως, ένας κόμβος επισκέπτεται ξανά σπάνια εκτός αν χρειαστεί για την δρομολόγηση ενός subSR που δεν μπορεί άμεσα να φτάσει έναν καλυμμένο κόμβο.

Όσο για τον υπολογισμό του αριθμού περιοχής του στόχου αναζήτησης, τμηματοποιούμε ένα SR σε subSRs βασιζόμενοι στα εύρη που βρίσκονται αποθηκευμένα στην τοπική ιστορία του τμήματος. Με την ανάθεση ενός αριθμού περιοχής σε κάθε subSR, μπορούμε να συγκρίνουμε την θέση του με τις περιοχές των συνδεδεμένων κόμβων για να αποφασίσουμε για την προώθηση της αναζήτησης.

Ο αλγόριθμος 2 δείχνει την διαδικασία τμηματοποίησης του χώρου αναζήτησης και υπολογισμού του αριθμού περιοχής για τα τμηματοποιημένα υποδιαστήματα. Όμοια με τον αλγόριθμο 1, σκανάρουμε την ιστορία της διαίρεσης με την σειρά (γραμμή 1), υπολογίζουμε το επόμενο άγνωστο bit της περιοχής δεδομένων για το SR (γραμμή 3), και χωρίζω το SR σε 2 subSRs (γραμμή 4-8). Ο υπολογισμός σταματά μόλις ο κόμβος δεν μπορεί να χωρίσει

περισσότερο το SR. Αυτό συμβαίνει όταν το ενημερωμένο SR είναι εκτός του εύρους της ιστορίας του συγκεκριμένου κόμβου (γραμμή 10-11).

Algorithm 2 partition($node\ n, range\ SR$)

Define: $subSRSet$ disjoint sub search range set of SR

```
1:  for b from RNum(SR).length to RNum(n).length
    do
2:    if (SR ⊆ HistoryRange(n)[b]) then
3:      RNum(SR)[b] ← RNum(n)[b]
4:      if (HistoryRange(n)[b] is lower part) then
5:        UpperBound(SR)[b] ← SplitPos(n)[b]
6:      else
7:        LowerBound(SR)[b] ← SplitPos(n)[b]
8:      Add SR's buddy subSR to subSRSet
9:    else
10:     RNum(SR)[b] ← 1 − RNum(n)[b]
11:     break
```

Αλγόριθμος 2

Όσον αφορά τον χώρο αναζήτησης του SQ1, μετά την αφαίρεση του εύρους του SQ-Starter όπως φαίνεται στην εικόνα 5.7(a), αφαιρούμε το αριστερό τμήμα SR σε ένα μοναδικό SR εκτιμώντας το ως 01, γιατί είναι εκτός του εύρους ιστορίας της περιοχής 8 στο δεύτερο χρονικό τμήμα. Το δεξί τμήμα SR χωρίζεται σε 2 τμήματα στο πρώτο χρονικό τμήμα: το ένα επικαλύπτει την περιοχή 1 και στην περιοχή 5 ανατίθεται ένας αριθμός περιοχής 1, και το άλλο επικαλύπτει την περιοχή 4 εκτιμάται ως 001.

Στην συνέχεια, θεωρούμε ένα SR το όριο του οποίου περιορίζεται από την συντεταγμένη του p_{md} στην διάσταση i. Έστω μ ο αριθμός των bits στην περιοχή δεδομένων που είναι μακρύτερα από το SQ-Starter και επικαλύπτει το SR. Τότε, ο αριθμός των εγγραφών για την διάσταση i είναι περίπου |m/d|. Επομένως, ο αριθμός των τμημάτων subSR είναι |m/d|, και ο γνωστός αριθμός των bits στην περιοχή δεδομένων του subSR μειώνεται κατά d κάθε φορά όταν ένα subSR πέφτει έξω από το εύρος της ιστορίας του SQ-Starter. Από την στιγμή που ο μέγιστος αριθμός των hops για να λύσεις ένα subSR είναι ο αριθμός των bits της περιοχής της, ο μέγιστος αριθμός των hops για να εκτελέσεις ένα SR είναι η αριθμητική πρόοδος του προηγούμενου, $O(d + 2d + \cdots + \left\lfloor \frac{m}{d} \right\rfloor d)$ και ασυμπτωτικά

$O(\frac{m\left(1+\frac{\square}{d}\right)}{2})$. Στην ομοιόμορφη κατανομή του φόρτου, το εύρος αναζήτησης διαιρείται σε μισά, οπότε το κόστος εκτέλεσης ενός subSR πάντα μειώνεται στα μισά. Εφόσον το μέσο μήκος μονοπατιού για να εντοπίσουμε τη μακρύτερη περιοχή που το subSR επικαλύπτει είναι O(logN), ο μέσος όρος των βημάτων για την εκτέλεση ενός SR είναι $O\left(logN + \frac{1}{2} * logN + \cdots + \frac{1}{2}^{\frac{logN}{d}} logN\right)$, πιο συγκεκριμένα $O(2\left(1 - \frac{1}{\sqrt[d]{N}}\right) logN)$.

5.6.3 αλγόριθμος αναζήτησης κορυφογραμμής

Παρακάτω παρουσιάζεται ο αλγόριθμος αναζήτησης που είναι για τοπική εκτέλεση σε κάθε hop αφού το SQ-Starter καθορίσει το χώρο αναζήτησης της κορυφογραμμής.

Κάθε subSR προωθείτε σε έναν κόμβο γείτονα που δεν κυριαρχείται, παιδί ή παρακείμενο κόμβο η περιοχή του οποίου καλύπτεται από το subSR (γραμμή 7-11 του αλγόριθμου 3). Εάν ένας τέτοιος κόμβος δεν βρεθεί, το subSR προωθείτε στον μακρύτερο γείτονα στην περίπτωση που επιτύχει (προηγείται) τον δεξιότερο (αριστερότερο) γείτονα (γραμμή 13-14), ή απλά στον παρακείμενο κόμβο (γραμμή 16). Η αναζήτηση κορυφογραμμής σε καθένα από τα επόμενα hops (γραμμή 21) εκτελείται παράλληλα.

Για να απαντήσουμε στο SQ1, στο πρώτο βήμα (εικόνα 5.7(a)) προωθούμε το subSR που έχει αριθμό 001 στον κόμβο 4 που μπορεί πλήρως να τον αναλύσει, περνάμε το subSR 01 στον δεξί γειτονικό κόμβο 10 (γραμμή 9) και επίσης στέλνουμε το subSR 1 στον δεξιότερο κόμβο 10 για επιπλέον προώθηση (γραμμή 14). Στο δεύτερο βήμα που φαίνεται στην εικόνα 5.7(b), ο κόμβος 10 αφαιρεί το subSR 01 από τα τοπικά δεδομένα (γραμμή 3) και το τμήμα (γραμμή 4) του subSR 0100, στην συνέχεια το προωθεί στο αριστερό παρακείμενο κόμβο 2 που μπορεί να τον αναλύσει πλήρως, και περνά subSR 1 στον δεξί παρακείμενο κόμβο 6 (γραμμή 16). Η εικόνα 5.7(c) επιδεικνύει τον χώρο αναζήτησης της κορυφογραμμής μετά την τοπική εκτέλεση του κόμβου 2. Το subSR που απομένει προωθείται στην συνέχεια στον γειτονικό κόμβο 5 (γραμμή 9), που λύνει το τμήμα 101 και στέλνει το ενημερωμένο subSR στον αριστερό του παρακείμενο κόμβο 1. Από την εικόνα 5.7(d), βλέπουμε ότι ο κόμβος 1 είναι το τελευταίο hop για την εκτέλεση του subSR 1. Η εικόνα 5.7 και η εικόνα 5.8 επιδεικνύουν την διαδικασία αφαίρεση τμημάτων από τον χώρο αναζήτησης και την προώθηση της αναζήτησης αντίστοιχα.

Algorithm 3 search_skyline(*node* n, *query* SQ, *search_range* SR)

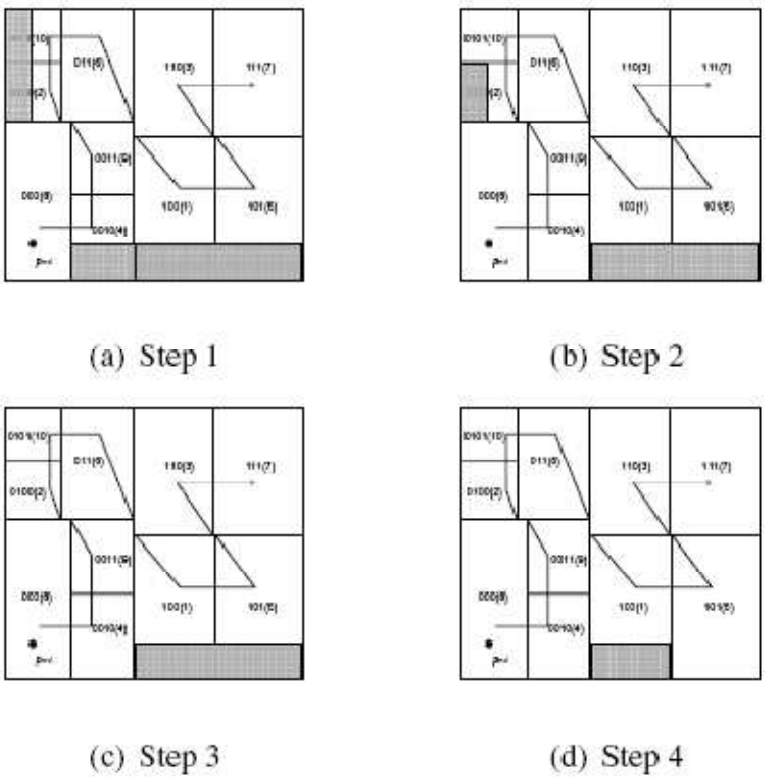
Define: RRT(n) right routing table of node n
Define: RChild(n) right child of node n
Define: RAdj(n) right adjacent of n
Define: $subSRSet$ disjoint search range set of SQ
Define: Dominated(m) if node m is dominated by p_{md}

1: **if** (Range(n) $\bigcap SR \neq \emptyset$) **then**
2: Compute local skyline not dominated by p_{md} and report to SQ-$Starter$
3: $SR \leftarrow$ Range(SR) − Range(n)
4: $subSRSet \leftarrow$ partition(n, SR)
5: **for all** sub-range $subSR$ **in** $subSRSet$ **do**
6: **if** (LowerBound($subSR$) > UpperBound(n)) **then**
7: $m \leftarrow$ NodeWhoseRegionCoveredBy (Region($subSR$)) in RRT(n)
8: **if** ((m not exist or Dominated(m)) and (RChild(n) not processed $subSR$) and (Region($subSR$) $\supseteq$ Region(RChild(n)))) **then**
9: $m \leftarrow$ RChild(n)
10: **if** ((m not exist || Dominated(m)) and (RAdj(n) not processed $subSR$) and (Region($subSR$) $\supseteq$ Region(RAdj(n)))) **then**
11: $m \leftarrow$ Radj(n)
12: **if** (m not exist || Dominated(m)) **then**
13: **if** (Region($subRQ$) $\succ$ Region(FarthestNodeInRRT(n))) **then**
14: $m \leftarrow$ FarthestNodeInRRT(n)
15: **else**
16: $m \leftarrow$ RAdj(n)
17: Map m to $subSR$ in $subSRSet$
18: **else**
19: // A similar process executes towards the left
20: **for all** ($m, subSR$) in $subSRSet$ **do**
21: search_skyline($m, SQ, subSR$)

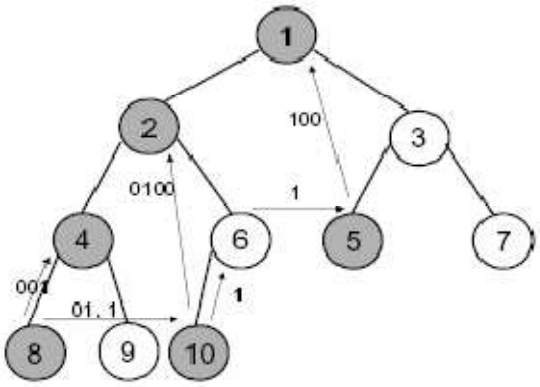
αλγόριθμος 3

Θεώρημα 5.6.2 *SSP απαντά μια αναζήτηση κορυφογραμμής σε* $O(\left(1+2d\left(1-\frac{1}{d\sqrt{N}}\right)\right)logN)$ *βήματα στην ομοιόμορφη κατανομή φόρτου ,* $O(\frac{m\left(1+\frac{m}{d}\right)}{2})$ *βήματα στην χειρότερη περίπτωση.*

Απόδειξη: η απόδειξη αυτή είναι γρήγορη δεδομένου των hops δρομολόγησης για τον εντοπισμό του SQ-Starter και την εκτέλεση ενός SR. Ο εντοπισμός του SQ-Starter παίρνει O(logN) hops για την ομοιόμορφη κατανομή φόρτου, O(m) στην χειρότερη περίπτωση. Η εκτέλεση κάθε SR απαιτεί O(2(1-1/$\sqrt[d]{N}$)logN) βήματα για την ομοιόμορφη κατανομή φόρτου, O(m(1+m/d)/2) βήματα στην χειρότερη περίπτωση. Αν προσθέσω τα δυο αυτά κόστη, παίρνουμε $O(\left(1+2d\left(1-\frac{1}{d\sqrt{N}}\right)\right)logN)$ για την ομοιόμορφη κατανομή φόρτου και $O(\left(1+\frac{m+d}{2}\right)m)$ στην χειρότερη περίπτωση.



Εικόνα 5.7. Εκτέλεση επίλυσης αναζήτησης κορυφογραμμής



Εικόνα 5.8. Αναζήτηση κορυφογραμμής

5.7 Query Load Balancing

Για να ισορροπήσουμε τον φόρτο αναζήτησης, τα παρακάτω δυο βήματα πρέπει να παρθούν. Πρώτα, το χώρος δεδομένων τμηματοποιείται σύμφωνα με τον φόρτο αναζήτησης, και η σύνδεση/αποχώρηση ενός κόμβου σχεδιάζεται ώστε να υπάρχει ισορροπία στον φόρτο. Δεύτερον, χρησιμοποιείται ένας μηχανισμός για δειγματοληψία και δυναμική ισορροπία φόρτου κατά την διάρκεια εκτέλεσης της αναζήτησης. Εφόσον το πρώτο βήμα περιγράφηκε στην ενότητα 3.4, θα παρουσιάσουμε μόνο το δεύτερο βήμα, ξεκινώντας από τον ορισμό του φόρτου αναζήτησης.

Ορισμός 5.7: *Ο φόρτος αναζήτησης ενός κόμβου είναι το άθροισμα του αριθμού της κορυφογραμμής που ανακτούμε από τα τοπικά αρχεία δεδομένων, και του αριθμού των μηνυμάτων που δρομολογούνται από τον κόμβο αλλά δεν οδηγούν σε τοπική αναζήτηση.*

Η διαδικασία εξισορρόπησης του φόρτου λειτουργεί ως εξής. Πρώτα, κάθε κόμβος ελέγχει αν υπάρχει ανισορροπία. Εάν υπάρχει, η διαδικασία μετακίνησης δεδομένων καλείται για να ισορροπήσει τον φόρτο.

Ο φόρτος αναζήτησης συγκεντρώνεται περιοδικά είτε από κάθε κόμβο από τους γειτονικούς κόμβους του πίνακα δρομολόγησης του και από παρακείμενους κόμβους είτε από την ανταλλαγή ειδοποιήσεων από αυτούς τους κόμβους. Η ανισορροπία καθορίζεται από την διαφορά δ του τοπικού κόμβου και του δείγματος του φόρτου αναζήτησης. Εάν το δ είναι μεγαλύτερο από ένα προκαθορισμένο όριο σ, μια ανισορροπία εντοπίζεται. Ωστόσο, το δείγμα φόρτου από τους διασυνδεδεμένους κόμβους μπορεί να μην αντικατοπτρίζει την καθολική κατανομή του φόρτου. Για να αντισταθμίσουμε την διαφορά, ξεκινάμε τυχαία δειγματοληψία όταν το πρώτο βήμα δεν εντοπίσει ανισορροπία και ακόμη ο τοπικός φόρτος είναι μεγαλύτερος από τον μέσο όρο του φόρτου που συγκεντρώσαμε. Ένας αριθμός από *logN* κόμβους που δεν είναι συνδεδεμένοι επιλέγονται με την αποστολή μηνυμάτων που έχουν επικολλήσει έναν μικρό μήκος hops δρομολόγησης και ακολουθούν ένα υπάρχον σύνδεσμο τυχαία σε κάθε hop. Το hop στο οποίο τερματίζεται το μήνυμα στέλνει πίσω το φόρτο του και το φόρτο δεδομένων που έχει συγκεντρώσει.

Για να ισορροπήσουμε τον φόρτο αναζήτησης, ένα φύλλο κόμβος βρίσκει ένα ελαφρύ από πλευράς φόρτου φύλλο κόμβο και το «αναγκάζει» να γίνει ένα από τα «παιδιά» του.

Βιβλιογραφία

- **Efficient Skyline Query Processing on Peer-to-Peer Networks**
- **W. T. Balke, U. G¨untzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.**
- **S. B¨orzs¨onyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.**

Κεφάλαιο 6

Υπολογισμός κορυφογραμμής σε δίκτυο αισθητήρων

Περιεχόμενα

6.1 Εισαγωγή

Τα ασύρματα δίκτυα αισθητήρων μπορούν να χρησιμοποιηθούν σε πολλά πεδία, π.χ. σε στρατιωτικές και αστικές εφαρμογές. Η τεχνική εξοικονόμησης ενέργειας για την επιμήκυνση της ζωής των αισθητήρων είναι μία από τις κύριες προκλήσεις για τα δίκτυα αισθητήρων με περιορισμό πόρων. Επομένως, έχει προταθεί η συνάθροιση δεδομένων μέσα στο δίκτυο (in-network) για την ενεργειακή απόδοση περιβαλλόντων με περιορισμό πόρων. Οι πιο πρόσφατες δουλειές στη συνάθροιση μέσα στο δικτύου υποστηρίζει μόνο μονοδιάστατα δεδομένα (π.χ., MIN και MAX). Για την υποστήριξη πολυδιάστατων δεδομένων χρησιμοποιείται το ερώτημα κορυφογραμμής. Το ερώτημα κορυφογραμμής επιστρέφει μία ομάδα δεδομένων που δεν κυριαρχούνται από κανένα άλλο σημείο σε όλες τις διαστάσεις. Η πλειοψηφία των προηγούμενων μεθόδων επεξεργασίας ερωτημάτων κορυφογραμμής (π.χ., BNL και BBS) λειτουργούν με συγκεντρωτική αποθήκευση. Οι μέθοδοι επεξεργασίας συγκεντρωτικών ερωτημάτων δεν έχουν ουσιαστικά λάθη όσον αφορά την ενεργειακή απόδοση σε περιβάλλοντα με υψηλό βαθμό συχνότητας συμβάντων. Παρακάτω θα παρουσιάσουμε έναν νέο αλγόριθμο για την επεξεργασία μέσα στο δίκτυο ερωτημάτων κορυφογραμμής. Ο προτεινόμενος αλγόριθμος μειώνει το κόστος επικοινωνίας και κατανέμει ίσα το φορτίο. Τα πειραματικά αποτελέσματα δείχνουν τα πλεονεκτήματα του αλγορίθμου μας σε συνάθροιση μέσα στο δίκτυο όσον αφορά την βελτίωση της ενεργειακής απόδοσης.

Οι πρόσφατες εξελίξεις στις τεχνολογίες επεξεργαστή, μνήμης, και ασύρματης επικοινωνίας έχουν οδηγήσει στη δημιουργία πολυ-λειτουργικών κόμβων χαμηλού κόστους και χαμηλής ενέργειας, που μπορούν να επικοινωνούν μεταξύ τους σε ασύρματες επικοινωνίες μικρής απόστασης χωρίς να είναι προσδεδεμένοι και έχουν σημαντική ικανότητα υπολογισμού και ανίχνευσης. Στο κοντινό μέλλον, θα αναπτυχθούν εκατοντάδες οικονομικοί κόμβοι για ένα πλήθος σκοπών. Για παράδειγμα, στον στρατιωτικό τομέα, τα δίκτυα αισθητήρων μπορούν να χρησιμοποιηθούν για την παρακολούθηση φιλικών δυνάμεων, εξοπλισμού και πυρομαχικών, για την επιβίωση στη μάχη και για πυρηνικές και βιολογικές επιθέσεις. Τα δίκτυα αισθητήρων μπορούν επίσης να εφαρμοστούν σε περιβαλλοντολογικές εφαρμογές που ανιχνεύουν πυρκαγιές σε δάση ή πλημμύρες και για την παρακολούθηση θαλάσσιων, εδαφικών και ατμοσφαιρικών περιεχομένων. Σε αυτά τα παραδείγματα, κάθε κόμβος αισθητήρα ανιχνεύει ένα ή περισσότερα φαινόμενα τη φορά. Οι πιο πολλές αναγνώσεις αισθητήρων που ονομάζονται από χαρακτηριστικά μπορούν να έχουν βαθμωτές τιμές (π.χ., θερμοκρασία, υγρασία, επίπεδο SO2). Τα χαρακτηριστικά αυτά

αποθηκεύονται στον εσωτερική τους μνήμη που ονομάζονται από ενδιάμεση μνήμη και μεταδίδονται στο σταθμό βάσης ως μία απάντηση στα ερωτήματα από το σταθμό βάσης.

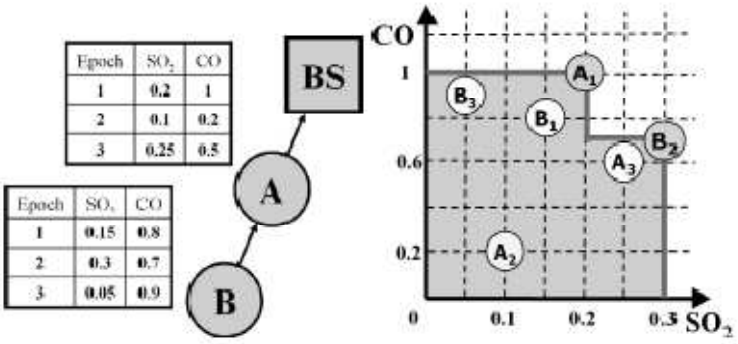
Στα δίκτυα αισθητήρων, εξακολουθούν να είναι κρίσιμα προβλήματα η περιορισμένη υπολογιστική ικανότητα, η περιορισμένη μπαταρία και το περιορισμένο εύρος ζώνης δικτύου των κόμβων αισθητήρων. Η τεχνική εξοικονόμηση ενέργειας για την επιμήκυνση τη ζωή των κόμβων αισθητήρων είναι μία από τις κύριες προκλήσεις για τα δίκτυα αισθητήρων με περιορισμό πόρων. Επομένως, έχει προταθεί η συνάθροιση δεδομένων μέσα στο δίκτυο για την ενεργειακή απόδοση των περιβαλλόντων με περιορισμό πόρων. Πολλές μέθοδοι συνάθροισης μέσα στο δίκτυο υποστηρίζουν συνάθροιση SQL (π.χ., MIN και MAX). Για παράδειγμα, "Πες μου την πιο ζεστή θέση για τις τελευταίες δύο ώρες." Ωστόσο, όταν υπάρχουν πάνω από δύο χαρακτηριστικά, δεν μπορούν να επηρεαστούν τα ενδιαφέροντα του χρήστη από την μέθοδο MIN/MAX κάθε χαρακτηριστικού. Το ερώτημα κορυφογραμμής υποκινείται από αυτό το πρόβλημα. Ως *Κορυφογραμμή* ορίζεται ένα σύνολο από σημεία που δεν κυριαρχούνται από κανένα άλλο σημείο. Ένα σημείο *κυριαρχεί* ένα άλλο σημείο εάν είναι το ίδιο καλό ή καλύτερο σε όλες τις διαστάσεις και καλύτερο τουλάχιστον σε μία διάσταση . Για απλότητα, υποθέτουμε ότι η κορυφογραμμή υπολογίζεται όσον αφορά τις συνθήκες MAX σε όλες τις διαστάσεις. Ωστόσο, ο προτεινόμενος αλγόριθμος μπορεί να εφαρμοστεί με οποιοδήποτε συνδυασμό συνθηκών με μία απλή τροποποίηση του αλγορίθμου. Το ερώτημα κορυφογραμμής είναι χρήσιμο για την υποστήριξη πολυκριτήριων αποφάσεων. Ένα υποκινούμενο παράδειγμα του ερωτήματος κορυφογραμμής σε δίκτυα αισθητήρων περιγράφεται παρακάτω.

6.2 Σενάριο

Κατασκευάζεται ένα νέο εργοστάσιο σε ένα προάστιο. Οι ερευνητές αναπτύσσουν αισθητήρες που συλλέγουν περιβαλλοντολογικά δεδομένα, CO και SO2, για να παρακολουθούν την μόλυνση του αέρα. Κάθε κόμβος αισθητήρα ανιχνεύει γεγονότα και αποθηκεύει τα αποτελέσματα στην δική του ενδιάμεση μνήμη κάθε 20 δευτερόλεπτα. Οι ερευνητές θα ήθελαν να ξέρουν την επίδραση του νέου κατασκευασμένου εργοστασίου.

Ερώτημα. *Αναφορά του χειρότερα επηρεασμένου χώρου όσον αφορά την μόλυνση του αέρα.*

Στο σενάριο αυτό, καθώς εκτελούνται περιοδικά τα ερωτήματα κορυφογραμμής (π.χ., κάθε 1 λεπτό), οι ερευνητές μπορούν να παρακολουθούν τον χειρότερα επηρεασμένο χώρο όσον αφορά την μόλυνση του αέρα. Οι ερευνητές αναζητούν έναν χώρο όπου ένας κόμβος αισθητήρα ανιχνεύει υψηλές τιμές CO και SO2. Και οι δύο τιμές CO και SO2 είναι σημαντικοί παράμετροι για τον καθορισμό του επιπέδου της μόλυνσης του αέρα. Μία απλή μέθοδος για **Ερώτημα** είναι η αποστολή όλων των μετρήσεων των αισθητήρων στο σταθμό βάσης. Το ερώτημα εκτελείται περιοδικά στο σταθμό βάσης. Στην Εικ. 6.1, οι κόμβοι αισθητήρων *A* και *B* μεταδίδουν τρεις πλειάδες που περιέχουν τιμές CO και SO2 (στην Εικ. 1, οι μετρήσεις των αισθητήρων είναι κανονικοποιημένες) αντίστοιχα, στο σταθμό βάσης. Καθώς δεν υπάρχει χώρος που να είναι καλύτερος και στις δύο συνθήκες (τιμές CO και SO2), δυστυχώς είναι δύσκολος ο καθορισμός του καλύτερου χώρου. Οι πλειάδες *A*1 και *B*2 που λαμβάνονται την εποχή 1 από τον αισθητήρα *A* και την εποχή 2 από τον αισθητήρα *B* αντίστοιχα, κυριαρχούν τις άλλες πλειάδες.



(a) Μετρήσεις αισθητήρα σε κάθε κόμβο αισθητήρα (b) Σχέση κυριαρχίας ανάμεσα στις μετρήσεις αισθητήρων

Εικ. 6.1 Σενάριο παραδείγματος.

Σημειώστε ότι η *A*1 κυριαρχεί την *B*2 όσον αφορά το επίπεδο CO αλλά η *B*2 κυριαρχεί της *A*1 όσον αφορά το επίπεδο SO2. Επομένως, οι δύο πλειάδες, είναι ασήμαντα γεγονότα όσον αφορά τον καθορισμό των τοποθεσιών ενδιαφέροντος. Αυτό φαίνεται στην Εικ. 6.1(b). Αυτά τα αποτελέσματα ερωτημάτων που ονομάζονται κορυφογραμμής είναι σημαντικά για την εξακρίβωση της σχέσης ανάμεσα στο καινούριο κατασκευασμένο εργοστάσιο και στον περιβάλλοντα χώρο του. Αν και έχουμε αναλογιστεί περιβάλλοντα δικτύου χωρίς απώλειες, τα δίκτυα αισθητήρων έχουν προδιάθεση στην απώλεια πακέτων. Πιστεύουμε ότι η «περιοδική» εκτέλεση ενός (απλού (ad-hoc) τελεστή κορυφογραμμής μπορεί να λύσει αυτό το σενάριο εργοστασιακής παρακολούθησης. Καθώς ο απλός (ad-hoc) τελεστής κορυφογραμμής μπορεί να υποστηρίζει την «περιοδική» εκτέλεση απλού (ad-hoc) τελεστή κορυφογραμμής, επικεντρωνόμαστε στην αποδοτική εκτέλεση ενός απλού (ad-hoc) τελεστή κορυφογραμμής.

Ωστόσο, η παραπάνω απλή μέθοδος περιλαμβάνει το μη απαραίτητο κόστος επικοινωνίας που προκαλείται από την μετάδοση μη πιστοποιημένων πλειάδων (π.χ., *A2, A3, B*1*, B*3). Στα δίκτυα αισθητήρων, παρατηρείται ότι οι κοντινά τοποθετημένοι αισθητήρες θα παίρνουν παρόμοια σήματα. Επομένως, έχουμε την ευκαιρία να εξαλείψουμε μη απαραίτητες επικοινωνίες που προκύπτουν από τη μετάδοση μη πιστοποιημένων αισθητήρων. Σε αυτή τη μελέτη, προτείνουμε μεθόδους επεξεργασίας μέσα στο δίκτυο για τα ερωτήματα κορυφογραμμής σε δίκτυα αισθητήρων. Οι προτεινόμενες μέθοδοι μπορούν να μειώσουν το κόστος επικοινωνίας. Οι συνεισφορές της μελέτης περιλαμβάνουν τα παρακάτω.

- Όσο γνωρίζουμε, πρόκειται για την πρώτη έρευνα για την επεξεργασία ερωτημάτων κορυφογραμμής σε δίκτυα αισθητήρων.
- Ορίζεται ο ρόλος της κεφαλής συστάδας (cluster) και του μέλους συστάδας για την κατανομή και μείωση των φορτίων.
- Προτείνονται μέθοδοι φιλτραρίσματος στη συστάδα της ενεργειακής απόδοσης στη συστάδα για την ελάττωση των δεδομένων που μεταφέρονται από τους αισθητήρες στο σταθμό βάσης.
- Τα πειραματικά αποτελέσματα υποδεικνύουν την ενεργειακή απόδοση των προτεινόμενων μεθόδων.

6.3 Επεξεργασία Μέσα στο Δίκτυο Ερωτημάτων Κορυφογραμμής σε Δίκτυα Αισθητήρων

6.3.1 Μοντέλο Συστήματος

Σε αυτή την εργασία, θεωρούμαι το ακόλουθο σύστημα δικτύου αισθητήρων:

- Θεωρούμαι ότι οι n αισθητήρες $S = \{S1, S2,..., Sn\}$ είναι τυχαία τοποθετημένοι σε μία ευρεία περιοχή, όπου n είναι το συνολικό πλήθος των κόμβων αισθητήρων και i ($1 \leq i \leq n$) είναι το αναγνωριστικό κόμβου.

- Κάθε κόμβος αισθητήρα διαθέτει το ίδιο κυκλικό ασύρματο εύρος. Ένας κόμβος αισθητήρα μπορεί να ελέγχει το ενεργειακό επίπεδο για τη μετάδοση των δεδομένων σύμφωνα με την απόσταση ανάμεσα στον αποστολέα και τον δέκτη. Δύο κόμβοι μπορούν να επικοινωνούν χρησιμοποιώντας το ίδιο ενεργειακό επίπεδο μετάδοσης. Για την επικοινωνία ανάμεσα σε έναν κόμβο αισθητήρα και στους άλλους κόμβους αισθητήρων που είναι εκτός από το εύρος του, ο αποστολέας χρησιμοποιεί ένα παράδειγμα επικοινωνίας πολλαπλών αναπηδήσεων, όπου οι ενδιάμεσοι κόμβοι σχετίζουν δεδομένα για τους γείτονες τους.

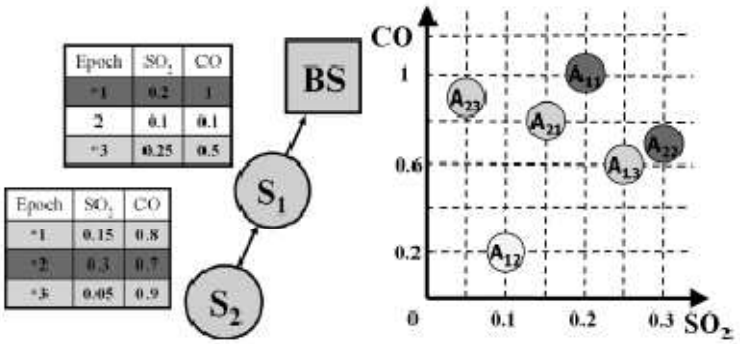
- Το Si αποθηκεύει ένα σύνολο δεδομένων που ανιχνεύτηκαν κατά τη διάρκεια της εποχής l, $Ai = \{Ai1, Ai2,..., Ail\}$, στην ενδιάμεση μνήμη. Η ανίχνευση γεγονότος περιγράφεται ως μία πλειάδα από τιμές χαρακτηριστικών $Ait = \{p1, p2,...,pm\}$, t ($1 \leq t \leq l$) που αντιπροσωπεύει την εποχή που ανιχνεύεται το γεγονός και κάθε χαρακτηριστικό pj($1 \leq j \leq m$) αντιπροσωπεύει την ένδειξη ενός αισθητήρα, ή ορισμένες τιμές που ανταποκρίνονται στην ανίχνευση.

- Ένα ερώτημα κορυφογραμμής SQ εκδίδεται από το σταθμό βάσης BS. Κάθε Si μπορεί να εκτελεί ένα χειριστή κορυφογραμμής που είναι ένα λογικό πρόγραμμα, και στέλνει πίσω το αποτέλεσμα του (π.χ., τα τοπικά αποτελέσματα κορυφογραμμής LSi) στο BS. Το BS συλλέγει LSi και επεξεργάζεται το τελικό ερώτημα κορυφογραμμής.

6.3.2 Περιγραφή Προβλήματος

Οι μέθοδοι κεντρικοποιημένης διαδικασίας ερωτήματος δεν έχουν λάθη σε σχέση με την ενεργειακή απόδοση όταν ανιχνεύονται συχνά γεγονότα. Η συνάθροιση μέσα στο δίκτυο μπορεί να απασχοληθεί για τη μείωση των δεδομένων που μεταφέρονται στο σταθμό βάσης. Σύμφωνα με το δέντρο ρουτίνας, αποφασίζεται ένα χρονοδιάγραμμα. Το χρονικό διάστημα επιλέγεται έτσι ώστε να υπάρχει αρκετός χρόνος για τον γονέα να συνδυάσει τα αποτελέσματα από το παιδί και να εκπέμπει το δικό του αποτέλεσμα στον γονέα του . Το *Si* λαμβάνει όλα τα τοπικά αποτελέσματα που αποστέλλονται δια μέσω των κόμβων παιδιών κατά τη διάρκεια του διαιρεμένου χρονικού διαστήματος. Το *Si* πραγματοποιεί τη συνάθροιση των τοπικών αποτελεσμάτων από διαφορετικούς αισθητήρες μαζί με το *LSi*, πριν την αναφορά του *LSi*. Το *Si* μεταδίδει το αποτέλεσμα συνάθροισης ως *LSi*.

Στην Εικ. 6.2, παρουσιάζεται ένα παράδειγμα από μία συνάθροιση μέσα στο δίκτυο για μία ερώτηση κορυφογραμμής. Τα *S*1 και *S*2 έχουν *LS*1{*A*11, *A*13} και *LS*2{*A*21, *A*22, *A*23} (στην Εικ. 6.2(a), η σημείωση ∗ αντιπροσωπεύει το σημείο τοπικής κορυφογραμμής) αντίστοιχα. Το *S*1 συναθροίζει τα δεδομένα στα *LS*1 και *LS*2. Εφόσον τα {*A*13} και {*A*21, *A*23} κυριαρχούνται αντίστοιχα από τα *A*22 και *A*11, *A*13, *A*21 και *A*23 κλαδεύονται. Το *S*1 δε χρειάζεται να αποστείλει αυτές τις μη πιστοποιημένες πλειάδες στο σταθμό βάσης. Επομένως, το *S*1 μειώνει το κόστος επικοινωνίας από τη συνάθροιση μέσα στο δίκτυο.



(a) Αναγνώσεις αισθητήρων και κόστος επικοινωνίας (b) Σχέση κυριαρχίας ανάμεσα στις αναγνώσεις αισθητήρων

Εικ. 6.2 Παράδειγμα από συνάθροιση μέσα στο δίκτυο.

Ωστόσο, είναι ακόμη μη απαραίτητη ροή δεδομένων (π.χ., το *S*2 μεταδίδει τα *A*21 και *A*23 στο *S*1). Παρατηρούμε ότι οι κοντινά τοποθετημένοι αισθητήρες τείνουν να λαμβάνουν

αναλογικά σήματα στα δίκτυα αισθητήρων. Η παρατήρηση αυτή δίνει την ευκαιρία για την περαιτέρω μείωση του συνόλου δεδομένων που μεταδίδεται στο *BS*. Σε αυτό το άκρο, χρησιμοποιούμε μία νέα μέθοδο φιλτραρίσματος για την περαιτέρω επιλογή μη-πιστοποιημένων πλειάδων.

6.3.3 MFT (Πλειάδα Φιλτραρίσματος Ελάχιστου-Σκορ)

Σε αυτή την ενότητα, προτείνουμε μία μέθοδο φιλτραρίσματος χρησιμοποιώντας ένα *MFT* για μία αποδοτική διαδικασία ερωτήματος κορυφογραμμής μέσα στο δίκτυο. Αν και η τοπική διαδικασία ερωτήματος κορυφογραμμής και η συνάθροιση μέσα στο δίκτυο μπορεί να μειώσει το πλήθος των δεδομένων που στέλνονται πίσω στο *BS*, ο κόμβος αισθητήρα μπορεί ακόμη να μεταφέρει πλειάδες που δεν ανήκουν στην τελική κορυφογραμμή.

Στο *LSi*, μπορεί να υπάρχει ένα *Ait* που μπορεί ενδεχομένως να «κλαδέψει» μη-πιστοποιημένες πλειάδες για την τελική κορυφογραμμή. Το *Ait* μπορεί να είναι ικανό να αποτρέψει μία ή παραπάνω μη-πιστοποιημένες πλειάδες στο *LSj* από το μεταφερθούν τοπικά.

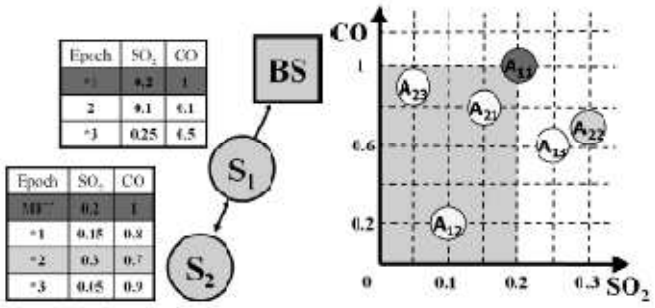
Είναι γνωστό από προγενέστερες μελέτες ότι η κοντινότερη πλειάδα από το αρχικό σημείο (0,...,0) υπάρχει στην κορυφογραμμή. Δανείζουμε αυτή την ενδιαφέρουσα ιδιοκτησία της κορυφογραμμής στην επιλογή του *MFT*. Εάν υιοθετήσουμε τον υπάρχον αλγόριθμο κορυφογραμμής SFS για την τοπική επεξεργασία ερωτήματος κορυφογραμμής, μπορούμε να πάρουμε εύκολα την κοντινότερη πλειάδα από το αρχικό σημείο. Πριν την επεξεργασία του τοπικού ερωτήματος κορυφογραμμής, αρχικά ένας αισθητήρας ταξινομεί τις ενδείξεις αισθητήρα στην ενδιάμεση μνήμη του σύμφωνα με την αύξουσα σειρά της τιμές $\sum_{j=1}^{m} \frac{1}{p_{ij}}$. Επιλέγουμε το *MFT* με βάση τις ακόλουθες τρεις συνθήκες.

1. Η πρώτη εμφανιζόμενη πλειάδα επιλέγεται ως *MFT* μετά το τέλος της ταξινόμησης σύμφωνα με την τιμή $\sum_{j=1}^{m} \frac{1}{p_{ij}}$, π.χ., η πλειάδα που έχει την ελάχιστη τιμή min($\sum_{j=1}^{m} \frac{1}{p_{ij}}$) ανάμεσα σε πλειάδες του *Ai*.

1. Εάν υπάρχουν πάνω από δύο πλειάδες που έχουν ελάχιστη τιμή min($\sum_{j=1}^{m} \frac{1}{p_{ij}}$), η πλειάδα με την μέγιστη τιμή max($\prod_{j=1}^{m} p_{ij}$) εμφανίζεται ως *MFT*, π.χ., η πλειάδα που έχει τη μέγιστη τιμή max($\prod_{j=1}^{m} p_{ij}$) ανάμεσα στις πλειάδες που ικανοποιούν την παραπάνω συνθήκη.

2. Εάν υπάρχουν παραπάνω από δύο πλειάδες που ικανοποιούν τις παραπάνω συνθήκες, η πρώτη αισθητήρια πλειάδα ανάμεσα τους επιλέγεται ως *MFT*.

Το Si λαμβάνει *SQ* με το *MFT* από τον προσδιορισμένο γονέα routing. Σύμφωνα με τις παραπάνω συνθήκες επιλογής, το *Si* επιλέγει



(a) Κόστος επικοινωνίας με *MFT* (b) Εφφέ *MFT*

Εικ. 6.3 Παράδειγμα *MFT*.

το MFTi ανάμεσα σε πλειάδες, όπου το ***i*** σημαίνει τον αναγνωριστή κόμβου. Το *MFT* ανανεώνεται ως μία πλειάδα που ικανοποιεί τις παραπάνω συνθήκες επιλογής ανάμεσα στο *MFT* και στο *MFTi*. Το *Si* αναμεταδίδει το *SQ* με το *MFT*. Επομένως, αυξάνεται η ικανότητα «κλαδέματος» του *MFT*.

Για παράδειγμα, στην Εικ. 6.3, όταν το *S*1 λαμβάνει *SQ* από το *BS*, το *S*1 συλλέγει *MFT*1 από τα *A*1{*A*11, *A*12, *A*13}. Αρχικά, το *S*1 ταξινομεί το *A*1 σε αύξουσα σειρά της τιμής $\sum_{j=1}^{2} \frac{1}{p_j}$ value. Το *S*1 βρίσκει υποψήφιες πλειάδες του *MFT*1 σύμφωνα με την πρώτη συνθήκη επιλογής. Τα *A*11 και *A*13 έχουν την ελάχιστη τιμή, $\sum_{j=1}^{2} \frac{1}{p_j}=6.$ Τότε το *A*11 επιλέγεται ως *MFT*1 σύμφωνα με την δεύτερη συνθήκη επιλογής, καθώς το *A*11 έχει $\prod_{j=1}^{2} p_j=0.2$ και το *A*13 έχει $\prod_{j=1}^{2} p_j=0.125.$. Το *S*1 αποφασίζει το *MFT* ως *MFT*1. Το *S*1 διαβιβάζει *SQ* με *MFT* στο *S*2. Το *S*2 βρίσκει το *LS*2 και στη συνέχεια αποστέλλει το *S*1 στο *LS*2 που φιλτράρονται προς το *MFT*. Το *MFT* κυριαρχεί τα *A*21 και *A*23 στο *LS*2. Το *S*2 αποστέλλει *LS*2{*A*22} αντί για το *LS*2{*A*21, *A*22, *A*23} στο *S*1. Το *S*2 μπορεί να δώσει κέρδος σε απλό έλεγχο κυριαρχίας.

6.3.4 MFTAC (MFT-Μέθοδος Εφαρμοσμένης Συνάθροισης με Βάση τη Συστοιχία (Applied Aggregation Method Based on Cluster))

Η στρατηγική φιλτραρίσματος αναμένεται να μειώσει δραστικά την μη αναγκαία επικοινωνία. Ωστόσο, η στρατηγική φιλτραρίσματος προσθέτει επιπλέον κόστος επικοινωνίας για τη διαβίβαση του *MFT*. Για την ελαχιστοποίηση του επιπλέον κόστους επικοινωνίας για τη διαβίβαση του *MFT*, εκμισθώνουμε τους αντιπροσώπους που είναι υπεύθυνοι για την επιλογή και διαβίβαση του *MFT*. Η εκλογή του αντιπροσώπου βασίζεται στη συσχέτιση ανάμεσα στις τιμές δεδομένων από κοντινά τοποθετημένους κόμβους αισθητήρων. Επομένως, εφαρμόζουμε τον αντιπρόσωπο στη μέθοδο συνάθροισης με φιλτράρισμα. Θα ονομάσουμε αυτή τη μέθοδο MFT-εφαρμοσμένη μέθοδο συνάθροισης με βάση τη συστοιχία (*MFTAC*).

Για τη διευκόλυνση της διαδικασίας ερωτήματος κορυφογραμμής και για την απλούστευση του προβλήματος αυτού, υιοθετούμε ένα ευρέως γνωστό σχήμα συστοιχίας που ονομάζεται HEED , που υποκινείται από τη συσχέτιση ανάμεσα στις ενδείξεις αισθητήρα και στην τοποθεσία αισθητήρα. Το HEED δημιουργήθηκε για την εκλογή κεφαλών συστοιχιών και για την διατήρηση συστοιχιών. Η κεφαλή συστοιχίας εκλέγεται σύμφωνα με ένα υβρίδιο την υπολειμματική ενέργεια κόμβου και την εγγύτητα του κόμβου με τους γείτονες του. Σε αυτή την εργασία, δανειζόμαστε τη δημιουργία συστοιχίας και τη συντήρηση αλγορίθμου από την προσέγγιση HEED.

Για την αποδοτική κατανομή φορτίου και για την ελαχιστοποίηση του επιπλέον κόστους για τη διαβίβαση του *MFT*, ορίζουμε επιπλέον ρόλους μίας κεφαλής συστοιχίας και ενός μέλους συστοιχίας .

Αλγόριθμος 1 Επεξεργασία τοπικού ερωτήματος στο *Chi* ***skyline_cluster_header*( )**

Είσοδος: *SQ* και *MFT*

Έξοδος: *LSi* και *MFTi*

έναρξη

01: Αύξουσα ταξινόμηση του *Ai* κατά $\sum_{j=1}^{m} \frac{1}{p_j}$

02: *LSi* = Ø; *skip* = false;

03: *temp* = 0; // το *temp* αποθηκεύει το προσωρινό μέγιστο $\prod_{j=1}^{m} p_j$

04: **για (for)** κάθε πλειάδα *mtp* του *Ait* που έχει ελάχιστο min$(\sum_{j=1}^{m} \frac{1}{p_j})$

05: **εάν (if)** (*temp* < $\prod_{j=1}^{m} p_j$ του mtp)

06: *temp* = $\prod_{j=1}^{m} p_j$;

07: *MFTi* = *mtp*;

08: if $\{(\sum_{j=1}^{m} \frac{1}{p_j}$ of $MFT > \sum_{j=1}^{m} \frac{1}{p_j}$ of $MFT_i)$
or $(\sum_{j=1}^{m} \frac{1}{p_j}$ of $MFT == \sum_{j=1}^{m} \frac{1}{p_j}$ of MFT_i
and $\prod_{j=1}^{m} p_j$ of $MFT < \prod_{j=1}^{m} p_j$ of $MFT_i)\}$

09: *MFTi* = *MFT*;

10: αποστολή *SQ* με *MFTi*;

11: **για κάθε (for** each) πλειάδα *tp* του *Ai*

12: **για κάθε (for** each) πλειάδα *stp* του *LSi*

13: **εάν (if)** (το *tp* κυριαρχείται από *stp* ή *MFT*)

14: *skip* = true; break;

15: **εάν (if)** {(το skip δεν είναι αληθές) ή (το *LSi* είναι κενό)} εισαγωγή *tp* στο *LSi*;

16: Συλλογή των τοπικών αποτελεσμάτων από τα μέλη της συστοιχίας και των προσδιορισμένων μελών μέχρι τη λήξη του χρονικού ορίου (timeout)

17: ***συνάθροιση***(συλλεγμένα δεδομένα και *LSi*, *MFT*);

τέλος (end)

***aggregation*( )**

Είσοδος: συλλεγμένα δεδομένα και *LSi* , *MFT*

Έξοδος: αποτελέσματα συνάθροισης *RA*

έναρξη (begin)

01: *T* είναι ένα σύνολο από πλειάδες στα συλλεγμένα δεδομένα και στο *LSi* ;

02: *RA* = Ø; *skip* = false;

03: Αύξουσα ταξινόμηση του *T* κατά $\sum_{j=1}^{m} \frac{1}{p_j}$;

04: **για** κάθε (**for** each) πλειάδα *tp* του *T*

05: **για** κάθε (**for** each) *stp* του *RA*

06: **εάν** (**if)**(το *tp* κυριαρχείται από *stp* ή *MFT*)

07: *skip* = true; break;

08: **if** {(το *skip* δεν είναι αληθές) ή (το *RA* είναι κενό)} εισαγωγή *tp* στο *RA*; **τέλος (end)**

Οι ρόλοι μίας κεφαλής συστάδας και ενός μέλους συστάδας ορίζεται ως εξής.

- Μία κεφαλή συστάδας *CHi* ανανεώνει το *MFT* και αποστέλλει *SQ* με ανανεωμένα *MFT* στα μέλη της συστάδας της καταχωρημένης ανταποκρινόμενης συστάδας και των προσδιορισμένων κεφαλών συστάδας.

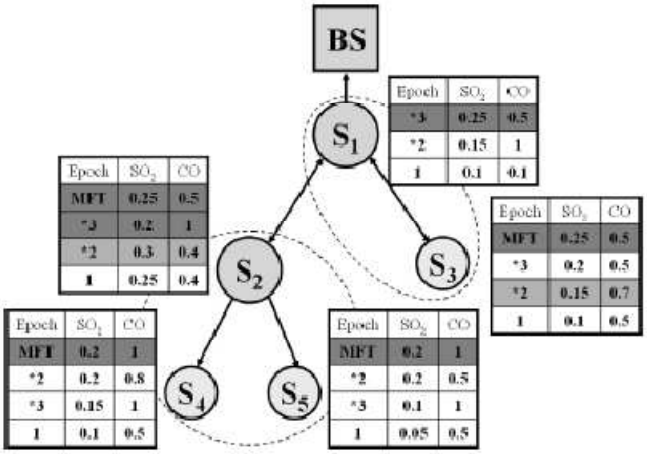
- Ένα μέλος συστάδας *CMk*, *k* είναι ένας αναγνωριστής κόμβου, τοπική επεξεργασία ερωτημάτων κορυφογραμμής και μεταφορά των φιλτραρισμένων με *MFT LSk* αντί για όλα τα *LSk* σε μία κεφαλή συστάδας της ανταποκρινόμενης συστάδας.

- Αφού το *CHi* λαμβάνει τοπικά αποτελέσματα από όλα τα μέλη συστάδας της ανταποκρινόμενης συστάδας και των προσδιορισμένων κεφαλών συστάδας κατά τη διάρκεια της προκαθορισμένης λήξης του χρονικού ορίου[1], το *CHi* υπολογίζει τη συνάθροιση των συλλεγμένων αντικειμένων μαζί με το *LSi*.

- Το *CHi* αποστέλλει αποτελέσματα πίσω στο *BS* μαζί με το μονοπάτι ρουτίνας.

[1] Το σύστημα TinyDB [10] χρησιμοποιεί αυτή την προσέγγιση για όλα τα ερωτήματα (όχι μόνο για ερωτήματα συνάθροισης).

Ως αποτέλεσμα, εάν και *CHi* προσθέτει επιπλέον κόστος για την επιλογή του *MFTi*, την ανανέωση του *MFT* και τη μεταφορά του *SQ* με *MFT*, το *CHi* μπορεί να μειώσει το μη απαραίτητο κόστος λήψης δεδομένων. Το *CMj* μπορεί να μειώσει το κόστος μετάδοσης των μη-πιστοποιημένων πλειάδων και το επιπλέον κόστος για την επιλογή και μετάδοση του *MFT*.



Εικ. 6.4 Παράδειγμα *MFTAC*.

6.3.5 Αλγόριθμος Κορυφογραμμής σε Δίκτυα Αισθητήρων

Σε αυτή την ενότητα, εξηγούμε το *MFTAC* με ένα παράδειγμα. Ο **Αλγόριθμος 1** παρουσιάζει τον ψευδοκώδικα για την απόκτηση ενός τοπικού ερωτήματος κορυφογραμμής στο *CH*. Ένας αλγόριθμος στο *CM* παραλείπεται, καθώς το *CM* πραγματοποιεί μόνο τα βήματα 1, 11, 12, 13, 14, και 15 στο *skyline_cluster_header*( ).

Για παράδειγμα, την Εικ. 6.4, αναπτύσσονται μόνο πέντε αισθητήρες. Υπάρχουν δύο συστάδες, που εμφανίζονται με τους διακεκομμένους κύκλους. Μία συστάδα αποτελείται από μία κεφαλή συστάδας, που αντιπροσωπεύεται από έναν μεγάλο συμπαγή κύκλο, και από μέλη συστάδας, που αντιπροσωπεύονται ως μικροί συμπαγείς κύκλοι. Όταν το *CH* εκπέμπει *SQ* κατά μήκος του μονοπατιού ρουτίνας, αποφασίζεται ένα χρονοδιάγραμμα. Το *SQ* εκδίδεται από *BS*. Αφού λάβει το *S*1 *SQ*, το *S*1 επιλέγει το *A*13 ως *MFT*1. Το *S*1 αποστέλλει *SQ* μαζί με *MFT*1 στα *S*2 και *S*3. Στη συνέχεια το *S*1 πραγματοποιεί τοπική επεξεργασία ερωτημάτων κορυφογραμμής, που είναι τα βήματα από 11 έως 15 στο *skyline_cluster_header*( ). Το *S*2 ανανεώνει το *MFT*2 ως *A*23. Το *SQ* μαζί με *MFT*2 αποστέλλεται στα *S*4 και *S*5. Τα *S*4 και *S*5 πραγματοποιούν τοπική επεξεργασία ερωτημάτων κορυφογραμμής, που είναι τα βήματα 1, 11, 12, 13, 14 και 15 in *skyline_cluster_header*( ). Τα *S*4 και *S*5 χρειάζεται να στείλουν *LS*4 είτε *LS*5 στο *S*2 μέχρι τη λήξη του χρονικού ορίου, καθώς το *MFT*2 κυριαρχεί σε όλα τα στοιχεία των *LS*4 και *LS*5, που είναι συσχετισμένο με το βήμα 16 στο *skylinecluster_header*( ). Το *S*2 μεταφέρει *LS*2{*A*22, *A*23} στο *S*1. Το *S*3 αποστέλλει μόνο *A*32, καθώς ένα άλλο στοιχείο του *LS*3, το *A*33, αποκλείεται από το *MFT*1. Το *S*1 φιλτράρει προς τα έξω τα *A*12 και *A*32 με την μέθοδο συνάθροισης, που είναι το βήμα 17 στο *skyline_cluster_header*( ).

Επομένως, το *S*1 αποστέλλει στο *BS* μόνο τα *A*13, *A*22, και στη συνέχεια τα *A*23 αντί για όλες τις πλειάδες.

Βιβλιογραφία

- **In-Network Processing for Skyline Queries in Sensor Networks**
- **I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," Computer and Telecommunications Networking, vol.38, no.4, pp.393–422, 2002.**
- **S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," Proc. International Conference on Data Engineering, pp.421–430,**