ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Τμήμα Πληροφορικής και Επικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση συστάδων με τον αλγόριθμο STING για εφαρμογές spatial data mining από συστήματα χωρικών δεδομένων

ΛΙΑΚΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΕΜ 490

Επιβλέπων: Κόκκινος Ιωάννης

Σέρρες, 2011

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Το θέμα της παρούσης πτυχιακής εργασίας είναι **Ανίχνευση συστάδων με τον αλγόριθμο STING για εφαρμογές spatial data mining από συστήματα χωρικών δεδομένων**.

Η εξόρυξη χωρικής γνώσης (spatial data mining) από βάσεις χωρικών δεδομένων αφορά εφαρμογές όπως τα Γεωγραφικά Συστήματα πληροφοριών αλλά και τα συστήματα βιοϊατρικής διάγνωσης και επεξεργασίας χωρικών συσχετίσεων ιατρικών εικόνων ή δεδομένων τηλεπισκόπησης, που απαιτούν γνωρίσματα θέσης στο χώρο. Για την αποδοτική εκτέλεση χωρικών ερωτημάτων είναι χρήσιμο τα γειτονικά (χωρικά) αντικείμενα να ομαδοποιούνται σε συστάδες στο δίσκο. Ο γεωγραφικός χώρος μπορεί να διαμεριστεί σε κελιά (cells) βάσει της εγγύτητας, οπότε αυτά τα κελιά αντιστοιχούν σε φυσικές θέσεις – blocks στο δίσκο. Η Συσταδοποίηση (clustering) σε μεγάλες βάσεις δεδομένων είναι η τεχνική εξόρυξης γνώσης από δεδομένα (data mining) από τις ιδιότητες των οποίων θα πρέπει να προσδιοριστούν συστάδες (ομάδες ή συμπλέγματα ή συγκροτήματα ή κλάσεις) για περαιτέρω κατάταξη. Οι συστάδες δεν είναι προκαθορισμένες.

Ο αλγόριθμος Statistical Information Grid-based (STING) χρησιμοποιεί μία ιεραρχική τεχνική διαίρεσης των χωρικών περιοχών σε ορθογώνια κελιά παρόμοια με ένα τετραδικό δένδρο. Η βάση χωρικών δεδομένων σαρώνεται μία φορά και για κάθε κελί καθορίζονται στατιστικές παράμετροι (μέση τιμή, διασπορά, μέγιστο, ελάχιστο, τύπος κατανομής). Έτσι μπορούν να απαντηθούν πολλά ερωτήματα εξόρυξης γνώσης από χωρικά δεδομένα, συμπεριλαμβανομένης και της συσταδοποίησης. Μια διάσχιση κατά πλάτος χρησιμοποιείται για την εξέταση του κατασκευασμένου δένδρου και εξετάζονται παιδιά σχετικών κόμβων. Ο υπολογισμός της πιθανότητας ένα κελί να είναι σχετικό με μία ερώτηση βασίζεται στο ποσοστό των αντικειμένων στο κελί που ικανοποιούν τους περιορισμούς της ερώτησης. Στην παρούσα πτυχιακή εργασία μελετήθηκε και ακολούθως υλοποιήθηκε ο αλγόριθμος STING σε περιβάλλον Borland C++ Builder και ένα γνωστό σύστημα βάσης δεδομένων.

1 DATA MINING – SPATIAL DATA MINING

1.1 ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ (DATA MINING)

Η Εξόρυξη Γνώσης από βάσεις δεδομένων ορίζεται ως η εύρεση πληροφοριών που είναι κρυμμένες σε μία βάση δεδομένων, η εξερευνητική ανάλυση δεδομένων, η ανακάλυψη καθοδηγούμενη από δεδομένα και η εξερευνητική μάθηση. Ειδικότερα, πρόκειται για την διαδικασία «ανακάλυψης» ενδιαφερόντων και εν δυνάμει χρήσιμων προτύπων (patterns), υπαρκτών σε μεγάλες βάσεις δεδομένων. Ο όρος «εξόρυξη» χρησιμοποιείται προκειμένου να τονισθεί ότι τα πρότυπα συνιστούν πολύτιμες πληροφορίες, κρυμμένες καλά μέσα σε μεγάλες βάσεις δεδομένων.

Τα βήματα της διαδικασίας εξόρυξης γνώσης είναι:

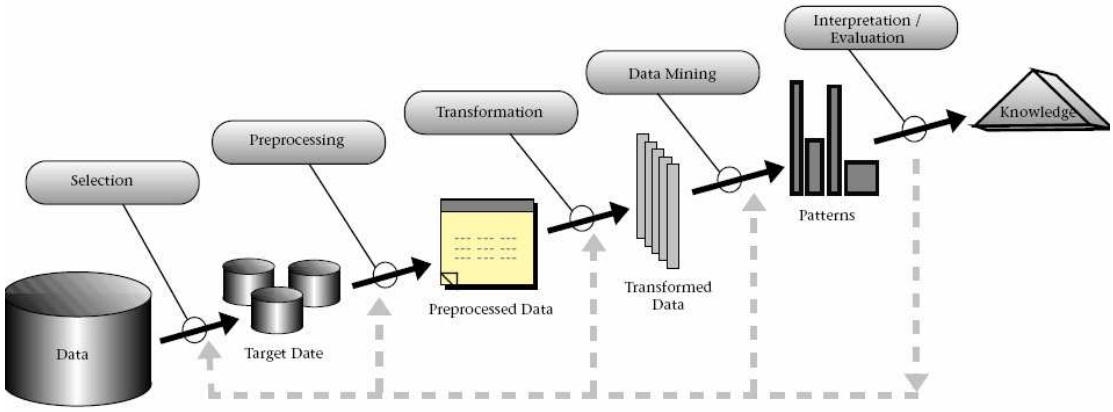
1. Επιλογή: Σε αυτό το βήμα συλλέγονται δεδομένα από διάφορες βάσεις δεδομένων, αρχεία και μη ηλεκτρονικές πηγές.

2. Προεπεξεργασία: Εδώ μπορεί να εκτελεσθούν ενέργειες όπως διόρθωση ή αφαίρεση λανθασμένων δεδομένων, ή συλλογή και εκτίμηση ελλιπών δεδομένων.

3. Μετασχηματισμός: Τα δεδομένα που προέρχονται από διαφορετικές πηγές χρειάζεται να μετατραπούν σε ένα κοινό σχήμα για την περαιτέρω επεξεργασία τους. Κάποια από αυτά ίσως απαιτηθεί να κωδικοποιηθούν, ή να μετασχηματισθούν σε πιο χρήσιμα σχήματα. Μπορεί να ελαττωθούν τα δεδομένα, προκειμένου για την μείωση του αριθμού των τιμών αυτών που θα ληφθούν υπόψη.

4. Εξόρυξη γνώσης από δεδομένα: Με βάση το είδος της εξόρυξης που πρόκειται να εκτελεσθεί, σε αυτό το βήμα εφαρμόζονται αλγόριθμοι στα τροποποιημένα δεδομένα, ώστε να προκύψουν τα προσδοκώμενα αποτελέσματα.

5. Ερμηνεία / αξιολόγηση: Είναι πολύ σημαντικό το πώς θα παρουσιασθούν στους χρήστες τα αποτελέσματα της εξόρυξης γνώσης, επειδή η χρησιμότητα ή μη των αποτελεσμάτων μπορεί να εξαρτάται ακριβώς από αυτήν την παρουσίαση.



Εικόνα. Τα διάφορα βήματα της ανακάλυψης γνώσης σε βάσεις δεδομένων

Η εξόρυξη γνώσης από δεδομένα περιλαμβάνει πολλούς διαφορετικούς αλγορίθμους για να εκπληρωθούν διαφορετικές εργασίες. Όλοι αυτοί οι αλγόριθμοι επιχειρούν να ταιριάξουν ένα μοντέλο στα δεδομένα, εξετάζοντας τα τελευταία και καθορίζοντας το μοντέλο αυτό που είναι το πλησιέστερο στα χαρακτηριστικά τους.

Το Προβλεπτικό Μοντέλο (Predictive Model) Κάνει μία πρόβλεψη για τις τιμές των δεδομένων, χρησιμοποιώντας γνωστά αποτελέσματα που έχει βρει από άλλα δεδομένα. Η μοντελοποίηση της πρόβλεψης μπορεί να γίνει με βάση τη χρήση ιστορικών δεδομένων. Οι πιο συνηθισμένες εργασίες εξόρυξης γνώσης από δεδομένα που χρησιμοποιούν αυτό το είδος μοντέλου, είναι :

η κατηγοριοποίηση (Απεικονίζει τα δεδομένα σε προκαθορισμένες ομάδες ή κατηγορίες – κλάσεις (classes).)

η παλινδρόμηση (Χρησιμοποιείται για να απεικονιστεί ένα στοιχειώδες δεδομένο σε μία πραγματική μεταβλητή πρόβλεψης.)

η ανάλυση χρονολογικών σειρών (Μελετάται η τιμή ενός γνωρίσματος καθώς μεταβάλλεται στο χρόνο.)

η πρόβλεψη (Μπορεί να θεωρηθεί σαν ένα είδος κατηγοριοποίησης)

Το Περιγραφικό Μοντέλο (Descriptive Model) Αναγνωρίζει πρότυπα ή συσχετίσεις στα δεδομένα. Αντίθετα από το προβλεπτικό, λειτουργεί σαν ένα μέσο που διερευνά τις ιδιότητες των δεδομένων που εξετάζονται και όχι για να προβλέπει νέες ιδιότητες. Οι πιο συνηθισμένες εργασίες εξόρυξης γνώσης από δεδομένα που χρησιμοποιούν το περιγραφικό μοντέλο, είναι :

η παρουσίαση συνόψεων (Απεικονίζει τα δεδομένα σε υποσύνολά τους με συνοδευτικές απλές περιγραφές.)

οι κανόνες συσχετίσεων (Η διαδικασία εκείνη της εξόρυξης γνώσης που αποκαλύπτει συσχετίσεις μεταξύ των δεδομένων.)

η παρουσίαση ακολουθιών (χρησιμοποιείται για να καθορισθούν σειριακά πρότυπα στα δεδομένα.)

η συσταδοποίηση (Clustering) : Η συσταδοποίηση αναφέρεται εναλλακτικά και σαν μη εποπτευόμενη μάθηση, ή τμηματοποίηση. Μπορεί να θεωρηθεί σαν μια διαμέριση ή τμηματοποίηση των δεδομένων σε ομάδες, που μπορεί να είναι ή να μην είναι διακριτές μεταξύ τους. Συνήθως επιτυγχάνεται με τον καθορισμό της ομοιότητας, ως προς προκαθορισμένα γνωρίσματα, Μια ειδική κατηγορία συσταδοποίησης ονομάζεται κατάτμηση (segmentation). Με την κατάτμηση, μια βάση δεδομένων χωρίζεται σε διακριτές ομάδες παρόμοιων εγγραφών που ονομάζονται τμήματα (segments). Η κατάτμηση συχνά θεωρείται πανομοιότυπη με την συσταδοποίηση.

1.2 ΧΩΡΙΚΗ ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ

Η Εξόρυξη Χωρικής Γνώσης (Spatial Mining) είναι εξόρυξη γνώσης που εφαρμόζεται σε βάσεις χωρικών δεδομένων ή χωρικά δεδομένα. Ορισμένες από τις εφαρμογές εξόρυξης χωρικής γνώσης εντάσσονται στα πεδία των γεωγραφικών συστημάτων πληροφοριών, γεωλογίας, περιβαλλοντικής επιστήμης, διαχείρισης πόρων, γεωργίας, ιατρικής και ρομποτικής.

- 1854 cholera epidemic
- Hero of the story: John Snow, MD
- Method: plot on a map
  - Cholera deaths
  - Public water pumps
- Discovery:
  - Deaths cluster around one pump
- The epidemic was shut down by removing the handle from the pump

Τα χωρικά δεδομένα είναι δεδομένα, τα οποία έχουν μια χωρική συνιστώσα (ή συνιστώσα θέσης). Μπορούν να θεωρηθούν ως δεδομένα αντικειμένων τα οποία βρίσκονται σε έναν φυσικό χώρο. Αυτό μπορεί να δηλώνεται ρητά με ένα ή περισσότερα γνωρίσματα θέσης, όπως η διεύθυνση ή το γεωγραφικό πλάτος / μήκος ή μπορεί να υπονοείται, όπως με μια διαμέριση της βάσης δεδομένων η οποία βασίζεται στη θέση. Επιπλέον, τα χωρικά δεδομένα μπορούν να προσπελασθούν χρησιμοποιώντας ερωτήσεις που περιέχουν χωρικούς τελεστές όπως οι τελεστές «κοντά», «βόρεια», «νότια», «γειτονικά» και «περιέχεται σε».

Τα χωρικά δεδομένα αποθηκεύονται σε βάσεις χωρικών δεδομένων που περιέχουν τόσο τη χωρική όσο και τη μη χωρική πληροφορία. Εξαιτίας της ενυπάρχουσας πληροφορίας της απόστασης που σχετίζεται με τα χωρικά δεδομένα, οι βάσεις χωρικών δεδομένων πολύ συχνά χρησιμοποιούν ειδικές δομές δεδομένων ή ευρετήρια τα οποία είναι χτισμένα με βάση την πληροφορία απόστασης ή τοπολογίας. Όσον αφορά στην εξόρυξη γνώσης, αυτή η πληροφορία απόστασης παρέχεται στη βάση για τις αναγκαίες μετρήσεις ομοιότητας. Η προσπέλαση των χωρικών δεδομένων μπορεί να είναι πιο πολύπλοκη από αυτή των μη χωρικών δεδομένων. Υπάρχουν ειδικές λειτουργίες και δομές δεδομένων που χρησιμοποιούνται για την προσπέλαση των χωρικών δεδομένων.

1.3 ΧΩΡΙΚΕΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Οι χωρικές βάσεις δεδομένων περιέχουν πολυδιάστατα δεδομένα σχετικά με ένα χώρο για τον οποίο έχουμε ρητή γνώση και δεδομένα για τα οποία γνωρίζουμε τις χωρικές τους ιδιότητες, όπως είναι η έκτασή τους και η θέση τους στο χώρο (όπως π.χ. περιοχές και σημεία). Τα αντικείμενα αυτά συνήθως αναπαρίστανται σε διανυσματική μορφή. Για παράδειγμα, μια χωρική βάση δεδομένων μπορεί να περιλαμβάνει χωρικές πληροφορίες για την περιοχή της Αττικής, δηλαδή το οδικό δίκτυο της περιοχής και σημεία ενδιαφέροντος, όπως είναι βενζινάδικα, φαρμακεία, νοσοκομεία, μηχανές αυτόματης ανάληψης μετρητών κτλ.

Τα χωρικά δεδομένα στη γενική περίπτωση χαρακτηρίζονται απο κάποιες ιδιότητες, που τα διαφοροποιούν από τα υπόλοιπα δεδομένα σχετικά με τον τρόπο διαχείρισης τους.

1. Η πιο χαρακτηριστική ιδιότητα των χωρικών δεδομένων είναι ότι είναι σύνθετα δεδομένα. Ένα χωρικό αντικείμενο μπορεί να αποτελείται από ένα μόνο σημείο, είτε από πολλές χιλιάδες σύνολα σημείων, αυθαίρετα κατανεμημένα στο χώρο. Ως εκ τούτου είναι γενικώς αδύνατο να χρησιμοποιηθούν παραδοσιακά συστήματα βάσεων δεδομένων, όπου κάθε πλειάδα έχει συγκεκριμένο μέγεθος, για να αποθηκευτούν σύνολα από τέτοια αντικείμενα.

2. Τα χωρικά δεδομένα είναι δυναμικά. Οι εισαγωγές και οι διαγραφές είναι αναμεμειγμένες με ενημερώσεις και γι’ αυτό το λόγο πρέπει οι δομές δεδομένων που θα χρησιμοποιηθούν να υποστηρίζουν αυτή τη δυναμική συμπεριφορά.

3. Οι χωρικές βάσεις δεδομένων τείνουν να είναι μεγάλες σε μέγεθος. Το πιο χαρακτηριστικό παράδειγμα, οι γεωγραφικοί χάρτες είναι συνήθως πολλά gigabytes σε μέγεθος. Γι’ αυτό απαιτείται η διάφανη προς το χρήστη χρησιμοποίηση δευτερεύουσας μνήμης.

4. Δεν υπάρχει πρότυπη και τυποποιημένη άλγεβρα ορισμένη επί χωρικών δεδομένων. Αυτό σημαίνει ότι δεν υπάρχει ένα καθορισμένο σύνολο τελεστών, αν και ορισμένοι από τους τελεστές χρησιμοποιούνται πιο συχνά από άλλους(όπως η τομή). Οι τελεστές που χρησιμοποιούνται εξαρτώνται άμεσα από το θεματικό πεδίο της κάθε εφαρμογής.

5. Πολλοί χωρικοί τελεστές δεν είναι κλειστοί. Για παράδειγμα, η τομή μεταξύ δύο πολυγώνων στο επίπεδο μπορεί να επιστρέψει οποιοδήποτε αριθμό από άσχετα μεταξύ τους σημεία, γωνιές και ξένα μεταξύ τους πολύγωνα. Αυτό δυσκολεύει περισσότερα τα πράγματα, όταν τέτοιοι τελεστές χρησιμοποιούνται επαναληπτικά.

Ας δούμε μερικά βασικά για τις χωρικές βάσεις δεδομένων. Μια Χωρική Βάση Δεδομένων (Spatial Database) χρησιμοποιείται για την αποθήκευση χωρικών αντικειμένων που αναπαρίστανται με χωρικούς τύπους δεδομένων και σχετίζονται με χωρικές συσχετίσεις. Τα χωρικά δεδομένα φέρουν πληροφορίες σχετικές με την τοπολογία και τις αποστάσεις των αντικειμένων αυτών. Για την οργάνωσή τους χρησιμοποιούνται κατάλληλες δομές ευρετηρίων και για την προσπέλασή τους κατάλληλες επερωτήσεις.

Οι Χωρικές Βάσεις Δεδομένων προέκυψαν από το επιτυχημένο σχεσιακό μοντέλο με κύριο σκοπό να εξυπηρετήσουν άλλες τεχνολογίες, όπως είναι τα GIS. Από το γεγονός αυτό και όσα έχουν προαναφερθεί για τα GIS, γίνεται προφανής η σημασία τους για τα συστήματα που παρέχουν υπηρεσίες εξαρτώμενες από την θέση.

Σήμερα πολλά πανεπιστήμια και ερευνητικές ομάδες δραστηριοποιούνται στο πεδίο των τεχνικών που μπορούν να χρησιμοποιηθούν σε τέτοιου είδους βάσεις (πχ spatial data mining). Επίσης πολλές εταιρίες έχουν ήδη παρουσιάσει συστήματα διαχείρισης βάσεων δεδομένων (DBMS) ενώ άλλες έχουν προσαρτήσει λειτουργίες διαχείρισης χωρικών δεδομένων στα υπάρχοντα συστήματά τους (τέτοια παραδείγματα αποτελούν τα εργαλεία Oracle Spatial και Oracle Locator που μας επιτρέπει να αποθηκεύουμε, να προσπελαύνουμε και να αναλύουμε τέτοια δεδομένα γρήγορα και αποτελεσματικά).

Οι βασικοί γεωμετρικοί τύποι από τους οποίους αποτελούνται όλα τα στοιχεία μιας τέτοιας βάσης είναι:

• Σημείο (Point)

• Γραμμή (Line)

• Πολύγωνο (Polygon)

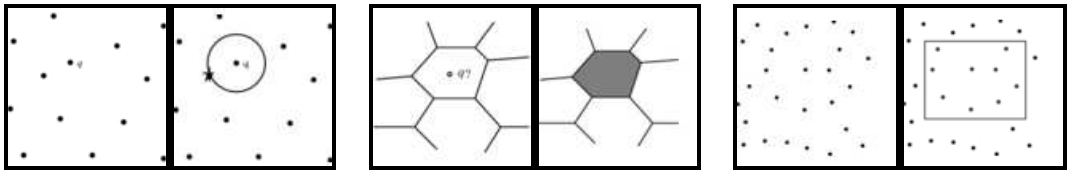
Το σημείο είναι ένα στοιχείο που αποτελείται από 2 συντεταγμένες (οι οποίες μπορεί να είναι και γεωγραφικό μήκος και πλάτος). Η γραμμή αποτελείται από ένα ζευγάρι σημείων και μπορεί να σχηματίσει μία σειρά γραμμών (line string). Το πολύγωνο αποτελείται από ένα σύνολο συνδεδεμένων γραμμών οι οποίες όμως πρέπει να σχηματίζουν ένα κλειστό δακτύλιο. Τα σημεία μπορούν, για παράδειγμα, να αναπαριστούν κάποια γεωγραφικά σημεία ιδιαίτερης σημασίας για τον χρήστη της βάσης δεδομένων, όπως τηλεφωνικούς θαλάμους ή φανάρια. Όμοια οι σειρές γραμμών μπορούν να αναπαριστούν δρόμους και τα πολύγωνα γεωγραφικά όρια. Βέβαια οι δυνατότητες αναπαράστασης δεν σταματούν εδώ αφού μπορούν να συνδυαστούν τα παραπάνω στοιχεία για να προκύψουν πιο περίπλοκες δομές (είναι πολύ συχνή η συνεργασία των Χωρικών Βάσεων Δεδομένων με εφαρμογές CAD/CAM, για να μοντελοποιούν αρκετά περίπλοκες δομές στον χώρο).

Οι πράξεις που μπορούν να εφαρμοστούν στα χωρικά δεδομένα είναι επερωτήσεις και υπάρχουν τρεις βασικές κατηγορίες :

- Επερώτηση διαστήματος: Βρίσκει τα αντικείμενα ενός συγκεκριμένου τύπου που βρίσκονται μέσα σε δεδομένη χωρική περιοχή ή σε συγκεκριμένη απόσταση από δοθείσα θέση. (πχ, όλα τα νοσοκομεία που βρίσκονται στην περιοχή της πόλης Αθήνας ή βρίσκει όλα τα ασθενοφόρα σε απόσταση πέντε μιλίων από τη θέση ενός ατυχήματος).

- Επερώτηση για τον κοντινότερο γείτονα: Βρίσκει ένα αντικείμενο συγκεκριμένου τύπου που βρίσκεται πιο κοντά σε δεδομένη θέση. (πχ, το περιπολικό που βρίσκεται πιο κοντά σε συγκεκριμένη θέση).

- Χωρικές συνενώσεις ή επικαλύψεις: Τυπικά συνενώνει αντικείμενα δύο τύπων με βάση κάποια χωρική συνθήκη, όπως αντικείμενα που τέμνονται ή επικαλύπτονται χωρικά ή που βρίσκονται σε κάποια απόσταση το ένα από το άλλο. (πχ, όλες τις πόλεις που βρίσκονται σ‘ έναν αυτοκινητόδρομο ή όλα τα σπίτια που απέχουν δύο μίλια από μια λίμνη.)



Nearest Neighbor Search Point Location Range Search

Βέβαια όλες αυτές οι πράξεις απαιτούν πολύ περίπλοκους αλγορίθμους έτσι ώστε να ελαχιστοποιηθεί η (αρκετά) μεγάλη χρονική πολυπλοκότητα τους. Η ανακάλυψη τέτοιων αλγορίθμων είναι μια από τις σημαντικές ερευνητικές περιοχές στις Χωρικές Βάσεις Δεδομένων επειδή τα χωρικά δεδομένα είναι η κυρίαρχη πηγή δεδομένων του πραγματικού κόσμου.

Στην μοντελοποίηση των χωρικών δεδομένων ο χώρος ορίζεται σαν ένα σύνολο από αντικείμενα στα οποία αναθέτονται χαρακτηριστικά, και συσχετίσεις με άλλα αντικείμενα. Η καταγραφή των δομικών στοιχείων του χώρου περνά στις βάσεις δεδομένων. Τα μοντέλα δεδομένων δεν έχουν στατική και σταθερή λειτουργία αφού μπορεί να επεκτείνονται σε μορφές διαφορετικές, όπως πχ. η διάδραση σε ένα σύγχρονο γραφικό περιβάλλον, συνεργασία με σύγχρονες γλώσσες προγραμματισμού, αντιμετώπιση πολυδιάστατων και δεδομένων ειδικών σκοπών.

Βέβαια οι υλοποιήσεις διαφέρουν πολύ μεταξύ τους αλλά συνήθως ακολουθούν τον παραπάνω κανόνα. Για παράδειγμα η Oracle με την Spatial επέκταση δεν αλλάζει την αρχιτεκτονική του συστήματος διαχείρισης (συμπεριλαμβανομένων των στοιχείων που ανήκουν στο μοντέλο δεδομένων που υποστηρίζει). Επεκτείνει όμως όλα τα δομικά τμήματα στης στρωματοποιημένης αρχιτεκτονικής της ώστε να υποστηρίζει τη νέα, χωρική μορφή δεδομένων.

Η εξέλιξη των χωρικών βάσεων δεδομένων διακρίνεται σε τέσσερις μεγάλες κατηγορίες υλοποιήσεων:

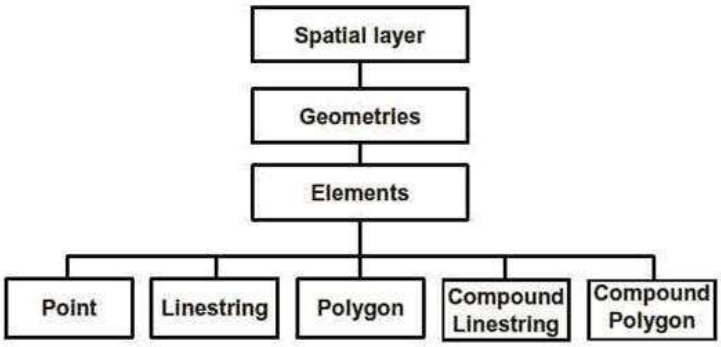
1. Πλήρως κανονικοποιημένοι πίνακες. Συνδεδεμένοι πίνακες συντεταγμένων ορίζουν την γεωμετρία. Κάθε εφαρμογή ασχολείται με τις λειτουργίες και την ακεραιότητα των δεδομένων.

2. Μεγάλα δυαδικά αντικείμενα (Binary Large Objects ή BLOBs). Δυαδικά κωδικοποιημένα αρχεία ορίζουν την γεωμετρία. Κάθε εφαρμογή ασχολείται με τις λειτουργίες και την ακεραιότητα των δεδομένων.

3. Χωρικές επεκτάσεις σε Σύστημα διαχείρισης βάσεων δεδομένων (Spatially Extended RDBMS). Συνεχίζει η σχεσιακή λογική. Η προτυποποίηση αποθήκευσης βασίζεται σε κάθε κατασκευαστή Σύστημα διαχείρισης βάσεων δεδομένων χωριστά. Δεν υπάρχει συμβατότητα ανάμεσα στα Σύστημα διαχείρισης βάσεων δεδομένων, κάτι που αποβαίνει δύσκολο στην υποστήριξη από τους κατασκευαστές συστημάτων GIS. Η επέκταση αντιμετωπίζει μόνο ένα μέρος από τις λειτουργίες και την ακεραιότητα των δεδομένων. Μερική υποστήριξη του ανερχόμενου ορισμού τύπων δεδομένων από τους χρήστες (User defined data types ή UDTs).

4. Ορισμοί τύπων δεδομένων από τους χρήστες (UDTs SQL3 και SQL/MM). Αρχή αντικειμενοσχεσιακής λογικής.

Ένα χωρικό μοντέλο δεδομένων στην εφαρμογή του είναι μια ιεραρχική δομή η οποία αποτελείται από τα στοιχειώδη χωρικά αντικείμενα μέχρι τα ολοκληρωμένα σύνολα χωρικών αντικείμενών. Οι αναπαραστάσεις αυτές ξεκινούν από το πρωταρχικό ≪στοιχείο≫ (element) το οποίο είναι το δομικό υλικό μιας γεωμετρίας (όπως ζεύγη συντεταγμένων). Μια γεωμετρία ή γεωμετρικό αντικείμενο είναι η αναπαράσταση μιας γεωμετρικής οντότητας συναποτελούμενης από διατάξεις πρωταρχικών στοιχείων. Μπορεί να είναι ομοιογενή ή ετερογενή σύνολα. (πχ. μια πολυγραμμή ή multiline string, μια κατασκευή γεωμετρίας από ένα πολύγωνο και ένα σημείο). Τα στρώματα (layers) είναι σύνολα από ετερογενείς γεωμετρίες οι οποίες όμως έχουν κοινές εννοιολογικές ιδιότητες (πχ. πόλεις και το οδικό δίκτυο που τις ενώνει). Ακολουθεί σχήμα που δείχνει αυτή τη δομη :



1.4 INDEXING ΧΩΡΙΚΩΝ ΔΕΔΟΜΕΝΩΝ

Μια σημαντική κλάση γεωμετρικών τελεστών που απαιτεί ειδική υποστήριξη στο φυσικό επίπεδο είναι η κλάση των χωρικών τελεστών αναζήτησης. Η ανάκτηση και η ενημέρωση χωρικών δεδομένων συνήθως εξαρτάται όχι μόνο από την τιμή κάποιων αλφαριθμητικών χαρακτηριστικών, αλλά και από τη χωρική θέση ενός αντικειμένου. Μια ερώτηση ανάκτησης σε μια χωρική βάση συχνά απαιτεί τη γρήγορη εκτέλεση μιας ερώτησης σημείου ή περιοχής. Δοθέντος ενός συνόλου γεωμετρικών αντικειμένων στο κ-διάστατο Ευκλείδιο χώρο Ε, αποθηκευμένου σε μια χωρική βάση δεδομένων, μια ερώτηση περιοχής (region query) υπολογίζει τα αντικείμενα της βάσης που επικαλύπτουν ένα δεδομένο διάστημα αναζήτησης. Η ερώτηση σημείου (point query) μπορεί να θεωρηθεί ως μια εκφυλισμένη ερώτηση περιοχής, που υπολογίζει τα αντικείμενα που περιέχουν ένα σημείο p. Οι δύο αυτές λειτουργίες απαιτούν πολύ γρήγορη πρόσβαση στα αντικείμενα της βάσης που χωρικά βρίσκονται μέσα με μια περιοχή. Επειδή τα χωρικά δεδομένα αναπαρίστανται με ειδικούς τύπους δεδομένων, για την οργάνωση τους χρησιμοποιούνται άλλοι τύποι ευρετηρίων από τους συνηθισμένους. Οι δομές αυτές είναι συνήθως πολυδιάστατα δέντρα.

Για την υποστήριξη των λειτουργιών χωρικών τελεστών αναζήτησης χρειάζονται μέθοδοι προσπέλασης πολυδιάστατων δεδομένων (**multidimensional access methods**). Οι «κλασσικοί» αλγόριθμοι indexing για μονοδιάστατα δεδομένα, όπως τα B και B+ δέντρα, δεν είναι κατάλληλα για indexing χωρικών και πολυδιάστατων δεδομένων. Γι’ αυτό το λόγο έχουν προταθεί πολλές άλλες δομές για την ευρετηρίαση πολυδιάστατων δεδομένων και τις μεθόδους προσπέλασης σε αυτά.

Οι μέθοδοι προσπέλασης για πολυδιάστατα δεδομένα (Multidimensional access methods) μπορούν να ταξινομηθούν ως μέθοδοι προσπέλασης σημείου (Point Access Methods - PAM) ή χωρικές μέθοδοι προσπέλασης (Spatial Access Methods - SAM). Οι μέθοδοι προσπέλασης σημείου εκτελούν τις αναζητήσεις σε δεδομένα σημείων (στοιχεία χωρίς έναν χωρικό βαθμό), ενώ οι χωρικές μέθοδοι προσπέλασης χειρίζονται τα χωρικά αντικείμενα που, εκτός από τη θέση τους, έχουν και χωρικές ιδιότητες όπως εμβαδόν και σχήμα.

Κατά ακολουθία οι μέθοδοι PAM μπορούν να ταξινομηθούν σε δύο κατηγορίες: πολυδιάστατες hashing μέθοδοι προσπέλασης και ιεραρχικές μέθοδοι προσπέλασης. Οι ιεραρχικές μέθοδοι προσπέλασης χρησιμοποιούν τις ιεραρχικές δομές δεδομένων για να διαχειριστούν τα πολυδιάστατα στοιχεία σημείων. Οι μέθοδοι προσπέλασης σημείου (PAM) που περιλαμβάνονται σε αυτήν την κατηγορία είναι το quadtree , το k-D tree το oct-tree και άλλα. Εκτός από αυτές τις memory-based ιεραρχικές δομές δεικτοδότησης πολυδιάστατων δεδομένων υπάρχουν και πολλές disk-based ιεραρχικές δομές για αναζήτηση ομοιότητας όπως τα k-d-B trees, Grid files, Pyramid tree, R-trees, R*-trees, R+-tree.

1.4.1 R-Trees

Τα R-δέντρα είναι από τα πιο πολυχρησιμοποιημένα. Κάθε κόμβος-φύλλο των R δέντρων είναι της μορφής (I, P), όπου I το ελάχιστο ορθογώνιο στο οποίο μπορεί να περιέχεται το χωρικό στοιχείο της ΧΒΔ στο οποίο δείχνει ο δείκτης P. Οι υπόλοιποι κόμβοι είναι πλειάδες που αποτελούνται από στοιχεία της μορφής (I, PC), όπου I είναι το ελάχιστο ορθογώνιο στο οποίο μπορούν να περιέχονται όλα τα ορθογώνια των παιδιών του συγκεκριμένου κόμβου. PC είναι ο δείκτης που δείχνει στο κόμβο-παιδί του συγκεκριμένου κόμβου. Οι υπόλοιπες ιδιότητες των δέντρων αυτών μοιάζουν πολύ με τις ιδιότητες των B-δέντρων. Στη συνέχεια προτάθηκαν πολλές παραλλαγές της δομής αυτής, με αποτέλεσμα να δημιουργηθεί μια «οικογένεια» από δομές που βασίζονται σε R δέντρα, από τις οποίες καμία δεν υπερτερεί των άλλων για όλα τα δυνατά σύνολα δεδομένων, αλλά κάθε τέτοια δομή απευθύνεται κυρίως σε συγκεκριμένες περιοχές εφαρμογών.

Τα αρχικά R-Trees είναι μια επέκταση των B+ δέντρων στον πολυδιάστατο χώρο, που αναπαριστούν τα δεδομένα ως διαστήματα(intervals) στο χώρο των πολλών διαστάσεων. Δηλαδή ένα R δέντρο είναι μια ιεραρχία από εμφωλευμένα n-διάστατα διαστήματα. Οι κόμβοι των δέντρων αυτών είναι πλειάδες της μορφής (I, tuple_id), όπου I = (I0, I1, I2,… In-1) και το Ii αναπαριστά την έκταση του περιβάλλοντος κουτιού (bounding box) του αντικειμένου I στην i-οστή διάσταση. Αν το v είναι ένα φύλλο του δέντρου, τότε το In(v) είναι το n-διάστατο ελάχιστο περιβάλλον κουτί (Minimum Bounding Box, MBB) των αντικειμένων που είναι αποθηκευμένα στο v. Μερικά από τα κύρια χαρακτηριστικά της δομής αυτής είναι:

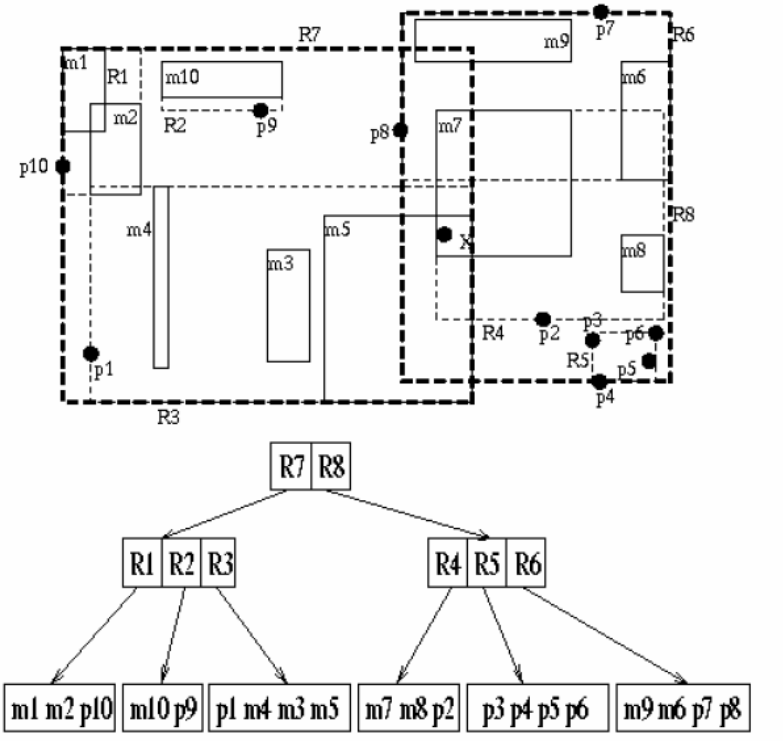
• Κάθε κόμβος περιέχει μεταξύ m και M εγγραφές (εκτός αν είναι η ρίζα).Το κάτω όριο αποτρέπει τον εκφυλισμό του δέντρου και εξασφαλίζει αποδοτική χρησιμοποίηση του αποθηκευτικού χώρου. Όποτε ο αριθμός των απογόνων ενός κόμβου πέσει κάτω από το m, ο κόμβος διαγράφεται και οι απόγονοί του κατανέμονται σε γειτονικούς κόμβους (condensation). Το άνω όριο M προκύπτει από την επιθυμητή ιδιότητα κάθε κόμβος του δέντρου να αντιστοιχεί σε μια ακριβώς σελίδα του δίσκου.

• Τα R-trees έχουν ισοζυγισμένο ύψος, δηλαδή όλα τα φύλλα βρίσκονται στο ίδιο βάθος

• Το ύψος ενός R-Tree είναι logm N , όπου N είναι ο αριθμός των εγγραφών του ευρετηρίου

Όταν δοθεί μια ερώτηση για μια περιοχή (πολυδιάστατο διάστημα) E η αναζήτηση γίνεται περίπου όπως και στα B δέντρα, δηλαδή ξεκινάει από τη ρίζα του δέντρου και βρίσκει όλες τις εγγραφές με τις οποίες το διάστημα E έχει κάποια επικάλυψη. Στη συνέχεια επισκεπτόμαστε όλους του κόμβους-παιδιά για τους οποίους ισχύει $I^n(v_i) \cap E \neq \varnothing$ μέχρι να φτάσουμε σε κόμβους φύλλα οπότε

μπορούμε να προσδιορίσουμε αν έχουμε ταίριασμα ή όχι (επειδή στο R-tree χειριζόμαστε μόνο bounding boxes των αντικειμένων, όταν ένα αντικείμενο δεν έχει το σχήμα ενός τέτοιου box (που είναι και η συνήθης περίπτωση), για να λυθεί πλήρως το πρόβλημα της αναζήτησης πρέπει να ελεγχθεί αυτό το σύνολο από υποψήφια αντικείμενα στα οποία δείχνουν τα φύλλα, για να διαπιστωθεί αν το δοθέν σημείο ανήκει στη γεωμετρία του αντικειμένου).



Εικόνα. R tree

Έπειτα από την εμφάνιση της αρχικής δομής του R-tree παρουσιάστηκαν και μια σειρά από δομές που ήταν παραλλαγές του R-tree και προσπαθούσαν να επιλύσουν κάποια προβλήματά του. Για παράδειγμα, το R+-tree χρησιμοποιεί την τεχνική της αποκοπής, δηλαδή απαιτεί να μην υπάρχει επικάλυψη μεταξύ των διαστημάτων In που βρίσκονται στο ίδιο επίπεδο του δέντρου. Με αυτή την τεχνική οι ερωτήσεις σημείου απαιτούν την αναζήτηση κατά μήκους ενός μόνο μονοπατιού του δέντρου, γεγονός που αυξάνει κατά πολύ την απόδοση της δομής δεδομένων.

Μια άλλη παραλλαγή, το R*-tree βασίζεται στην παρατήρηση ότι η εισαγωγή των αντικειμένων στο δέντρο έχει πολύ μεγάλο αντίκτυπο στην ολική του απόδοση και γι’ αυτό το λόγο χρησιμοποιεί την τεχνική της επιβεβλημένης επανεισαγωγής (forced-reinsert). Σύμφωνα με αυτή όταν ένας κόμβος υπερχειλίζει δεν πρέπει να σπάσει αμέσως, αλλά είναι καλύτερα να αφαιρέσουν έναν αριθμό από εγγραφές του κόμβο και να τον επανεισάγουν στο δέντρο.

1.4.2 QUAD TREES

Οι παραλλαγές των R-trees που περιγράφηκαν στις παραπάνω παραγράφους είναι οι πλέον κατάλληλες και ευρέως χρησιμοποιούμενες για τη χωρική ευρετηρίαση μη-σημειακών αντικειμένων, όπως είναι πολύγωνα, γραμμές κτλ. Στην παρούσα εργασία, ωστόσο, μας ενδιαφέρει ιδιαίτερα η απόδοση της χωρικής δομής στην ευρετηρίαση σημειακών αντικειμένων, χωρίς να δίνεται έμφαση στην επεξεργασία και απάντηση ερωτημάτων πρόβλεψης. Έτσι οι προηγούμενες δομές αποδεικνύονται αργές σε σχέση με άλλες δομές, όπως τα quad trees που περιγράφονται παρακάτω.

Τα Quad-Trees προτάθηκαν από τους R.Finkel και J.L.Bentley το 1974. Πρόκειται για μια από τις πιο έντονα μελετημένες δομές δεδομένων η οποία στηρίζεται στις αρχές της περιοδικά επαναλαμβανόμενης διάσπασης *(divide and conquer* methods)

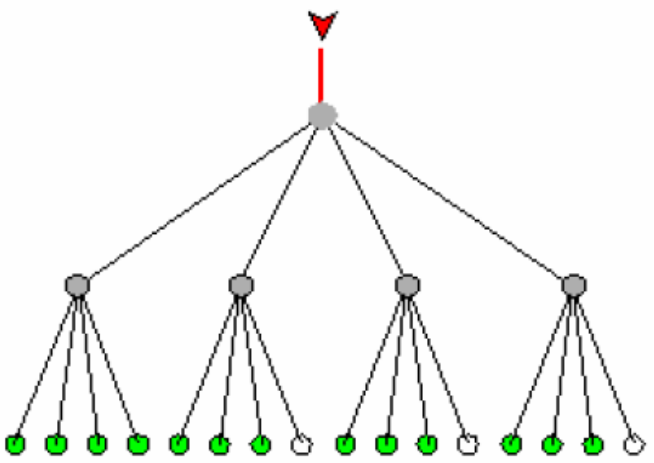
Στην απλούστερη του μορφή το Quad –Tree είναι όμοιο με το δυαδικό δένδρο, με εξαίρεση ότι κάθε κόμβος-γονέας έχει τέσσερις απογόνους (κάποιοι από τους οποίους μπορεί να είναι κενοί), για αυτό και ονομάζεται τετραδικό (Quad) δένδρο. Το Quad – Tree είναι μια μη ισορροπημένη (unbalanced) χωρική δομή δεδομένων η οποία διασπάει περιοδικά τον χώρο (δυσδιάστατο χώρο) σε τέσσερα ίσου μεγέθους ασυνάρτητα τεταρτημόρια, μέχρι κάθε κόμβος φύλλο να περιέχει ένα μόνο σημείο. Η περιοχή στην οποία διασπάτε ο χώρος μπορεί να είναι τετράγωνη, ορθογώνια ή να έχει αυθαίρετα σχήματα ανάλογα με την εκδοχή του Quad – Tree. Το Quad – Tree το συναντάμε συχνά στην βιβλιογραφία και ως Q - Tree. Υπάρχουν πολλές παραλλαγές του Quad – Tree, όλες όμως έχουν κάποια κοινά χαρακτηριστικά:

- Διασπούν τον χώρο σε προσαρμόσιμα κελιά
- Κάθε κελί έχει μια μέγιστη χωρητικότητα. Όταν το κελί φθάσει την μέγιστη χωρητικότητα του διασπάτε.
- Ο δενδρικός κατάλογος ακολουθεί τη χωρική διάσπαση του Quad – Tree.

Τα Quad Trees είναι μια οικογένεια δέντρων με πολλές παραλλαγές. Το κοινό τους στοιχείο είναι ότι αποτελούν ιεραρχικές δομές δεδομένων, που βασίζουν τη λειτουργία τους στην αναδρομική αποσύνθεση (decomposition) του χώρου. Οι διάφορες παραλλαγές quadtrees διαφοροποιούνται με βάση το είδος των δεδομένων που αναπαριστούν, τη μέθοδο (αρχή) με την οποία γίνεται η αποσύνθεση του χώρου (από μεγαλύτερες σε μικρότερες περιοχές) και το αν η ευκρίνεια (resolution) της δομής καθορίζεται στατικά ή μπορεί να μεταβάλλεται.

Εδώ θα ασχοληθούμε με quad trees για την ευρετηρίαση σημειακών αντικειμένων στο διδιάστατο χώρο, καθώς όταν έχουμε να κάνουμε με αντικείμενα που έχουν έκταση (περιοχές) ή μήκος (γραμμές) τυχαίου σχήματος, τότε τα R-Trees εμφανίζουν πολύ καλύτερη απόδοση σε σχέση με τα quad trees.

Ένα quad tree φαίνεται στο σχήμα που ακολουθεί

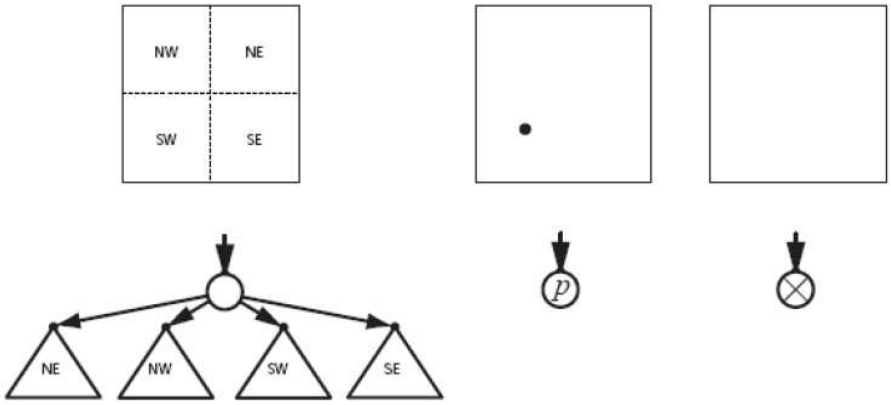


Σε ένα quad tree κάθε εσωτερικός κόμβος του έχει 4 κόμβους - παιδιά. Κάθε κόμβος αντιστοιχεί σε μια περιοχή του χώρου και κάθε παιδί του αντιστοιχεί σε ένα υποσύνολο του πατρικού χώρου. Εδώ ασχολούμαστε με περιπτώσεις στις οποίες ο χωρισμός του χώρου γίνεται κανονικά δηλαδή κάθε κόμβος είναι τετραγωνικού σχήματος και τα παιδιά του είναι ισομεγέθη. Έτσι κάθε κόμβος αντιστοιχεί σε μια τετραγωνική περιοχή του χώρου και κάθε παιδί του αντιστοιχεί στο ¼ του χώρου του.

Συγκεκριμένα ο γονικός κόμβος διαμερίζεται στα τέσσερα τεταρτημόριά (quadrants) του, δηλαδή το κάθε παιδί του αντιστοιχεί σε ένα τεταρτημόριό του και για το λόγο αυτό μπορούμε να συμβολίσουμε τα παιδιά του με βάση τη θέση τους ως προς το κέντρο του πατρικού τετραγώνου δηλ. NW (north-west), NE (north-east), SW (south-west), SE (south-east). Φύλλα του δέντρου είναι οι κόμβοι εκείνοι για τους οποίους δεν απαιτείται περεταίρω διαμέριση. Τα δέντρα του τύπου αυτού ονομάζονται region quadtrees.

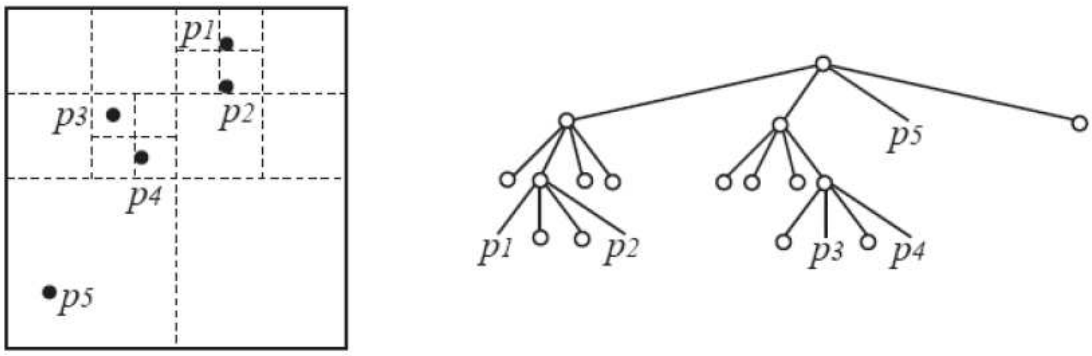
Σε περιπτώσεις που θέλουμε να κάνουμε για παράδειγμα indexing μια εικόνα (που είναι μια πολύ συχνή χρήση των quadtrees) κάθε φύλλο απλά κρατά την τιμή του χρώματος για την περιοχή αυτή(ο χώρος αποσυντίθεται σε περιοχές με το ίδιο χρώμα). Αντίθετα εδώ μας ενδιαφέρει για κάθε περιοχή του χώρου να γνωρίζουμε ποια σημειακά αντικείμενα οριοθετούνται από αυτή. Γι’ αυτό το λόγο σε κάθε φύλλο αποθηκεύουμε τα σημειακά αντικείμενα που περιέχει. Αν θεωρήσουμε ότι κάθε φύλλο του δέντρου χωράει μόνο ένα σημειακό αντικείμενο (το ίδιο ισχύει και για περισσότερα σημεία), τότε μπορεί να έχουμε τρία είδη κόμβων:

- Εσωτερικούς κόμβους (δηλαδή κόμβοι που δεν είναι φύλλα)
- Κόμβους – φύλλα που περιέχουν ένα σημείο
- Κόμβους – φύλλα που είναι άδειοι.



Εικόνα. Είδη κόμβων Quad tree

Το είδος αυτό του δέντρου, που συσχετίζει τεταρτημόρια με σημεία ονομάζεται PR Quadtree (Point – Region Quadtree). Παρακάτω φαίνεται ο αναδρομικός χωρισμός του αρχικού χώρου μετά την προσθήκη πέντε σημείων, καθώς και το αντίστοιχο δέντρο που προκύπτει.



Εικόνα. Παράδειγμα αναδρομικού χωρισμού του χώρου

Σε ένα PR quadtree κάθε κόμβος είναι μια εγγραφή που περιέχει τις ακόλουθες καταχωρήσεις:

• Τέσσερις δείκτες προς τα παιδιά του κόμβου, καθένας από τους οποίους δηλαδή αντιστοιχεί σε ένα τεταρτημόριο. Αν ο P είναι δείκτης προς ένα κόμβο και το I είναι ένα τεταρτημόριο, οι καταχωρήσεις αυτές αναφέρονται ως SON(P, I).

• Το πέμπτο πεδίο, NODETYPE, έχει την τιμή BLACK αν πρόκειται για φύλλο που περιέχει σημείο, WHITE αν πρόκειται για κενό φύλλο και GRAY αν πρόκειται για κόμβο που δεν είναι φύλλο.

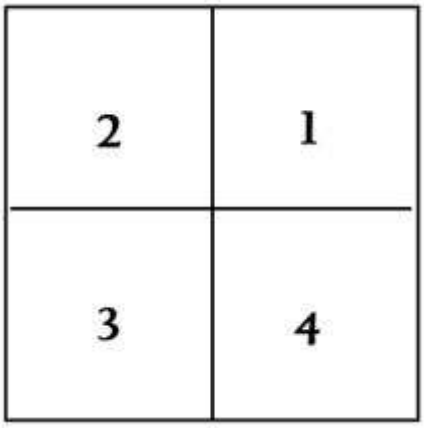
• Το πεδίο NAME, δηλαδή το όνομα του σημειακού αντικειμένου που είναι αποθηκευμένο στον κόμβο

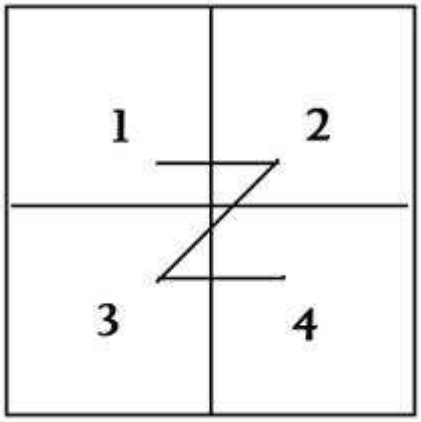
• Τα πεδία XCOORD και YCOORD, δηλαδή τις συντεταγμένες του σημειακού αντικειμένου.

Θεωρούμε στο παράδειγμα ότι κάθε αντικείμενο είναι μοναδικό.

1.4.3 ΔΙΑΔΙΚΑΣΙΑ ΔΗΜΙΟΥΡΓΙΑΣ ΕΝΟΣ QUAD – TREE

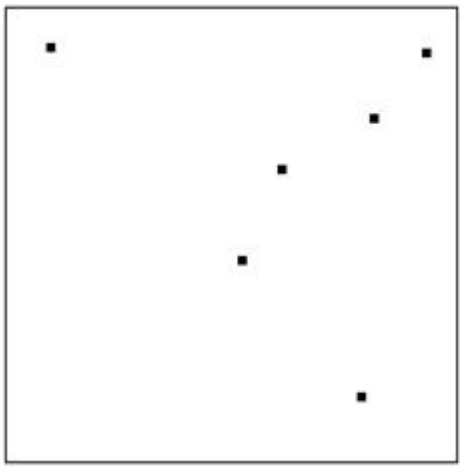
Η πιο συνηθισμένη μέθοδος απόδοσης τιμών στα τεταρτημόρια ενός Quad – Tree είναι αυτή της **εικόνας 1** (c ordering) όπου οι αριθμοί παρατάσσονται με σειρά αντίθετη αυτής των δεικτών του ρολογιού. Άλλη γνωστή μέθοδος απόδοσης τιμών στα τεταρτημόρια ενός Quad Tree είναι της **εικόνας 2** (Z ordering) όπου οι αριθμοί παρατάσσονται έτσι ώστε να σχηματίζεται ένα νοητό Z.

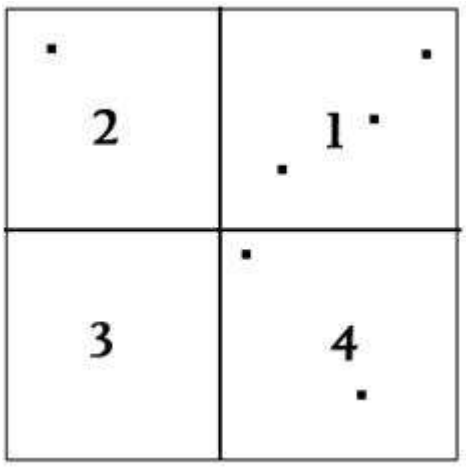




Εικόνα 1: C ordering **Εικόνα 2:** Z ordering

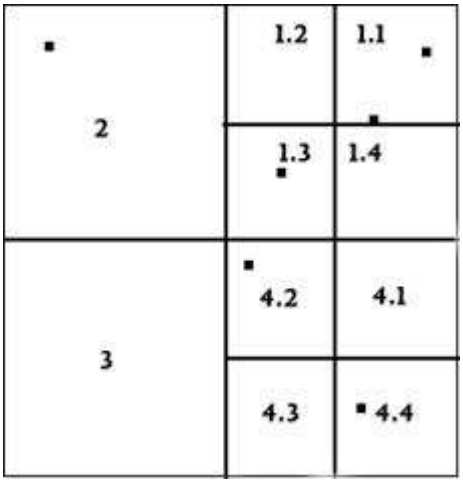
Στις παρακάτω εικόνες παρουσιάζεται το παράδειγμα της δημιουργίας ενός Quad – Tree. Στην πρώτη εικόνα φαίνονται τα σημεία που έχουν επιλεχθεί για την δημιουργία του Quad – Tree. Η πρώτη εικόνα αποτελεί την ρίζα του δένδρου. Στην συνέχεια χωρίζεται σε τεταρτημόρια όπου σε κάθε ένα από αυτά αντιστοιχεί ένας αριθμός (1, 2, 3, 4). Στα τεταρτημόρια που έχουμε περισσότερα του ενός σημεία η διαδικασία συνεχίζεται έως κάθε σημείο να ανήκει σε διαφορετικό τεταρτημόριο, **εικόνα 6**.



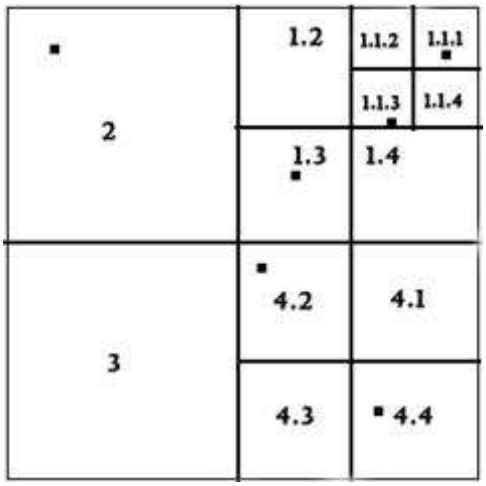


Εικόνα 3: Τα σημεία που επιλέχθηκαν **Εικόνα 4**: Επίπεδο 1

Στην **Εικόνα 4** παρατηρούμε ότι τα τεταρτημόρια 1 και 4 περιέχουν περισσότερα του ενός σημεία. Η διαδικασία συνεχίζεται, **Εικόνα 5**, όπου τα τεταρτημόρια αυτά έχουν χωριστεί εκ νέου σε τέσσερα νέα αλλά το τεταρτημόριο 1.1 συνεχίζει να περιέχει περισσότερα του ενός σημεία. Για το λόγω αυτό η διαδικασία συνεχίζεται και ολοκληρώνεται στην **Εικόνα 6** όπου κάθε σημείο ανήκει σε διαφορετικό τεταρτημόριο.

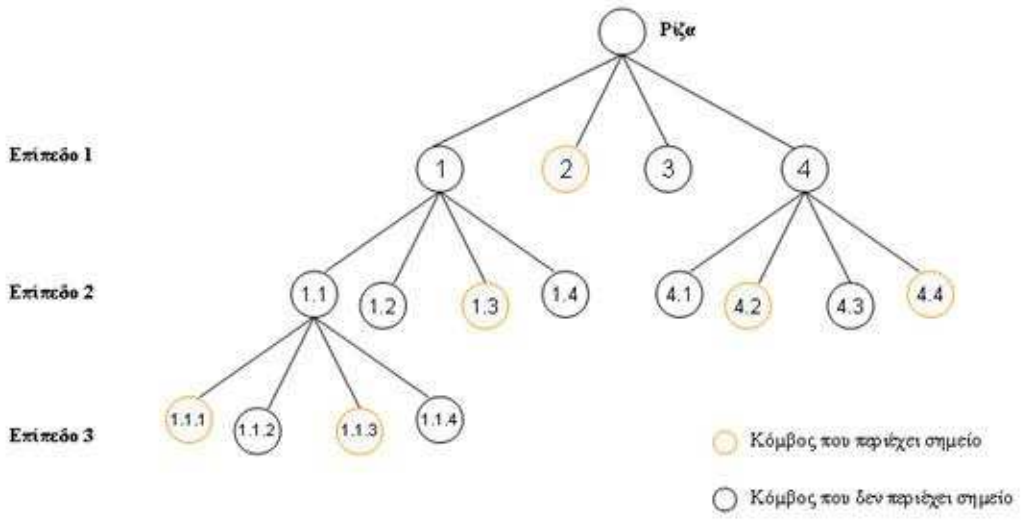


Εικόνα 5: Επίπεδο 2



Εικόνα 6: Επίπεδο 3

Το τετραδικό δένδρο (Quad - Tree) που σχηματίζεται από την εφαρμογή του αλγορίθμου στα σημεία της εικόνας 8, φαίνεται στην **Εικόνα 7**.



Εικόνα 7: Το Quad – Tree των σημείων της εικόνας 18

Χρήσεις του Quad-Tree

Το Quad – Tree και οι παραλλαγές του έχουν πολλές εφαρμογές, οι περισσότερες από τις οποίες σχετίζονται με τον τομέα της εικόνας. Ειδικότερα το Quad - Tree χρησιμοποιείται:

- Στον τομέα των Computer Graphics για την αναπαράσταση μιας εικόνας. Επιτρέπει την αναπαράσταση μιας εικόνας σε διαφορετικά επίπεδα ανάλυσης.
- Στον τομέα της ψηφιακής επεξεργασίας εικόνας (Image Processing) για την σύμπτυξη (Compact) ή συμπίεση (Compress) μιας εικόνας. Με τον όρο compact εννοούμε την συμπίεση χωρίς απώλειες: Μπορούμε να ανακτήσουμε ακριβώς την πρωτότυπη εικόνα. Αντίθετα, με τον όρο Compress εννοούμε την συμπίεση με απώλειες: Μπορούμε να ανακτήσουμε μια κοντινή προσέγγιση της πρωτότυπης εικόνας.

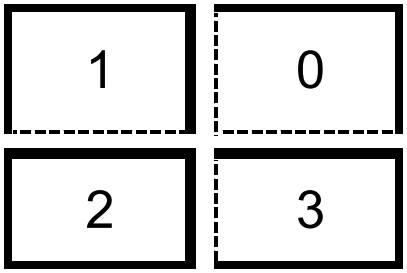
Άλλες χρήσεις στον τομέα αυτό είναι για τον τεμαχισμό μίας εικόνας με βάση το χρώμα ή το σχήμα των αντικειμένων που περιέχει ή για τον εντοπισμό αντικειμένων σε μια εικόνα (Object Modeling).

Το Quad – Tree χρησιμοποιείται από αρκετές μεθόδους για την βάση-περιεχομένου ανάκτηση εικόνας (Content-Based Image Retrieval), με σκοπό την σύλληψη των χωρικών χαρακτηριστικών της εικόνας, για παράδειγμα χρώμα, υφή, σχήμα

- Χρησιμοποιείται από σχεδιαστικά προγράμματα τύπου CAD - Computer-Aided Design.
- Χρησιμοποιείται στα γεωγραφικά συστήματα πληροφορίας - Geographical Information Systems (GIS).
- Χρησιμοποιείται από στον τομέα της χαρτογράφηση.
- Χρησιμοποιείται από βάσεις δεδομένων εικόνων (Image databases).

Τα τελευταία χρόνια το Quad – Tree έχει ενσωματωθεί σε πολλές εφαρμογές.

Ένα σημείο ανήκει σε ένα μόνο τεταρτημόριο, και την υλοποίησή μας οι έλεγχοι της ισότητας (η κλειστότητα) γίνονται στις πλευρές των τεταρτημορίων με συνεχόμενη γραμμή. Στις πλευρές με διακεκομένη γραμμή ελέγχονται μόνο ανισότητες.

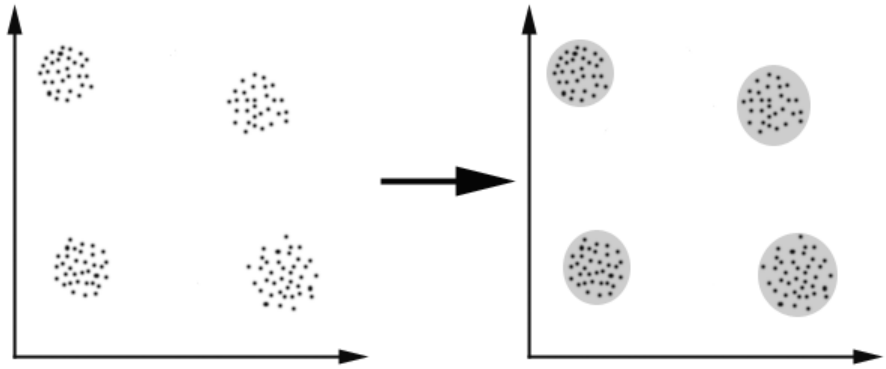


2 CLUSTERING – GRID CLUSTERING

2.1 Συσταδοποίηση

Η Συσταδοποίηση ή ομαδοποίηση ή εύρεση συγκροτήσεων μπορεί να θεωρηθεί το σημαντικότερο πρόβλημα μη επιβλεπόμενης μηχανικής μάθησης. Ένας ορισμός της Συσταδοποίησης θα μπορούσε να είναι " η διαδικασία οργάνωσης αντικειμένων σε ομάδες, των οποίων μέλη είναι παρόμοια κατά κάποιο τρόπο".

Ένας cluster είναι επομένως μια συλλογή αντικειμένων που είναι "όμοια" μεταξύ τους και είναι "ανόμοια" με τα αντικείμενα που ανήκουν σε άλλους clusters. Το παραπάνω μπορεί να παρουσιαστεί με ένα απλό γραφικό παράδειγμα:



Σχήμα : Παράδειγμα ομαδοποίησης

2.1.1 Οι στόχοι της συσταδοποίησης

Ο στόχος είναι να καθοριστεί η εγγενής ομαδοποίηση σε ένα σύνολο unlabeled δεδομένων. Αλλά πώς να αποφασιστεί τι αποτελεί μια καλή ομαδοποίηση; Μπορεί να αποδειχθεί ότι δεν υπάρχει κάποιο απόλυτα "βέλτιστο" κριτήριο που θα ήταν ανεξάρτητο από τον τελικό στόχο της ομαδοποίησης. Συνεπώς, είναι ο χρήστης που πρέπει να παρέχει αυτό το κριτήριο, κατά τέτοιο τρόπο ώστε το αποτέλεσμα της ομαδοποίησης να ανταποκρίνεται στις ανάγκες του.

Παραδείγματος χάριν, θα μπορούσαμε να ενδιαφερθούμε για την εύρεση αντιπροσώπων για ομοιογενείς ομάδες (μείωση δεδομένων), για την εύρεση "φυσικών clusters" και την περιγραφή των άγνωστων ιδιοτήτων τους ("φυσικοί" τύποι δεδομένων), για την εύρεση χρήσιμων και κατάλληλων σχηματισμών ομάδας ("χρήσιμες" κλάσεις δεδομένων) ή για την εύρεση ασυνήθιστων αντικειμένων δεδομένων (ανίχνευση outliers).

2.1.2 *Πιθανές εφαρμογές*

Οι αλγόριθμοι ομαδοποίησης μπορούν να εφαρμοστούν σε πολλούς τομείς, παραδείγματος χάριν:

- **Μάρκετινγκ**: εύρεση ομάδων πελατών με παρόμοια συμπεριφορά, δοθείσης μιας μεγάλης βάσης δεδομένων με δεδομένα πελατών που περιέχουν τις ιδιότητες τους και παρελθόντα αρχεία αγοράς.
- **Βιολογία:** ταξινόμηση των φυτών και τα ζώων με βάση τα χαρακτηριστικά γνωρίσματα τους.
- **Βιβλιοθήκες**: Ταξινόμηση βιβλίων.
- **Ασφάλεια**: προσδιορισμός των ομάδων κατόχων πολιτικών ασφάλειας μηχανοκίνητων οχημάτων με υψηλό μέσο κόστος αξίωσης, προσδιορισμός απατών.
- **Προγραμματισμός πόλης**: προσδιορισμός των ομάδων σπιτιών σύμφωνα με τον τύπο σπιτιών, την αξία και τη γεωγραφική θέση τους
- **Μελέτες σεισμού**: ομαδοποίηση των παρατηρηθέντων επίκεντρων σεισμού προς προσδιορισμό των επικίνδυνων ζωνών.
- **WWW:** ταξινόμηση εγγράφων, ομαδοποίηση των δεδομένων των weblogs προς ανακάλυψη των ομάδων παρόμοιων σχεδίων πρόσβασης.

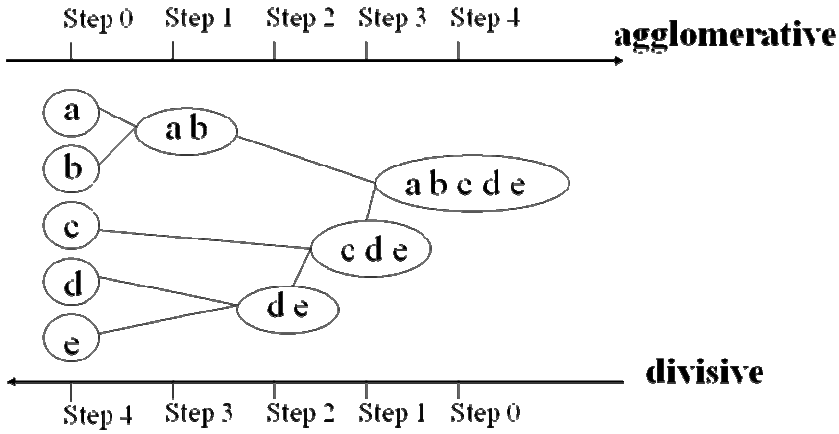
2.2 Αλγόριθμοι Συσταδοποίησης

Τα τελευταία χρόνια, διάφοροι αλγόριθμοι ομαδοποίησης έχουν προταθεί και είναι διαθέσιμοι. Οι αλγόριθμοι ομαδοποίησης μπορούν να ταξινομηθούν με βάση:

- Τον τύπο δεδομένων εισόδου του αλγορίθμου.
- Το κριτήριο ομαδοποίησης που ορίζει την ομοιότητα μεταξύ data points. Π.χ.:
  - Απόσταση: δύο ή περισσότερα αντικείμενα ανήκουν στον ίδιο cluster εάν είναι “κοντά” σύμφωνα με μια δεδομένη απόσταση (σε αυτήν την περίπτωση γεωμετρική απόσταση). Αυτό καλείται ομαδοποίηση βασισμένη-στην-απόσταση(distance-based clustering).
  - *Έννοια*: δύο ή περισσότερα αντικείμενα ανήκουν στον ίδιο cluster εάν αυτός καθορίζει μια έννοια κοινή για όλα αυτά τα αντικείμενα. Με άλλα λόγια, τα αντικείμενα ομαδοποιούνται σύμφωνα με τη συμφωνία τους με περιγραφικές έννοιες, όχι σύμφωνα με απλά μέτρα ομοιότητας. Αυτό καλείται εννοιολογική ομαδοποίηση(conceptual clustering).
- Τη θεωρία και τις θεμελιώδεις έννοιες πάνω στις οποίες οι τεχνικές ανάλυσης της ομαδοποίησης είναι βασισμένες (π.χ. ασαφής θεωρία(fuzzy theory), στατιστικές).

Κατά συνέπεια σύμφωνα με τη μέθοδο που υιοθετείται για τον ορισμό των clusters, οι αλγόριθμοι μπορούν ευρέως να ταξινομηθούν στους ακόλουθους τύπους :

- Το Partitional clustering, που προσπαθεί να αποσυνθέσει ευθέως το σύνολο δεδομένων σε ένα σύνολο από ασύνδετους clusters.. Πιο συγκεκριμένα προσπαθεί να καθορίσει έναν ακέραιο αριθμό partitions που βελτιστοποιούν μια ορισμένη συνάρτηση κριτηρίου. Η συνάρτηση κριτηρίου μπορεί να υπογραμμίσει την τοπική ή σφαιρική δομή των δεδομένων και η βελτιστοποίησή της είναι μια επαναληπτική διαδικασία.
- Την Ιεραρχική ομαδοποίηση (Hierarchical Clustering), που προχωρά διαδοχικά είτε συγχωνεύοντας μικρότερους clusters σε μεγαλύτερους, είτε με το διαχωρισμό των μεγαλύτερων clusters. Το αποτέλεσμα του αλγορίθμου είναι ένα δέντρο από clusters, που ονομάζεται δενδρόγραμμα (dendrogram), το οποίο επιδεικνύει πώς συσχετίζονται οι clusters. Με την κοπή του δενδρογράμματος σε επιθυμητό επίπεδο, λαμβάνεται μια ομαδοποίηση των δεδομένων σε ασύνδετες ομάδες. Η ιεραρχική ομαδοποίηση σύμφωνα με τη μέθοδο που παράγει τους clusters μπορεί περαιτέρω να διαιρεθεί σε:
    - Συσσωρευτική ομαδοποίηση (*Agglomerative Clustering*). Παράγει μια ακολουθία σχημάτων ομαδοποίησης φθίνοντος αριθμού clusters σε κάθε βήμα. Το σχήμα ομαδοποίησης που παράγεται σε κάθε βήμα προκύπτει από το προηγούμενο με τη συγχώνευση των δύο πιο κοντινών clusters σε έναν.
    - Διαχωριστική ομαδοποίηση(*Divisive Clustering*). Αυτή η ομαδοποίηση παράγει μια ακολουθία σχημάτων ομαδοποίησης αύξοντος αριθμού clusters σε κάθε βήμα. Σε αντίθεση με τους συσσωρευτική ομαδοποίηση, η ομαδοποίηση που παράγεται σε κάθε βήμα προκύπτει από το προηγούμενο με το διαχωρισμό ενός cluster σε δύο.



Παράδειγμα Συσσωρευτικής και Διαχωριστικής ομαδοποίησης

- Την Ομαδοποίηση βασισμένη-στην-πυκνότητα (Density-based clustering). Η βασική ιδέα αυτού του τύπου ομαδοποίησης είναι να ομαδοποιηθούν τα γειτονικά αντικείμενα ενός συνόλου δεδομένων σε clusters με βάση συνθήκες πυκνότητας.
- Την Ομαδοποίηση βασισμένη-στο-πλέγμα (Grid-based clustering). Αυτός ο τύπος αλγορίθμων προτείνεται κυρίως για τη χωρική ανάσυρση δεδομένων. Το κύριο χαρακτηριστικό τους είναι ότι κβαντοποιούν το χώρο σε ένα πεπερασμένο αριθμό κελιών (cells) και έπειτα κάνουν όλες τις διαδικασίες πάνω στο κβαντοποιημένο χώρο.

Για κάθε μια από τις ανωτέρω κατηγορίες υπάρχει ένας πλούτος υποκατηγοριών και διαφορετικών αλγορίθμων για την εύρεση των clusters. Κατά συνέπεια, σύμφωνα με τον τύπο μεταβλητών που επιτρέπονται στο σύνολο δεδομένων μπορεί να ταξινομηθούν σε:

- Στατιστικούς (Statistical), οι οποίοι είναι βασισμένοι στις έννοιες στατιστικής ανάλυσης. Χρησιμοποιούν τα μέτρα ομοιότητας (similarity measures) για να χωρίσουν τα αντικείμενα και περιορίζονται στα αριθμητικά δεδομένα.
- Εννοιολογικούς(Conceptual), οι οποίοι χρησιμοποιούνται για να ομαδοποιήσουν τα κατηγορικά δεδομένα. Ομαδοποιούν τα αντικείμενα σύμφωνα με τις έννοιες που φέρουν.

  Ένα άλλο κριτήριο ταξινόμησης είναι ο τρόπος που χειρίζεται η ομαδοποίηση την αβεβαιότητα, από την άποψη της επικάλυψης των clusters.

- Ασαφούς ομαδοποίησης (Fuzzy Clustering), που χρησιμοποιεί τεχνικές ασάφειας για να ομαδοποιήσει τα δεδομένα και θεωρεί ότι ένα αντικείμενο μπορεί να είναι ανήκει σε περισσότερους του ενός clusters. Αυτός ο τύπος αλγορίθμων οδηγεί σε σχήματα ομαδοποίησης που είναι συμβατά με την καθημερινή εμπειρία ζωής δεδομένου ότι χειρίζονται την αβεβαιότητα των πραγματικών δεδομένων.
- "Στιβαρής" ομαδοποίησης (Crisp Clustering), που ασχολείται με μη επικαλυπτόμενες διαχωρίσεις, που σημαίνει ότι ένα data point είτε ανήκει σε μια κλάση είτε όχι. Οι περισσότεροι από τους αλγορίθμους ομαδοποίησης οδηγούν σε "στιβαρούς" clusters, και μπορούν έτσι να ταξινομηθούν στη "στιβαρή" ομαδοποίηση.
- Ομαδοποίησης δικτύου Kohonen, η οποία είναι βασισμένη στις έννοιες των νευρωνικών δικτύων. Το δίκτυο Kohonen έχει τους κόμβους εισόδου και εξόδου. Το στρώμα εισόδου (κόμβοι εισόδου) έχει έναν κόμβο για κάθε ιδιότητα της εγγραφής, καθένα συνδεόμενο με κάθε κόμβο εξόδου (στρώμα εξόδου). Κάθε σύνδεση συνδέεται με ένα βάρος, το οποίο καθορίζει τη θέση του αντίστοιχου κόμβου εξόδου. Κατά συνέπεια, σύμφωνα με έναν αλγόριθμο, που αλλάζει τα βάρη κατάλληλα, οι κόμβοι εξόδου κινούνται προς το σχηματισμό clusters.

Γενικά, οι αλγόριθμοι ομαδοποίησης είναι βασισμένοι σε ένα κριτήριο για την αξιολόγηση της ποιότητας ενός δεδομένου διαχωρισμού. Συγκεκριμένα, παίρνουν ως είσοδο μερικές παραμέτρους (π.χ. αριθμός clusters, πυκνότητα των clusters) και προσπαθούν να ορίσουν τον καλύτερο διαχωρισμό ενός συνόλου δεδομένων για τις δεδομένες παραμέτρους. Κατά συνέπεια, ορίζουν ένα διαχωρισμό ενός συνόλου δεδομένων βασισμένοι σε ορισμένες υποθέσεις και όχι απαραιτήτως τον "καλύτερο" που ταιριάζει στο σύνολο δεδομένων.

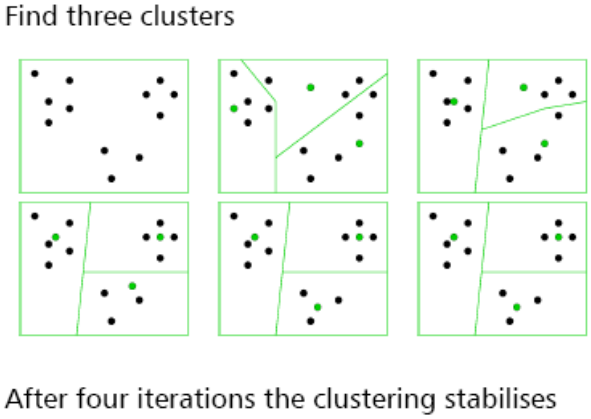
Ας δούμε όμως μερικούς αντιπροσωπευτικούς αλγορίθμους των κατηγοριών partition-based, hierarchical, grid-based, density-based, και model-based για να μπορέσουμε να κατανοήσουμε καλύτερα την έννοια της ομαδοποίησης.

2.2.1 Partitional algorithms

Σε αυτήν την κατηγορία, ο K-Means είναι ένας συνήθως χρησιμοποιημένος αλγόριθμος. Ο στόχος της ομαδοποίησης K-Means είναι η βελτιστοποίηση μιας συνάρτησης-κριτηρίου που περιγράφεται από την εξίσωση

$$E = \sum_{i=1}^{c} \sum_{x \in C_i} d(x, m_i)$$

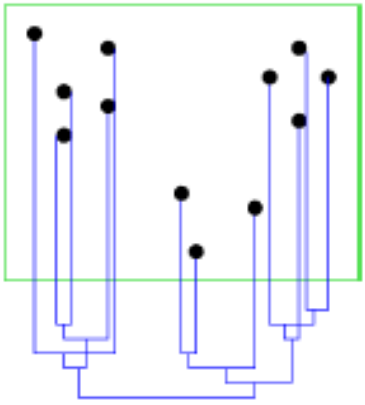
Στην ανωτέρω εξίσωση, το m_i είναι το κέντρο (centroid) του cluster C_i, ενώ $d(x, m_i)$ είναι η ευκλείδεια απόσταση μεταξύ ενός σημείου x και του m_i. Το κέντρο ενός cluster είναι το μέσο σημείο στον πολυδιάστατο χώρο που ορίζεται από τις διαστάσεις. Υπό μία έννοια είναι το κέντρο βαρύτητας του cluster. Δεν αποτελεί απαραίτητα υπαρκτό σημείο του cluster, μπορεί δηλαδή να είναι νοητό. Κατά συνέπεια, η συνάρτηση-κριτήριο E προσπαθεί να ελαχιστοποιήσει την απόσταση κάθε σημείου από το κέντρο του cluster στην οποία το σημείο ανήκει. Συγκεκριμένα, ο αλγόριθμος αρχίζει με την αρχικοποίηση ενός συνόλου κέντρων clusters c. Κατόπιν, αναθέτει κάθε αντικείμενο του συνόλου δεδομένων στον cluster του οποίου το κέντρο είναι το κοντινότερο, και επαναϋπολογίζει τα κέντρα. Η διαδικασία συνεχίζεται μέχρι τα κέντρα των clusters να σταματήσουν να μεταβάλλονται ή η συνάρτηση κριτηρίου να μεταβληθεί ελάχιστα.



2.2.2 Ιεραρχικοί αλγόριθμοι (Hierarchical Algorithms)

Μερικοί αντιπροσωπευτικοί ιεραρχικοί αλγόριθμοι ομαδοποίησης είναι οι παρακάτω:

Ο Αλγόριθμος BIRCH (Balanced Iterative Reducing and Clustering) χρησιμοποιεί μια ιεραρχική δομή δεδομένων που ονομάζεται CF-tree για το διαχωρισμό των εισερχόμενων data points με έναν επαυξητικό και δυναμικό τρόπο. Το CF-tree είναι ένα height-balanced δέντρο, το οποίο αποθηκεύει τα χαρακτηριστικά ομαδοποίησης και είναι βασισμένο σε δύο παραμέτρους: στον *παράγοντα διακλάδωσης* B και το κατώτατο όριο(threshold) T, που είναι σχετιζόμενες με τη διάμετρο ενός cluster (Η διάμετρος (ή ακτίνα) κάθε cluster πρέπει να είναι μικρότερη από το T). Ο BIRCH μπορεί τυπικά να βρει μια καλή ομαδοποίηση με ένα μοναδικό πέρασμα των δεδομένων και να βελτιώσει την ποιότητα περαιτέρω με μερικά πρόσθετα περάσματα. Είναι επίσης ο πρώτος αλγόριθμος ομαδοποίησης που μπορεί να χειριστεί το θόρυβο αποτελεσματικά. Εντούτοις, δεν αντιστοιχεί πάντα σε ένα φυσικό cluster, δεδομένου ότι κάθε κόμβος στο CF-tree μπορεί να κρατήσει έναν περιορισμένο αριθμό καταχωρήσεων λόγω του μεγέθους του. Επιπλέον, είναι ευαίσθητος-στη-διάταξη δεδομένου ότι μπορεί να παράγει διαφορετικούς clusters για τις διαφορετικές διατάξεις των ίδιων δεδομένων εισόδου.



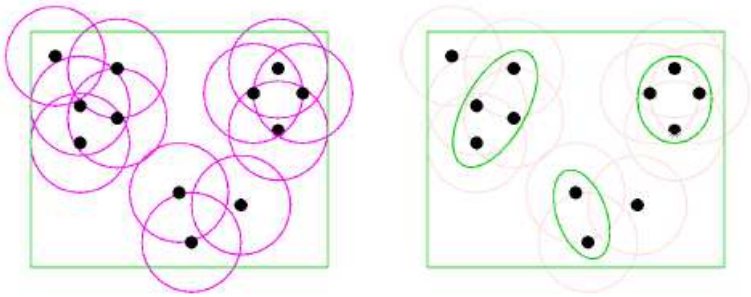
Ο CURE απεικονίζει κάθε cluster με ένα συγκεκριμένο αριθμό σημείων που παράγονται με την επιλογή καλά-διεσπαρμένων σημείων και έπειτα το στένεμα τους προς το centroid του cluster κατά ένα προσδιορισμένο κλάσμα. Χρησιμοποιεί έναν συνδυασμό τυχαίας δειγματοληψίας και ομαδοποίησης διαχωρισμάτων για να χειριστεί μεγάλες βάσεις δεδομένων.

Ο ROCK, είναι ένας στιβαρός αλγόριθμος ομαδοποίησης για Boolean και κατηγορικά δεδομένα. Εισάγει δύο νέες έννοιες, οι οποίες είναι “γείτονες” και “συνδέσεις ενός σημείου”, και είναι βασισμένο σε αυτές προκειμένου να μετρηθούν η ομοιότητα/εγγύτητα μεταξύ ενός ζευγαριού data points.

2.2.3 Αλγόριθμοι βασισμένοι-στην-πυκνότητα (Density-based)

Οι βασισμένοι στην πυκνότητα αλγόριθμοι τυπικά θεωρούν τους clusters ως πυκνές περιοχές αντικειμένων στο χώρο δεδομένων που χωρίζονται από περιοχές χαμηλής πυκνότητας.

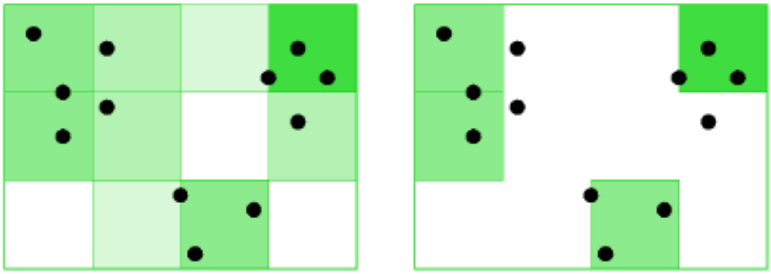
Ένας ευρέως γνωστός αλγόριθμος αυτής της κατηγορίας είναι ο DBSCAN . Η βασική ιδέα στον DBSCAN είναι ότι για κάθε σημείο σε ένα cluster, η γειτονιά μιας δεδομένης ακτίνας πρέπει να περιέχει τουλάχιστον έναν ελάχιστο αριθμό σημείων. Ο DBSCAN μπορεί να χειριστεί το θόρυβο (outliers) και να ανακαλύψει clusters αυθαίρετης μορφής. Επιπλέον, ο DBSCAN χρησιμοποιείται ως βάση για έναν επαυξητικό αλγόριθμο ομαδοποίησης που προτείνεται στο.



Λόγω της βασισμένης-στη-πυκνότητα φύσης του, η εισαγωγή ή η διαγραφή ενός αντικειμένου έχει επιπτώσεις στην τρέχουσα ομαδοποίηση μόνο στη γειτονιά του αντικειμένου και έτσι μπορούν να δοθούν αποδοτικοί αλγόριθμοι βασισμένοι στον DBSCAN για επαυξητικές εισαγωγές και διαγραφές σε μια υπάρχουσα ομαδοποίηση.

2.2.4 Αλγόριθμοι βασισμένοι-στο-πλέγμα (Grid-based)

Πρόσφατα διάφοροι αλγόριθμοι ομαδοποίησης έχουν παρουσιαστεί για τα χωρικά δεδομένα (spatial data), γνωστοί ως αλγόριθμοι βασισμένοι-στο-πλέγμα (Grid-based algorithms). Αυτοί οι αλγόριθμοι κβαντοποιούν το χώρο σε ένα πεπερασμένο αριθμό κελιών (cells) και έπειτα κάνουν όλες τις διαδικασίες πάνω στο κβαντοποιημένο χώρο.



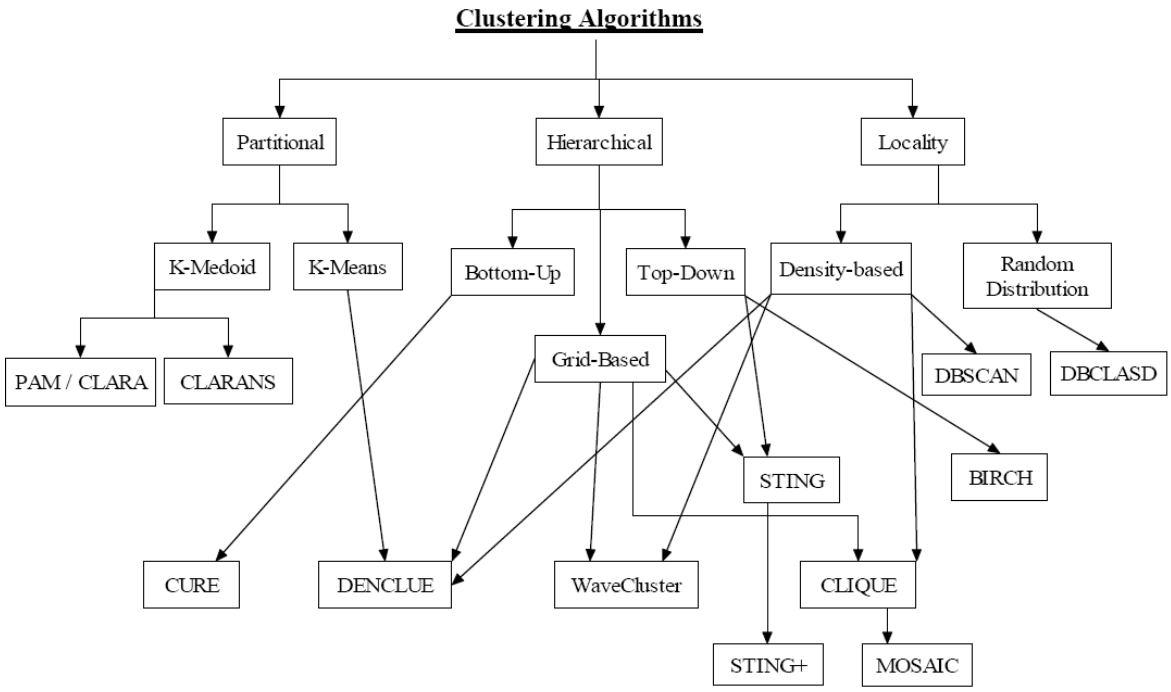
Calculate density in each grid cell. Discard cells with too low density

Ο STING (Statistical Information Grid-based method) είναι αντιπροσωπευτικός αλγόριθμος αυτής της κατηγορίας. Διαιρεί τη χωρική περιοχή σε ορθογώνια κελιά χρησιμοποιώντας μια ιεραρχική δομή. Ο STING σαρώνει το σύνολο δεδομένων και υπολογίζει τις στατιστικές παραμέτρους (όπως ο μέσος όρος, η απόκλιση, ελάχιστο, μέγιστο και τύπος διασποράς) του κάθε αριθμητικού χαρακτηριστικού των αντικειμένων μέσα στα κελιά. Κατόπιν παράγει μια ιεραρχική δομή των κελιών πλέγματος ώστε να απεικονίσει τις πληροφορίες ομαδοποίησης σε διαφορετικά επίπεδα. Με βάση αυτήν την δομή ο STING επιτρέπει τη χρήση των πληροφοριών της ομαδοποίησης για αναζήτηση μέσω queries ή την αποδοτική ανάθεση ενός νέου αντικειμένου στους clusters.

Ο WaveCluster είναι ο πιο πρόσφατος βασισμένος-στο-πλέγμα αλγόριθμος που προτείνεται στη βιβλιογραφία. Χρησιμοποιεί τεχνικές επεξεργασίας σήματος (μετασχηματισμός wavelets) για τη μεταφορά των χωρικών δεδομένων στο πεδίο των συχνοτήτων. Πιο συγκεκριμένα, πρώτα συμπτύσσει τα δεδομένα με την επιβολή μιας πολυδιάστατης δομής πλέγματος επάνω στο χώρο δεδομένων . Κάθε κελί πλέγματος συμπτύσσει τις πληροφορίες της ομάδας σημείων που βρίσκονται σε αυτό. Κατόπιν χρησιμοποιεί έναν μετασχηματισμό wavelet για να μετασχηματίσει τον αρχικό χώρο χαρακτηριστικών. Στο μετασχηματισμό wavelet, η συνέλιξη με μια κατάλληλη συνάρτηση οδηγεί σε ένα μετασχηματισμένο χώρο όπου γίνονται διακριτοί οι φυσικοί clusters των δεδομένων. Κατά συνέπεια, μπορούμε να προσδιορίσουμε τους clusters μέσω της εύρεσης των πυκνών περιοχών στο μετασχηματισμένο πεδίο. Δεν απαιτείται η a-priori γνώση του ακριβή αριθμού των clusters στον WaveCluster.

2.2.5 Αλγόριθμοι βασισμένοι σε μοντέλα (Model-based)

Οι Βασισμένοι σε μοντέλα (Model-based): υποθέτουν ένα μοντέλο για κάθε συστάδα και βρίσκουν το καλύτερο ταίριασμα των δεδομένων σε αυτό διανέμοντας κάθε σημείο δεδομένου στη συστάδα στην οποία αναμένεται να έχει τη μεγαλύτερη πιθανότητα να ανήκει. Προσπαθούν να βελτιστοποιήσουν το ταίριασμα μεταξύ των δεδομένων και ενός μαθηματικού μοντέλου που θα τα περιγράφει. Ο Αλγόριθμος EM (Expectation, Maximization) δημιουργεί συστάδες διανέμοντας με επαναληπτική επανατοποθέτηση, κάθε σημείο δεδομένου, στη συστάδα στην οποία αναμένεται να έχει τη μεγαλύτερη πιθανότητα να ανήκει. Ο EM με το Gaussian Mixture Model βρίσκει την εκτίμηση μέγιστης πιθανοφάνειας (maximum likelihood) για μία παράμετρο ‘μέσου’. Ο Hierarchical Model-based clustering (HMBC) συνενώνει ζεύγη συστάδων που αναλογούν στην ελάχιστη μείωση της μεταξύ τους πιθανοφάνειας.



2.3 Εγκυρότητα Ομαδοποίησης

Η εφαρμογή ενός αλγορίθμου ομαδοποίησης σε ένα σετ δεδομένων στοχεύει, υποθέτοντας ότι το σετ δεδομένων προσφέρει μια τέτοια τάση ομαδοποίησης, στην ανακάλυψη των έμφυτων διαμερισμών του. Ωστόσο, η διαδικασία ομαδοποίησης γίνεται αντιληπτή ως μία ανεπίβλεπτη διαδικασία, καθώς δεν υπάρχουν προκαθορισμένες κλάσεις και παραδείγματα που θα έδειχναν τι είδος επιθυμητής σχέσης ανάμεσα στα δεδομένα πρέπει να θεωρείται έγκυρη. Έπειτα, οι διάφοροι αλγόριθμοι ομαδοποίησης βασίζονται σε κάποιες υποθέσεις για να ορίσουν ένα διαμερισμό του σετ δεδομένων. Κατά συνέπεια μπορεί να συμπεριφερθούν με διαφορετικό τρόπο ανάλογα με:

- Τα χαρακτηριστικά του σετ δεδομένων (γεωμετρία και κατανομή πυκνότητας των clusters) και
- Τις τιμές των παραμέτρων εισόδου.

Συνεπώς, αν τεθούν στις παραμέτρους του αλγορίθμου ομαδοποίησης ακατάλληλες τιμές, η μέθοδος ομαδοποίησης θα καταλήξει σε ένα σχήμα διαμερισμού που δε θα είναι βέλτιστο για το συγκεκριμένο σετ δεδομένων οδηγώντας σε λάθος αποφάσεις.

Είναι εμφανές ότι ένα πρόβλημα που αντιμετωπίζουμε στην ομαδοποίηση είναι το να αποφασίσουμε τον βέλτιστο αριθμό clusters που ταιριάζει σε ένα σετ δεδομένων. Ορίζουμε τον όρο “βέλτιστο” σχήμα ομαδοποίησης ως το αποτέλεσμα της εκτέλεσης ενός αλγορίθμου ομαδοποίησης, που ταιριάζει καλύτερα στον έμφυτο διαμερισμό του σετ δεδομένων. Είναι δύσκολο να ορίσουμε πότε ένα αποτέλεσμα

ομαδοποίησης είναι αποδεκτό, κατά συνέπεια έχουν αναπτυχθεί διάφορες τεχνικές και δείκτες ελέγχου της εγκυρότητας της ομαδοποίησης.

2.3.1 Αξιολόγηση της ομαδοποίησης

Η διαδικασία αξιολόγησης των αποτελεσμάτων ενός αλγορίθμου ομαδοποίησης ονομάζεται αξιολόγηση της εγκυρότητας των clusters (cluster validity assessment). Δύο κριτήρια μέτρησης έχουν προταθεί για την αξιολόγηση και την επιλογή ενός βέλτιστου σχήματος ομαδοποίησης:

- Συνοχή (compactness): Η απόσταση μεταξύ των μελών κάθε cluster πρέπει να είναι όσο το δυνατόν πιο μικρή. Ένα κοινό μέτρο της συνοχής είναι η διακύμανση(variance) που πρέπει να είναι ελάχιστη.
- Διαχωρισμός (separation): Τα clusters πρέπει να είναι μεταξύ τους πολύ διαχωρίσιμα. Υπάρχουν τρεις κοινές προσεγγίσεις για την μέτρηση της απόστασης μεταξύ δύο διαφορετικών clusters:
    - Απλός σύνδεσμος(simple linkage): απόσταση μεταξύ των πιο κοντινών μελών των clusters.
    - Πλήρης σύνδεσμος (complete linkage): απόσταση μεταξύ των πιο απόμακρων μελών των clusters.
    - Σύγκριση των κέντρων(Comparison of centroids): απόσταση μεταξύ των κέντρων των clusters

Οι δύο πρώτες προσεγγίσεις είναι βασισμένες στις στατιστικές δοκιμές και το σημαντικό μειονέκτημά τους είναι το υψηλό υπολογιστικό κόστος τους. Επιπλέον, οι σχετικοί με αυτές τις προσεγγίσεις δείκτες στοχεύουν στη μέτρηση του βαθμού στον οποίο ένα σύνολο στοιχείων επιβεβαιώνει ένα διευκρινισμένο a- priori σχήμα. Αφ' ετέρου, η τρίτη προσέγγιση στοχεύει στην εύρεση του καλύτερου σχήματος ομαδοποίησης που ένας αλγόριθμος συγκέντρωσης μπορεί να καθοριστεί κάτω από ορισμένες υποθέσεις και παραμέτρους.

3 Ο ΑΛΓΟΡΙΘΜΟΣ STING

STatistical INformation Grid-based

3.1 ΕΙΣΑΓΩΓΗ

Γενικά, εξόριξη χωρικών δεδομένων (spatial data mining), ή ανακάλυψη γνώσης μέσα σε βάσεις χωρικών δεδομένων (databases) είναι η εξαγωγή της υπονοούμενης γνώσης, χωρικών σχέσεων και ανακάλυψη ενδιαφερόντων χαρακτηριστικών και σχεδίων που δεν αναπαριστούνται ρητά στις βάσεις δεδομένων. Αυτές οι τεχνικές μπορούν να παίξουν σημαντικό ρόλο στην κατανόηση των χωρικών δεδομένων και στη σύλληψη των εγγενών σχέσεων μεταξύ χωρικών (spatial) και μη χωρικών δεδομένων (nonspatial). Επιπλέον, τέτοιες σχέσεις μπορούν να χρησιμοποιηθούν για να παρουσιάσουν τα στοιχεία κατά τρόπο συνοπτικό και να αναδιοργανώσουν τις χωρικές βάσεις δεδομένων για να προσαρμόσουν τα σημασιολογικά στοιχεία και να επιτευχτεί υψηλότερη απόδοση. Η εξόριξη χωρικών δεδομένων (spatial data mining) έχει ευρείες εφαρμογές σε πολλούς τομείς, όπως Συστήματα GIS, εξερεύνηση βάσεων δεδομένων εικόνας, ιατρική απεικόνιση.

Το ποσό των χωρικών δεδομένων που λαμβάνονται από δορυφόρο, ιατρική απεικόνιση και από άλλες πηγές έχει αυξηθεί σημαντικά τα τελευταία χρόνια. Μια μεγάλη πρόκληση όσο αναφορά τη εξόριξη χωρικών δεδομένων είναι η αποδοτικότητα των αλγόριθμων κατά την εξόριξη χωρικών δεδομένων λόγω του συχνά τεράστιου ποσού χωρικών στοιχείων και τη πολυπλοκότητα των χωρικών τύπων και χωρικών μεθόδων. Σε αυτό το κεφάλαιο παρουσιάζουμε τον αλγόριθμο STING(STatistical INformation Grid-based method) Έναν αλγόριθμο που βασίζεται στο πλέγμα και μπορεί να επεξεργαστεί πολλές «region oriented» ερωτήσεις σε ένα σύνολο σημείων.

3.2 GRID CELL HIERARCHY

3.2.1 HIERARCHICAL STRUCTURE

Διαιρούμε τη χωρική περιοχή σε ορθογώνια κελιά (π.χ., χρησιμοποιώντας το γεωγραφικό πλάτος και το γεωγραφικό μήκος) και υιοθετούμε μια ιεραρχική δομή. Αφήστε τη ρίζα (root) ιεραρχίας να είναι στο επίπεδο 1 της ιεραρχίας και τα παιδιά (clildren) του σε επίπεδο 2, κ.λπ. Ένα κελί στο επίπεδο ι αντιστοιχεί στην ένωση των περιοχών των παιδιών του σε επίπεδο ι + 1. Σε αυτό το κεφάλαιο κάθε κελί (εκτός από τα φύλλα) έχει 4 παιδιά και κάθε παιδί αντιστοιχεί σε ένα τεταρτημόριο του γονικού κελιού. Η ρίζα (root) στο επίπεδο 1 αντιστοιχεί σε ολόκληρη τη χωρική περιοχή (που υποθέτουμε είναι ορθογώνια για απλότητα). Το μέγεθος των κελιών στα επίπεδα των φύλλων εξαρτώνται από την πυκνότητα των αντικειμένων. Εμπειρικά, επιλέγουμε ένα μέγεθος έτσι ώστε ο μέσος αριθμός αντικειμένων σε κάθε κελί να είναι στη σειρά από μερικές δωδεκάδες έως μερικές χιλιάδες. Επιπλέον, ένας επιθυμητός αριθμός στρωμάτων θα μπορούσε να ληφθεί με την αλλαγή του αριθμού κελιών που διαμορφώνουν ένα υψηλότερου επιπέδου κελί. Σε αυτό το κεφάλαιο , θα χρησιμοποιήσουμε 4 ως προκαθορισμένη αξία εκτός αν διευκρινίζεται αλλιώς. Υποθέτουμε το διάστημά μας είναι δύο διαστάσεων αν και είναι πολύ εύκολο να γενικευτεί αυτή η ιεραρχική δομή σε μοντέλα περισσότερων διαστάσεων. Σε δύο διαστάσεις, η ιεραρχική δομή είναι φαίνεται στο σχήμα

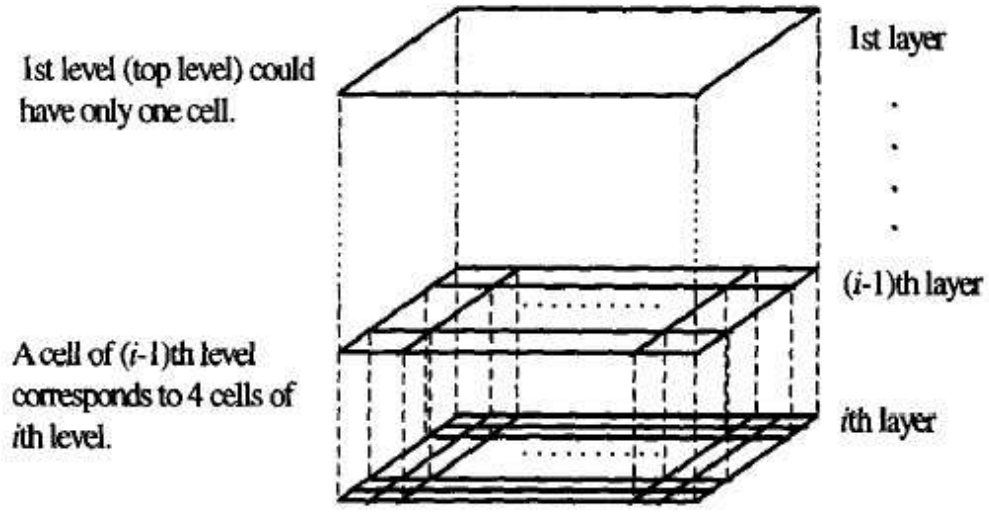


Figure 1: Hierarchical Structure

Για κάθε κελί, έχουμε εξαρτημένες και μη εξαρτημένες ιδιότητες για τις παραμέτρους. Η εξαρτημένη παράμετρος είναι:

N - Αριθμός αντικειμένων (σημεία) σε αυτό το κελί

Όσον αφορά στις παραμέτρους εξαρτώμενες από ιδιότητες, υποθέτουμε ότι για κάθε αντικείμενο, οι ιδιότητές του έχουν αριθμητικές τιμές.

Για κάθε αριθμητική ιδιότητα, έχουμε τις ακόλουθες τέσσερις παραμέτρους για κάθε κελί:

- m -μέσος όρος όλων των τιμών σε αυτό το κελί
- s - σταθερή απόκλιση όλων των τιμών των ιδιοτήτων σε αυτό το κελί
- min. - η ελάχιστη αξία των ιδιοτήτων σε αυτό το κελί
- Max - η μέγιστη αξία των ιδιοτήτων σε αυτό το κελί

3.2.2 PARAMETER GENERATION

Παράγουμε την ιεραρχία των κελιών με τις σχετικές παραμέτρους τους όταν φορτώνονται τα στοιχεία στη βάση δεδομένων. Οι τιμές n,n,s,min and max των κατώτατων επιπέδων υπολογίζονται άμεσα από τα στοιχεία. Οι παράμετροι των υψηλότερου επιπέδου κελιών μπορούν να υπολογιστούν εύκολα από τις παραμέτρους του χαμηλότερων επιπέδων κελιών. Έστω n,m,s,min,max, dist να είναι οι παράμετροι των συγκεκριμένων κελιών και ni, mi, si, mini, maxi και disti ναι είναι οι παράμετροι που αντιστοιχούν στα κατώτερα επίπεδα κελιών. Τα n,m,s,min και max μπορούν να υπολογιστούν ως εξής :

$$n = \sum_i n_i$$

$$m = \frac{\sum_i m_i n_i}{n}$$

$$s = \sqrt{\frac{\sum_i (s_i^2 + m_i^2) n_i}{n} - m^2}$$

$$min = \min_i (min_i)$$

$$max = \max_i (max_i)$$

3.3 ΤΥΠΟΙ ΕΡΩΤΗΜΑΤΩΝ

Αν οι στατιστικές πληροφορίες που είναι αποθηκευμένες στην ιεραρχική δομή του STING δεν είναι αρκετές να απαντήσουν ένα ερώτημα, τότε έχουμε αναδρομή στη βαθύτερη βάση δεδομένων. Για αυτό τον λόγο , μπορούμε να υποστηρίξουμε οποιοδήποτε ερώτημα που μπορεί να εκφραστεί με την κάτι σαν SQLγλώσσα που περιγράφεται παρακάτω σ΄ αυτό το κεφάλαιο. Παρόλα αυτά, οι στατιστικές πληροφορίες στη δομή του STING μπορούν να απαντήσουν πολλά συνηθισμένα ερωτήματα πολύ αποδοτικά και συχνά δεν χρειάζεται να προσπελάσουμε όλη τη βάση δεδομένων. Ακόμα και όταν οι στατιστικές πληροφορίες δεν επαρκούν για να απαντηθεί ένα ερώτημα , μπορούμε να περιορίσουμε το σύνολο πιθανών επιλογών.

Ο STING μπορεί να χρησιμοποιηθεί για να διευκολύνει διάφορα είδη ερωτημάτων χωρικών δεδομένων. Το πιο συχνό ερώτημα είναι το ερώτημα περιοχής, το οποίο ζητάει να επιλεγούν περιοχές που ικανοποιούν συγκεκριμένες συνθήκες (παράδειγμα 1). Ένα άλλο είδος ερωτήματος επιλέγει περιοχές και επιστρέφει μια λειτουργία της περιοχής, όπως το σύνολο μερικών ιδιοτήτων μέσα στην περιοχή (παραδειγμα 2).

Παράδειγμα 1. Επέλεξε το μέγιστο σύνολο περιοχών που έχουν τουλάχιστον 100 σπίτια ανά μονάδα περιοχής και τουλάχιστον 70% των τιμών των σπιτιών είναι πάνω απο 400 χιλ $ και με συνολική περιοχή τουλαχιστον 100 μονάδες με 90% ασφάλεια.

SELECT REGION

FROM house-map

WHERE DENSITY IN (100,∞)

AND price RANGE (400000.∞)

WITH PERCENT(0.7,1)

AND AREA (100,∞)

AND WITH CONFIDENCE 0.9

Παράδειγμα 2. Επέλεξε την ηλικιακή κλίμακα των σπιτιών στις μέγιστες περιοχές όπου υπάρχοθν τουλάχιστον 100 σπίτια ανά μονάδα περιοχής και τουλάχιστον 70% των σπιτιών έχουν τιμή μεταξύ 150 χιλ $ και 300 χιλ $ με περιοχή τουλάχιστον 100 μονάδες στην Καλιφόρνια.

SELECT REGION

FROM house-map

WHERE DENSITY IN (100,∞)

AND price RANGE (150000.300000)

WITH PERCENT(0.7,1)

AND AREA (100,∞)

AND LOCATION California

3.4 ΑΛΓΟΡΙΘΜΟΣ

Με την ιεραρχική δομή των κελιών πλέγματος έτοιμη προς χρήση, μπορούμε να χρησιμοποιήσουμε μια από επάνω προς τα κάτω προσέγγιση απάντησης στις ερωτήσεις εξόρυξης χωρικών δεδομένων. Για κάθε ερώτηση, αρχίζουμε με την εξέταση των κελιών σε ένα υψηλού επιπέδου στρώμα. Σημειώστε ότι δεν είναι απαραίτητο να αρχίσει με τη ρίζα μπορούμε να αρχίσουμε από ένα ενδιάμεσο στρώμα.

Αρχίζοντας με τη ρίζα, υπολογίζουμε την πιθανότητα ότι αυτό το κελί είναι σχετικό με την ερώτηση σε κάποιο επίπεδο εμπιστοσύνης που χρησιμοποιεί τις παραμέτρους αυτού του κελιού (ακριβώς πώς αυτό υπολογίζεται περιγράφεται αργότερα). Αυτή η πιθανότητα μπορεί να οριστεί ως το ποσοστό των αντικειμένων σε αυτό το κελί που ικανοποιούν τους όρους ερώτησης. Αφότου λάβουμε το διάστημα που θέλουμε, ονομάζουμε αυτό το κελί για να είμαστε σχετικοί ή μη σχετικοί στο διευκρινισμένο επίπεδο εμπιστοσύνης. Όταν τελειώνουμε το τρέχον στρώμα, προχωράμε στο επόμενο χαμηλότερο επίπεδο κελιών και επαναλαμβάνουμε την ίδια διαδικασία. Η μόνη διαφορά είναι ότι αντί να περάσουμε από όλα τα κελιά, εξετάζουμε μόνο εκείνα τα κελιά που είναι παιδιά των σχετικών κελιών του προηγούμενου στρώματος. Αυτή η διαδικασία συνεχίζεται έως ότου τελειώνουμε στο χαμηλότερο στρώμα επιπέδων.

Στις περισσότερες περιπτώσεις αυτά τα σχετικά κελιά και οι σχετικές στατιστικές πληροφορίες τους είναι αρκετά να δώσουν ένα ικανοποιητικό αποτέλεσμα στην ερώτηση. Κατόπιν, βρίσκουμε όλες τις περιοχές διαμορφωμένες από τα σχετικά κελιά και τις επιστρέφουμε. Εντούτοις, σε σπάνιες περιπτώσεις (οι άνθρωποι μπορούν να θελήσουν το πολύ ακριβές αποτέλεσμα για πρόσθετους λόγους, π.χ. στρατιωτικούς), αυτές οι πληροφορίες δεν είναι αρκετές να απαντήσουν στην ερώτηση. Κατόπιν, πρέπει να ανακτήσουμε εκείνα τα στοιχεία που περιέχονται

στα σχετικά κελιά από τη βάση δεδομένων και κάνουν κάποια περαιτέρω επεξεργασία.

Αφότου έχουμε ονομάσει όλα τα κελιά ως σχετικά ή μη σχετικά, μπορούμε εύκολα να βρούμε όλες τις περιοχές που ικανοποιούν την πυκνότητα που διευκρινίζεται από μια BFS αναζήτηση (breadth-first search). Για κάθε σχετικό κελί, εξετάζουμε τα κελιά μέσα σε μια ορισμένη απόσταση από το κέντρο του τρέχοντος κελιού που βλέπει εάν η μέση πυκνότητα μέσα σε αυτήν την μικρή περιοχή είναι μεγαλύτερη από την πυκνότητα που διευκρινίζεται. Σε αυτή την περίπτωση, αυτή η περιοχή μαρκάρετε και όλα τα σχετικά κελιά που εξετάσαμε ακριβώς μπαίνουν σε μια ουρά αναμονής (queue).

Κάθε φορά παίρνουμε ένα κελί από την ουρά και επαναλαμβάνουμε την ίδια διαδικασία, μονο που συγκεκριμένα κελιά που είναι σχετικά με το ερώτημα και δεν έχουν εξεταστεί νωρίτερα μπαίνουν στην σειρά. Όταν η σειρά αναμονής είναι κενή, έχουμε προσδιορίσει μια περιοχή. Η απόσταση που χρησιμοποιούμε ανωτέρω υπολογίζεται από τη διευκρινισμένη πυκνότητα του κελιού κατώτατου επιπέδου. Σαν αποτέλεσμα, αυτή η απόσταση μπορεί μόνο να φθάσει στα γειτονικά κελιά. Σε αυτήν την περίπτωση, πρέπει ακριβώς να εξετάσουμε τα γειτονικά κελιά και να βρούμε τις περιοχές που διαμορφώνονται από τα συνδεδεμένα κελιά.

Παραδείγματος χάριν, εάν τα αντικείμενα στη βάση δεδομένων μας είναι σπίτια και η τιμή είναι μια από τις ιδιότητες, κατόπιν ένα είδος ερώτησης θα μπορούσε να είναι «βρες τις περιοχές με περιοχή τουλάχιστον A όπου ο αριθμός σπιτιών ανά περιοχή μονάδων είναι τουλάχιστον c και τουλάχιστον β% των σπιτιών έχουν την τιμή μεταξύ του α και του b με (1 - α) εμπιστοσύνης» όπου a < b. Εδώ, το α θα μπορούσε να τείνει στο μείον άπειρο και το b θα μπορούσε να τείνει στο συν άπειρο. Αυτή η ερώτηση μπορεί να γραφτεί ως εξής :

SELECT REGION

FROM house-map

WHERE DENSITY IN [c, -)

AND price RANGE [a, b] WITH PERCENT [p%, l]

AND AREA [A, -)

AND WITH CONFIDENCE 1 – a

Αλγόριθμος στατιστικών πληροφοριών βασισμένος σε πλέγμα:

1. Καθορίστε ένα στρώμα για να αρχίσετε.

2. Για κάθε κελί αυτού του στρώματος, υπολογίζουμε το διάστημα εμπιστοσύνης (ή την κατ' εκτίμηση σειρά) της πιθανότητας ότι αυτό το κελί είναι σχετικό με την ερώτηση.

3. Από το διάστημα που υπολογίζεται ανωτέρω, ονομάζουμε το κελί σχετικό ή μη σχετικό.

4. Εάν αυτό το στρώμα είναι το κατώτατο στρώμα, πηγαίνετε στο βήμα 6 διαφορετικά, πηγαίνετε στο βήμα 5.

5. Πηγαίνουμε κάτω από τη δομή ιεραρχίας από ένα επίπεδο. Πηγαίνετε στο βήμα 2 για εκείνα τα κελιά που διαμορφώνουν τα σχετικά κύτταρα του υψηλότερου επιπέδου στρώματος.

6. Εάν η προδιαγραφή της ερώτησης συναντιέται, πηγαίνετε στο βήμα 8 διαφορετικά, πηγαίνετε στο βήμα 7.

7. Ανακτηστε τα στοιχεια των σχετικων κελιων και κανετε περαιτερω επεξεργασια. Επιστρέψτε το αποτέλεσμα που καλύπτει την απαίτηση της ερώτησης. Πηγαίνετε στο βήμα 9.

8. Βρείτε τις περιοχές των σχετικών κελιων. Επιστρέψτε εκείνες τις περιοχές που καλύπτουν την απαίτηση της ερώτησης. Πηγαίνετε στο βήμα 9.

9. τελος

3.5 Ανάλυση του Αλγορίθμου STING.

Στον παραπάνω αλγόριθμο, το βήμα 1 χρειάζεται σταθερό χρόνο. Τα Βήματα 2 και 3 απαιτούν σταθερό χρόνο ώστε κάθε κελί να υπολογίσει το διάστημα εμπιστοσύνης ή την αναλογία διακύμανσης καθώς επίσης και ένα σταθερό χρόνο για να ονομαστεί το κελί ώς σχετικό ή μη σχετικό. Το οποίο σημαίνει ότι χρειαζόμαστε σταθερό χρόνο για τα βήματα 1, 2 και 3. Ο συνολικός χρόνος είναι μικρότερος ή ίσος με τον συνολικό αριθμό των κελιων στον γράφο μας. Παρατηρήστε ότι ο συνολικός αριθμός κυττάρων είναι 1.33K, όπου K είναι ο αριθμός κελιών στο κατώτατο στρώμα. Χρησιμοποιούμε τον παράγοντα 1,33 επειδή ο αριθμός των κελιά ενός στρώματος είναι πάντα το ένα τέταρτο του αριθμού κελιών στο στρώμα του επομένου επίπεδου. Οπότε η συνολική πολυπλοκότητα είναι O(K). Αλλά στην πραγματικότητα χρειάζεται να εξεταστούν λιγότερα κελιά καθώς πολλά κελιά των ανώτερων επιπέδων είναι μή σχετικά. Στο βήμα 8, ο χρόνος που χρειάζεται για να σχηματιστεί μία

περιοχή είναι γραμμικά ανάλογος του αριθμού των κελιών. Οπότε η πολυπλοκότητα του βήματος 8 είναι O(K). Συνήθως, το βήμα 7 δεν είναι αναγκαίο, οπότε η συνολική πολυπλοκότητα είναι O(K). Ακόμα και σε περίπτωση που είναι αναγκαίο να κάνουμε το βήμα 7, δεν χρειάζεται να ανακτησουμε όλα τα δεδομένα. Οπότε ο χρόνος που χρειάζεται σε αυτό το βήμα είναι μικρότερος από γραμμικός. Εν κατακλείδι ο αλγόριθμος αυτός υπερέχει των άλλων.

3.6 Ποιοτική ανάλυση του Αλγορίθμου STING.

Ο Αλγόριθμος STING χρησιμοποιεί στατιστικές πληροφορίες για να προσεγγίσει τα αναμενόμενα αποτελέσματα κάθε ερώτησης. Επομένως, θα μπορούσε να είναι ανακριβής δεδομένου ότι τα δεδομένα θα μπορούσαν να είναι αυθαίρετα τοποθετημένα. Εντούτοις, κάτω από τον ακόλουθο ικανοποιητικό όρο, ο STING μπορεί να εγγυηθεί ότι εάν μια περιοχή ικανοποιεί τον ορισμό της ερώτησης, τότε θα επιστρέψει αποτέλεσμα.

Ορισμός 1: Έστω F μια περιοχή. Το πλάτος του F ορίζεται ως το μέγεθος της πλευράς του μεγαλύτερου τετραγώνου που μπορεί να χωρέσει στην περιοχή F.

Επαρκής Προυπόθεση:

Έστω A η ελάχιστη περιοχή και c η πυκνότητα που ορίζεται από το ερώτημα. Έστω R η περιοχή που ικανοποιεί το ερώτημα με W το πλάτος της. Εάν W2 - 4([W/L + 1])≥ A, όπου L είναι το μέγεθος της πλευράς του κατωτέρου επιπέδου κελιών, τότε ο STING πρέπει να επιστρέψει το R.

Έστω S το μέγιστο τετράγωνο στην R με W το πλάτος του. Έστω I το σύνολο των κελίων του κατωτέρου επιπέδου που εμπεριέχονται στην στο S. Το I χωρίζεται σε δύο υποσύνολα I1 και I2. I1 είναι τα κελία που βρίσκονται στα όρια του S και I2 τα κελία που εμπεριέχονται στο S. Είναι προφανές ότι τα κελιά του I1 συνδέονται. Ένα τμήμα γραμμών μήκους W μπορεί να διασχύσει το πολύ [W/L + 1] κελία κατωτέρου επιπέδου. Οπότε ο συνολικός αριθμός στοιχείων του I1 είναι το πολύ 4[W/L + 1]. Το συνολικό εμβαδόν των κελιών στο I1 είναι το πολύ 4[W/L + 1]L2 και το εμβαδόν της περιοχής S είναι W2. Οπότε το ελάχιστο εμβαδόν των κελιών του I2 είναι W2 - 4[W/L + 1]L2 ο STING μπορεί να ονομάσει τα κελιά του I2 ως σχετικά. Και εφόσον W2 - 4[W/L + 1]L2 ≥ A , ο STING είναι σίγουρο ότι θα επιστρέψει το R.

3.7 Η οριακή συμπεριφορά του STING είναι αντίστοιχη του DBSCAN.

Οι περιοχές που επιστρέφει ο STING αποτελούν μία προσέγγιση του αποτελέσματος του DBSCAN. Όσο η κοκκοποίηση προσεγγίζει το μηδέν, οι περιοχές που επιστρέφει ο STING προσεγγίζουν το αποτέλεσμα του DBSCAN. Προκειμένου να συγκρίνουμε τον STING με τον DBSCAN, χρησιμοποιούμε μόνο τον αριθμό των σημείων εφόσον ο DBSCAN ομαδοποιεί τα σημεία με βάση την χωρική θέση. Ο DBSCAN έχει δύο παραμέτρους: το Eps και το MinPts. Στην δικιά μας περίπτωση ο STING έχει μόνο μία παράμετρο την πυκνότητα c.

Θέτουμε $c = \frac{\text{MinPts}+1}{\text{Eps}^2 \cdot \pi} = \frac{k+1}{\text{Eps}^2 \cdot \pi}$ για να προσεγγίσουμε τα αποτελέσματα του DBSCAN. Ο λόγος: η πυκνότητα κάθε περιοχής μέσα σε μία συστάδα που ανιχνεύεται από τον DBSCAN είναι το ελάχιστον $\frac{\text{MinPts}+1}{\text{Eps}^2 \cdot \pi}$ εφόσον για κάθε σημείο υπάρχουν τουλάχιστον MinPts σημεία σε απόσταση Eps. Στον STING, για κάθε κελί, εάν $n < S \times c$ (όπου n ο αριθμός σημείων του κελιού και S η περιοχή κάθε κελιού στο κατώτατο επίπεδο) ονομάζουμε το σημείο ως μή σχετικό, αλλιώς το ονομάζουμε σχετικό. Όταν σχηματίζουμε τις περιοχές από σχετικά κελιά, θέτουμε την εξεταζόμενη απόσταση ώς $d = \max(l, \sqrt{\frac{k+1}{c\pi}})$, όπου l είναι το μέγεθος της πλευράς του κατωτέρου επιπέδου κελιών. Όταν η κοκκοποίηση είναι πολύ μικρή το $\sqrt{\frac{k+1}{c\pi}}$ είναι το μέγιστο. Όταν η κοκκοποίηση προσεγγίζει το μηδέν το μέγεθος κάθε κελιού πλησιάζει το μηδέν. Οπότε εάν υπάρχει έστω και ένα σημείο στο κελί, το κελί ονομάζεται ώς σχετικό. Αυτό που πρέπει να γίνει στην συνέχεια είναι να δημιουργήσουμε την απόσταση που θα επιστραφεί σε σχέση με την απόσταση και την πυκνότητα. Μπορούμε να δούμε ότι

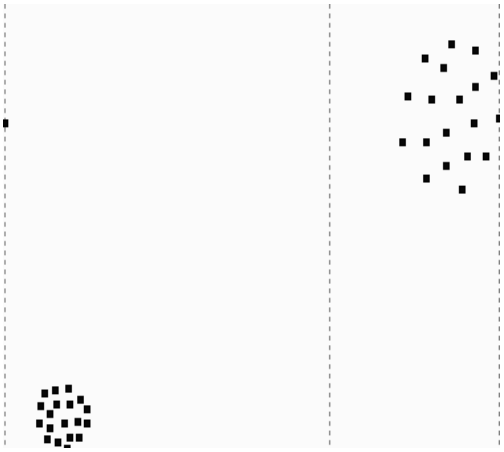
$$d = \sqrt{\frac{k+1}{c\pi}} = \sqrt{\frac{k+1}{\frac{k+1}{\text{Eps}^2 \cdot \pi} \cdot \pi}} = \text{Eps}.$$

Για κάθε σχετικό κελί εξετάζουμε την περιοχή γύρω του σε απόσταση d για να δούμε αν η πυκνότητα είναι μεγαλύτερη από c. Το οποίο είναι αντίστοιχο με το να ελέγχεις εάν ο αριθμός των σημείων είναι μεγαλύτερος από $c \times \pi d^2 = k + 1$. Δηλαδή τα αποτελέσματα του STING προσεγγίζουν αυτά του DBSCAN όταν η κοκκοποίηση πλησιάζει το μηδέν.

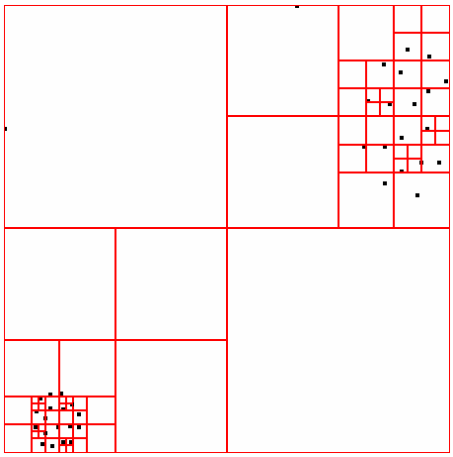
4 ΑΠΟΤΕΛΕΣΜΑΤΑ

Αποτελέσματα σε διάφορα dataset για παράμετρο density = 300

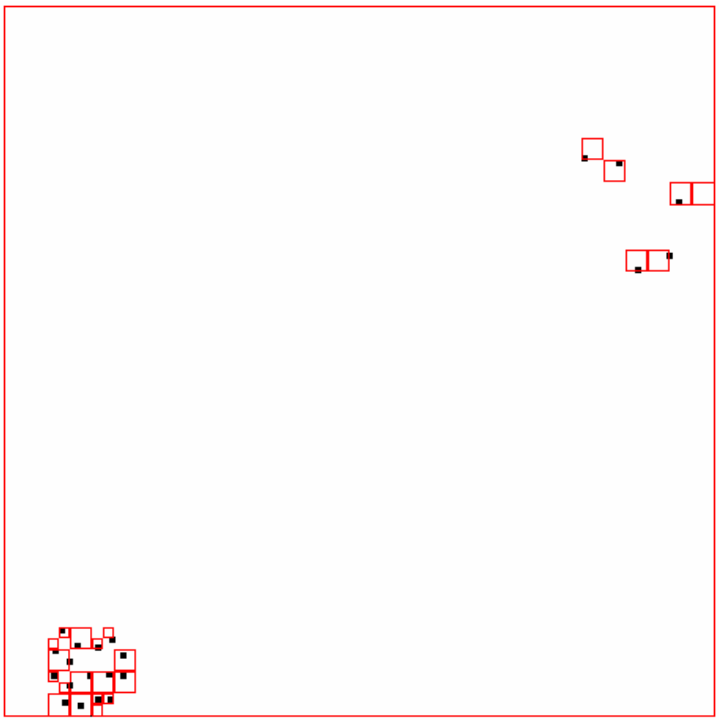
DATASET 1



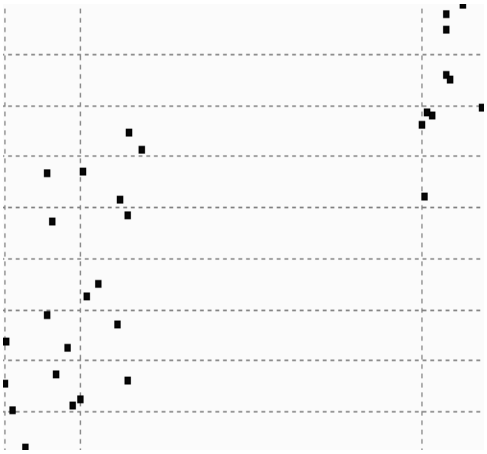
QUAD-TREE



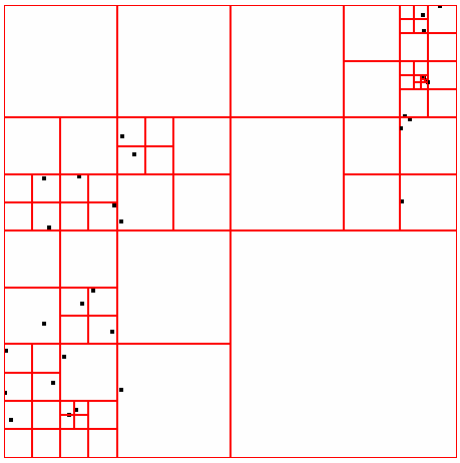
STING RESULT



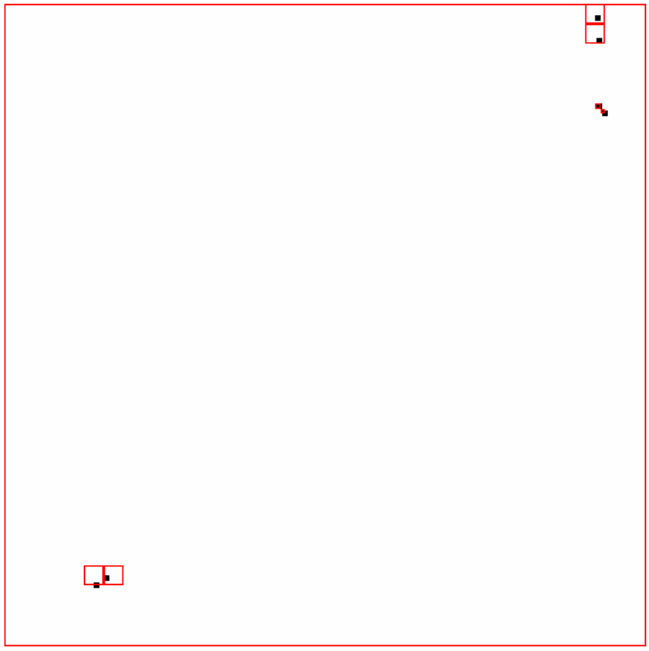
DATASET 2



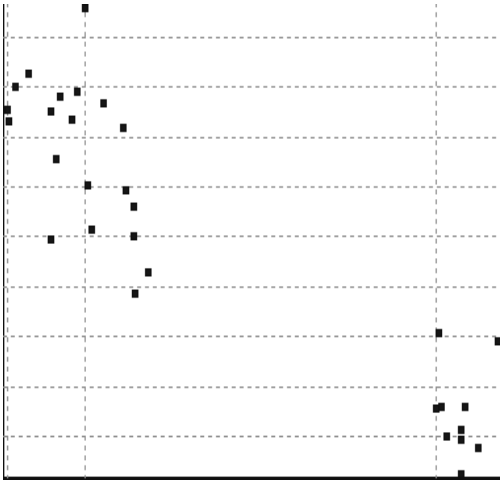
QUAD-TREE



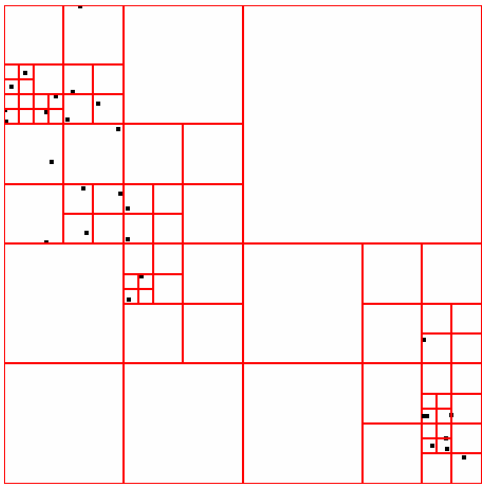
STING RESULT



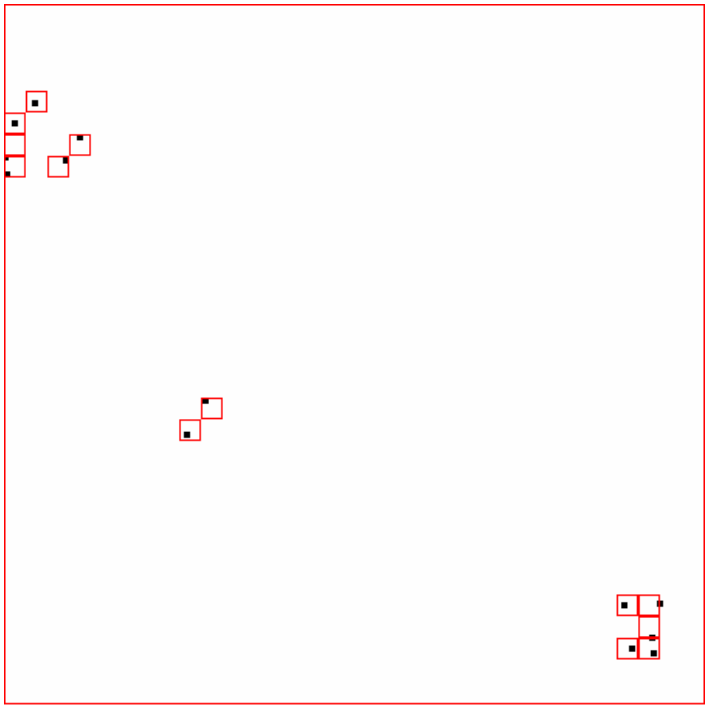
DATASET 3



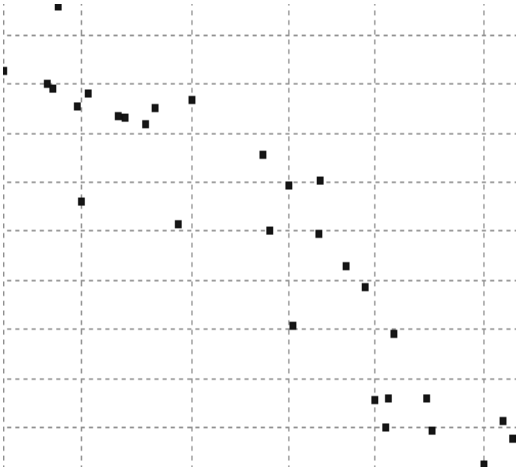
QUAD-TREE



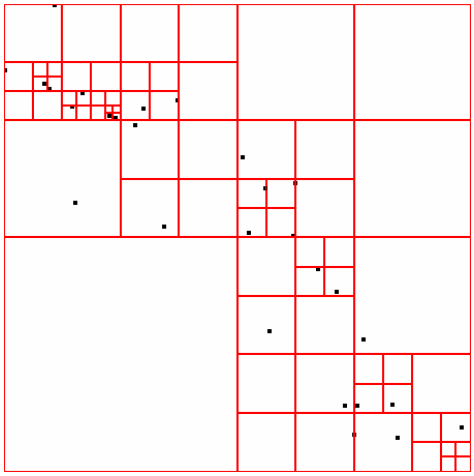
STING RESULT



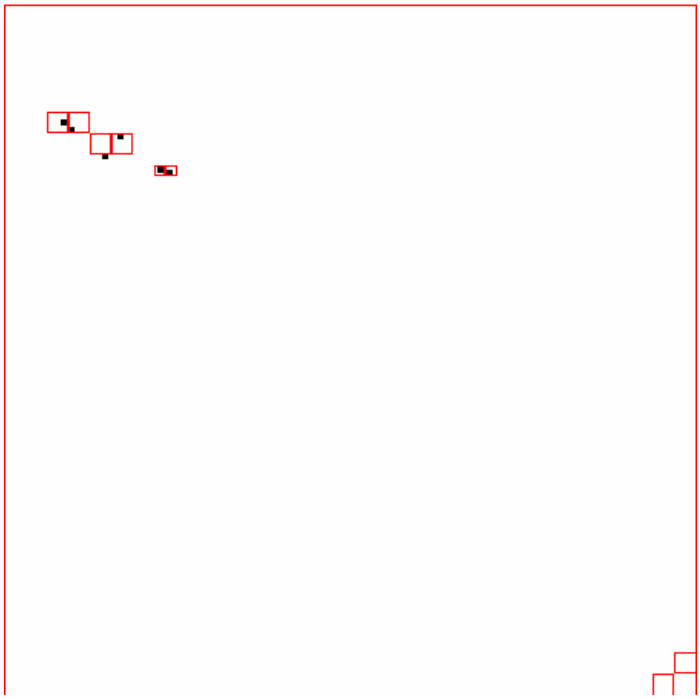
DATASET 4



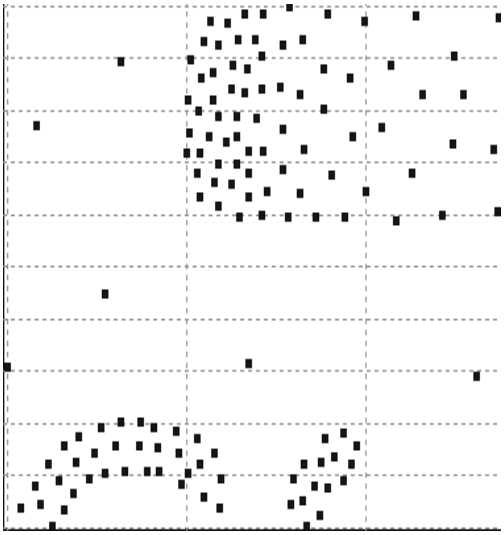
QUAD-TREE



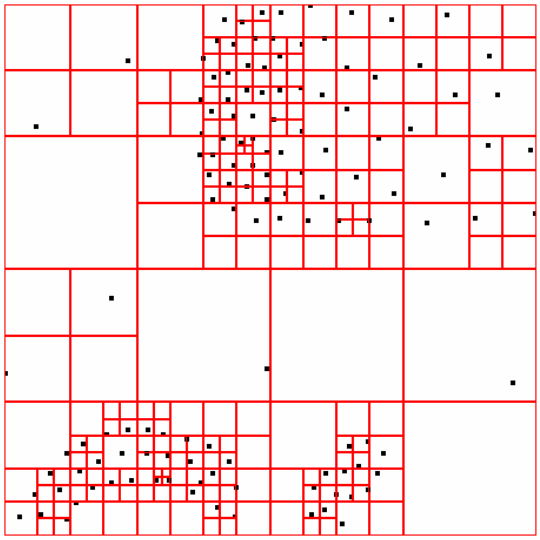
STING RESULT



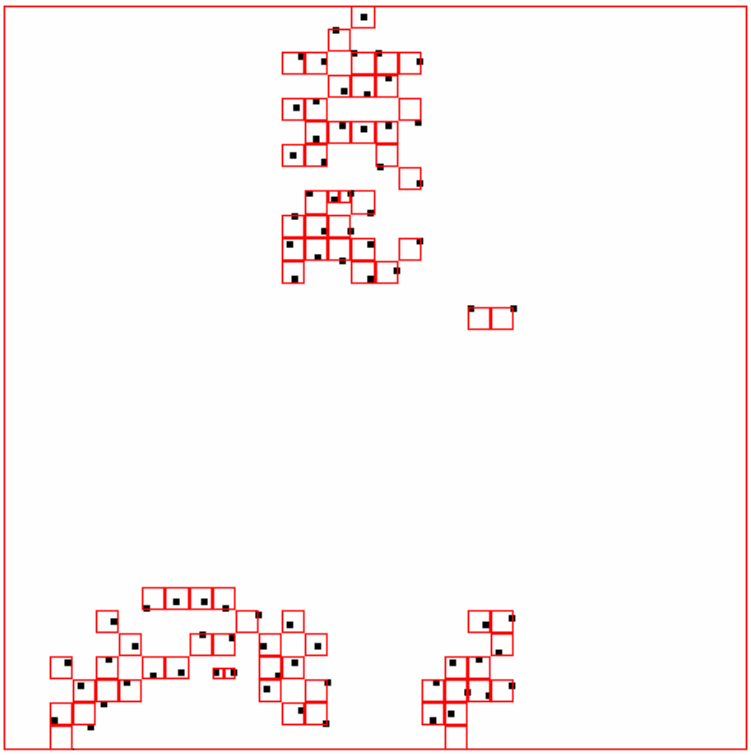
DATASET 52



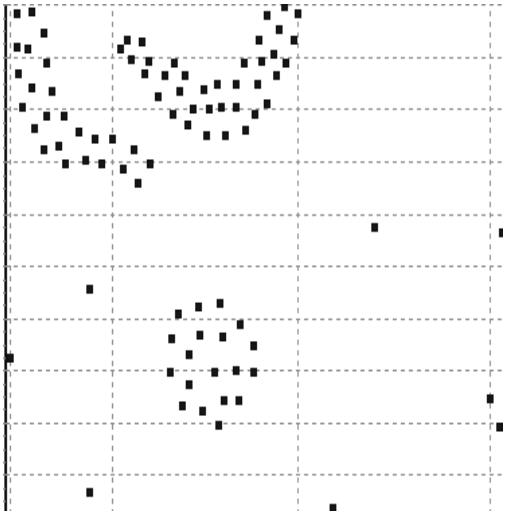
QUAD-TREE



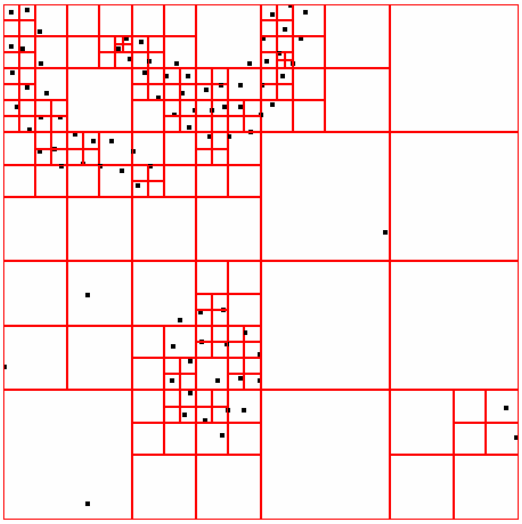
STING RESULT



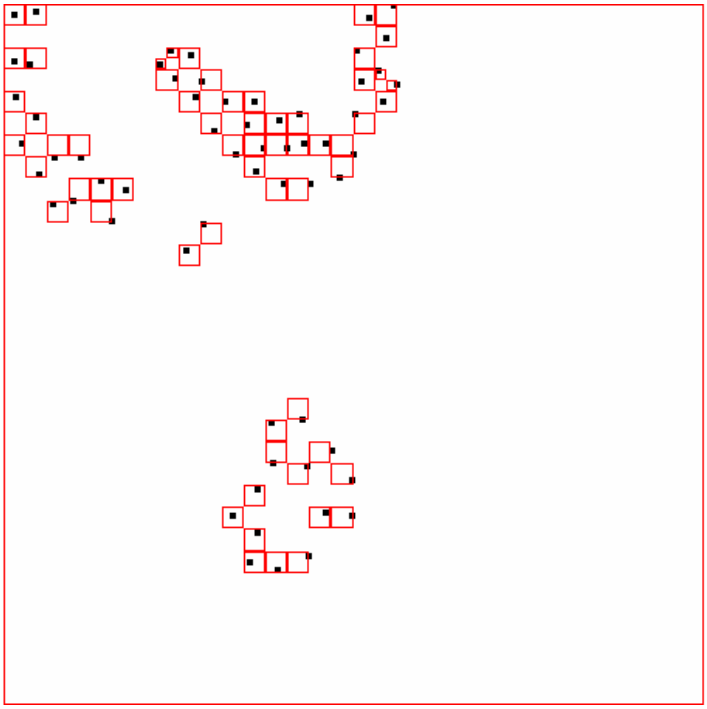
DATASET 53



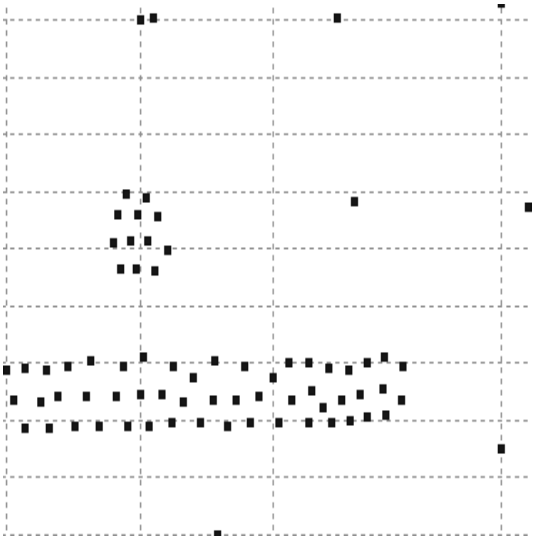
QUAD-TREE



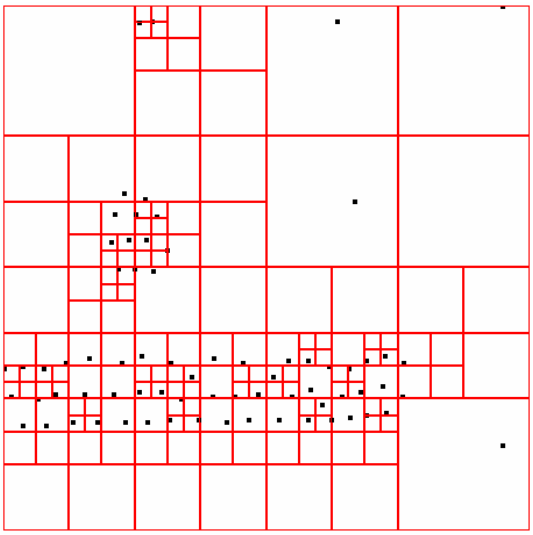
STING RESULT



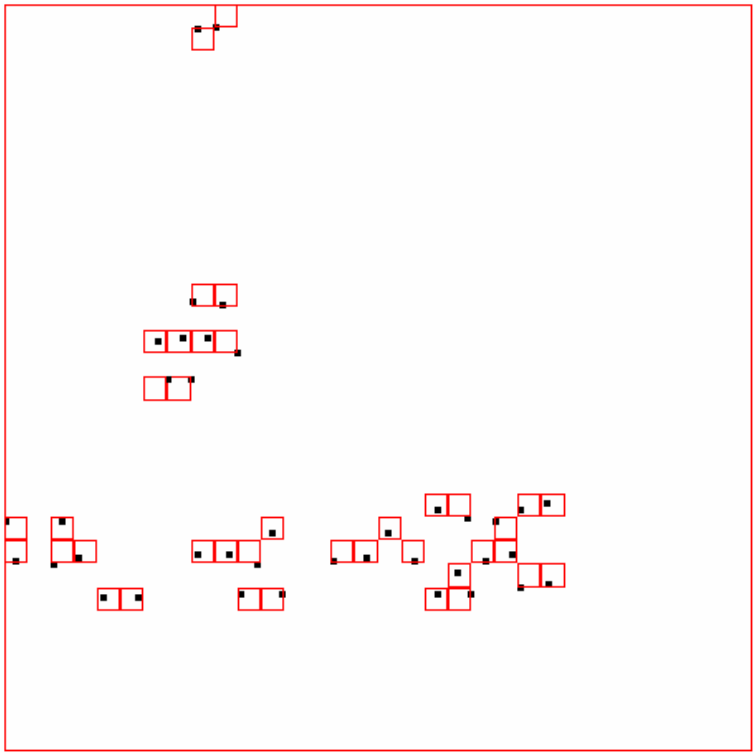
DATASET 54



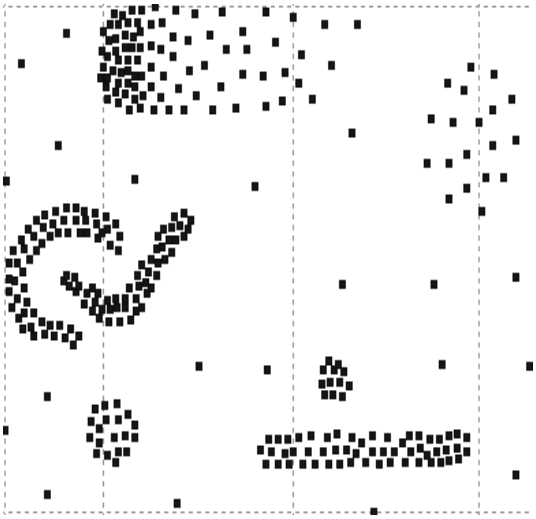
QUAD-TREE



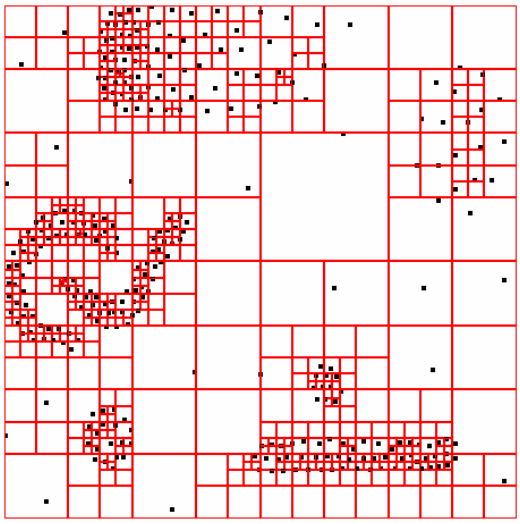
STING RESULT



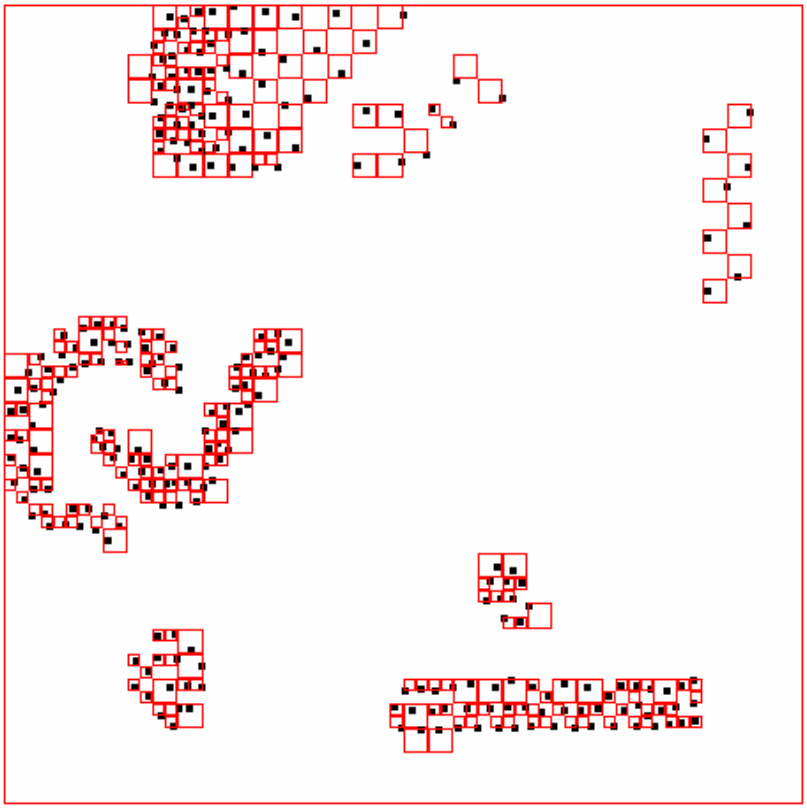
DATASET 6



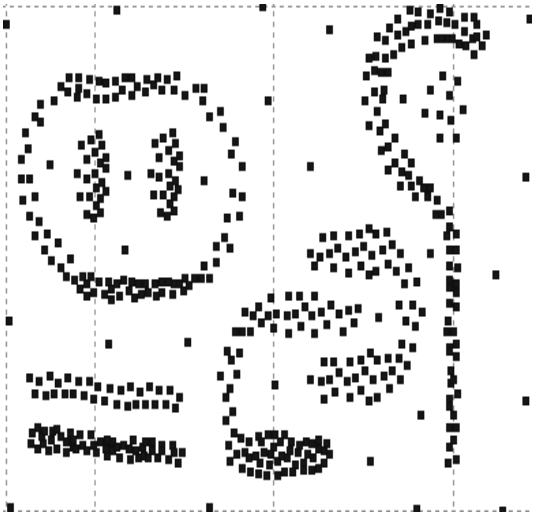
QUAD-TREE



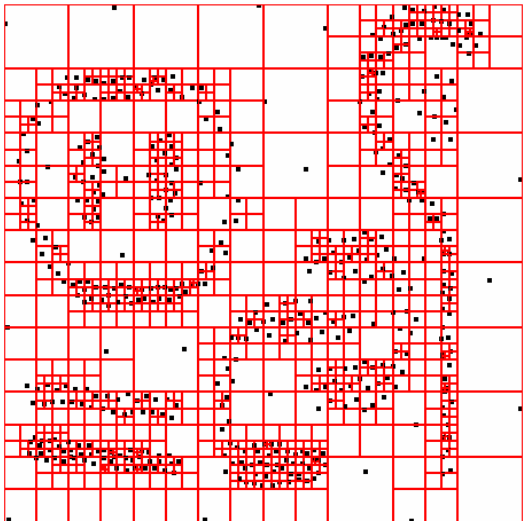
STING RESULT



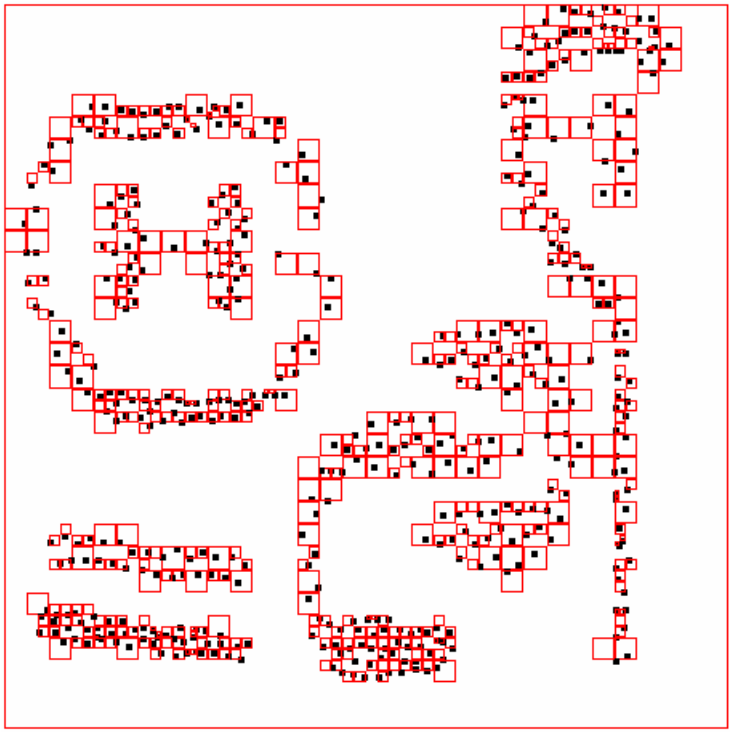
DATASET 7



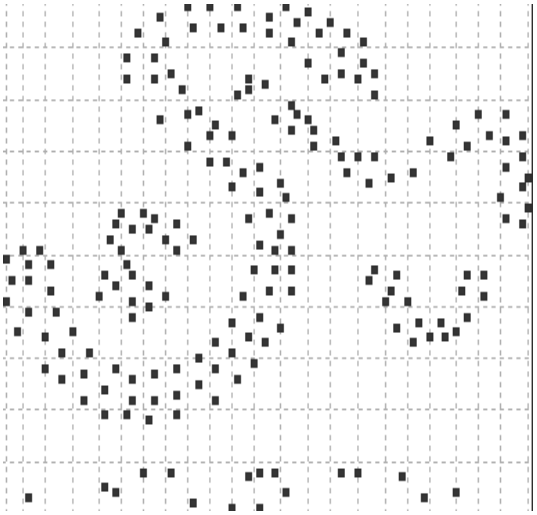
QUAD-TREE



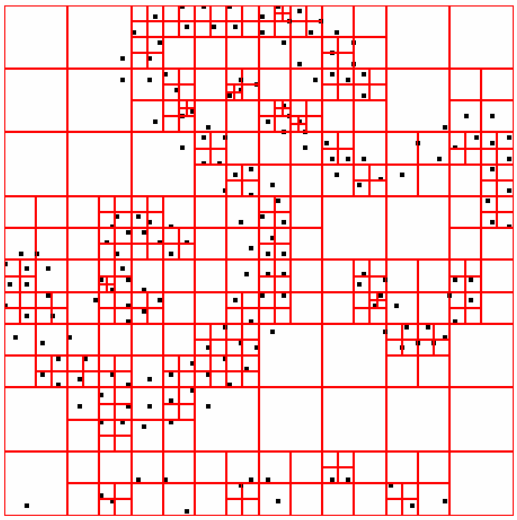
STING RESULT



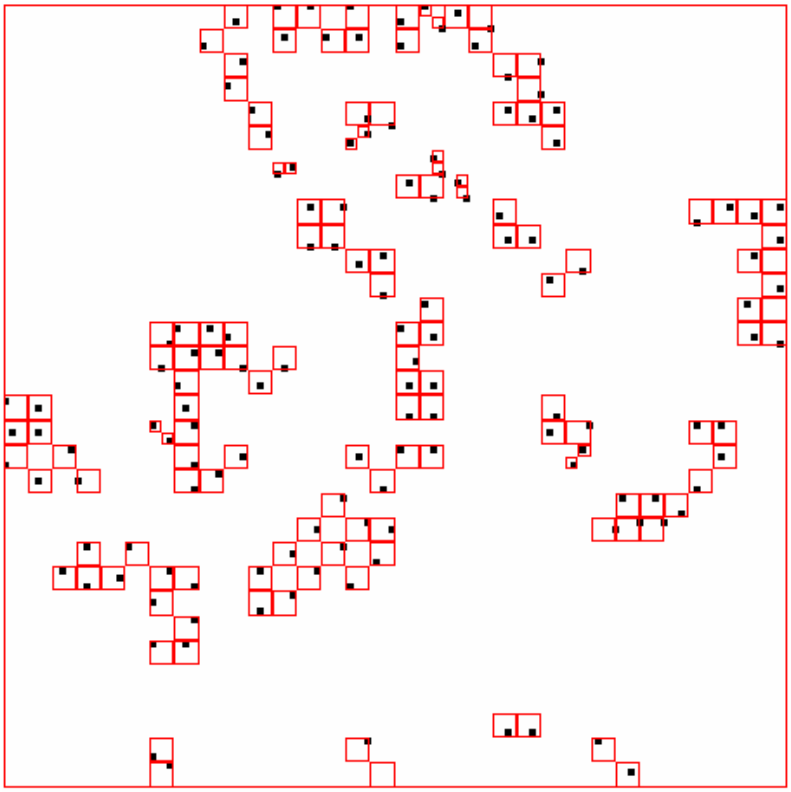
DATASET 8



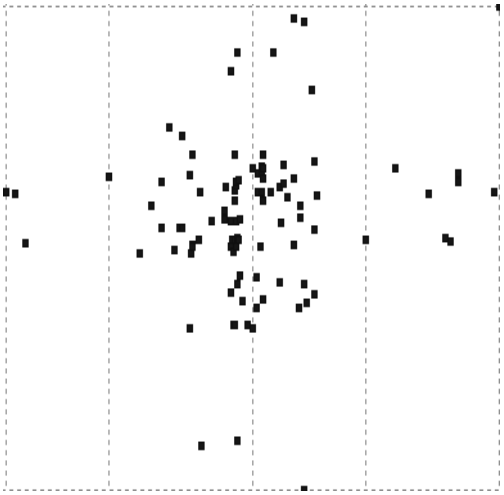
QUAD-TREE



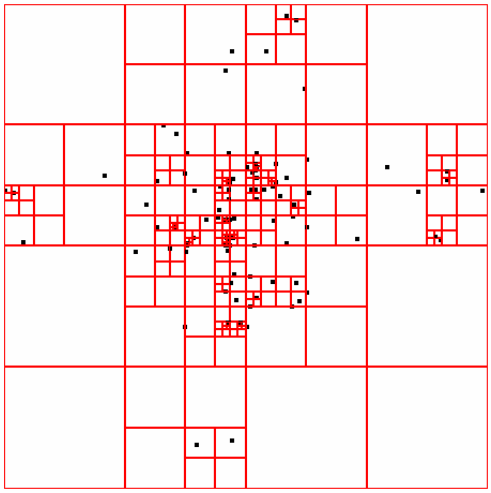
STING RESULT



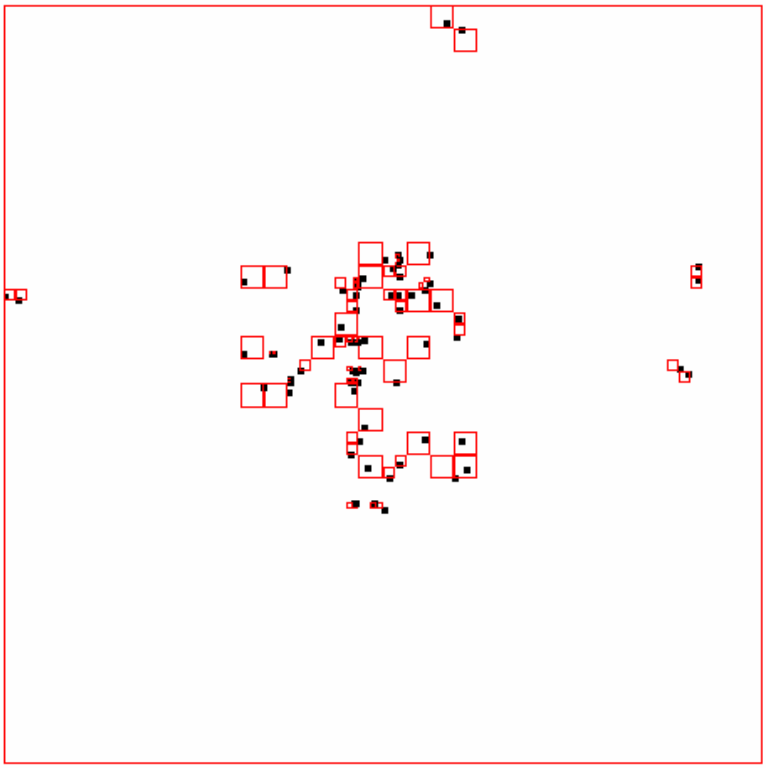
DATASET 9



QUAD-TREE



STING RESULT



5 ΚΩΔΙΚΑΣ ΕΦΑΡΜΟΓΗΣ

5.1 ΥΛΟΠΟΙΗΜΕΝΕΣ ΜΕΘΟΔΟΙ

void setnode(struct Node *xy, double x, double y, double w, double h)

Η συγκεκριμένη μέθοδος αρχικοποιεί ένα κόμβο με τις τιμές που δίνουμε από τα ορίσματα. Όπου double x και double y είναι το σημείο x,y και double w,double h είναι το πλάτος κ ύψος αντίστοιχα.

double findMinX(struct Node* n)

Η findMinX() παίρνει ως όρισμα ένα Node n και επιστρέφει την μικρότερη συντεταγμένη X μέσα στο Node.

double findMaxX(struct Node* n)

Η findMaxX() παίρνει ως όρισμα ένα Node n και επιστρέφει την μεγαλύτερη συντεταγμένη X μέσα στο Node.

double findMinY(struct Node* n)

Η findMinY() παίρνει ως όρισμα ένα Node n και επιστρέφει την μικρότερη συντεταγμένη Y μέσα στο Node.

double findMaxX(struct Node* n)

Η findMaxY() παίρνει ως όρισμα ένα Node n και επιστρέφει την μεγαλύτερη συντεταγμένη Y μέσα στο Node.

void qtree:: calcStatistics(struct Node* n)

Η calcStatistics() παίρνει ως όρισμα ένα Node n και υπολογίζει τα stats για το συγκεκριμένο κόμβο. Η κλήση της γίνεται μέσω της BuildQuadTree(). Αυτό γίνεται διοτι στον αλγόριθμο STING για κάθε κελί καθορίζονται στατιστικές παράμετροι (μέση τιμή, διασπορά, τύπος κατανομής). Με τη λήψη αυτής της πληροφορίας μπορούν να απαντηθούν πολλά αιτήματα για εξόρυξη γνώσης από δεδομένα, εξετάζοντας τα στατιστικά που δημιουργήθηκαν για τα κελιά

int pointArray_size(struct Node *n)

Η παραπάνω μέθοδος παίρνει ως όρισμα μια δομή τύπου node και μου επιστρέφει το μέγεθος του πίνακα σημείων που υπάρχει στο συγκεκριμένο κόμβο. Αυτή η μέθοδος μου είναι απαραίτητη κατά την δημιουργία του δέντρου όπου βλέπω σε κάθε κόμβο πόσα σημεία έχω και κάνω μετέπειτα τις διασπάσεις.

void qtree:: FindNofNodes(Node *n, Node *nParent, int index)

Η παραπάνω μέθοδος χρησιμοποιείται για να υπολογίσουμε τον αριθμό των σημείων κάθε παιδιού, παίρνει ως ορίσματα το Node n, ο οποίος είναι το παιδί για το οποίο υπολογίζουμε τον αριθμό των σημείων την δεδομένη στιγμή, τον Node το nParent, ο οποίος είναι ο πατρικός Node και τελευταίο όρισμα είναι το index, το οποίο δηλώνει το παιδί για το οποίο υπολογίζουμε τα σημεία την δεδομένη στιγμή. Η συγκεκριμένη μέθοδος καλείται πριν την δέσμευση χώρου για τον υπολογισμό του αριθμού των σημείων που πρέπει να δεσμεύσουμε. Η διαδικασία είναι απλή, για κάθε σημείου του πατρικού Node ελέγχουμε αν είναι στα όρια του κάθε παιδιού.

void qtree:: PrintQuadTree(Node *n, int depth)

Η παραπάνω μέθοδος είναι μια βοηθητική για εμάς και μόνο. Αυτό που ουσιαστικά κάνει είναι να τυπώνει το quadTree που έχουμε δημιουργήσει. Με αυτό το τρόπο είναι πολύ εύκολο για εμάς να δούμε αν έχει δημιουργηθεί ένα σωστό δέντρο.

void qtree:: LoadFromFile(char* filename, struct Node* root)

Η παραπάνω μέθοδος LoadFromFile χρησιμοποιείται για να φορτώσουμε δεδομένα από ένα αρχείο εισόδου. Το πρώτο όρισμα είναι το filename το οποίο είναι το όνομα του αρχείου εισόδου, εάν το αρχείο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο τότε αρκεί μόνο το όνομα του αρχείου αλλιώς απαιτείται ολόκληρο το filepath. Το δεύτερο όρισμα είναι το Node* root το οποίο είναι και η ρίζα του δέντρου, για την οποία είναι απαραίτητο να έχει δεσμευτεί χώρος.

Το αρχείο εισόδου πρέπει να έχει συγκεκριμένη μορφή. Στην πρώτη γραμμή είναι απαραίτητο να υπάρχει ο αριθμός των σημείων. Οι επόμενες γραμμές είναι για τα σημεία, μία γραμμή για κάθε σημείο. Ένα σημείο ορίζεται από 2 δεκαδικούς αριθμούς, οπότε σε κάθε γραμμή πρέπει να υπάρχουν δύο δεκαδικοί αριθμοί με ένα κενό ανάμεσα τους.

Node* qtree:: BuildNode(Node *n, Node *nParent, int index)

Μια από τις πιο βασικές μεθόδους του προγράμματος μας είναι η παραπάνω. Σε γενικές γραμμές αυτό που κάνει είναι να δημιουργεί ένα box με τα σημεία για τον κόμβο. Εν συνεχεία καθορίζει ποιο σημείο είναι για κάθε box. Η θέση του κόμβου παιδί βασισμένη με index (0-3) καθορίζεται ως εξής :

| 1 | 0 |

| 2 | 3 |

Σε πρώτη φάση αρχικοποιούμαι το νέο κόμβο και έπειτα με τη χρήση μια switch καθορίζουμε τα νέα πλάτη και ύψη για το κάθε box. Σύμφωνα με τον αλγόριθμο το ύψος και το πλάτος του box για ένα παιδί είναι το μισό του ύψους και του πλάτους του πατρικού κόμβου.

Αφού έχουμε δημιουργήσει τα τεταρτημόρια προχωράμε στο επόμενο βήμα όπου τοποθετούμε τα σημεία στα τεταρτημόρια που αντιστοιχούν. Οπότε έχουμε μια ανάλογη με παραπάνω switch και ανάλογα με το τεταρτημόριο που πρέπει να είναι το κάθε σημείο δημιουργούμε ένα δυναμικό πίνακα και προσθέτουμε εκεί το σημείο. Εδώ αξίζει να σημειωθεί ότι αυτοί οι δυναμικοί πίνακες στο τέλος γίνονται delete. Διατηρούνται μόνο οι πίνακες που έχουν ένα σημείο μόνο μέσα και αυτοί είναι συνήθως οι πίνακες των τελευταίων επιπέδων κάθε τεταρτημόριου.

void qtree:: BuildQuadTree(Node *n)

Η μέθοδος δημιουργίας του quadTree. Αναδρομικά δημιουργούμε το δέντρο χρησιμοποιώντας εσωτερικά την μέθοδο BuildNode() που αναλύσαμε πιο πάνω. Ως όρισμα η BuildQuadTree παίρνει μια δομή τύπου node.

void qtree:: DeleteQuadTree(Node *n)

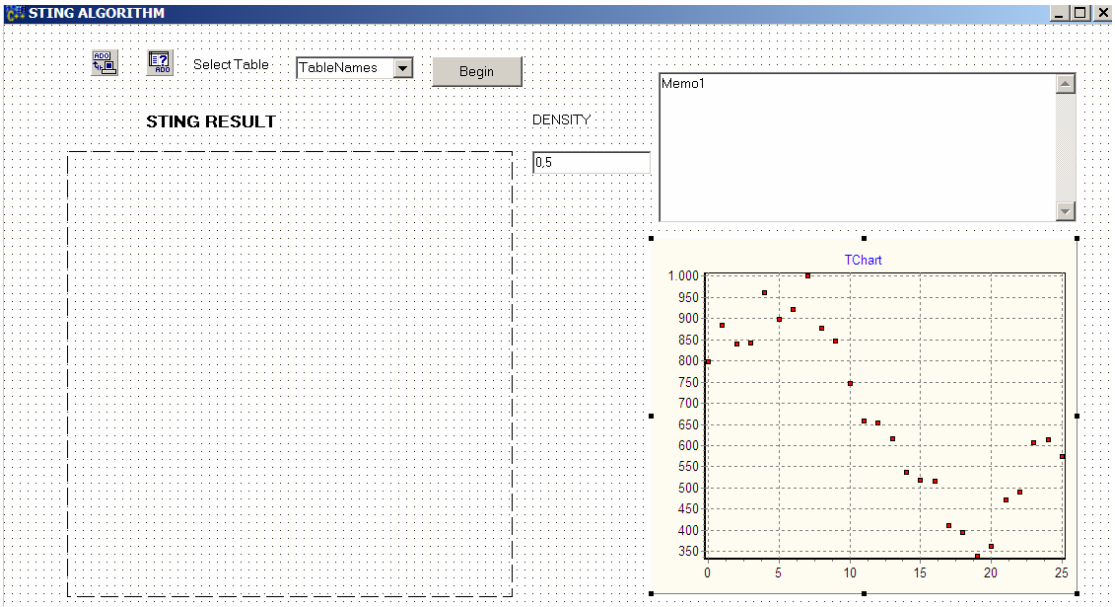
Όπως και με τη δημιουργία του δέντρου έτσι και εδώ έχουμε μια μέθοδο η οποία καταστρέφει αναδρομικά το δέντρο μας και απελευθερώνει τη δεσμευμένη μνήμη. Σαν όρισμα παίρνει τον πατρικό κόμβο.

void printStats(Node *n)

Μέθοδος που τυπώνει τα στατιστικά ενός κόμβου (μέση τιμή, μέγιστο, ελάχιστο, τυπική απόκλιση). Η κλήση της γίνεται μέσα από την αναδρομική Print Statistics().

void qtree:: PrintStatistics(Node *n, int depth)

Μέθοδος που τυπώνει όλο το δέντρο με τα στατιστικά για τον κάθε κόμβο



//----------ΚΩΔΙΚΑΣ ΤΗΣ ΦΟΡΜΑΣ----------

```
#include <vcl.h>
#include<conio.h>
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
using namespace std;
#include<math.h>

#include "qtree.h"



#pragma hdrstop

#include "Form1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
```

```
TForm1 *Form1;

/*********  KATHOLIKES METABLHTES (GLOBALS) *****************/
struct Record
   {
   long PrimaryKey;
   vector<double> Attributes;
   int ClusterID;
   };

/*************global variables************************/

double** dataPoints=NULL;    /* data points array[dataRows][dataColumns]  */
long int dataRows ;         // number of rows
int  dataColumns ;          // number of columns
int * membership = NULL;

int dataSize; //to plithos twn simeiwn. To pairnoume apo to arxeio
int maxcellpoints;

/******************** function prototypes ***********************/

double** loadDataTable(long int *,  int *, AnsiString );

void Create2DGraph(TChart *ChartXY, double** OurData, int recsize);



//---------------------------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

/*****************************************************************************/


//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void drawNode(TPaintBox *PaintBox1, struct Node* AX)
{

int  i, Xpos, Ypos, X1, Y1 , X2, Y2 ;

 double left, top, right, bottom ;

  if (AX->pointArray != NULL)
  for (i=0; i<AX->nof_points; i++){
     Xpos = ( AX->pointArray[i].x  *(PaintBox1->Width) ) / 1  + 1 ;
     Ypos = ( (1 - AX->pointArray[i].y)*(PaintBox1->Height ) ) / 1 + 1;
     PaintBox1->Canvas->Pen->Color = clBlack;
     PaintBox1->Canvas->Pen->Width = 3;
     PaintBox1->Canvas->MoveTo(Xpos,Ypos);
     PaintBox1->Canvas->Rectangle(Xpos-1,Ypos+1,Xpos+1,Ypos-1);
      //its upper left corner at the point (X1, Y1) and
      //its lower right corner at the point (X2, Y2)
  }
```

```
// NEED QTREE MIN_X, MIN_Y, AND ADD NODE->WIDTH, AND NODE->HEIGHT TO THEM
// RECURSIVELY TO FIND CURRENT   , THEN
// X1 = MIN_X
// Y1 = MIN_Y + NODE->HEIGHT
// X2 = MIN_X + NODE->WIDTH
// Y2 = MIN_Y

X1 = ( AX->posX*(PaintBox1->Width) ) / 1  ;
Y1 = ( (1 - (AX->posY + AX->height) )*(PaintBox1->Height ) ) / 1 ;
X2 = ( (AX->posX + AX->width) *(PaintBox1->Width) ) / 1  ;
Y2 = ( (1 - AX->posY )*(PaintBox1->Height ) ) / 1 ;

PaintBox1->Canvas->Pen->Color = clRed;
PaintBox1->Canvas->Pen->Width = 1;
// PaintBox1->Canvas->MoveTo(Xpos,Ypos);
PaintBox1->Canvas->Rectangle(X1,Y1,X2,Y2);
//its upper left corner at the point (X1, Y1) and
//its lower right corner at the point (X2, Y2)


  Form1->Memo1->Lines->Add("points = "+ AnsiString(AX->nof_points)+ ", area=" +
        AnsiString((AX->width * AX->height )));


}
//---------------------------------------------------------------------------
void PostOrderDrawLeafs(TPaintBox *PaintBox1, struct Node* AX, double density)
{

  if (AX==NULL){return;}
```

```
  if (AX->child[0] != NULL)  PostOrderDrawLeafs(PaintBox1, AX->child[0],density);
  if (AX->child[1] != NULL)  PostOrderDrawLeafs(PaintBox1, AX->child[1],density);
  if (AX->child[2] != NULL)  PostOrderDrawLeafs(PaintBox1, AX->child[2],density);
  if (AX->child[3] != NULL)  PostOrderDrawLeafs(PaintBox1, AX->child[3],density);


  if ( ( (double)AX->nof_points/(AX->width * AX->height ) >= density) &&
       (AX->pointArray != NULL) ) drawNode(PaintBox1, AX);

 // drawNode(PaintBox1, AX);

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int i, j, recordsize;
  int counter, index ;
  double density ;


  // Initialize the root node
  //dhmiourgw ton root kombo t dentrou
  struct Node * rootNode = new Node;
  qtree mytree;

  density = Edit1->Text.ToDouble();

 Memo1->Lines->Clear() ;
 PaintBox1->Enabled =false;
 PaintBox1->Enabled=true;
```

```
Application->ProcessMessages() ;

dataPoints = loadDataTable(&dataRows, &dataColumns,  Form1->ComboTables->Text) ;

mytree.setRootNodeData(rootNode, dataPoints, dataRows);

//     dataSize = mytree.LoadFromFile("inputfile.dat", rootNode);
    // dhmiourgia dentrou

    mytree.BuildQuadTree(rootNode);

    mytree.PrintQuadTree(rootNode);

       //mytree.PrintStatistics(rootNode);
  //   printStats(rootNode);
    //mytree.DeleteQuadTree(rootNode);

  Memo1->Lines->Add("dataRows=" + AnsiString(dataRows) + ", dataColumns="+
AnsiString(dataColumns));

  Memo1->Lines->Add("rootNode->width=" + AnsiString(rootNode->width) +
          ", rootNode->height="+ AnsiString(rootNode->height));

Create2DGraph(Chart2, dataPoints, dataRows)  ;
```

```
 Application->ProcessMessages() ;

   PaintBox1->Canvas->Brush->Style = bsClear ;     // transparent rectangles
   PaintBox1->Canvas->Rectangle(0,0,PaintBox1->Width ,PaintBox1->Height);

    //draw Points InPaint box
    PostOrderDrawLeafs(PaintBox1, rootNode, density);




    for(i=0; i<dataRows; i++) free(dataPoints[i]);
    free(dataPoints);

 ShowMessage("OK");

}
//---------------------------------------------------------------------------

 void Create2DGraph(TChart *ChartXY, double** OurData, int recsize)
{   //this routine creates a TChart
 ChartXY->SeriesList->Series[0]->Clear() ;
  int i, exist;  //for cluster count
  unsigned j;

      //create colorArray values
      //red 0x000000FF    green 0x0000FF00-----blue 0x00FF0000
      // custom  made Color pallete
```

```
  Application->ProcessMessages() ;



      ChartXY->SeriesList->Series[0]->ColorEachPoint= true;
      for (i = 0; i < recsize; i++)
        ChartXY->SeriesList->Series[0]-> AddXY((const double)OurData[i][0] ,
       (const double)OurData[i][1],AnsiString(OurData[i][0]),
        0x00600000);




}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
double** loadDataTable(long int *dataRows,  int *dataColumns, AnsiString table)
{
     /*loads data points from user specified table */
    long int i, rows;
    int j, columns;
    double** temp;


  AnsiString sql;
  sql = "Select * from " + table + " order by cod";
  Form1->ADOQuery1->SQL->Text = sql;
  Form1->ADOQuery1->Open();

  rows = Form1->ADOQuery1->RecordCount;
  columns = Form1->ADOQuery1->FieldCount - 2 ;   //first is recordID, last is clusterID
  ShowMessage("the record size is " + AnsiString(rows));
```

```
   temp = (double**)malloc(rows*sizeof(double*));              //allocate array

    /* get memory for each row */
    for (i=0; i<rows; i++) {
         temp[i] = (double*)malloc(columns*sizeof(double));
         for (j=0; j<columns; j++) {
              temp[i][j] = Form1->ADOQuery1->Fields->Fields[j+1]->Value;
          }
          Form1->ADOQuery1->Next();
     }

  Form1->ADOQuery1->Close();

    *dataRows = rows;      //apodidei taytoxrona kai tis time sthn main
    *dataColumns = columns;
     return temp;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
ADOConnection1->GetTableNames(ComboTables->Items, false);
}
```

//----------ΚΩΔΙΚΑΣ ΤΟΥ QTREE.H ----------

```
// QTREE.H
#ifndef _QTREE_
#define _QTREE_

#define MAX_POINTS_PER_LEAF  1
using namespace std;

class qtree;

struct Point  {//domh gia kathe shmeio
      double x;
      double y;
};

struct Attribstatistics
{//domh gia statistika stoixeia twn attributes(x,y)
// For each numerical attribute (X,Y), we have the following five parameters for each cell:
      double mmean; //m-mean of all values in this cell
      double sdev;//s - standard deviation of all values of the attribute in this cell
      double minv;//min -the minimum value of the attribute in this cell
      double maxv;//max- the maximum value of the attribute in this cell
       //distribution - the type of distribution that the attribute value in this cell follows

};

struct Node
{//domh gia kathe kombo
      double posX; //TO MIN X TOY CURRENT PARTITION
      double posY; //TO MIN Y TOY CURRENT PARTITION
```

```
    double width; //platos  (EUROS STON X) MAX(X)-MIN(X)
    double height; //ypsos  (EUROS STON Y)  MAX(Y)-MIN(Y)
    int nof_points;//points contained under this node
    struct Node *child[4]; //pointer sta paidia
    struct Point *pointArray; //pointer se pinaka domwn p krataei ta shmeia
    struct Attribstatistics* stats;
};

// ----------------------------Basiki Klasi-----------------------------------
class qtree{
public:
    int CheckTypeofNode(struct Node *n);//elegxos gia ton typo tou kombou
    void BuildQuadTree(struct Node *n); //methodos gia dhmiourgia t quadtree
    void PrintQuadTree(struct Node *n, int depth = 0); // methodos gia ektypwsh t quadtree
    void DeleteQuadTree(struct Node *n); //methodos gia diagrafh tou quadtree
    void FindNofNodes(Node *n, Node *nParent, int index);//methodos gia ton ypologismo twn nodes sto sigekrimeno kombo
    Node *BuildNode(struct Node *n, struct Node  *nParent, int index); // methodos gia dhmiouria enws kombou
    int LoadFromFile(char* filename, struct Node* root); //methodos gia tin fortwsi dedomenwn apo arxeioy.
    void calcStatistics(struct Node* n);//methodos gia ton ypologismo twn stats
    void setRootNodeData(struct Node* root, double** dataPoints, int size) ;
};

#endif
```

//----------ΚΩΔΙΚΑΣ ΤΟΥ QTREE.CPP ----------

```
// QTREE.CPP


#include<stdio.h>

#include<stdlib.h>

#include <fstream>

#include <iostream>

#include <cmath>

#include "qtree.h"


void setnode(struct Node *xy, double x, double y, double w, double h)

{//arxikopoihsh enws kombou me ta stoixeia p theloume

      xy->posX = x;           //TO MIN X TOY CURRENT PARTITION

      xy->posY = y;           //TO MIM Y TOY CURRENT PARTITION

      xy->width= w;           //TO EUROS (SPREAD) THS DIASTASHS X , MAX(X)-MIN(X)

      xy->height= h;          //TO EUROS (SPREAD) THS DIASTASHS Y, MAX(Y)-MIN(Y)


      xy->nof_points = 0;


      for (int i = 0; i < 4; i++)          xy->child[i] = NULL; //initialize ta paidia ws null


      xy->stats = NULL;

}


//synarthsh gia na brisoume to mikrotero x sto box p hmaste

double findMinX(struct Node* n)

{

   int i;

   double tmp = n->pointArray[0].x;

   for (i=1; i < n->nof_points; i++)
```

```
    {//ws to megethos tou kombou
       if (tmp > n->pointArray[i].x) tmp = n->pointArray[i].x;
    }
    return tmp;
}

//synarthsh gia na brisoume to megalytero x sto box p hmaste
double findMaxX(struct Node* n)
{
    int i;
    double tmp = n->pointArray[0].x;
    for (i=1; i < n->nof_points; i++)//lathos elegxos mallon gt theloume to megethos tou box
    {//ws to megethos tou kombou
       if (tmp < n->pointArray[i].x) tmp =  n->pointArray[i].x;
    }
    return tmp;
}

//synarthsh gia na brisoume to mikrotero y sto box p hmaste
double findMinY(struct Node* n)
{
    int i;
    double tmp = n->pointArray[0].y;
    for (i=1; i < n->nof_points; i++)
    {//ws to megethos tou kombou
       if (tmp >  n->pointArray[i].y) tmp =  n->pointArray[i].y;
    }
    return tmp;
}
```

```
//synarthsh gia na brisoume to megalytero y sto box p hmaste
double findMaxY(struct Node* n)
{
    int i;
    double tmp = n->pointArray[0].y;
    for (i=1; i < n->nof_points; i++)
    {//ws to megethos tou kombou
       if (tmp <  n->pointArray[i].y) tmp =  n->pointArray[i].y;
    }
    return tmp;
}
//-------------------------------------------------------------------------------------------
void printNodeData(struct Node *n)
{
 int i;
 for(i=0;i<n->nof_points;i++) printf("[%2.3lf,%2.3lf]", n->pointArray[i].x, n->pointArray[i].y);
 printf("\n");

}
//-------------------------------------------------------------
void qtree:: calcStatistics(struct Node* n){//
    //synarthsh gia ton ypologismo statistikwn
    double sumx=0.0, sumy =0.0;
    double varianceX =0.0, varianceY = 0.0;
    int N;
    double averageX, averageY;

    if (n->nof_points <=1) return;

    N = n->nof_points;//o arithoms twn simeiwn
```

```
    n->stats = new Attribstatistics[2];   // ENA GIA TA X KAI ENA GIA TA Y
    n->stats[0].minv = n->posX;
    n->stats[1].minv = n->posY;
    n->stats[0].maxv = findMaxX(n);
    n->stats[1].maxv = findMaxY(n);

    for (int i = 0; i < N; i++)
            {
       sumx += n->pointArray[i].x;
       sumy += n->pointArray[i].y;
            }

    averageX = n->stats[0].mmean = sumx / (double) N;
    averageY =  n->stats[1].mmean = sumy / (double) N;


    for (int i = 0; i < N; i++) {
       varianceX += (averageX - n->pointArray[i].x) * (averageX - n->pointArray[i].x);
       varianceY += (averageY - n->pointArray[i].y) * (averageY - n->pointArray[i].y);
    }


    n->stats[0].sdev = varianceX / (double) (N );
    n->stats[1].sdev = varianceY / (double) (N );

}

//---------------------------------------------------------------------

int pointArray_size(struct Node *n)
```

```
{//pairnei orisma ena kombo k m epistrefei to megethos t pinaka p periexei


      return n->nof_points;
}
//---------------------------------------------------------------------------
void qtree:: FindNofNodes(Node *n, Node *nParent, int index)
{
   int i;
         /*
       h thesh tou kombou paidi basismenh me index (0-3), kathorizetai ws ekshs:
       | 1 | 0 |
       | 2 | 3 |
       */
          // each coordinate partition is [](] meaning (>= and <=) , (> and <=)


   switch(index)    //TESSERIS CASES MIA GIA KATHE TETARTHMORIO (index= 0, 1, 2, 3)
   {
       case 0: // box-->0
           for(i = 0; i < nParent->nof_points; i++)
           {
               //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto deksi panw
tetarthmorio
               if(   nParent->pointArray[i].x >  nParent->posX + nParent->width/2.0
                 && nParent->pointArray[i].y >  nParent->posY + nParent->height/2.0
                 && nParent->pointArray[i].x <= nParent->posX + nParent->width
                 && nParent->pointArray[i].y <= nParent->posY + nParent-> height)
               {
                 //an perasei olous tous elegxous to shmeio auksano ton arithmo shmeion toy node
                  n->nof_points++;
               }
```

```
        }
        break;
    case 1: // box-->1
        for(i = 0; i < nParent->nof_points; i++)
        {
            //elegxw ola ta shmeia tou patrikou kombou gia na dw an einai sto aristero panw tetarthmorio
            if(   nParent->pointArray[i].x >= nParent->posX
              && nParent->pointArray[i].y >  nParent->posY + nParent->height/2.0
              && nParent->pointArray[i].x <= nParent->posX + nParent->width/2.0
              && nParent->pointArray[i].y <= nParent->posY + nParent->height)
            {
              //an perasei olous tous elegxous to shmeio auksano ton arithmo shmeion toy paidiou
               n->nof_points++;
            }
        }
        break;
    case 2: // box-->2
        for(i = 0; i < nParent->nof_points; i++)
        {
            //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto aristero katw tetarthmorio
            if(   nParent->pointArray[i].x >= nParent->posX
              && nParent->pointArray[i].y >= nParent->posY
              && nParent->pointArray[i].x <= nParent->posX + nParent->width/2.0
              && nParent->pointArray[i].y <= nParent->posY + nParent->height/2.0)
            {
              //an perasei olous tous elegxous to shmeio auksano ton arithmo shmeion toy paidiou
               n->nof_points++;
```

```
            }
        }
        break;
    case 3: // box-->3
        for(i = 0; i < nParent->nof_points; i++)
        {
            //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto deksi katw tetarthmorio
            if(   nParent->pointArray[i].x > nParent->posX +  nParent->width/2.0
              && nParent->pointArray[i].y >= nParent->posY
              && nParent->pointArray[i].x <= nParent->posX + nParent->width
              && nParent->pointArray[i].y <= nParent->posY + nParent->height/2.0)
            {
                //an perasei olous tous elegxous auksano ton arithmo shmeion toy paidiou
                n->nof_points++;
            }
        }
        break;

    }
}
//-------------------------------------
// anadromikh methodos gia th dhmiourgia t dentrou
//-------------------------------------
void qtree:: BuildQuadTree(Node *n)
{
    //orismos kombou
    Node * nodeIn = new Node;

    //kalw thn pointArray_size k pairnw ton arithmo twn shmeiwn gia ton kombo n
```

```
    int points = pointArray_size(n);

   //  printNodeData(n);

    if(points > MAX_POINTS_PER_LEAF)
    {//an ta shmeia eina panw apo 100
        for(int k =0; k < 4; k++)
        {//dhmiourgw clilds k kanw anadromikh klhsh

            n->child[k] = new Node;
            nodeIn = BuildNode(n->child[k], n, k);   //TO nodeIn = n->child[k]
            //SET ATTRIBUTE STATISTICS (nodeIn)
            calcStatistics(nodeIn);

            BuildQuadTree(nodeIn);
        }
    }
}
//-------------------------------------
// Dhmiourgia enws kombou
//-------------------------------------
Node* qtree:: BuildNode(Node *n, Node *nParent, int index) //BUILD ONE CHILD NODE  AT A TIME
{
    int numParentPoints, i,j = 0;

    // 1) dhmiourgei to box me ta shmeia gia ton kombo
    // 2) kathorizei poio shmeio einai se kathe box

    /*
```

```
 h thesh tou kombou paidi basismenh me index (0-3), kathorizetai ws ekshs:
| 1 | 0 |
| 2 | 3 |
*/

//arxikopoihsh kombou              nodeIn = BuildNode(n->child[k], n, k);
setnode(n, 0, 0, 0, 0);

switch(index)
{
    case 0: // box-->0

        n->posX = nParent->posX + nParent->width/2.0 ;
        n->posY = nParent->posY + nParent->height/2.0 ;
        break;

    case 1: // box-->1

        n->posX = nParent->posX;
        n->posY = nParent->posY + nParent->height/2.0 ;
        break;

    case 2: // box-->2

        n->posX = nParent->posX;
        n->posY = nParent->posY;
        break;

    case 3: // box-->3
```

```
            n->posX = nParent->posX+nParent->width/2.0 ;
            n->posY = nParent->posY;
            break;

    }

    // to platos k to ypsos t box gia ena paidi einai to miso tou ypsous kai tou platous tou patera kombou
    n->width = nParent->width/2.0 ;
    n->height = nParent->height/2.0 ;

    // omoia me panw tha bgaloume ta shmeia sta box gia to paidi(child) node
    numParentPoints = pointArray_size(nParent);

        // each coordinate partition is [](] meaning (>= and <=) , (> and <=)
    switch(index)
    {
        case 0: // box-->0
            FindNofNodes(n, nParent, index);
            n->pointArray = new struct Point[n->nof_points];

            for(i = 0; i < numParentPoints; i++)
            {

                //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto deksi panw tetarthmorio
                if(nParent->pointArray[i].x > nParent->posX + nParent->width/2.0
                && nParent->pointArray[i].y > nParent->posY + nParent->height/2.0
                && nParent->pointArray[i].x <= (nParent->posX + nParent->width)
                && nParent->pointArray[i].y <= (nParent->posY + nParent->height) )
                {
```

```
                    //an perasei olous tous elegxous to shmeio to prosthetw sto pinaka tou
paidiou
                    n->pointArray[j].x = nParent ->pointArray[i].x;
                    n->pointArray[j].y = nParent ->pointArray[i].y;
                    j++;
                }


            }

        break;

        case 1: // box-->1
            FindNofNodes(n, nParent, index);
            n->pointArray = new struct Point[n->nof_points];

            for(i = 0; i < numParentPoints; i++)
            {

                //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto aristero
panw tetarthmorio
                if(nParent->pointArray[i].x >= nParent->posX
                && nParent->pointArray[i].y >  nParent->posY + nParent->height/2.0
                && nParent->pointArray[i].x <= nParent->posX + nParent->width/2.0
                && nParent->pointArray[i].y <= nParent->posY + nParent->height)
                {
                    //an perasei olous tous elegxous to shmeio to prosthetw sto pinaka tou
paidiou
                    n->pointArray[j].x = nParent ->pointArray[i].x;
                    n->pointArray[j].y = nParent ->pointArray[i].y;
```

```
                    j++;
                }
            }

        break;

        case 2: // box-->2
            FindNofNodes(n, nParent, index); //ypologismos arithmou shmeiwn
            n->pointArray = new struct Point[n->nof_points]; //desmeysh xwrou isi me ta shmeia

            for(i = 0; i < numParentPoints; i++)
            {

                //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto aristero katw tetarthmorio
                if(nParent->pointArray[i].x >= nParent->posX
                && nParent->pointArray[i].y >= nParent->posY
                && nParent->pointArray[i].x <= nParent->posX + nParent->width/2.0
                && nParent->pointArray[i].y <= nParent->posY + nParent->height/2.0)
                {
                    //an perasei olous tous elegxous to shmeio to prosthetw sto pinaka tou paidiou
                    n->pointArray[j].x = nParent ->pointArray[i].x;
                    n->pointArray[j].y = nParent ->pointArray[i].y;
                    j++;
                }
            }

        break;
```

```
        case 3: // box-->3
            FindNofNodes(n, nParent, index);
            n->pointArray = new struct Point[n->nof_points];

            for(i = 0; i < numParentPoints; i++)
            {
                //elegxw ola ta shmeua tou patrikou kombou gia na dw an einai sto deksi katw tetarthmorio
                if(nParent->pointArray[i].x > nParent->posX +  nParent->width/2.0
                && nParent->pointArray[i].y >= nParent->posY
                && nParent->pointArray[i].x <= nParent->posX + nParent->width
                && nParent->pointArray[i].y <= nParent->posY + nParent->height/2.0)
                {
                    //an perasei olous tous elegxous to shmeio to prosthetw sto pinaka tou paidiou
                     n->pointArray[j].x = nParent ->pointArray[i].x;
                     n->pointArray[j].y = nParent ->pointArray[i].y;
                     j++;
                }

             // bug if(nParent->pointArray[i].x == 61.500 ){printf("FOUND, %lf \n",(nParent->posX + nParent->width));}
             // bug if(nParent->pointArray[i].x == (nParent->posX + nParent->width) ){printf("FOUND\n");}

            }

            delete[] nParent->pointArray; //efoson einai to teleutaio child, apodesmeuoume to xwro pou kratousame gia to parent node.
            nParent->pointArray = NULL;
        break;
```

```
    }
    return n;
}

//-------------------------------------
// methodos p typwnei to dentro....gia debug kyriws.
//-------------------------------------
void qtree:: PrintQuadTree(Node *n, int depth)
{
    for (int i = 0; i < depth; i++)              //gia kathe ena epipedo vathous bazei ena tab
        printf("\t");

    if (n->child[0] == NULL)          //kombos fyllou
    {

        if(n->nof_points!=0)      //ean exei simeia
        {
            cout << "Points: " << n->nof_points << " [";     //emfanise arithmo simeiwn
            for(int i=0; i < n->nof_points; i++){
                cout << " (" << n->pointArray[i].x << " , " << n->pointArray[i].y << ")";
            }
            cout << "]\n";
        /*
        if(n->nof_points!=0){
            cout << "x = " << n->pointArray[0].x << " y = " << n->pointArray[0].y << "\n";
        }
*/                  }
        else if(n->nof_points==0)     //ean den exei simeia
                {
            cout << "Points: " << 0 << "\n";
```

```
        }    //typwse 0
        return;
    }
    else if (n->child[0] != NULL)    //ean einai inner node
    {
        //cout << "Points: " << n->nof_points << " with limits " << n->posX << " to " << n->posX +
n->width << ", " << n->posY << " to " << n->posY + n->height <<" Childs: \n";
        //For testing only
        /*for(int i=0; i < n->nof_points; i++){
            cout << " (" << n->pointArray[i].x << " , " << n->pointArray[i].y << ") ";
        }
        cout << "]\n";*/

        //printf("points %i, Childs:\n",n->nof_points);
        cout << "Points: " << n->nof_points << " Childs: \n";
        for (int i = 0; i < 4; i++)
            PrintQuadTree(n->child[i], depth + 1);
        return;
    }

}
/*
void printStats(Node *n)
{
   cout << "Mean : (" << n->attr.mmean.x << " , " << n->attr.mmean.y << ") , \n" ;
   cout << "Standard Deviation : (" << n->attr.sdev.x << " , " << n->attr.sdev.y << ") , \n";
   cout << "Minimum Point : (" << n->attr.minv.x << " , " << n->attr.minv.y << ") , \n";
   cout << "Maximum Point : (" << n->attr.maxv.x << " , " << n->attr.maxv.y << ") , \n";
}*/
//-------------------------------------
```

```
// methodos p typwnei Statistics....gia debug kyriws.
//-------------------------------------
/*
void qtree:: PrintStatistics(Node *n, int depth)
{
      for (int i = 0; i < depth; i++)
            printf("\t");

      if (n->child[0] == NULL)
      {
            printStats(n);
            return;
      }
      else if (n->child[0] != NULL)
      {
            printf("Children:\n");
            for (int i = 0; i < 4; i++)
                  PrintQuadTree(n->child[i], depth + 1);
            return;
      }
}
*/
//-------------------------------------
// anadromikh methodos gia th diagrafh t dentrou
//-------------------------------------
void qtree:: DeleteQuadTree(Node *n)
{     //PostOrderDelete
      if (n==NULL){return;}
      if (n->child[0] != NULL) {
            DeleteQuadTree(n->child[0]) ;
```

```
            delete (n->child[0]);
            n->child[0]==NULL;
            }
    if (n->child[1] != NULL) {
            DeleteQuadTree(n->child[1]) ;
            delete (n->child[1]);
            n->child[1]==NULL;
            }
    if (n->child[2] != NULL) {
            DeleteQuadTree(n->child[2]) ;
            delete (n->child[2]);
            n->child[2]==NULL;
            }
    if (n->child[3] != NULL) {
            DeleteQuadTree(n->child[3]) ;
            delete (n->child[3]);
            n->child[3]==NULL;
            }

}//-------------------------------------------------------------------------
void qtree::setRootNodeData(struct Node* root, double** dataPoints, int size)
{
  int i ;
  double minX, minY, maxX, maxY;

  root->pointArray = new struct Point[size];

  for (i=0; i < size; i++) { //pairnw tosa stoixeia apo to arxeio oso einai to datasize.
     root->pointArray[i].x = dataPoints[i][0];
     root->pointArray[i].y = dataPoints[i][1];
```

```
  }

  root->nof_points = size;

  minX = findMinX(root) ;
  minY = findMinY(root) ;
  maxX = findMaxX(root) ;
  maxY = findMaxY(root) ;

  //NORMALIZE DATA IN POINT ARRAY
  for (i=0; i < size; i++) { //pairnw tosa stoixeia apo to arxeio oso einai to datasize.
     root->pointArray[i].x = (root->pointArray[i].x  - minX )/(maxX-minX) ;
     root->pointArray[i].y = (root->pointArray[i].y  - minY )/(maxY-minY) ;
  }

  minX = findMinX(root) ;
  minY = findMinY(root) ;
  maxX = findMaxX(root) + 0.00000000001 ;       // without it, an arithmetic bug occur
  maxY = findMaxY(root) + 0.00000000001 ;       // without it, an arithmetic bug occur

   setnode(root, minX, minY, maxX-minX , maxY-minY);//arxikopoisi toy rootNode.

   root->nof_points = size;  //reset back to normal

   //setattributestatistics
    calcStatistics(root);

}
//-----------------------------------------------------------------------------
//synarthsh pou pairnei ta shmeia apo ena arxeio k ta perna sto prwto kombo
```

```
int qtree:: LoadFromFile(char* filename, struct Node* root)
{
    struct Point temp;
    ifstream fp;
    int size;
    int i;
    int j;
    double x,y;
    double minX, minY, maxX, maxY;

    fp.open(filename, ios::in);//anoigma arxeiou

    if(fp.is_open())
    {
        fp >> size;
        root->pointArray = new struct Point[size];

        for (i=0; i < size; i++)
        { //pairnw tosa stoixeia apo to arxeio oso einai to datasize.

            fp >> temp.x >> temp.y;
            root->pointArray[i] = temp;
        }
    }
    else
    {
       printf("Can not open the input file\n");
       system("pause");
       exit(1);
    }
```

```
    root->nof_points = size;

//    printf("\n minX = %lf, minY = %lf, maxX = %lf, maxY = %lf,\n",
//            findMinX(root), findMinY(root), findMaxX(root), findMaxY(root));

   minX = findMinX(root) ;
   minY = findMinY(root) ;
   maxX = findMaxX(root)  + 0.00000000001 ;     // without it, an arithmetic bug occur
   maxY = findMaxY(root)  + 0.00000000001 ;     // without it, an arithmetic bug occur

    setnode(root, minX, minY, maxX-minX , maxY-minY);//arxikopoisi toy rootNode.

 //     setnode(root, findMinX(root)-1.0, findMinY(root)-1.0, findMaxX(root)-findMinX(root)+2.0,
findMaxY(root)-findMinY(root)+2.0);//arxikopoisi toy rootNode.
//            setnode(root, findMinX(root), findMinY(root), findMaxX(root)-findMinX(root),
findMaxY(root)-findMinY(root));//arxikopoisi toy rootNode.

    root->nof_points = size;  //reset back to normal


      //setattributestatistics
    calcStatistics(root);
    return size;
}
```

6 ΒΙΒΛΙΟΓΡΑΦΙΑ

W. Wang, J. Yang and R.Muntz, “STING: A statistical information grid approach to spatial data mining” Proceedings of the International Very Large Databases Conference, pages 186-195, 1997.

M.H. Dunham, “Data Mining introductory and advanced topics” Prentice Hall 2004.

R. Xu, and D. Wunsch II, “Survey of Clustering Algorithms”, IEEE Transactions on Neural Networks, Vol. 16, No. 3, May 2005.

Han, J., & Kamber, M. (2001). “Data mining: concepts and techniques (Morgan-Kaufman Series)”. San Diego: Academic Press.

H. Samet. Recent developments in linear quadtree-based geographic information systems. Image and Vision Computing, 5, (3), pp.187-197, Aug. 1987.

H. Samet. The design and analysis of spatial data structures. Addison-Wesley Publishing Co., 1989.

A. Guttman. R-trees: A dynamic index structure for spatial searching. Proc. A CM SIGMOD Int. Conf. on Management of Data, pages 47-57, 1984.