ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Τμήμα Πληροφορικής και Επικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ο αλγόριθμος Simulated Annealing στην κατευθυνόμενη στοχαστική αναζήτηση της βέλτιστης επιλογής χαρακτηριστικών για κατάταξη προτύπων χρονοσειρών από βάση δεδομένων Oracle με τον αλγόριθμο συσταδοποίησης DBSCAN».

Σπουδαστής: Αυγούστης Γεώργιος

Επιβλέπων Καθηγητής: Δρ. Κόκκινος Ιωάννης

Σέρρες Αύγουστος 2006

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΞΟΡΥΞΗ ΓΝΩΣΗΣ

1.1 Εισαγωγή

Η πρόοδος στην απόκτηση ψηφιακών δεδομένων και στην τεχνολογία αποθήκευσης τους, είχε σαν αποτέλεσμα την αύξηση του μεγέθους των βάσεων δεδομένων. Αυτό συνέβη σε όλους τους τομείς της ανθρώπινης ενασχόλησης. Ξεκινώντας από την καθημερινή ζωή όπως δεδομένα συναλλαγών, δεδομένα τηλεφωνικών κλήσεων και στατιστικά δεδομένα που χρησιμοποιούνται από τις κυβερνήσεις ή άλλους οργανισμούς και φτάνοντας μέχρι και σε πιο εξεζητημένους όπως μοριακές βάσεις δεδομένων και ιατρικά αρχεία. Δεν αποτελεί έκπληξη λοιπόν το ενδιαφέρον να απομονώσουμε τα επιθυμητά δεδομένα και να «εξορύξουμε» από αυτά πληροφορίες που μπορεί να είναι χρήσιμες στον ιδιοκτήτη της βάσης.

Απλές ερωτήσεις που μπορούν να εκφραστούν σε μία δομημένη γλώσσα ερωτήσεων, δεν αρκούν για να υποστηρίξουν τις αυξανόμενες απαιτήσεις για πληροφορίες. Η εξόρυξη γνώσης από δεδομένα (data mining) παρεμβαίνει προκειμένου να ικανοποιήσει αυτές τις ανάγκες.

Εξέλιξη της τεχνολογίας των βάσεων δεδομένων

Δεκαετία 1960	συλλογή δεδομένων, δημιουργία βάσεων, ιεραρχικά (IMS) και δικτυωτά συστήματα
Δεκαετία 1970	Σχεσιακό μοντέλο, υλοποίηση σχεσιακών συστημάτων διαχείρισης βάσεων δεδομένων (RDBMS)
Δεκαετία 1980	Εμπορικά RDBMS, γλώσσα SQL-1, πρώτα αντικείμενο-σχεσιακά και αντικειμενοστραφή μοντέλα, επαγωγικές βάσεις δεδομένων
Δεκαετία 1990	γλώσσα SQL-2, εξόρυξη γνώσης, αποθήκες δεδομένων, βάσεις πολυμέσων
Δεκαετία 2000-σήμερα	εξόρυξη γνώσης με πληθώρα επιστημονικών εφαρμογών, γλώσσα SQL-3, διαχείριση δεδομένων συνεχούς ροής, τεχνολογίες διαδικτύου και παγκόσμια συστήματα ανάκλησης πληροφοριών

1.2 Ορισμοί του data mining

Ο όρος "data mining" έκανε τις πρώτες εμφανίσεις του στη στατιστική επεξεργασία δεδομένων όπου αφορούσε την αποτροπή εξαγωγής μη έγκυρων συμπερασμάτων ως αποτέλεσμα της υπερ-χρήσης δεδομένων.

Το θεώρημα του Bonferron λέει ότι σε προβλήματα όπου είναι δυνατό να εξαχθούν πάρα πολλά πιθανά συμπεράσματα, τότε μερικά από αυτά τα συμπεράσματα θα βγουν αληθή για καθαρά στατιστικούς λόγους, δίχως να έχουν καμία εγκυρότητα.

Ένα πολύ γνωστό παράδειγμα αποτελεί αυτό του David Rhine, ενός παραψυχολόγου στη δεκαετία του 1950 που εξέτασε εκατοντάδες μαθητές για να ανακαλύψει αν διαθέτει κάποιος από αυτούς “υπεραισθητική αντίληψη” ζητώντας τους να μαντέψουν στη σειρά 10 χαρτιά, κόκκινα ή μαύρα. Βρήκε ότι περίπου 1/1000 μάντεψε το σωστό χρώμα και στα δέκα χαρτιά, και αντί να συνειδητοποιήσει ότι αυτό ακριβώς είναι που περιμένεις όταν χιλιάδες άτομα μαντεύουν τυχαία, τους κατέταξε στην κατηγορία των ατόμων με “υπεραισθητική αντίληψη”. Όταν επανεξέτασε αυτούς που τα είχαν μαντέψει σωστά είδε ότι τη δεύτερη φόρα δεν τα πήγαν καλύτερα από το μέσο όρο. Το συμπέρασμα που έβγαλε από τη δεύτερη εξέταση ήταν “αν πεις σε κάποιον ότι έχει υπεραισθητική αντίληψη τότε τη χάνει! ”.

Ένας γενικός ορισμός σύμφωνα με τον Jeffrey D. Ullman (από τους πρωτοπόρους στην έρευνα των βάσεων δεδομένων) για την εξόρυξη γνώσης είναι “**η ανακάλυψη χρήσιμων συνόψεων από δεδομένα**”.

Δηλαδή η εύρεση πληροφοριών που είναι κρυμμένες σε μία βάση δεδομένων. Εναλλακτικά η εξόρυξη γνώσης ονομάζεται και εξερευνητική ανάλυση δεδομένων, ανακάλυψη γνώσης και συμπερασματική μάθηση. Οι όροι ανακάλυψη γνώσης σε βάσεις δεδομένων (Knowledge Discovery in Databases –εν συντομία KDD) και εξόρυξη γνώσης από δεδομένα (data mining) συχνά χρησιμοποιούνται εναλλακτικά για την ίδια έννοια.

Η εξόρυξη γνώσης χρησιμοποιεί αλγόριθμους για την ανάλυση των αρκετά μεγάλων συνόλων από δεδομένα και την εύρεση ανυποψίαστων σχέσεων και την σύνοψη αυτών με νέους τρόπους κατανοητούς και χρήσιμους στον ιδιοκτήτη. Οι σχέσεις και οι συνόψεις που παράγονται μέσω της εξόρυξης δεδομένων συχνά παρουσιάζονται ως μοντέλα ή πρότυπα.

Η λειτουργία του «Data Mining» έχει να κάνει ουσιαστικά με δεδομένα που έχουν συλλεχθεί ήδη για κάποιο άλλο σκοπό. Αυτό σημαίνει πως οι στόχοι της εξόρυξης γνώσης δεν επηρεάζουν τον τρόπο με τον οποίο συλλέγονται τα δεδομένα. Αυτή θα μπορούσε να είναι μία διαφορά της εξόρυξης δεδομένων με τις στατιστικές, όπου τα δεδομένα συλλέγονται με συγκεκριμένους τρόπους για την απάντηση συγκεκριμένων ερωτημάτων. Για αυτόν τον λόγο η μέθοδος του Data Mining συχνά αναφέρεται ως δευτερεύουσα ανάλυση δεδομένων.

Ο ορισμός για το Data Mining παραπάνω αναφέρει και την έννοια του μεγέθους των συνόλων δεδομένων. Εάν χρησιμοποιούντο μόνο μικρά σύνολα δεδομένων τότε, θα συζητούσαμε μόνο την κλασσική διερευνητική ανάλυση στοιχείων όπως ασκείται από τους στατιστικούς αναλυτές. Όταν ερχόμαστε αντιμέτωποι με μεγάλα μεγέθη βάσεων δεδομένων δημιουργούνται και καινούρια προβλήματα. Τέτοια προβλήματα μπορεί να είναι ακόμα και προβλήματα αποθήκευσης και προσπέλασης των δεδομένων αυτών αλλά υπάρχουν και άλλα πιο θεμελιώδη όπως το πώς να αναλύσουμε τα δεδομένα μέσα σε ένα λογικό χρονικό όριο και πώς να αποφασίσουμε εάν μια προφανής σχέση είναι μόνο ένα περιστατικό πιθανότητας που δεν απεικονίζει οποιαδήποτε υποκείμενη πραγματικότητα. Συχνά τα διαθέσιμα στοιχεία περιλαμβάνουν μόνο ένα δείγμα από τον πλήρη πληθυσμό. Ο στόχος μπορεί να είναι *να γενικεύσουμε* από το δείγμα στον πληθυσμό. Παραδείγματος χάριν, μπορεί να επιθυμήσουμε να προβλέψουμε πώς οι μελλοντικοί πελάτες είναι πιθανό να συμπεριφερθούν ή να καθορίσουμε τις ιδιότητες των πρωτεϊνικών δομών που δεν έχουμε δει ακόμα. Μερικές φορές μπορούμε να θελήσουμε να συνοψίσουμε ή *να συμπιέσουμε* ένα πολύ μεγάλο σύνολο στοιχείων κατά τέτοιο τρόπο ώστε το αποτέλεσμα είναι πιο κατανοητό, χωρίς οποιαδήποτε έννοια της γενίκευσης. Αυτό το ζήτημα θα προέκυπτε, παραδείγματος χάριν, εάν είχαμε τα πλήρη στοιχεία απογραφής για τα εκατομμύρια συγκεκριμένων χωρών ή μιας καταγραφής βάσεων δεδομένων των μεμονωμένων λιανικών συναλλαγών.

1.3 Εφαρμογές και Επιστημονικοί Κλάδοι

Μερικά παραδείγματα επιτυχιών στις εφαρμογές εξόρυξης γνώσης είναι:

- Δένδρα απόφασης κατασκευάζονται από ιστορικά τραπεζικών δανείων για την παραγωγή αλγορίθμων που αποφασίζουν πότε να εγκρίνεται ένα δάνειο.
- Πρότυπα συμπεριφοράς ταξιδιωτών εξάγονται για την διαχείριση των εκπτώσεων σε τιμές ξενοδοχείων, αεροπλάνων, ταξιδιωτικών πακέτων κλπ.
- Η παρατήρηση ότι καταναλωτές που αγοράζουν ένα προϊόν (π.χ. πάνες), είναι πιθανό να αγοράσουν και κάποιο άλλο (π.χ. μπίρες) επιτρέπει στα σούπερμαρκετ να τοποθετούν τα δύο προϊόντα κοντά. Τοποθετώντας και τσιπς μεταξύ τους αυξάνει την πιθανότητα πωλήσεων και στα τρία.
- Συσταδοποίηση αντικειμένων του ουρανού ως προς τα επίπεδα ακτινοβολίας τους σε διαφορετικές συχνότητες επιτρέπει στους αστρονόμους να διαχωρίσουν μεταξύ των γαλαξιών, των κοντινών άστρων και άλλων ουράνιων σωμάτων.
- Σύγκριση γονότυπου διαφορετικών ανθρώπων επιτρέπει την ανακάλυψη γονιδίων που ερμηνεύουν την εμφάνιση διαβήτη ή άλλων κληρονομικών ασθενειών.

Με την εξέλιξη της εξόρυξης γνώσης και την αναγνώριση της αποτελεσματικότητάς της, πολλοί διαφορετικοί επιστημονικοί κλάδοι συγκλίνανε προς αυτήν την κατεύθυνση, όπως:

- Στατιστική,
- Μηχανική μάθηση (είναι ο αντίστοιχος όρος στην Τεχνητή νοημοσύνη),
- Αλγόριθμοι συσταδοποίησης,
- Τεχνικές Οπτικοποίησης των αποτελεσμάτων,
- Ανάκτηση πληροφοριών,
- Βάσεις Δεδομένων, όπου τα δεδομένα πολλά και τα ερωτήματα περίπλοκα.

1.4 Λειτουργίες του Data Mining

Είναι βολικό να κατηγοριοποιήσουμε την εξόρυξη δεδομένων σε λειτουργίες ανάλογα με τον στόχο που έχει θέσει κάποιος που κάνει ανάλυση των δεδομένων. Η παρακάτω κατηγοριοποίηση δεν είναι μοναδική και θα μπορούσαν να γίνουν περαιτέρω κατηγοριοποιήσεις αλλά είναι αντιπροσωπευτική.

1. Exploratory Data Analysis (εξερευνητική ανάλυση δεδομένων **EDA):** Ο στόχος εδώ είναι να εξερευνήσουμε τα δεδομένα χωρίς να έχουμε κάποια ξεκάθαρη άποψη για το τι ψάχνουμε. Τυπικά οι τεχνικές EDA είναι *interactive* αλλά και *visual* και υπάρχουν πολλές μέθοδοι που κάνουν γραφική αναπαράσταση για σχετικά λιγο-διάστατα σύνολα δεδομένων. Όσο αυξάνεται ο αριθμός των διαστάσεων γίνεται όλο και πιο δύσκολο να οπτικοποιήσουμε το σύννεφο των σημείων σε ν-διαστάσεις. Για ν>3 ή 4, χρησιμοποιούντα τεχνικές προστασίας (όπως ανάλυση κυρίων τμημάτων), οι οποίες παράγουν απεικονίσεις των δεδομένων σε μικρότερες διαστάσεις. Μεγάλος αριθμός από περιπτώσεις είναι δύσκολο να απεικονιστεί αποτελεσματικά. Παρόλα αυτά κάπου εδώ εισάγονται οι έννοιες της κλίμακας κα της λεπτομέρειας. Δεδομένα χαμηλότερης ανάλυσης μπορούν να απεικονιστούν ή να συνοψιστούν με κόστος έλλειψης πιθανών σημαντικών λεπτομερειών.

2. Descriptive Modeling: Ο στόχος του Descriptive Modeling είναι να περιγράψει όλα τα δεδομένα. Παραδείγματα τέτοιων περιγραφών περιλαμβάνουν τα μοντέλα για την ολική κατανομή πιθανοτήτων των δεδομένων (εκτίμηση πυκνότητας), μοντέλα διαμερισμού του ν-διάστατου χώρου σε ομάδες (ανάλυση συστάδων και κατάτμηση) και μοντέλα που περιγράφουν την σχέση μεταξύ μεταβλητών (dependency modeling). Στην ανάλυση κατάτμησης, για παράδειγμα, ο στόχος είναι να ομαδοποιήσουμε παρόμοια δεδομένα όπως στην κατάτμηση των εμπορικών βάσεων δεδομένων. Εδώ ο στόχος είναι να χωρίσουμε τα δεδομένα σε ομοιογενείς ομάδες έτσι ώστε παρόμοιοι άνθρωποι (εάν τα δεδομένα αναφέρονται σε ανθρώπους) να κατατάσσονται στην ίδια ομάδα.

Αυτό επιτρέπει στους διαφημιστές και τους εμπόρους να στοχεύσουν αποτελεσματικότερα να κατευθύνουν τις προωθήσεις τους σε αυτούς που είναι πιο πιθανό να ανταποκριθούν. Ο αριθμός των ομάδων καθορίζεται από τον ερευνητή και δεν υπάρχει κάποιος βέλτιστος αριθμός. Αυτό έρχεται σε αντίθεση με την ανάλυση συστάδων, όπου ο στόχος είναι να ανακαλύψουμε «φυσικές» ομάδες στα δεδομένα, στις επιστημονικές βάσεις για παράδειγμα.

3. Predictive Modeling: Classification and Regression: Ο στόχος εδώ είναι να κατασκευάσουμε ένα μοντέλο που θα επιτρέπει την τιμή μιας μεταβλητής να προβλεφθεί από τις γνωστές τιμές άλλων μεταβλητών. Η έννοια πρόβλεψη χρησιμοποιείται με την γενική της έννοια και δεν υπονοείται κάποια έννοια της χρονικής συνέχειας. Η κύρια διαφορά μεταξύ της πρόβλεψης και της περιγραφής είναι ότι η πρόβλεψη έχει σαν στόχο (αντικείμενο) μία μοναδική μεταβλητή(π.χ τιμή μετοχής), ενώ στα προβλήματα περιγραφής δεν υπάρχει κάποια μεταβλητή που να είναι μοναδική για το πρότυπο.

4. Discovering Patterns and Rules: Οι τρεις παραπάνω λειτουργίες έχουν να κάνουν με την κατασκευή προτύπων. Άλλες μέθοδοι εξόρυξης δεδομένων εστιάζονται πάνω στην ανίχνευση προτύπων. Ένα παράδειγμα είναι η ανίχνευση ψεύτικης συμπεριφοράς μέσω της ανίχνευσης περιοχών του χώρου καθορίζοντας τους διαφορετικούς τύπους συναλλαγών όπου τα σημεία των δεδομένων διαφέρουν σημαντικά από τα υπόλοιπα. Μια άλλη χρήση είναι στην αστρονομία, όπου η ανίχνευση των ασυνήθιστων αστεριών ή των γαλαξιών μπορεί να οδηγήσει στην ανακάλυψη των προηγουμένως άγνωστων φαινομένων. Ακόμα ένας άλλος στόχος είναι η εύρεση συνδυασμών αντικειμένων που συμβαίνουν συχνά σε βάσεις δεδομένων όπου καταγράφονται συναλλαγές. Όπως για παράδειγμα τα προϊόντα από το μανάβικο αγοράζονται συχνά όλα μαζί. Πάνω σε αυτό το πρόβλημα έχει εστιαστεί το ενδιαφέρον και έχει αντιμετωπιστεί με την χρήση αλγοριθμικών τεχνικών που βασίζονται σε κανόνες συσχετισμού.

5. Retrieval by Content: Εδώ ο χρήστης έχει ένα πρότυπο που τον ενδιαφέρει και επιθυμεί να βρει παρόμοια αντικείμενα. Αυτή η τεχνική χρησιμοποιείται περισσότερο σε σύνολα δεδομένων που περιέχουν κείμενο ή εικόνες. Όσο αναφορά στο κείμενο το πρότυπο μπορεί να είναι ένα σύνολο από λέξεις και ο χρήστης να θέλει να βρει σχετικά έγγραφα μέσα σε ένα μεγάλο σύνολο από σχετικά έγγραφα. Όσο αναφορά στις εικόνες ο χρήστης μπορεί να έχει μία απλή εικόνα ή μία περιγραφή εικόνας και να επιθυμεί να βρει παρόμοιες εικόνες από ένα μεγάλο σύνολο δεδομένων που αποτελείται από εικόνες. Και στις δύο περιπτώσεις ο ορισμός της ομοιότητας είναι κρίσιμος.

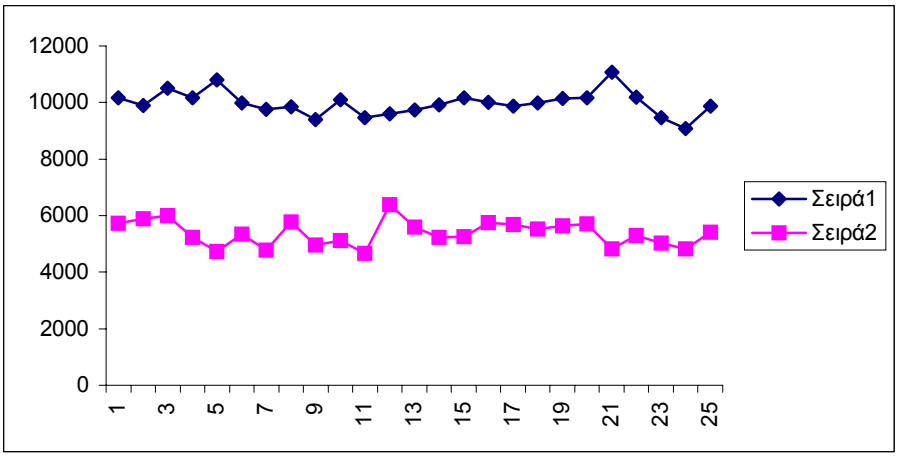
Παρόλο που οι παραπάνω πέντε λειτουργίες έχουν ξεκάθαρες διαφορές, έχουν πολλά κοινά σημεία. Για παράδειγμα η έννοια της ομοιότητας και της απόστασης χρησιμοποιείται από πολλές μεθόδους. Ακόμα είναι προφανές πως για διαφορετικές λειτουργίες χρειάζονται και διαφορετικές δομές μοντέλων και προτύπων, όπως διαφορετικές δομές μπορεί να χρειάζονται για διαφορετικά είδη δεδομένων.

Χάρη στην εξέλιξη των υπολογιστών και την τεχνολογία συλλογής των δεδομένων, μπορούν πλέον να συλλεχθούν τεράστιοι όγκοι δεδομένων. Αυτοί οι όγκοι περιέχουν συχνά πολύτιμη πληροφορία. Το «δύσκολο» είναι να εξάγουμε την πολύτιμη αυτή πληροφορία από τον μεγάλο αυτό όγκο έτσι ώστε οι ιδιοκτήτες των δεδομένων να μπορούν να επενδύσουν σε αυτή. Το Data Mining είναι μια νέα αρχή, η οποία αναζητά να κάνει ακριβώς αυτό. Με το «κοσκίνισμα» των δεδομένων με στόχο την σύνοψη αυτών και την εύρεση προτύπων.

1.5 Ανάλυση Χρονοσειρών

Με την ανάλυση χρονολογικών σειρών ή χρονοσειρών (time series), μελετάται η τιμή ενός γνωρίσματος καθώς μεταβάλλεται στο χρόνο. Οι τιμές λαμβάνονται σε ίσα χρονικά διαστήματα. Για παράδειγμα οι τιμές μιας χρονοσειράς θα αποτελούνται απλά από τους αριθμούς {4, 7, 1, 3, 5, 8, 2, 9, 1, 2}. Μία ακολουθία δεδομένων χρονολογικών σειρών αποτελείται από μετρήσεις ίσων χρονικών διαστημάτων που μπορούν να αφορούν τιμές μετοχών, εγκεφαλογραφήματα, εξωτερικές θερμοκρασίες, παρακολούθηση του καιρού, έξαρσης ηλιακών εκρήξεων.

Για να παρασταθούν οπτικά οι χρονοσειρές χρησιμοποιείται ένα διάγραμμα χρονοσειρών όπως το παρακάτω.

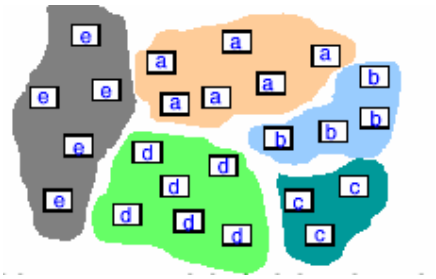
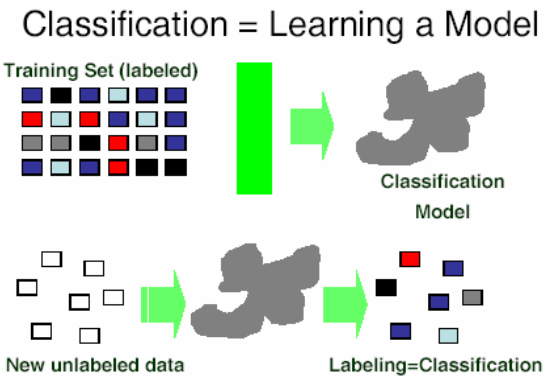


Τρεις βασικές λειτουργίες πραγματοποιούνται στην ανάλυση χρονοσειρών:

- Καθορισμός ομοιότητας ανάμεσα σε δύο διαφορετικές χρονοσειρές με μετρικά μεγέθη.
- Εξέταση της δομής της κάθε χρονοσειράς, εξαγωγή χαρακτηριστικών γνωρισμάτων για κάθε μία και πιθανή κατάταξή τους σε κατηγορίες ή κλάσεις.
- Χρήση διαγραμμάτων για πρόβλεψη μελλοντικών τιμών.

1.6 Ανάλυση Ομάδων (Clustering Analysis)

ΔΙΑΦΟΡΕΣ ΣΥΣΤΑΔΟΠΟΙΗΣΗΣ & ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ

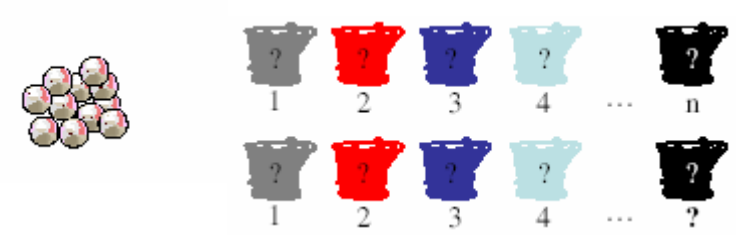
ΣΥΣΤΑΔΟΠΟΙΗΣΗ (CLUSTERING, GROUPING, PARTITIONING)	**ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ (CLASSIFICATION)**
Η γενική διαδικασία της ομαδοποίησης ομοειδών αντικείμενων σε συμπλέγματα (συστάδες ή κλάσεις ή κατηγορίες).  • Τα αντικείμενα δεν έχουν ετικέτες εκ των προτέρων (δεν υπάρχουν δεδομένα εκπαίδευσης) • Χρειάζεται κάποια έννοια ομοιότητας ή εγγύτητας (ποια γνωρίσματα) • Συνήθως δεν γνωρίζουμε πόσες κλάσεις θα πρέπει να παραχθούν • Συνήθως δεν γνωρίζουμε το κριτήριο χαρακτηρισμού για τα μέλη των κλάσεων • Συνήθως δεν γνωρίζουμε τι ετικέτες θα βάλουμε σε κάθε κλάση	Ο στόχος είναι η οργάνωση και κατηγοριοποίηση δεδομένων σε διακριτές κλάσεις.  • Πρώτα δημιουργείται ένα μοντέλο βασισμένο στην κατανομή των δεδομένων εκπαίδευσης • Έπειτα το μοντέλο αυτό χρησιμοποιείται για να κατατάξει νέα δεδομένα • Με δοσμένο το μοντέλο είναι δυνατή η πρόβλεψη νέας κλάσης από νέα δεδομένα

Η **κατηγοριοποίηση** είναι επιτηρούμενη (supervised) διαδικασία επειδή γνωρίζουμε τον αριθμό των κλάσεων και τα ιδιαίτερα χαρακτηριστικά τους (τις ετικέτες)

Παράδειγμα: ξεχώρισε ένα σύνολο από μπάλες γνωστού χρωματισμού στα αντίστοιχα καλάθια γνωστού αριθμού και χρώματος

Η **Συσταδοποίηση** είναι μη επιτηρούμενη (unsupervised) διαδικασία επειδή συνήθως δεν γνωρίζουμε ούτε τον αριθμό των κλάσεων ούτε τα ιδιαίτερα χαρακτηριστικά τους.

Παράδειγμα: ξεχώρισε ένα σύνολο από μπάλες άγνωστου χρωματισμού σε αντίστοιχα καλάθια άγνωστου αριθμού και χρώματος

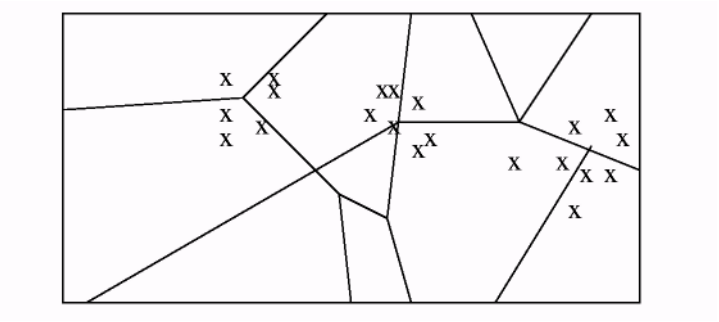


Συστάδα είναι μία συλλογή από αντικείμενα που είναι όμοια μεταξύ τους και έτσι είναι δυνατόν να τα χειριστούμε συλλογικά σαν μία ομάδα.

1.7 Συσταδοποίηση

Εάν δοθούν ορισμένα σημεία στο χώρο -συνήθως πολυδιάστατο χώρο- ομαδοποίησέ τα σε μικρό αριθμό συστάδων (clusters-διαμορφώνω συστάδα σημαίνει συμπυκνώνω σημεία) που θα αποτελούνται από τα σημεία που βρίσκονται “κοντά” με κάποια έννοια. Μερικές εφαρμογές:

1. Πολλά χρόνια πριν, κατά τη διάρκεια ξεσπάσματος επιδημίας χολέρας στο Λονδίνο, ένας γιατρός έκανε ένα σχεδιάγραμμα των περιπτώσεων πάνω σε χάρτη που φαίνεται στο παρακάτω σχήμα



Οι Συστάδες των κρουσμάτων χολέρας δείχνουν τα μολυσμένα πηγάδια

Η οπτικοποίηση των δεδομένων έδειξε, όπως φαίνεται και από την εικόνα, ότι οι περιπτώσεις χολέρας συγκεντρώνονταν γύρω από συγκεκριμένες διασταυρώσεις οδών, όπου βρίσκονταν τα μολυσμένα πηγάδια. Έτσι βρέθηκε όχι μόνο η αιτία της επιδημίας αλλά υποδείχθηκε και ο τρόπος για να αντιμετωπιστεί. Στις περισσότερες φορές βέβαια η εξόρυξη γνώσης δεν είναι τόσο εύκολη επειδή υπάρχουν πολλές διαστάσεις που καθιστούν την οπτικοποίηση αυτής της μορφής πολύ δύσκολη.

2. Το πρόγραμμα Skycat συσταδοποίησε 2×10^9 ουράνια σώματα σε αστέρια, γαλαξίες, κβάζαρ κ.α. Κάθε αντικείμενο ήταν ένα σημείο σε χώρο 7 διαστάσεων, με κάθε διάσταση να παριστά μία ζώνη συχνοτήτων του φάσματος ακτινοβολίας τους. Το πρόγραμμα Sloan Sky Survey είναι ακόμη πιο φιλόδοξο καθώς προσπαθεί να κατατάξει ολόκληρο το ορατό σύμπαν.

3. Τα έγγραφα μπορούν να θεωρηθούν ως σημεία σε έναν πολυδιάστατο χώρο, όπου κάθε διάσταση αντιστοιχεί σε μία πιθανή λέξη. Η θέση του εγγράφου σε κάθε διάσταση είναι ο αριθμός των εμφανίσεων της λέξης μέσα στο έγγραφο (ή απλά 1 αν εμφανίζεται και 0 αν όχι). Συστάδες εγγράφων σε αυτό το χώρο αντιστοιχούν σε ομάδες με κοινό θέμα.

1.8 Μέτρα Απόστασης στην Συσταδοποίηση

Για να βρεθεί το πότε ένα σύνολο από σημεία βρίσκονται αρκετά κοντά το ένα με το άλλο για να διαμορφώσουν μία συστάδα, χρειάζεται ένα μέτρο απόστασης D(x,y) που δείχνει πόσο μακριά βρίσκονται τα σημεία x και y. Τα συνήθη αξιώματα για ένα μέτρο απόστασης είναι:

D(x,x) = 0. Το σημείο έχει απόσταση 0 από τον εαυτό του.

D(x,y) = D(y,x). Η απόσταση είναι συμμετρική.

D(x,y) < D(x,z) + D(z,y). Η τριγωνική ανισότητα.

Συνήθως τα σημεία θεωρούνται σε έναν Ευκλείδειο πολυδιάστατο χώρο, όπου η απόσταση μεταξύ δύο σημείων $x = [x_1, x_2, ...x_k]$ και $y = [y_1, y_2, ...y_k]$ δίνεται από:

1. Συνήθης απόσταση ("L_2 norm"): $\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$
2. Απόσταση Manhattan ("L_1 norm"): $\sum_{i=1}^{k} |x_i - y_i|$
3. Μέγιστη των διαστάσεων ("L_∞ norm"): $\max_{i=1}^{k} |x_i - y_i|$

Παρακάτω παρουσιάζονται αντιπροσωπευτικοί αλγόριθμοι συσταδοποίησης

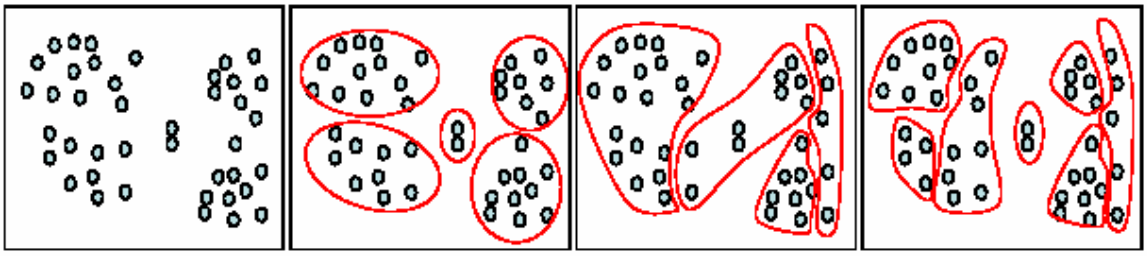
2 ΑΛΓΟΡΙΘΜΟΙ ΣΥΣΤΑΔΟΠΟΙΗΣΗΣ

2.1 Εισαγωγή και εφαρμογές συσταδοποίησης

Συσταδοποίηση είναι η διαδικασία ομαδοποίησης αντικειμένων με όμοια χαρακτηριστικά και η κατάταξη σε κλάσεις ή συστάδες ή συμπλέγματα. Στην Συσταδοποίηση οι συστάδες δεν είναι προκαθορισμένες αλλά προσδιορίζονται από τα δεδομένα. Αυτό το πρόβλημα έχει προσελκύσει το ενδιαφέρον πολλών ερευνητών εδώ και πολλά χρόνια. Με το πέρασμα των χρόνων δημιουργούνται και νέες τεχνικές συσταδοποίησης.

Η συσταδοποίηση έχει χρησιμοποιηθεί σε πολλά πεδία εφαρμογών, συμπεριλαμβανομένων της βιολογίας, ιατρικής, γενετικής, ανθρωπολογίας, μάρκετινγκ, οικονομίας, χρήσεων γης, ασφαλειών, ανάπτυξη πόλεων, αστρονομίας κλπ. Εφαρμογές της συσταδοποίησης περιλαμβάνουν την ταξινομία φυτών και ζώων, την κατηγοριοποίηση βάσει ασθένειας, ομαδοποίηση χρονοσειρών από όργανα μετρήσεων, την εύρεση γονιδίων με όμοιες λειτουργίες, την ομαδοποίηση καταναλωτών, την ομαδοποίηση χρονοσειρών τιμών μετοχών, την αναγνώριση περιοχών με όμοια χρήση γης, την εύρεση ομάδων ασφαλιστικών συμβολαίων με υψηλές ζημίες την ομαδοποίηση σπιτιών βάσει γεωγραφικής απόστασης ή βάσει μεγέθους, εύρεση συμπλεγμάτων ραδιογαλαξιών, την επεξεργασία εικόνας, την αναγνώριση προτύπων, και την ανάκτηση κειμένων. Ένα από τα πρώτα πεδία στα οποία χρησιμοποιήθηκε η συσταδοποίηση ήταν η βιολογική ταξινομία. Πρόσφατες χρήσεις της συσταδοποίησης περιλαμβάνουν την εξέταση των δεδομένων των αρχείων λειτουργίας του Web (Web logs) για τον εντοπισμό προτύπων σχετικά με τον τρόπο χρήσης του δικτύου.

Παράδειγμα συσταδοποίησης σπιτιών: Δοσμένου ενός συνόλου δεδομένων η συσταδοποίηση μπορεί να γίνει σύμφωνα με πολλές και διαφορετικές ιδιότητες.



Οι αρχικές ομάδες των σπιτιών | Συσταδοποίηση βάσει γεωγραφικής απόστασης | Συσταδοποίηση βάσει αντικειμενικής αξίας | Συσταδοποίηση βάσει αντικειμενικής αξίας και μεγέθους

Σαν εργαλείο από μόνο του η Συσταδοποίηση μπορεί να χρησιμοποιηθεί για να βρεθούν οι κατανομές των δεδομένων, τα χαρακτηριστικά κάθε κλάσης, ο καθορισμός της κλάσης ενός νέου παραδείγματος. Σαν βήμα προ-επεξεργασίας για άλλους αλγορίθμους μπορεί να χρησιμοποιεί τα κέντρα των συστάδων για να αναπαριστά τα δεδομένα μέσα σε αυτές.

2.2 Συσταδοποίηση σε βάσεις δεδομένων

Η Συσταδοποίηση (clustering) στις βάσεις δεδομένων γίνεται σύμφωνα με τα καθορισμένα γνωρίσματα των εγγραφών.

ΚΩΔΣΠΙΤΙΟΥ	ΑΠΟΣΤΑΣΗ Χ	ΑΠΟΣΤΑΣΗ Υ	ΑΞΙΑ	ΜΕΓΕΘΟΣ

Όταν εφαρμόζεται συσταδοποίηση σε πραγματικές βάσεις δεδομένων, προκύπτουν πολλά ενδιαφέροντα προβλήματα.

Ο χειρισμός των *ακραίων σημείων* (outliers) είναι δύσκολος. Τα στοιχεία αυτά δεν ανήκουν στην πράξη σε καμία συστάδα· μπορούν να θεωρηθούν σαν μεμονωμένες συστάδες. Ωστόσο, αν ένας αλγόριθμος συσταδοποίησης επιχειρήσει να βρει μεγαλύτερες συστάδες, αυτά τα στοιχεία αναγκαστικά θα τοποθετούν σε κάποια ευρύτερη συστάδα. Καθώς αυτή η διαδικασία μπορεί να συνδυάσει δύο υπάρχουσες συστάδες και να αφήσει το απομονωμένο σημείο στη δική του συστάδα, μπορεί να οδηγήσει σε φτωχή συσταδοποίηση.

Τα δεδομένα είναι δυναμικά μέσα στη βάση δεδομένων και υποδηλώνουν ότι η σύσταση των συστάδων μπορεί να αλλάξει στην πορεία του χρόνου.

Η ερμηνεία της σημασιολογίας κάθε συστάδας ενδέχεται να είναι δύσκολη. Συνεπώς, όταν ολοκληρωθεί η διαδικασία συσταδοποίησης δημιουργώντας ένα σύνολο συστάδων, μπορεί να μην είναι προφανής η ακριβής σημασία της κάθε συστάδας.

Δεν υπάρχει μία και μόνη σωστή λύση σε ένα πρόβλημα συσταδοποίησης. Στην πραγματικότητα, μπορούν να βρεθούν πολλές απαντήσεις. Το ακριβές πλήθος των συστάδων που απαιτούνται δεν είναι τόσο εύκολο να προσδιοριστεί. Για παράδειγμα, έστω ότι έχουμε ένα σύνολο δεδομένων για τα φυτά που συλλέχθηκαν κατά τη διάρκεια μιας εκδρομής. Αν δεν έχουμε καμία προηγούμενη γνώση σχετικά με την ταξινομία των φυτών και επιχειρήσουμε να χωρίσουμε αυτό το σύνολο δεδομένων σε παρόμοιες ομάδες, δε θα είναι προφανές πόσες ομάδες θα πρέπει να δημιουργηθούν.

Ένα άλλο σχετικό θέμα είναι τι δεδομένα θα πρέπει να χρησιμοποιηθούν για τη συσταδοποίηση. Σε αντίθεση με τη μάθηση κατά τη διάρκεια της διαδικασίας κατηγοριοποίησης, όπου υπάρχει εκ των προτέρων κάποια γνώση σχετικά με το ποια πρέπει να είναι τα γνωρίσματα της κατηγοριοποίησης, στη συσταδοποίηση δεν υπάρχει επιβλεπόμενη μάθηση για να βοηθήσει τη διαδικασία. Η συσταδοποίηση μπορεί να θεωρηθεί παρόμοια με τη μη επιβλεπόμενη μάθηση.

Μπορούμε τώρα να συνοψίσουμε μερικά βασικά χαρακτηριστικά της συσταδοποίησης:

- Ο (βέλτιστος) αριθμός συστάδων δεν είναι γνωστός.
- Μπορεί να μην υπάρχει καμία εκ των προτέρων γνώση σχετικά με τις συστάδες.
- Τα αποτελέσματα των συστάδων είναι δυναμικά.

Η ανάλυση των συστάδων επιτυγχάνεται με την ελαχιστοποίηση της ομοιότητας μεταξύ διαφορετικών ομάδων και την μεγιστοποίηση της ομοιότητας των αντικειμένων που βρίσκονται μέσα στην ίδια ομάδα.

Μία καλή μέθοδος συσταδοποίησης παράγει συστάδες στις οποίες

- η δια-συσταδική (inter-class) ομοιότητα των αντικειμένων είναι χαμηλή και αντίστοιχα η ανομοιότητα υψηλή
- η ενδο-συσταδική (intra-class) ομοιότητα των αντικειμένων είναι υψηλή και αντίστοιχα η ανομοιότητα χαμηλή

Στην πτυχιακή εργασία χρησιμοποιούμε σαν μέτρο ανομοιότητας την inter-class distance η οποία βασίζεται στην υπόθεση ότι αντικείμενα διαφορετικών κλάσεων είναι απομακρυσμένα μεταξύ τους

Πολλές λειτουργίες κατά το Data-Mining εστιάζονται είτε στο πώς τα αντικείμενα των δεδομένων είναι ομαδοποιημένα ή ποια αντικείμενα μπορούν να θεωρηθούν απόμακρα από τον σχηματισμό ομάδας. Είναι αλήθεια πως υπάρχει ένα πλήθος από στρατηγικές, η κάθε μία από αυτές έχει και τα δικά της πλεονεκτήματα αλλά και μειονεκτήματα. Παρακάτω παρουσιάζονται μερικές τυπικές απαιτήσεις για μία καλή τεχνική συσταδοποίησης στην εξόρυξη δεδομένων (Data Mining).

Εξελιξιμότητα: Η μέθοδος της συσταδοποίησης θα πρέπει να έχει εφαρμογή σε πολύ μεγάλες βάσεις δεδομένων και η απόδοση θα πρέπει να μειώνεται σχεδόν γραμμικά με την αύξηση της βάσης δεδομένων.

Μεταβλητότητα: Τα αντικείμενα προς συσταδοποίηση μπορεί να είναι διαφορετικού είδους- αριθμητικού τύπου η τύπου bool κ.τ.λ. Μια μέθοδος θα ήταν ιδανική εάν μπορούσε να είναι κατάλληλη για πολλούς τύπους δεδομένων.

Ικανότητα να ανακαλύπτονται συστάδες με διαφορετική μορφή: Αυτή είναι μία πολύ σημαντική απαίτηση συσταδοποίηση δεδομένων που έχουν διαφορετικό σχήμα. Πολλοί αλγόριθμοι συσταδοποίησης μπορούν να ανακαλύψουν συστάδες με σφαιρικό σχήμα.

Ελάχιστη αρχική παραμετροποίηση: Η μέθοδος θα πρέπει να απαιτεί ένα ελάχιστο ποσό γνώσης του χώρου όπου θα εφαρμοστεί. Παρόλα αυτά οι πιο πρόσφατοι αλγόριθμοι έχουν αρκετές παραμέτρους για εισαγωγή και αυτό δεν τους κάνει πολύ πρακτικούς για επιλύσεις πραγματικών προβλημάτων.

Ανεπηρέαστος από τον θόρυβο: Αυτό είναι σημαντικό γιατί ο θόρυβος (λανθασμένα ή τυχαία στοιχεία) υπάρχει παντού σε πραγματικές συνθήκες. Ένας καλός αλγόριθμος συσταδοποίησης πρέπει να έχει την ικανότητα να αποδίδει ικανοποιητικά ακόμα και σε μεγάλες ποσότητες θορύβου.

Καμία ευαισθησία στην σειρά εισαγωγής των δεδομένων: Η μέθοδος θα πρέπει να δίνει συνεπή αποτελέσματα ανεξάρτητα από την σειρά εισαγωγής των δεδομένων.

Ρυθμιζόμενοι σε υψηλές διαστατικότητες: Η ικανότητα να χειρίζονται υψηλής διαστατικότητας δεδομένα είναι πολύ ενδιαφέρον αλλά πραγματικά σύνολα δεδομένων είναι πολύ συχνά πολυδιάστατα.

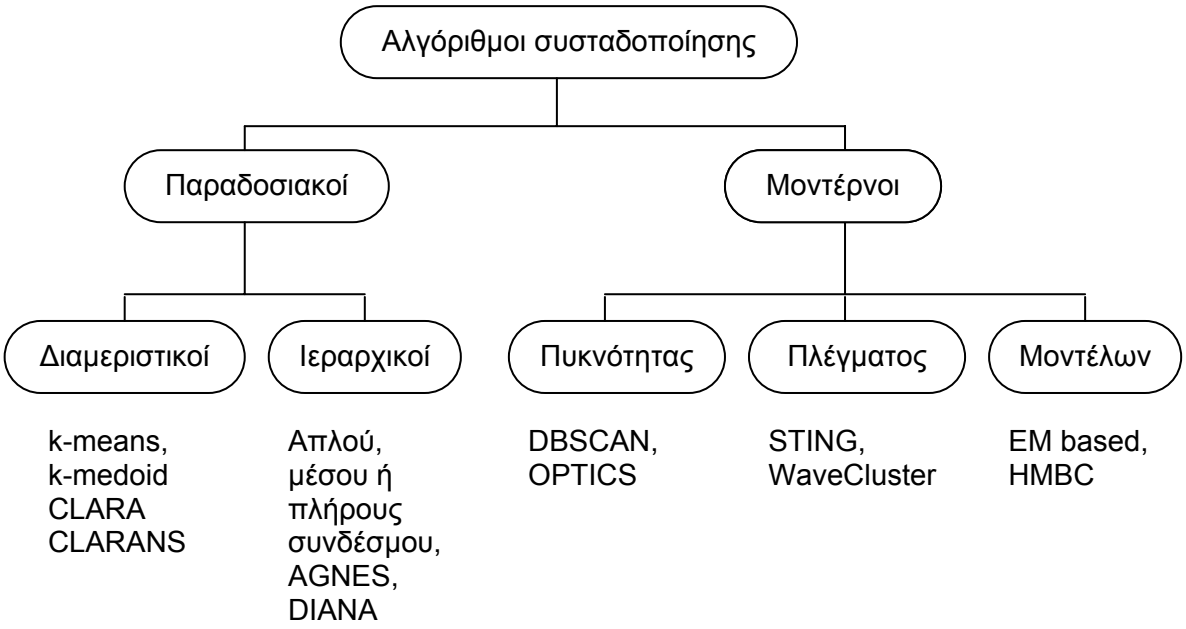
Ανατρέχοντας την ιστορία των αλγόριθμων συσταδοποίησης θα δούμε πως δεν υπάρχει κάποιος αλγόριθμος που να πληροί ταυτόχρονα όλες τις προαναφερθείσες απαιτήσεις. Πρόσφατα έχουν υπάρξει αρκετοί νέες μέθοδοι συσταδοποίησης που προσφέρουν πολλά πλεονεκτήματα και πιο ολοκληρωμένες λύσεις.

Παρακάτω θα παρουσιάσουμε διαφορετικές προσεγγίσεις στην συσταδοποίηση και θα τις ομαδοποιήσουμε σύμφωνα με την θεμελιώδη προσέγγισή τους. Θα παρουσιάσουμε τις βασικές αρχές και έννοιες τους.

2.3 Ταξινόμηση των μεθόδων συσταδοποίησης

Υπάρχει ένας μεγάλος αριθμός αλγορίθμων που ασχολούνται με την συσταδοποίηση, οι οποίοι έχουν μελετηθεί στην εξόρυξη γνώσης, την μηχανική μάθηση, την Στατιστική και την αναγνώριση προτύπων.

Γενικά, οι αλγόριθμοι μπορούν να ομαδοποιηθούν σε πέντε ομάδες.



Διαμεριστικοί αλγόριθμοι (Partitioning algorithms): κατασκευάζουν πολλές διαμερίσεις (partitions) των δεδομένων και τις αξιολογούν σύμφωνα με κάποιο κριτήριο απόστασης. Οι πιο γνωστοί είναι οι k-means και k-medoid.

Ιεραρχικοί αλγόριθμοι (Hierarchy algorithms): δημιουργούν μία ιεραρχική δομή (δενδρόγραμμα) αποσύνθεσης του συνόλου δεδομένων (ή αντικειμένων) σύμφωνα με κάποιο κριτήριο. Υπάρχει η διαιρετική και η συσσωρευτική προσέγγιση.

Βασισμένοι σε πυκνότητα (Density-based): βασίζονται στην συνεκτικότητα και συναρτήσεις πυκνότητας των σημείων των δεδομένων, έτσι οι συστάδες κατασκευάζονται σύμφωνα με κριτήρια πυκνότητας και συνεκτικότητας.

Βασισμένοι σε πλέγμα (Grid-based), χρησιμοποιούν μία πολλαπλών επιπέδων υψηλής ανάλυσης κοκκοποιημένη δομή για να αναλύσουν τις συστάδες. Σε κάθε επίπεδο η ανάλυση μεγαλώνει. Παράδειγμα ο STING και ο WaveCluster.

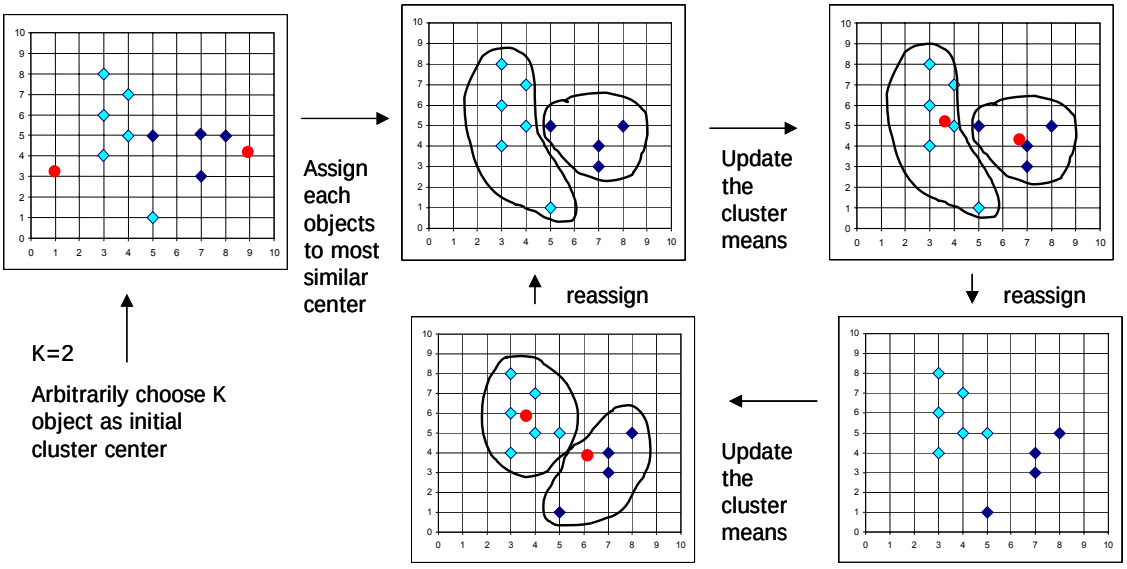
Βασισμένοι σε μοντέλα (Model-based): υποθέτουν ένα μοντέλο για κάθε συστάδα και βρίσκουν το καλύτερο ταίριασμα των δεδομένων σε αυτό διανέμοντας κάθε σημείο δεδομένου στη συστάδα στην οποία αναμένεται να έχει τη μεγαλύτερη πιθανότητα να ανήκει. Η εκτίμηση γίνεται μέσω της μέγιστης πιθανοφάνειας (maximum likelihood). Τυπικό παράδειγμα ο EM (Expectation Maximization) με το Gaussian Mixture Model και ο HMBC (Hierarchical Model-based clustering).

2.4 Διαμεριστικοί Αλγόριθμοι

Οι μέθοδοι προσπαθούν να διασπάσουν ένα σύνολο από n αντικείμενα σε k συστάδες (συμπλέγματα) με βάση την απόστασή τους. Το επιθυμητό πλήθος *k* των συμπλεγμάτων δίνεται ως είσοδος. Η βασική ιδέα του διαμερισμού είναι αρκετά διαισθητική και η διαδικασία γίνεται ουσιαστικά για να επιτευχθούν σταθερά βέλτιστα κριτήρια κατά επανάληψη. Οι πιο γνωστοί αλγόριθμοι που ανήκουν σε αυτή την κατηγορία είναι οι **k-means** και **k-medoid** όπου η κάθε συστάδα αντιπροσωπεύεται από το κέντρο βάρους (Centroid = $\sum_{i=1}^{n} p_i / N$) της συστάδας στον **k-means** ή από το πιο κεντρικό σημείο (Medoid) της συστάδας στην μέθοδο **k-medoid**. Μόλις επιλεχτούν οι αντιπρόσωποι συστάδων, τα σημεία των αντικειμένων ορίζονται σε αυτούς τους αντιπροσώπους, και επαναληπτικά, νέοι καλύτεροι αντιπρόσωποι επιλέγονται και τα σημεία επανεκχώρούνται έως ότου δεν γίνεται καμία αλλαγή.

Ο αλγόριθμος k-means με δοσμένο τον αριθμό των συστάδων (π.χ. θέλουμε 2 συστάδες) έχει τέσσερα βήματα

1. διαχώρισε τυχαία τα αντικείμενα σε k μη κενά υποσύνολα
2. υπολόγισε το κέντρο της κάθε συστάδας (το μέσο σημείο).
3. εκχώρησε το κάθε αντικείμενο στη συστάδα με το πλησιέστερο κέντρο
4. γύρνα στο βήμα 2, σταμάτησε όταν δεν θα υπάρχει νέα εκχώρηση



Η επανάληψη σταματά όταν συγκλίνει η συνάρτηση κριτηρίου, για παράδειγμα ελαχιστοποίησε το τετραγωνικό σφάλμα της μέσης τιμής για όλες τις συστάδες

$$\frac{\sum_{j=1}^{k} \sum_{i=1}^{size(K_j)} \left\| t_{ji} - C_j \right\|^2}{k}$$

Η πολυπλοκότητα του k-means είναι O(tkn), όπου **n** τα αντικείμενα, **k** οι συστάδες και **t** οι επαναλήψεις.

Συγκρίνοντας με τις παραλλαγές του k-means:

Πολυπλοκότητα του PAM (Partitioning Around Medoids, 1987): $O(k(n-k)^2)$,

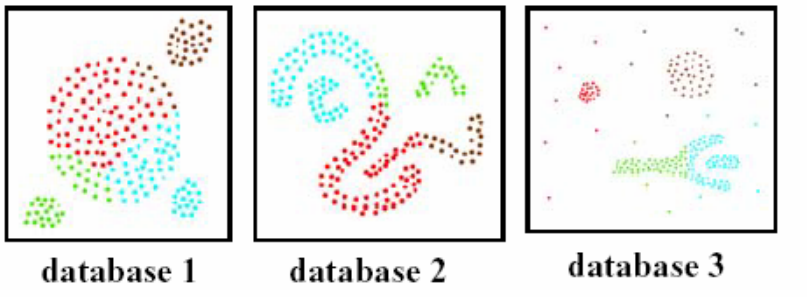
Πολυπλοκότητα του CLARA (πολλαπλά δείγματα+ PAM): $O(ks^2 + k(n-k))$

Ο αριθμός των συστάδων πρέπει να προκαθοριστεί. Συχνά τερματίζει σε τοπικό βέλτιστο. Το καθολικό βέλτιστο μπορεί να βρεθεί και εδώ με τεχνικές όπως προσομοιωμένη ανόπτηση (simulated annealing) και γενετικούς αλγόριθμους. Δεν χειρίζεται θόρυβο ή ακραία σημεία, και δεν βρίσκει συστάδες με μη σφαιρικό σχήμα.

Ο CLARANS αποτελεί έναν βελτιωμένο αλγόριθμο k-medoid. Χρησιμοποιεί τυχαία δειγματοληψία. Ο CLARANS είναι βελτιωμένος για μεγάλα σύνολα και αντιμετωπίζει το πρόβλημα των απομονωμένων σημείων (outliers). Απαιτεί επιπλέον σαν είσοδο το μέγιστο πλήθος γειτόνων (maxneighbor) και το πλήθος των δειγμάτων (numlocal) που θα εξεταστούν. Όλες οι μέθοδοι που ακολουθούν την λειτουργία του διαμερισμού είναι παρόμοιας ποιότητας και οι βασικές δυσκολίες με αυτές τις μεθόδους είναι :

- Ο αριθμός *k* των συστάδων που πρέπει να βρεθούν δίνεται ως είσοδος απαιτώντας έτσι μία αρχική γνώση για το πεδίο η οποία συχνά σε πραγματικά προβλήματα δεν είναί γνωστή.
- Είναι δύσκολο να βρεθούν συστάδες με μεγάλες διαφοροποιήσεις στο μέγεθος. Σαν αποτέλεσμα μεγάλες συστάδες τείνουν να σπάνε παρόλη την ομοιογένεια τους.
- Είναι κατάλληλες μόνο για σφαιρικές συστάδες.

Αυτό είναι πραγματικά ένα γενικό πρόβλημα για όλες τις μεθόδους που χρησιμοποιούν διαμερισμό επειδή χρησιμοποιούν μόνο ένα κέντρο βάρους για να αντιπροσωπεύσουν μια συστάδα, και η συγκέντρωση όλων των άλλων σημείων αποφασίζεται από τη σχετική κοντινότητα τους στα κέντρα βάρους των συστάδων. Οι περισσότερες μέθοδοι που χρησιμοποιούν διαμερισμό έχουν παρόμοια αποτελέσματα.



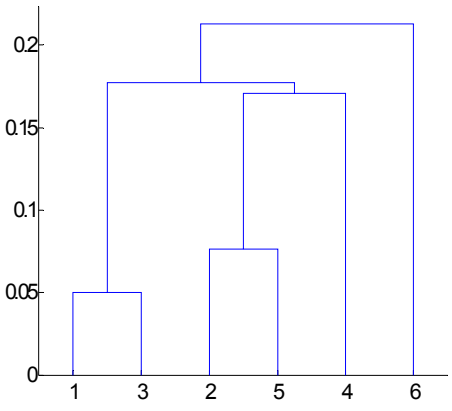
Παραδείγματα συστάδων που βρέθηκαν με τον CLARANS

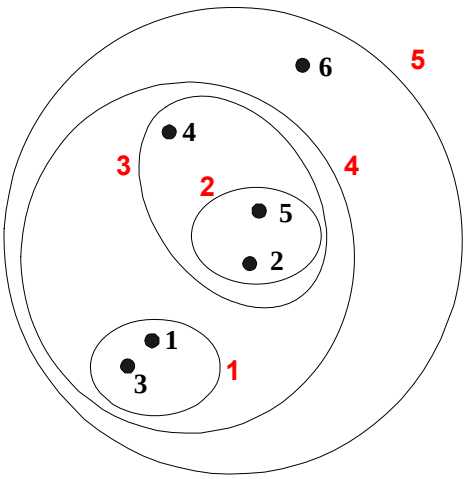
2.5 Ιεραρχικοί Αλγόριθμοι

Ένας αλγόριθμος που χρησιμοποιεί την ιεραρχία παράγει ένα δεντρόδιάγραμμα που αναπαριστά μια ένθετη σχέση ομαδοποίησης μεταξύ των αντικειμένων.

Τα κύρια χαρακτηριστικά των ιεραρχικών αλγορίθμων είναι:

- Δημιουργούνται πολλά σετ από συστάδες.
- Ο αριθμός των συστάδων δεν δίνεται ως είσοδος.
- Η ιεραρχία των συστάδων αναπαριστάται με ένα δενδρόγραμμα.
- Τα φύλλα του δένδρου αποτελούνται από ένα αντικείμενο.
- Κάθε αντικείμενο βρίσκεται σε μία συστάδα.
- Η ρίζα του δενδρογράμματος περιέχουν όλα τα αντικείμενα.
- Οι εσωτερικοί κόμβοι παριστάνουν τις συστάδες που δημιουργούνται από την ένωση πολλών φύλλων ή και κόμβων-παιδιών.
- Κάθε επίπεδο παριστάνει ένα κατώφλι απόστασης που χρησιμοποιείται για να συνενώσει τις συστάδες.
- Αν η απόσταση μεταξύ δύο συστάδων είναι μικρότερη από το κατώφλι απόστασης τότε συνενώνονται σε μία συστάδα.
- Η απόσταση αυξάνεται με τα επίπεδα.





Αν η ιεραρχία για την συσταδοποίηση είναι από κάτω προς τα πάνω, στην αρχή κάθε αντικείμενο αποτελεί και μια συστάδα και μετά μικρές συστάδες ενώνονται σε μεγαλύτερες σε κάθε επίπεδο ιεραρχίας μέχρι το τελευταίο επίπεδο ιεραρχίας. Αυτό το είδος της μεθόδου ονομάζεται συσσωρευτική. Η αντίστροφη μέθοδος λέγεται διαιρετική ιεραρχική μέθοδος.

Οι δύο κύριες προσεγγίσεις ιεραρχικών αλγορίθμων συσταδοποίησης είναι:

Συσσωρευτική μέθοδος (bottom-up): συνένωνε τις συστάδες επαναληπτικά.

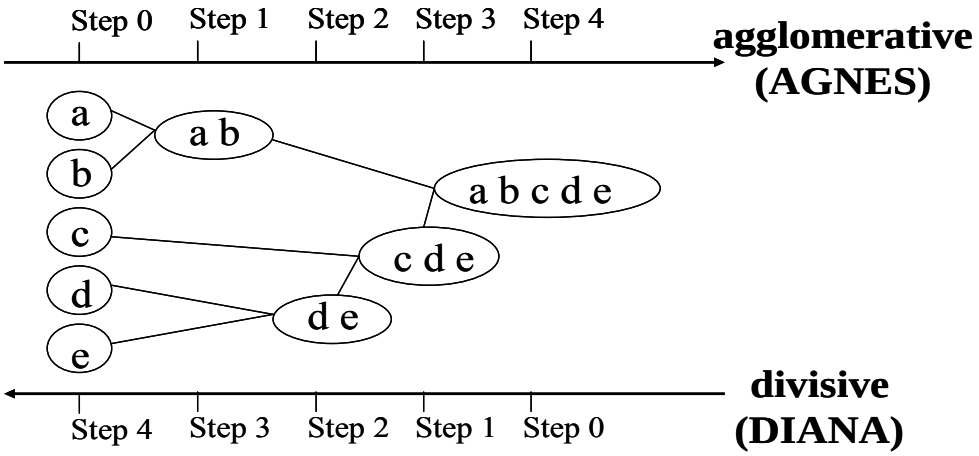
- ξεκίνα τοποθετώντας κάθε αντικείμενο στη δική του συστάδα και
- έπειτα συνένωνε αυτές τις ατομικές συστάδες σε όλο και μεγαλύτερες,
- μέχρι όλα τα αντικείμενα να βρεθούν σε μία συστάδα.

Οι περισσότεροι ιεραρχικοί αλγόριθμοι (π.χ. AGNES) ανήκουν σε αυτή την κατηγορία. Διαφέρουν ως προς τον καθορισμό των μέτρων ομοιότητας μεταξύ των συστάδων.

Διαιρετική μέθοδος (top-down): διαχώριζε τις συστάδες επαναληπτικά

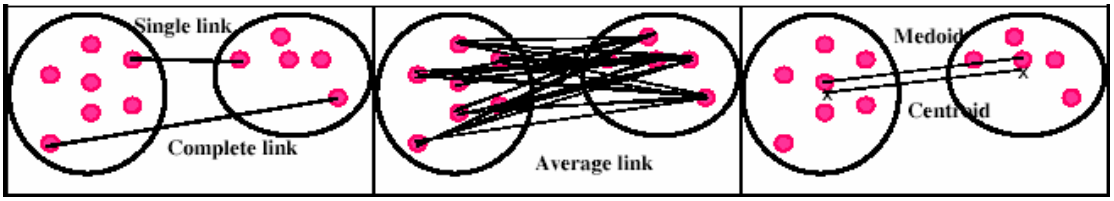
- κάνει το αντίστροφο ξεκινώντας από μία συστάδα που έπειτα
- διασπά σε όλο και μικρότερα κομμάτια,

Οι διαιρετικοί αλγόριθμοι (π.χ. DIANA) είναι σπανιότεροι. Χρησιμοποιούν πίνακα αποστάσεων ως κριτήριο συσταδοποίησης. Δεν απαιτούν αρχικό ορισμό των συστάδων, αλλά χρειάζονται ένα κριτήριο τερματισμού.



Συσσωρευτική και Διαιρετική μέθοδος

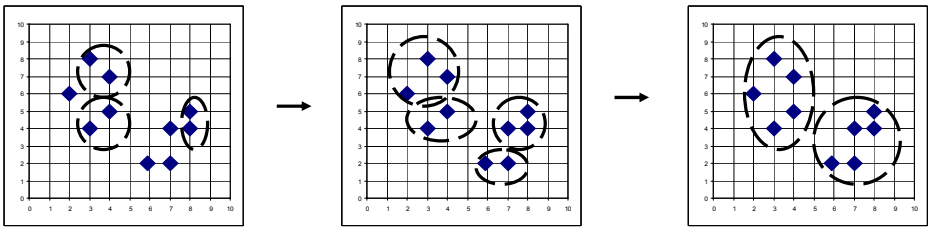
Η απόσταση μεταξύ των συστάδων έχει πολλές ερμηνείες.



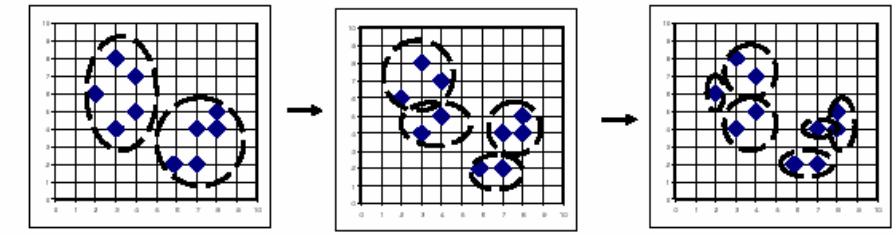
- Απλού συνδέσμου (Single link) – η απόσταση μεταξύ δύο συστάδων είναι η μικρότερη απόσταση από κάθε μέλος της μίας συστάδας με κάθε μέλος της άλλης.

- Πλήρους συνδέσμου (Complete link) – η απόσταση μεταξύ δύο συστάδων είναι η μεγαλύτερη απόσταση από κάθε μέλος της μίας συστάδας με κάθε μέλος της άλλης.
- Μέσου συνδέσμου (Average link) – η μέση απόσταση μεταξύ καθενός μέλους της μίας συστάδας με κάθε ένα μέλος της άλλης.
- Κέντρου βάρους (Centroid) – η απόσταση μεταξύ δύο συστάδων είναι η απόσταση μεταξύ των δύο κέντρων.
- Κεντρικού σημείου (Medoid) – η απόσταση μεταξύ δύο συστάδων είναι η απόσταση των δύο πιο κεντρικών σημείων τους.

Ο Συσσωρευτικός ιεραρχικός αλγόριθμος **AGNES** (Agglomerative Nesting) παρουσιάστηκε από τους Kaufmann και Rousseeuw το 1990, και εφαρμόζεται σε προγράμματα στατιστικής ανάλυσης, όπως το S plus. Χρησιμοποιεί την μέθοδο απλού συνδέσμου και πίνακα ανομοιότητας σημείων (dissimilarity matrix). Συνενώνει κόμβους που έχουν τη μικρότερη ανομοιότητα.



Ο διαιρετικός ιεραρχικός αλγόριθμος **DIANA** (Divisive Analysis) παρουσιάστηκε από τους Kaufmann και Rousseeuw το 1990, και εφαρμόζεται σε προγράμματα στατιστικής ανάλυσης, όπως το S plus. Εφαρμόζει την αντίστροφη σειρά του AGNES. Τελικά κάθε κόμβος διαμορφώνει μία συστάδα.

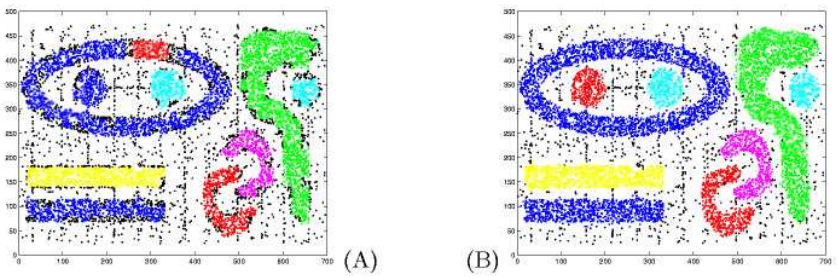


2.6 Άλλες ιεραρχικές μέθοδοι

Η μέθοδος **BIRCH** εισήγαγε την έννοια της συσταδοποίησης αντικειμένων αλλά και το CF-δέντρο. Αρχικά διαχωρίζει τα αντικείμενα ιεραρχικά χρησιμοποιώντας την δομή ενός CF-δέντρου. Σταδιακά προσαρμόζει την ποιότητα των συστάδων. Η μέθοδος BIRCH είναι κατάλληλη για μεγάλες βάσεις δεδομένων, αλλά δυστυχώς δεν αποδίδει καλά εάν οι συστάδες δεν είναι σφαιρικού σχήματος ή εάν έχουν μεγάλες διαφοροποιήσεις ως προς το μέγεθος.

Η μέθοδος **CURE** χρησιμοποιεί ένα σύνολο από σημεία, αντί να χρησιμοποιεί μόνο ένα σημείο για να εκπροσωπεί μία συστάδα σύμφωνα με τις μεθόδους που είναι βασισμένες σε κέντρα βάρους. Αυτός ο σταθερός αριθμός των αντιπροσωπευτικών σημείων της συστάδας επιλέγεται έτσι ώστε να είναι καλά διεσπαρμένα και μετά να μειώνονται ως προς το κέντρο βάρους της κάθε συστάδας σύμφωνα με τον παράγοντα μείωσης. Μετά οι συστάδες συγχωνεύονται επανειλημμένα βάση της ομοιότητάς τους. Η ομοιότητα μεταξύ δύο συστάδων μετριέται με την ομοιότητά του πιο κοντινού ζεύγους από τα αντιπροσωπευτικά σημεία που ανήκουν σε διαφορετικές συστάδες. Η μέθοδος CURE αγνοεί τις πληροφορίες αλληλοσυνεκτηκότητας των αντικειμένων μέσα στην συστάδα.

Η μέθοδος **ROCK** ενεργεί πάνω σε έναν παραγόμενο γράφημα ομοιοτήτων. Αντί να χρησιμοποιεί την έννοια της απόστασης για να μετρήσει την ομοιότητα μεταξύ των σημείων, χρησιμοποιείται η έννοια των δεσμών η οποία περιέχει περισσότερες γενικές πληροφορίες του διαστήματος συστάδων έναντι του μέτρου ομοιότητας απόστασης που εξετάζει μόνο την τοπική απόσταση μεταξύ δύο σημείων. Το πρόβλημα του αλγορίθμου είναι ότι δεν είναι επιτυχής στο να κανονικοποιεί δεσμούς συστάδων. Το αποτέλεσμα του **ROCK** δεν είναι καλό για πολύπλοκες συστάδες με ποικίλες πυκνότητες δεδομένων. Ακόμα είναι πολύ ευπαθής στην επιλογή των παραμέτρων και ευαίσθητος στον θόρυβο.



Οι Ιεραρχικοί αλγόριθμοι είναι κατάλληλοι για συσταδοποίηση αντικειμένων με φυσική ιεραρχία, όπως στην ταξινομία φυτικού και ζωικού βασιλείου.

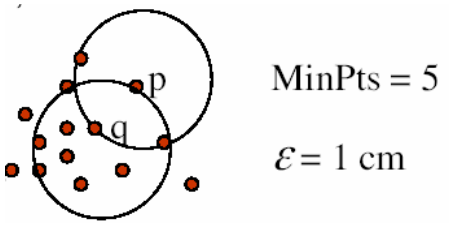
2.7 Μέθοδοι βασισμένες στην Πυκνότητα

Τα πλεονεκτήματα των μεθόδων που είναι βασισμένες στην πυκνότητα είναι ότι μπορούν να ανακαλύψουν συστάδες με αυθαίρετες μορφές, δεν χρειάζεται να προκαθοριστεί ο αριθμός των συστάδων, αναγνωρίζουν τα ακραία σημεία και δεν επηρεάζονται από θόρυβο. Ο αλγόριθμος **DBSCAN** *{Density-Based Spatial Clustering of Applications with Noise)* δημιουργεί συστάδες με ένα ελάχιστο μέγεθος και πυκνότητα. Ως πυκνότητα ορίζεται το ελάχιστο πλήθος σημείων που απέχουν συγκεκριμένη απόσταση μεταξύ τους. Αυτό αντιμετωπίζει το πρόβλημα των απομονωμένων σημείων καθώς εγγυάται ότι ένα τέτοιο σημείο (ή ένα μικρό σύνολο απομονωμένων σημείων) δε θα δημιουργήσει συστάδα. Μία παράμετρος εισόδου, ***MinPts****,* δείχνει το ελάχιστο πλήθος σημείων σε κάποια συστάδα. Επιπλέον, για κάθε σημείο συστάδας θα πρέπει να υπάρχει κάποιο άλλο σημείο στη συστάδα, η απόσταση του οποίου από το αρχικό σημείο να είναι μικρότερη απ' το κατώφλι εισόδου, ***Eps****.* Η ***Eps****-γειτονιά,* ή αλλιώς απλά, *γειτονιά,* ενός σημείου είναι το σύνολο των σημείων σε απόσταση ***Eps*** από το σημείο. Το επιθυμητό πλήθος συστάδων, *k,* δεν αποτελεί είσοδο αλλά αντιθέτως προσδιορίζεται από τον ίδιο τον αλγόριθμο.

Ο αλγόριθμος **DBSCAN** χρησιμοποιεί μια νέα έννοια για την πυκνότητα. Στο παρακάτω σχήμα ορίζονται τα σημεία που είναι *άμεσα προσεγγίσιμα με βάση την πυκνότητα* (directly density-reachable). Το πρώτο μέρος του ορισμού εξασφαλίζει ότι το δεύτερο σημείο είναι αρκετά κοντά στο πρώτο σημείο. Το δεύτερο μέρος του ορισμού εξασφαλίζει ότι υπάρχουν αρκετά σημεία *πυρήνες* (core points) σε κοντινές μεταξύ τους αποστάσεις. Τα σημεία αυτά σχηματίζουν το βασικό τμήμα της συστάδας δεδομένου ότι βρίσκονται όλα κοντά μεταξύ τους. Ένα άμεσα προσεγγίσιμο σημείο με βάση την πυκνότητα θα πρέπει να βρίσκεται κοντά σε ένα απ' αυτά τα σημεία πυρήνες, αλλά δεν χρειάζεται να είναι και το ίδιο πυρήνας. Στην περίπτωση αυτή, καλείται *οριακό σημείο* (border point).

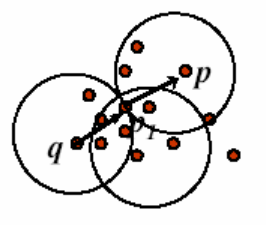
Δοθέντων των τιμών *Eps* και *MinPts,* ένα σημείο *ρ* είναι **άμεσα προσεγγίσιμο με βάση την πυκνότητα** από το σημείο *q* αν

- $\text{dis}(p, q) \leq Eps$ και
- $|\{r \mid \text{dis}(r, q) \leq Eps\}| \geq MinPts$

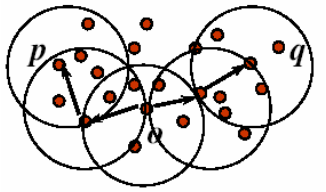


Ένα σημείο *ρ* σε σχέση με κάποιο άλλο *q* καλείται *προσεγγίσιμο με βάση την πυκνότητα* (density-reachable) από το σημείο *q* αν υπάρχει αλυσίδα *n* σημείων $p_1, p_2, p_3, \ldots p_n$ (όπου $p_1=p$, $p_n=q$) από το ένα σημείο στο άλλο που περιέχει μόνο σημεία, καθένα εκ των οποίων είναι άμεσα προσεγγίσιμο με βάση την πυκνότητα από το προηγούμενο σημείο. Αυτό εγγυάται ότι κάθε συστάδα θα έχει ένα σύνολο σημείων που βρίσκονται πολύ

κοντά σε μεγάλο αριθμό σημείων (πυρήνες) και κάποια άλλα σημεία που είναι αρκετά κοντά σε έναν τουλάχιστον πυρήνα (οριακά σημεία).



Ένα σημείο *ρ* σε σχέση με κάποιο άλλο *q* καλείται *συνδεδεμένο με βάση την πυκνότητα* (density-connected) με το σημείο *q* αν υπάρχει σημείο *ο* τέτοιο ώστε και τα δύο σημεία p και q είναι *προσεγγίσιμα με βάση την πυκνότητα* από το σημείο *ο*. Αυτό εγγυάται μία γέφυρα που μπορεί να ενώσει δύο σημεία και αποτελεί μία συμμετρική σχέση.

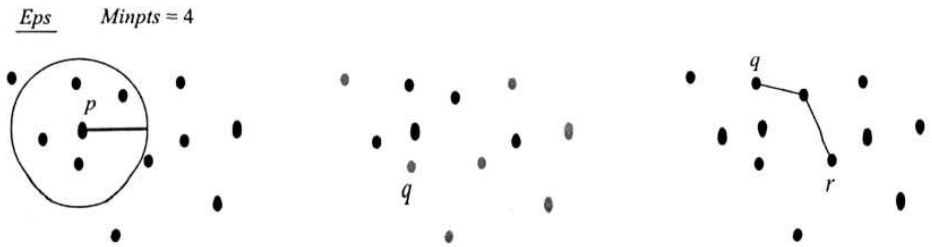


Για κάθε ζεύγος p,q θα ισχύει: Εάν το p ανήκει στη συστάδα C και το q είναι *προσεγγίσιμο με βάση την πυκνότητα* τότε και το q θα ανήκει στη συστάδα C.

Για κάθε ζεύγος p,q της συστάδας C θα ισχύει: το σημείο *ρ* θα είναι *συνδεδεμένο με βάση την πυκνότητα* με το σημείο *q*.

Άρα μία συστάδα ορίζεται να είναι ένα σύνολο από σημεία *συνδεδεμένα με βάση την πυκνότητα* με τη μέγιστη *προσεγγισιμότητα με βάση την πυκνότητα*. Ο θόρυβος είναι τα σημεία που δεν ανήκουν σε καμία συστάδα.

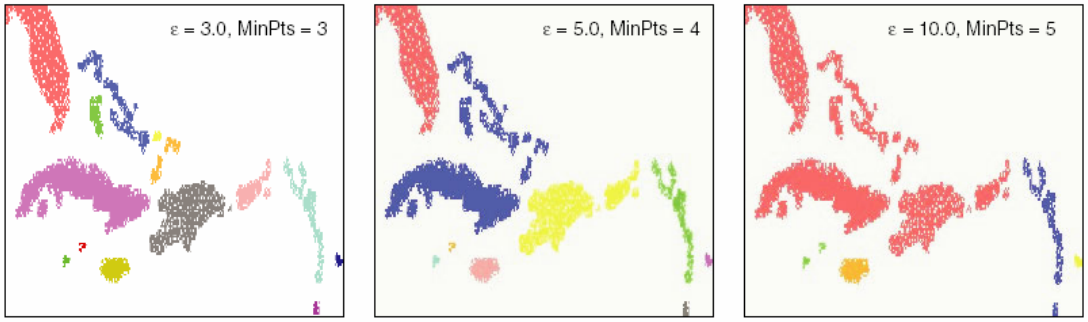
Στο παρακάτω σχήμα φαίνονται 12 σημεία. Η τιμή *Eps* απεικονίζεται από την κάθετη γραμμή. Στο τμήμα (α) του σχήματος φαίνεται ότι υπάρχουν τέσσερα σημεία στη γειτονιά του σημείου *ρ*. Όπως φαίνεται και από το σχήμα, το *ρ* αποτελεί σημείο πυρήνα επειδή έχει 4 (τιμή ***MinPts****)* σημεία στη γειτονιά του. Στο τμήμα (β) του σχήματος φαίνονται τα συνολικά 5 σημεία πυρήνες. Προσέξτε πως από τα 4 σημεία της γειτονιάς του *ρ* μόνο 3 είναι πυρήνες. Αυτά τα 4 σημεία καλούνται άμεσα προσεγγίσιμα από το σημείο *ρ* με βάση την πυκνότητα. Το σημείο *q* δεν είναι πυρήνας και γι' αυτό καλείται οριακό σημείο. Έχουμε χωρίσει τα σημεία σε ένα σύνολο πυρήνων, που όλοι βρίσκονται κοντά μεταξύ τους, στα οριακά σημεία που βρίσκονται κοντά σε έναν τουλάχιστον πυρήνα και τέλος στα εναπομείναντα σημεία που δε βρίσκονται κοντά σε κανέναν πυρήνα. Στο τμήμα (γ) του σχήματος φαίνεται ότι παρόλο που το σημείο *ρ* δεν είναι πυρήνας, είναι προσεγγίσιμο με βάση την πυκνότητα από το σημείο ***q***.



(α) Eps-γειτονιά β) Σημεία πυρήνες (γ)Σημεία προσεγγίσιμα με βάση την πυκνότητα

Εξαιτίας των περιορισμών σχετικά με το τι απαρτίζει μια συστάδα, όταν τερματίζει ο αλγόριθμος μπορεί να υπάρχουν σημεία που δεν θα συνδέονται με καμία συστάδα. Τα σημεία αυτά προσδιορίζονται ως θόρυβος.

Αν και ο **DBSCAN** υπερτερεί σημαντικά σε σχέση με τους αλγόριθμους που παρουσιάστηκαν μέχρι τώρα δεν βρίσκει πάντα αυτό που υπόσχεται καθώς εξαρτάται από τα **Eps** και **MinPts** και δεν αναγνωρίζει αποτελεσματικά τις συστάδες όταν η πυκνότητά τους ποικίλλει πολύ. Επιτυχή αλλά και ανεπιτυχή αποτελέσματα του αλγορίθμου φαίνονται στο παρακάτω σχήμα. Εάν αυξήσουμε πολύ το **Eps** τότε ο θόρυβος δημιουργεί γέφυρες οι οποίες προκαλούν κανονικές συστάδες να συγχωνεύονται.

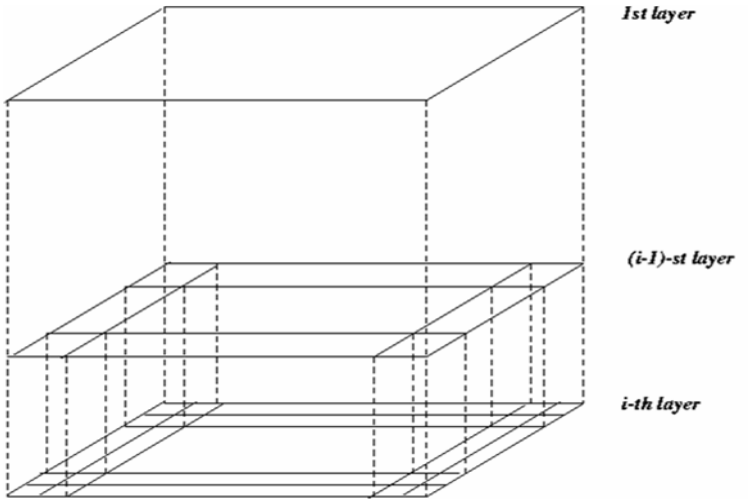


Συσταδοποίηση των νησιών της Καραϊβικής (20.000 σημεία) με τον DBSCAN. Στην αριστερή εικόνα τα βρήκε όλα. Η παράμετρος MinPts πρέπει να είναι γενικά >=4, αλλά ο αλγόριθμος είναι ευαίσθητος στην επιλογή του Eps (στην εικόνα το ε) που χαρακτηρίζει ένα μέτρο της μέσης απόστασης των σημείων.

Από τους ίδιους συγγραφείς ο **OPTICS** είναι μια προέκταση του **DBSCAN**. Αντί να παράγει ένα σύνολο από αποτελέσματα συσταδοποίησης με μια προκαθορισμένη ακτίνα **Eps**, ο **OPTICS** παράγει μια αθροιστική σειρά της βάσης δεδομένων παρουσιάζοντας έτσι την δομή συσταδοποίησης με βάση την πυκνότητα. Αυτή η ενέργεια της διάταξης των συστάδων ουσιαστικά περιέχει την πληροφορία για κάθε επίπεδο στις συσταδοποίησης της βάσης δεδομένων. Ο περιορισμός του **OPTICS** είναι ότι είναι πιο κατάλληλος για αριθμητικά δεδομένα. Ακόμα ο χρήστης πρέπει να καθορίσει την παράμετρο **MinPts**.

2.8 Grid-Based Αλγόριθμοι

Οι μέθοδοι που είναι βασισμένες σε πλέγμα, αρχικά κβαντικοποιούν τον χώρο συσταδοποίησης σε έναν πεπερασμένο αριθμό κελιών και έπειτα εφαρμόζουν την μέθοδο της συσταδοποίησης στα κελιά που έχουν προηγουμένως δημιουργηθεί. Το κύριο πλεονέκτημα των μεθόδων αυτών είναι πως η ταχύτητά τους εξαρτάται μόνο στην ανάλυση τους πλέγματος και όχι στο μέγεθος της βάσης δεδομένων. Τέτοιες μέθοδοι είναι κατάλληλες περισσότερο για σύνολα δεδομένων υψηλής πυκνότητας με μεγάλο πλήθος αντικειμένων σε μικρό χώρο.



Μία γνωστή μέθοδος η STING (a STatistical INformation Grid approach) από τους Wang, Yang και Muntz το1997.

Μια πρόσφατη προσέγγιση στον χώρο της συσταδοποίησης ο WaveCluster από τους Sheikholeslami, Chatterjee, και Zhang (VLDB'98), που βασίζεται σε κυματοειδείς μετασχηματισμούς. Η μέθοδος αυτή χρησιμοποιείται ως φίλτρο για τον καθορισμό την αναλογία συχνότητας του σήματος. Ένας κυματοειδής μετασχηματισμός ενός χωρικού αντικειμένου το αποσυνθέτει σε μια ιεραρχία από χωρικές εικόνες. Αυτές μπορούν να χρησιμοποιηθούν για κλιμάκωση μιας εικόνας σε διαφορετικά μεγέθη.

2.9 Model-Based Αλγόριθμοι

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 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.

2.10 Σύγκριση των μεθόδων συσταδοποίησης

Η ΚΑΤΑΛΛΗΛΗ ΜΕΘΟΔΟΣ ΣΥΣΤΑΔΟΠΟΙΗΣΗΣ ΘΑ ΠΡΕΠΕΙ ΝΑ

1. Χειρίζεται ικανοποιητικά πολλά δεδομένα
2. Αναγνωρίζει ακραία σημεία καθώς και θόρυβο
3. Αναγνωρίζει συστάδες ποικίλου μεγέθους, πυκνότητας και σχήματος.
4. Χειρίζεται δεδομένα πολλών διαστάσεων
5. Παρέχει συστηματικό τρόπο για την αναγνώριση του αριθμού των συστάδων
6. Απαιτεί μικρό βαθμό παραμετροποίησης και μικρό αριθμό παραμέτρων

Μέθοδοι	**1**	**2**	**3**	**4**	**5**	**6**
Partitioning	Yes/No	No	No	Yes	No	No
Hierarchical	No	No	Yes/No	Yes	Yes/No	Yes/No
Density-based	Yes	Yes	Yes	No	Yes/No	Yes/No
Grid based	Yes	Yes	Yes/No	No	Yes/No	Yes/No
Model-based	No	Yes	Yes/No	No	Yes	Yes/No

Κλιμάκωση σε μεγάλες βάσεις δεδομένων

Οι Partitioning, Hierarchical και Model-based μέθοδοι δεν κλιμακώνονται ικανοποιητικά όταν ο αριθμός των δεδομένων είναι πολύ μεγάλος, ειδικά για τη δεύτερη και τρίτη η κλιμάκωση είναι απαγορευτική. Οι Grid based μέθοδοι είναι πολύ αποτελεσματικές στις κλιμάκωση επειδή συμπυκνώνουν τα δεδομένα, με ενδεχόμενο φυσικά να χαθούν πληροφορίες. Οι Density-based μέθοδοι κλιμακώνονται ικανοποιητικά και ανταγωνίζονται τις Grid based δίχως το ρίσκο της συμπύκνωσης των δεδομένων.

Παρότι δεν φαίνεται στον πίνακα σύγκρισης ο DBSCAN είναι δυνατόν να χειριστεί αποτελεσματικά δεδομένα με περισσότερες από 20 διαστάσεις με κατάλληλη ενσωμάτωση των διαστάσεων [IBM report 2006] με multi dimensional scaling (MDS), και η επιλογή της παραμέτρου Eps μπορεί να προεκτιμηθεί από τις μέσες αποστάσεις των σημείων.

ΑΝΑΦΟΡΕΣ

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

Jiawei Han and Micheline Kamber “Data Mining: Concepts and Techniques”, Morgan Kaufmann Publishers, August 2000.

R. Ng and J. Han “Efficient and effective clustering method for spatial data mining”, in Proc. Conf. on Very Large Data Bases , pp 144-155, 1994.

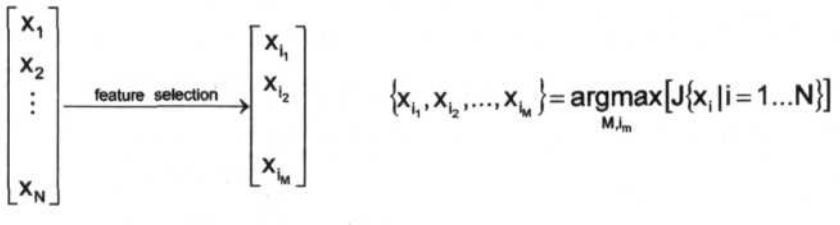
M. Ester, H.-P. Kriegel, J. Sander, X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, in Proc. ACM-SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, pp 226-231,1996.

Osmar R. Zaïane “Machine Learning and Data Mining” Dunham University, Lecture notes 2005.

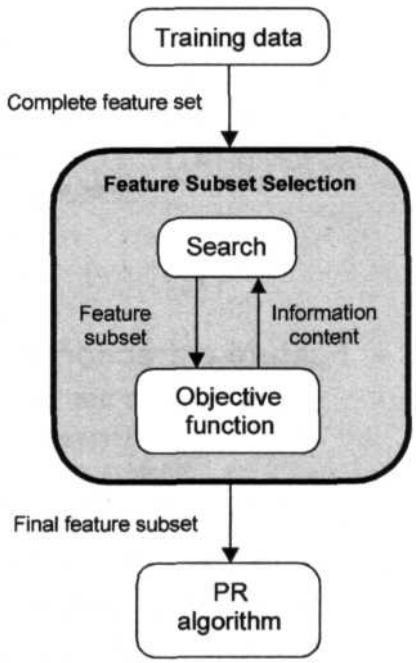
3 ΕΠΙΛΟΓΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ

3.1 Εισαγωγή

Η επιλογή χαρακτηριστικών είναι η επιλογή ενός υποσυνόλου από τα υπάρχοντα χαρακτηριστικά χωρίς μετασχηματισμό. Η επιλογή χαρακτηριστικών (Feature Selection) ονομάζεται και αλλιώς FSS (Feature Subset Selection), την επιλογή δηλαδή ενός υποσυνόλου χαρακτηριστικών. Ο ορισμός του FSS μπορεί να ορισθεί ως εξής: Δεδομένου ενός συνόλου χαρακτηριστικών $X = \{x_i \mid i = 1...N\}$ να βρεθεί ένα υποσύνολο $Y_M = \{x_{i1}, x_{i2}, ..., x_{iM}\}$, με M<N που να βελτιστοποιεί μια αντικειμενική συνάρτηση $J(Y)$



Η μέθοδος επιλογής χαρακτηριστικών απαιτεί μια *στρατηγική αναζήτησης* για να γίνει η επιλογή των υποψηφίων χαρακτηριστικών και μία *αντικειμενική συνάρτηση* με την χρήση της οποίας θα γίνεται η αξιολόγηση αυτών.



3.2 Στρατηγική Αναζήτησης

Έστω ότι χρειάζεται να βρεθούν τα δέκα καλύτερα χαρακτηριστικά (M=10) από ένα σύνολο είκοσι (N=20).

Η διεξοδική αξιολόγηση των υποσυνόλων των χαρακτηριστικών περιλαμβάνει

$\binom{N}{M} = \frac{N!}{(N-M)!\,M!}$ συνδυασμούς για σταθερή τιμή του M και

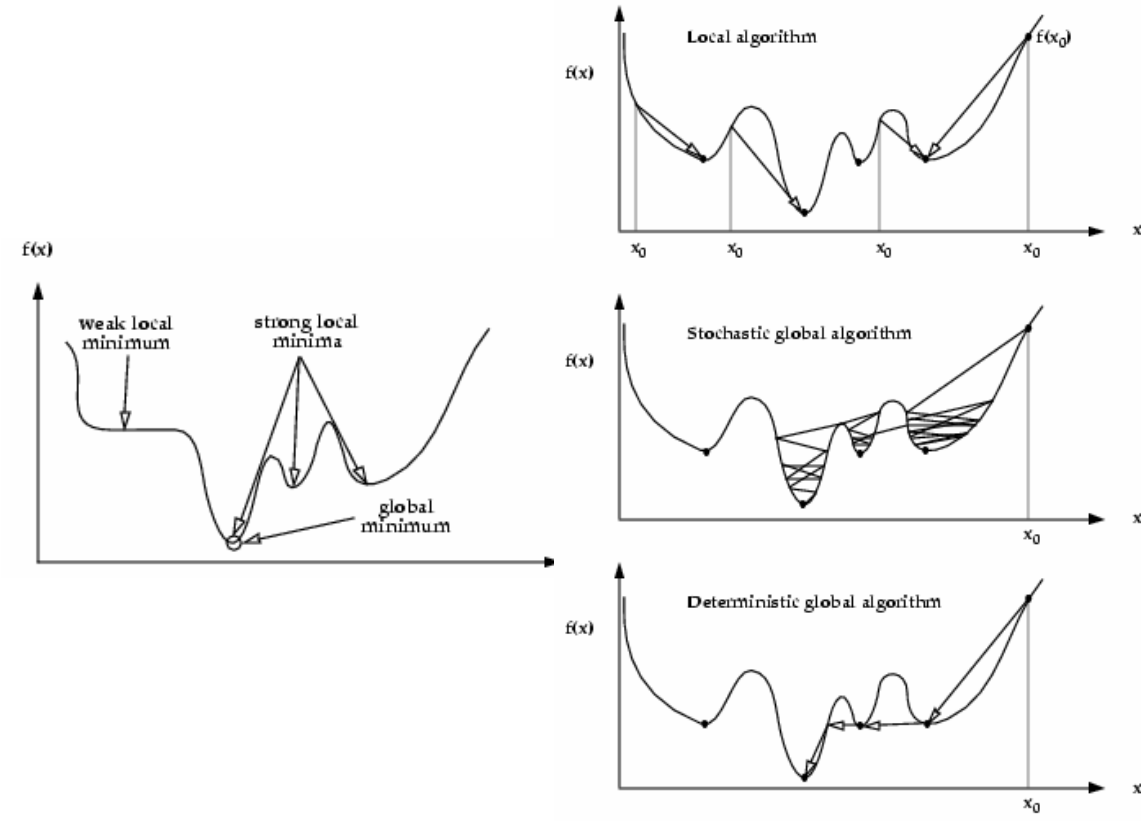
2^N συνδυασμούς εάν και το M θα πρέπει να βελτιστοποιηθεί.

Ο αριθμός των συνδυασμών είναι μη πραγματοποιήσιμος ακόμα και για μέτριες τιμές των M και N και θα πρέπει να χρησιμοποιηθεί μια διαδικασία αναζήτησης.

Για παράδειγμα, η διεξοδική αξιολόγηση των δέκα από τα είκοσι χαρακτηριστικά περιλαμβάνει 184,756 υποσύνολα χαρακτηριστικών. Διεξοδική αξιολόγηση των δέκα από τα εκατό περιλαμβάνει περισσότερα από 10^{13} υποσύνολα χαρακτηριστικών.

Για αυτόν τον λόγο χρειάζεται μία στρατηγική αναζήτησης που θα κατευθύνει την διαδικασία του Feature Subset Selection δεδομένου ότι ερευνά το διάστημα όλων του πιθανών συνδυασμών των χαρακτηριστικά.

Στην παρακάτω εικόνα φαίνεται αριστερά ποιο είναι το γενικό πρόβλημα και δεξιά ποιες οι πιθανότερες στρατηγικές.



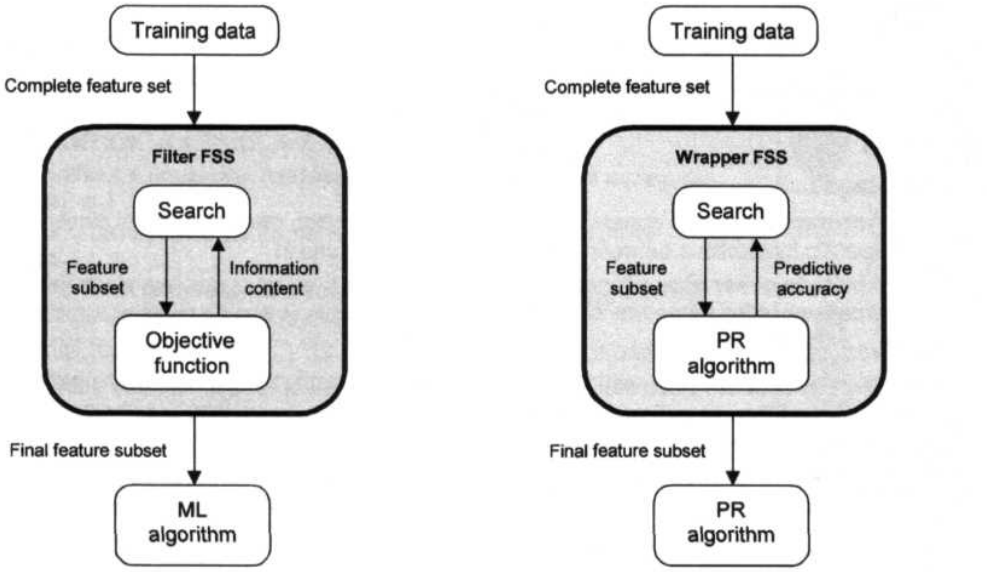
3.3 Αντικειμενική Συνάρτηση

Η αντικειμενική συνάρτηση αξιολογεί τα υποψήφια υποσύνολα και επιστρέφει ένα μέτρο της καταλληλότητας, ένα σήμα απόκρισης που χρησιμοποιείται από την στρατηγική έρευνας για διαλέξει καινούριους υποψήφιους.

Οι αντικειμενικές συναρτήσεις χωρίζονται σε δύο ομάδες:

Filters (Φίλτρα): Η αντικειμενική συνάρτηση αξιολογεί τα υποσύνολα των χαρακτηριστικών βάση του περιεχομένου της πληροφορίας τους, τυπικά η δια-κλασική απόσταση (interclass distance), αλλιώς η στατιστική εξάρτηση ή μέτρα information-theoretic.

Wrappers (Περιτύλιγμα): Η αντικειμενική συνάρτηση είναι ένας ταξινομητής προτύπων, ο οποίος αξιολογεί τα υποσύνολα των χαρακτηριστικών βάση της ακρίβειας πρόβλεψης τους (predictive accuracy), βάση της στατιστικής δειγματοληψίας (statistical resampling), ή διασταύρωση της εγκυρότητας (cross-validation).



3.4 Τύποι Φίλτρων

3.4.1 Μέτρα διαχωρισμού κλάσεων ή απόστασης.

Αυτές οι μέθοδοι χρησιμοποιούν το μέτρο της απόστασης για να καθορίσουν την διαχωριστικότητα μεταξύ των κλάσεων όπως την Ευκλειδια Απόσταση, την απόσταση Mahalanobis και άλλες.

3.4.2 Μέτρα συσχέτισης και information-theoretic.

Αυτές οι μέθοδοι βασίζονται στην λογική πως τα καλά υποσύνολα χαρακτηριστικών περιέχουν χαρακτηριστικά με υψηλή συσχέτιση με την κλάση αλλά ασυσχέτιστα σε σχέση με τα άλλα.

Μέτρα γραμμικής συσχέτισης. Η γραμμική συσχέτιση μπορεί να μετρηθεί χρησιμοποιώντας τον παράγοντα συσχέτισης

$$J(Y_M) = \frac{\sum_{i=1}^{M} \rho_{ic}}{\sum_{i=1}^{M} \sum_{j=i+1}^{M} \rho_{ij}}$$

όπου το ρ_{ic} είναι ο παράγοντας συσχέτισης μεταξύ χαρακτηριστικού i και την ετικέτα της κλάσης και το ρ_{ij} είναι ο παράγοντας συσχέτισης μεταξύ των χαρακτηριστικών i και j.

Μη γραμμικά μέτρα συσχέτισης. Η συσχέτιση είναι ικανή για την μέτρηση της γραμμικής εξάρτησης. Ένα πιο δυνατό μέτρο είναι αυτό της αμοιβαίας πληροφορίας $I(Y_κ;C)$

$$J(Y_M) = I(Y_M;C) = H(C) - H(C \mid Y_M) = \sum_{c=1}^{C} \int_{Y_M} P(Y_M, \omega_c) \lg \frac{P(Y_M, \omega_c)}{P(Y_M)P(\omega_c)} dx$$

Η αμοιβαία πληροφορία μεταξύ του ανύσματος του χαρακτηριστικού και της ετικέτας της κλάσης $I(Y_M;C)$ υπολογίζει το ποσό με το οποίο μειώνεται η αβεβαιότητα στην κλάση H(C) βάση της γνώσης του ανύσματος του χαρακτηριστικού $H(C|Y_M)$. Όπου H() η εντροπία.

3.5 Φίλτρα και Wrappers Πλεονεκτήματα και Μειονεκτήματα

3.5.1 Πλεονεκτήματα Φίλτρων

Γρήγορη Εκτέλεση:Τα φίλτρα περιλαμβάνουν γενικά έναν μη-επαναληπτικό υπολογισμό στο σύνολο δεδομένων, το οποίο μπορεί εκτελεστεί πολύ γρηγορότερα από μια άσκηση εκπαίδευσης ταξινομητών.

Γενικότητα: Δεδομένου ότι τα φίλτρα αξιολογούν τις εγγενείς ιδιότητες των στοιχείων, παρά τις αλληλεπιδράσεις τους με έναν ιδιαίτερο ταξινομητή, τα αποτελέσματά τους εκθέτουν περισσότερη γενικότητα. Η λύση θα είναι "καλή " για μία μεγαλύτερη οικογένεια των ταξινομητών.

3.5.2 Μειονεκτήματα Φίλτρων

Τάση να επιλεχτούν τα μεγάλα υποσύνολα: Από την στιγμή που οι συναρτήσεις που χρησιμοποιούν τα φίλτρα είναι μονοτονικές, το φίλτρο τείνει να επιλέξει ολόκληρο το σύνολο χαρακτηριστικών ως μια βέλτιστη λύση. Αυτό αναγκάζει το χρήστη για να επιλέξει αυθαίρετη διακοπή στον αριθμό των χαρακτηριστικών που πρέπει να επιλεχθούν.

3.5.3 Πλεονεκτήματα Wrappers

Ακρίβεια: τα wrappers επιτυγχάνουν γενικά καλύτερα ποσοστά αναγνώρισης από τα φίλτρα δεδομένου ότι είναι συντονισμένα στις συγκεκριμένες αλληλεπιδράσεις μεταξύ του ταξινομητή και του συνόλου δεδομένων

Δυνατότητα να γενικεύσει: Τα wrappers έχουν έναν μηχανισμό για να αποφύγουν το υπέρ - ταίριασμα , δεδομένου ότι χρησιμοποιούν τυπικά μέτρα cross-validation. προφητικής ακρίβειας

3.5.4 Μειονεκτήματα Wrappers

Αργή εκτέλεση: Δεδομένου ότι τα wrappers πρέπει να εκπαιδεύσουν έναν ταξινομητή για κάθε υποσύνολο χαρακτηριστικών γνωρισμάτων (ή αρκετούς ταξινομητές εάν η cross-validation χρησιμοποιείται), η μέθοδος μπορεί να καταστεί αδύνατη για τις εξαντλητικές μεθόδους υπολογισμών.

Έλλειψη γενικότητας : Η λύση στερείται τη γενικότητα δεδομένου ότι είναι δεμένη στην προκατάληψη του ταξινομητή που χρησιμοποιείται κατά την λειτουργία αξιολόγησης. Το "βέλτιστο" υποσύνολο χαρακτηριστικών γνωρισμάτων θα είναι συγκεκριμένο για τον ταξινομητή υπό κατασκευή.

3.6 Στρατηγικές Αναζήτησης

Υπάρχει πολλές στρατηγικές αναζήτησης στον χώρο των καταστάσεων ενός προβλήματος (χώρος των δυνατών λύσεων) οι οποίες μπορούν να ομαδοποιηθούν σε τρεις γενικές κατηγορίες ιδιαίτερα δημοφιλείς και στην τεχνητή νοημοσύνη.

Σειριακοί αλγόριθμοι

Οι αλγόριθμοι αυτοί προσθέτουν ή αφαιρούν χαρακτηριστικά διαδοχικά, αλλά έχουν την τάση να παγιδεύονται σε τοπικά ελάχιστα.

- Σειριακή Προς τα εμπρός επιλογή (Sequential Forward Selection)
- Σειριακή Προς τα πίσω επιλογή (Sequential Backward Selection)
- Επιλογή συν -L πλην -R (Plus-l Minus-r Selection)
- Αμφίδρομη αναζήτηση (Bidirectional Search)
- Sequential Floating Selection

Εκθετικοί αλγόριθμοι

Οι αλγόριθμοι αυτοί αξιολογούν έναν αριθμό υποσυνόλων χαρακτηριστικών, ο οποίος αυξάνεται εκθετικά με την διάσταση του χώρου αναζήτησης.

- Εξαντλητική αναζήτηση (Exhaustive Search)
- διακλάδωση και οριοθέτηση (Branch and Bound)
- Εκτίμηση μονοτονικότητας με διακλάδωση και οριοθέτηση (Approximate Monotonicity with Branch and Bound)
- Ακτινική αναζήτηση (Beam Search)

Στοχαστικοί Αλγόριθμοι

Οι αλγόριθμοι αυτοί ενσωματώνουν την τυχαία επιλογή στην διαδικασία αναζήτησης για να αποφύγουν την παγίδευση σε τοπικά ελάχιστα.

- Τυχαία παραγωγή και σειριακή επιλογή (Random Generation and Sequential Selection)
- Προσομοιωμένη ανόπτηση (Simulated Annealing)
- Γενετικοί αλγόριθμοι (Genetic Algorithms)
- (άλλοι όπως στοχαστική ακτινική αναζήτηση, στοχαστική αναρρίχηση λόφων- hill climbing)

3.6.1 Σειριακοί αλγόριθμοι

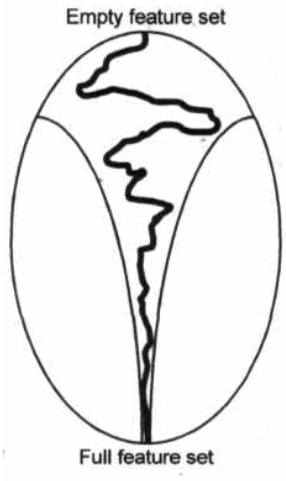
Οι αλγόριθμοι αυτοί προσθέτουν ή αφαιρούν χαρακτηριστικά διαδοχικά, αλλά έχουν την τάση να παγιδεύονται σε τοπικά ελάχιστα. Αντιπροσωπευτικοί αλγόριθμοι παρουσιάζονται παρακάτω.

Σειριακή Προς τα εμπρός επιλογή (Sequential Forward Selection)

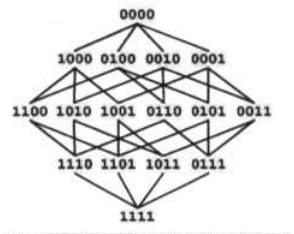
Ο αλγόριθμος αυτός είναι ο πιο απλός αλγόριθμος «άπληστης» αναζήτησης. Αρχίζοντας από ένα άδειο σύνολο, προσθέτουμε διαδοχικά το χαρακτηριστικό χ^+ το οποίο προκύπτει από την αντικειμενική συνάρτηση $J(Y_k+x^+)$ όταν συνδυάζεται με τα χαρακτηριστικά Y_k τα οποία έχουν ήδη επιλεχθεί.

Αλγόριθμος Sequential Forward Selection

1. Start with the empty set $Y_0=\{\varnothing\}$
2. Select the next best feature $x^+ = \underset{x \notin Y_k}{\operatorname{argmax}}\left[J(Y_k + x)\right]$
3. Update $Y_{k+1}=Y_k+x^+$; k=k+1
4. Go to 2



Εδώ θα πρέπει να παρατηρήσουμε πως ο Sequential Forward Selection (SFS) λειτουργεί καλύτερα όταν το βέλτιστο υποσύνολο έχει λίγα χαρακτηριστικά. Όταν η αναζήτηση είναι κοντά σε ένα άδειο σύνολο τότε ένας μεγάλος αριθμός από καταστάσεις μπορεί αξιολογηθεί. Ενώ όταν βρισκόμαστε κοντά σε ένα γεμάτο σύνολο τότε η περιοχή που εξετάζεται από τον Sequential Forward Selection είναι μικρότερη αφού τα περισσότερα χαρακτηριστικά έχουν ήδη επιλεχθεί. Ο χώρος αναζήτησης λύσεων σχεδιάστηκε σαν μια έλλειψη έτσι ώστε να δοθεί έμφαση στο γεγονός ότι όσο προχωράμε προς το γεμάτο σύνολο μειώνονται οι καταστάσεις. Σαν παράδειγμα παρουσιάζεται παρακάτω ένας χώρος καταστάσεων για 4 χαρακτηριστικά. Φαίνεται πως ο χώρος καταστάσεων στο κέντρο είναι μεγαλύτερος στο δέντρο αναζήτησης.

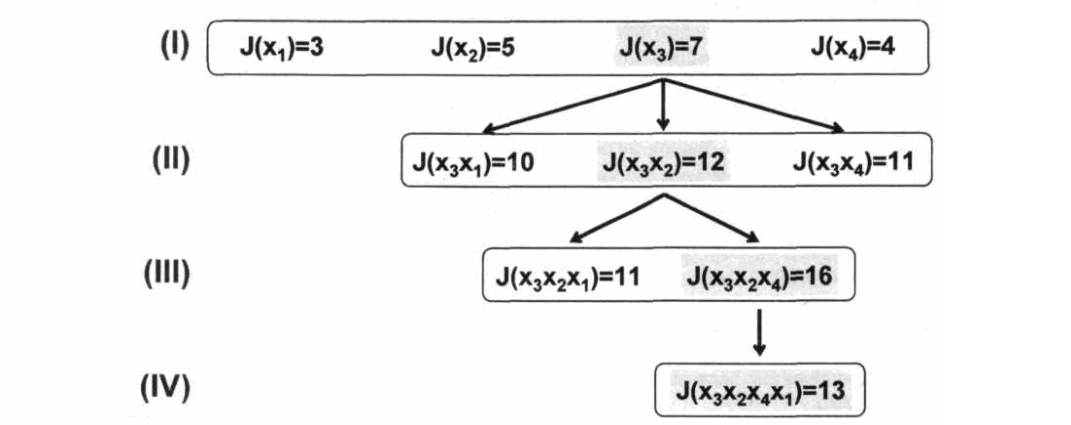


Το μειονέκτημα του SFS είναι ότι αδυνατεί να αφαιρέσει χαρακτηριστικά τα οποία καθίστανται ξεπερασμένα μετά την πρόσθεση άλλων χαρακτηριστικών. Αν υποθέσουμε πως έχουμε την παρακάτω αντικειμενική συνάρτηση

$$J(X) = -2x_1x_2 + 3x_1 + 5x_2 - 2x_1x_2x_3 + 7x_3 + 4x_4 - 2x_1x_2x_3x_4$$

όπου χ_κ είναι δείκτες μεταβλητών που καθορίζουν εάν το κ-οστό χαρακτηριστικό έχει επιλεχθεί (χ_κ=1) ή όχι (χ_κ=0).

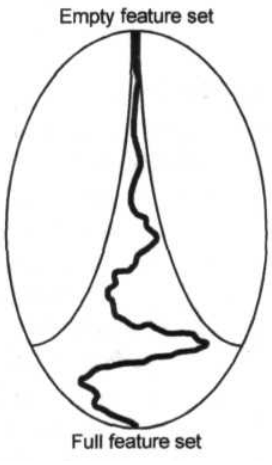
Λύση



Σειριακή Προς τα πίσω επιλογή (Sequential Backward Selection)

Ο αλγόριθμος αυτός λειτουργεί ουσιαστικά αντίθετα από τον Sequential Forward Selection. Αρχίζοντας από ένα γεμάτο σύνολο, αφαιρούμε διαδοχικά το χαρακτηριστικό x^- το οποίο προκύπτει από την μικρότερη μείωση της τιμής της αντικειμενικής συνάρτησης $J(Y- x^-)$.

Σε αυτό το σημείο θα πρέπει να παρατηρήσουμε πως η αφαίρεση ενός χαρακτηριστικού θα οδηγήσει σε μία αύξηση της τιμής της αντικειμενικής συνάρτησης $J(Y- x^-) > J(Y)$. Τέτοιες συναρτήσεις λέγονται μη μονότονες.



Ο Αλγόριθμος Sequential Backward Selection

1. Start with the full set $Y_0=X$
2. Remove the worst feature $x^- = \underset{x \in Y_k}{\arg\max}[J(Y_k - x)]$
3. Update $Y_{k+1}=Y_k-x^-$; $k=k+1$
4. Go to 2

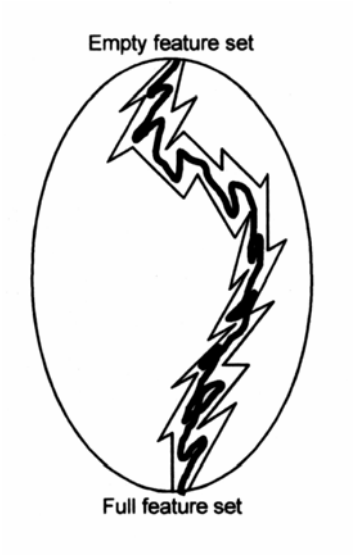
Ο Sequential Backward Selection (SBS) λειτουργεί καλύτερα όταν το βέλτιστο υποσύνολο χαρακτηριστικών έχει μεγάλο αριθμό χαρακτηριστικών γνωρισμάτων από την στιγμή που ο SBS ξοδεύει τον χρόνο του ερευνώντας μεγάλα υποσύνολα. Ο κύριος περιορισμός του SBS είναι η ανικανότητά του να επαναξιολογήσει την χρησιμότητα ενός χαρακτηριστικού αφού απορριφθεί.

Συν L Πλην R Επιλογή (Plus-L Minus-R Selection)

Ο αλγόριθμος Επιλογής συν L πλην R (LRS) αποτελεί ουσιαστικά μια γενίκευση του Sequential Forward Selection και Sequential Backward Selection. Εάν L>R ο LRS ξεκινά από το άδειο σύνολο και προσθέτει επανειλημμένα L χαρακτηριστικά και αφαιρεί R χαρακτηριστικά, ενώ εάν L<R τότε ο LRS ξεκινά από ένα γεμάτο σύνολο και επανειλημμένα αφαιρεί R χαρακτηριστικά τα οποία ακολουθούνται από L προσθέσεις χαρακτηριστικών.

Αλγόριθμος Plus-L Minus-R

1. If L>R then
   start with the empty set $Y=\{\varnothing\}$
   else
   start with the full set $Y=X$
   go to step 3
2. Repeat L times
   $$x^+ = \underset{x \in Y_k}{\operatorname{argmax}}[J(Y_k + x)]$$
   $$Y_{k+1} = Y_k + x^+; \quad k = k+1$$
3. Repeat R times
   $$x^- = \underset{x \in Y_k}{\operatorname{argmax}}[J(Y_k - x)]$$
   $$Y_{k+1} = Y_k - x^-; \quad k = k+1$$
4. Go to 2



Ο LRS προσπαθεί να αντισταθμιστεί για τις αδυναμίες των SFS και SBS με μερικές ικανότητες οπισθοδρόμησης.

Ο κύριος περιορισμός του είναι η έλλειψη μιας θεωρίας που θα βοηθήσει να προβλεφθεί ο αριθμός των L και R.

Αμφίδρομη αναζήτηση (Bidirectional Search -BDS)

Η αμφίδρομη αναζήτηση είναι ουσιαστικά μια παράλληλη εφαρμογή των SFS (Sequential Forward Selection) και του SBS (Sequential Backward Selection). Ο SFS εκτελείται από το άδειο σύνολο και ο SBS από το γεμάτο. Για να εγγυηθούμε πως ο SFS και ο SBS συγκλίνουν προς την ίδια λύση θα πρέπει να εξασφαλίσουμε πως τα χαρακτηριστικά που έχουν επιλεχθεί από τον SFS δεν θα αφαιρεθούν από τον SBS και τα χαρακτηριστικά που έχουν ήδη αφαιρεθεί από τον SBS δεν θα επιλεχθούν από τον SFS. Για παράδειγμα, πριν ο SFS προσπαθήσει να προσθέσει ένα νέο χαρακτηριστικό ελέγχει πρώτα εάν έχει αφαιρεθεί από τον SBS και εάν έχει, προσπαθεί να προσθέσει το δεύτερο καλύτερο κ.τ.λ. Κατά αντιστοιχία λειτουργεί και ο SBS.

Αλγόριθμος Bidirectional Search

1. Start SFS with the empty set $Y_F=\{\varnothing\}$
2. Start SBS with the full set $Y_B=X$
3. Select the best feature

$$x^+ = \underset{\substack{x\in Y_{F_k} \\ x\in Y_{B_k}}}{\operatorname{argmax}}\left[J\left(Y_{F_k}+x\right)\right]$$

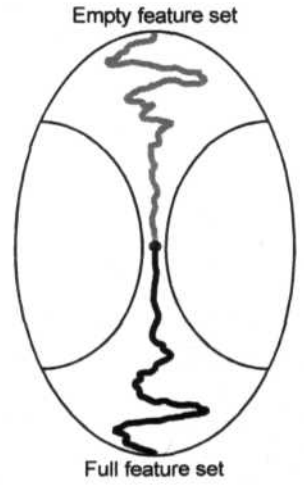
$$Y_{F_{k+1}} = Y_{F_k} + x^+$$

3. Remove the worst feature

$$x^- = \underset{\substack{x\in Y_{B_k} \\ x\in Y_{F_{k+1}}}}{\operatorname{argmax}}\left[J\left(Y_{B_k}-x\right)\right]$$

$$Y_{B_{k+1}} = Y_{B_k} - x^-; \quad k = k+1$$

4. Go to 2



Σειριακά κυμαινόμενη επιλογή (Sequential Floating Selection SFFS and SFBS)

Οι μέθοδοι της σειριακά κυμαινόμενης επιλογής χαρακτηριστικών είναι μία επέκταση των LRS αλγόριθμων (συν L πλην R Επιλογή) με ευέλικτες ικανότητες οπισθοδρόμησης. Αντί να φτιάχνει τις τιμές των L και R, αυτές οι κυμαινόμενες μέθοδοι επιτρέπουν σε αυτές τις τιμές να καθοριστούν από τα δεδομένα. Η διαστατικότητα του υποσυνόλου κατά την διάρκεια της αναζήτησης μπορεί να θεωρηθεί κυμαινόμενη προς τα πάνω ή κάτω. Υπάρχουν δύο κυμαινόμενες μέθοδοι, ο SFFS (Sequential Floating Forward Selection), ο οποίος αρχίζει από το άδειο σύνολο και μετά από κάθε βήμα προς τα εμπρός εκτελεί βήματα προς τα πίσω όσο αυξάνει η αντικειμενική συνάρτηση και ο SFBS (Sequential Floating Backward Selection), ο οποίος αρχίζει από το γεμάτο σύνολο και μετά από κάθε βήμα προς τα πίσω ο SFBS εκτελεί βήματα προς τα εμπρός όσο η αντικειμενική συνάρτηση αυξάνει.

Αλγόριθμος Sequential Floating Forward Selection (Ανάλογα και ο SFBS)

1. Start with the empty set Y={∅}
2. Select the best feature

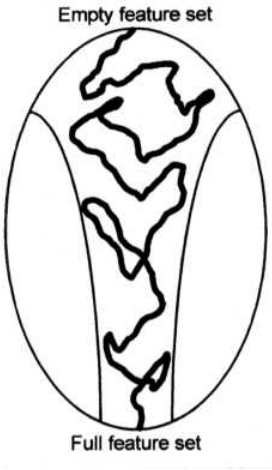
$$x^+ = \underset{x \notin Y_k}{\operatorname{argmax}}[J(Y_k + x)]$$

$$Y_k = Y_k + x^+; \quad k = k+1$$

3. Select the worst feature*

$$x^- = \underset{x \in Y_k}{\operatorname{argmax}}[J(Y_k - x)]$$

4. If $J(Y_k - x^-) > J(Y_k)$ then
   $Y_{k+1} = Y_k - x$; $k = k+1$
   go to Step 3
   else
   go to Step 2



Για την αποφυγή ατελείωτων επαναλήψεων είναι απαραίτητη η καταγραφή όλων των βημάτων.

3.6.2 Εκθετικοί αλγόριθμοι

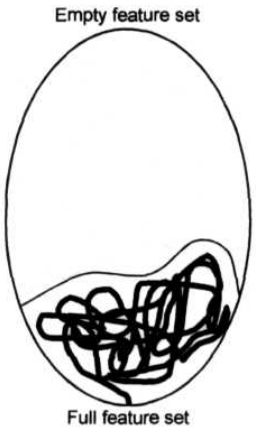
Οι αλγόριθμοι αυτοί αξιολογούν έναν αριθμό υποσυνόλων, ο οποίος αυξάνεται εκθετικά με την διαστατικότητα του χώρου αναζήτησης. Οι πιο αντιπροσωπευτικοί αλγόριθμοι παρουσιάζονται παρακάτω.

Διακλάδωση και οριοθέτηση (Branch and Bound -B&B)

Οι αλγόριθμοι διακλάδωσης και οριοθέτησης (B&B) χρησιμοποιούνται ευρύτατα στην επιχειρησιακή έρευνα. Ο αλγόριθμος εγγυάται να βρει το βέλτιστο υποσύνολο χαρακτηριστικών γνωρισμάτων βάση της προϋπόθεσης μονοτονίας της αντικειμενικής συνάρτησης δηλαδή

$$J(x_{i_1}) < J(x_{i_1}, x_{i_2}) < J(x_{i_1}, x_{i_2}, x_{i_3}) < \ldots < J(x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_N})$$

Η προϋπόθεση αυτή δηλώνει πως η πρόσθεση των χαρακτηριστικών μπορεί μόνο να αυξήσει την τιμή της αντικειμενικής συνάρτησης. Ο Branch and Bound ξεκινά από ολόκληρο το σύνολο και αφαιρεί χαρακτηριστικά χρησιμοποιώντας την στρατηγική depth-first. Οι κόμβοι των οποίων η αντικειμενική συνάρτηση είναι χαμηλότερη από την τρέχουσα καλύτερη, δεν εξερευνούνται καθώς η μονοτονία εγγυάται ότι τα «παιδιά» δεν θα έχουν καλύτερη λύση.



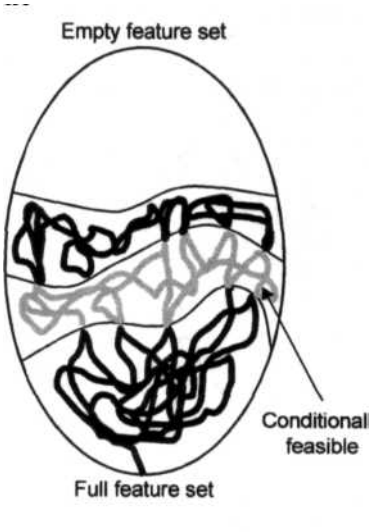
Εκτίμηση μονοτονίας με Διακλάδωση και Οριοθέτηση (Approximate Monotonicity with Branch and Bound)

Ο αλγόριθμος αυτός είναι μια παραλλαγή της διακλάδωσης με οριοθέτηση. Ο AMB&B επιτρέπει την χρήση μη μονότονων συναρτήσεων- ταξινομητών ουσιαστικά- ελαφραίνοντας έτσι την συνθήκη αποκοπής, η οποία τερματίζει την αναζήτηση στον συγκεκριμένο κόμβο. Αν υποθέσουμε ότι εκτελούμε τον αλγόριθμο θέτοντας ένα κατώφλι ποσοστού λάθους E(Y)=τ αντί για τον αριθμό των χαρακτηριστικών M.

Με τον AMB&B , ένα υποσύνολο χαρακτηριστικών Y θα θεωρηθεί

- **Εφικτό**, εάν $E(Y) \le \tau$
- **Υπό συνθήκες εφικτό**, εάν $\mathrm{E(Y)} \le \tau(1+\Delta)$
- **Ανέφικτο**, εάν $\mathrm{E(Y)} \ge \tau(1+\Delta)$

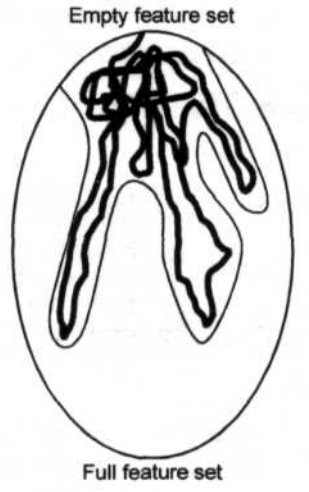
Το Δ είναι η ανοχή που τοποθετείται στο κατώφλι για να προσαρμοστεί για μη μονότονες συναρτήσεις. Αντί να περιορίζουμε την αναζήτηση σε εφικτούς κόμβους, όπως κάνει ο B&B, ο AMB&B επιτρέπει την αναζήτηση να εξερευνήσει και τους υπό συνθήκη εφικτούς κόμβους με την ελπίδα πως αυτοί οι κόμβοι θα οδηγήσουν σε μία εφικτή λύση.



Εντούτοις, ο AMB&B δεν θα επιστρέψει υπό συνθήκη εφικτούς κόμβους ως λύσεις, απλά επιτρέπει στην αναζήτηση να τους εξερευνήσει. Αλλιώς δεν θα ήταν διαφορετικός από τον B&B με υψηλότερο κατώφλι της τάξης του $\tau(1+\Delta)$.

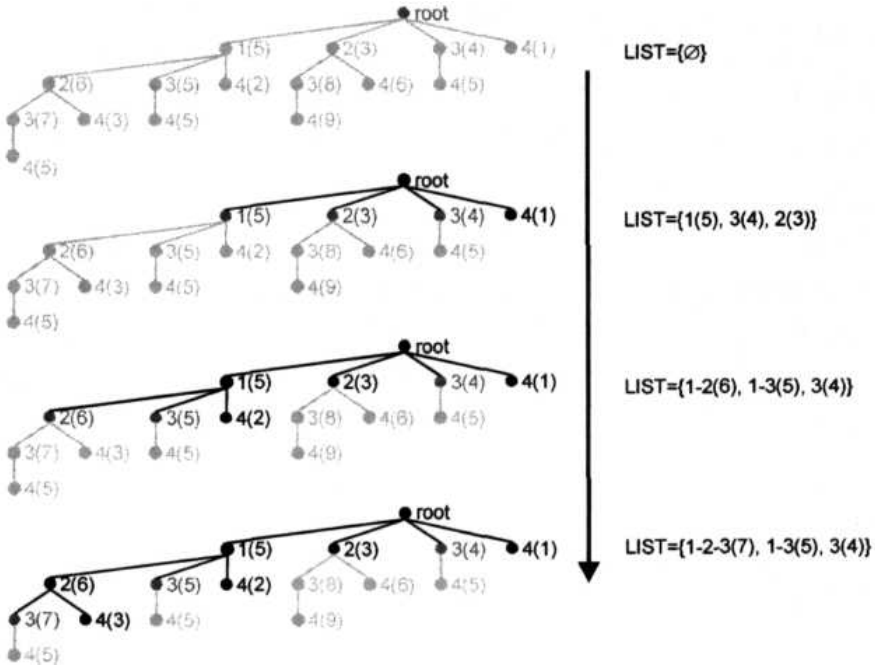
Ακτινική αναζήτηση (Beam Search)

Η ακτινική αναζήτηση (Beam Search) είναι μια παραλλαγή της αναζήτησης πρώτα στο καλύτερο (best first) με οριοθετημένη σειρά για τον περιορισμό της αναζήτησης.



Η σειρά οργανώνει καταστάσεις από την καλύτερη στην χειρότερη με τις καλύτερες καταστάσεις να τοποθετούνται στην αρχή της σειράς. Σε κάθε επανάληψη ο BS αξιολογεί όλες τις πιθανές καταστάσεις που προκύπτουν από την πρόσθεση χαρακτηριστικών στο υποσύνολο και τα αποτελέσματα εισέρχονται σε μια σειρά σε κατάλληλες θέσεις. Θα πρέπει να παρατηρήσουμε σε αυτό το σημείο ότι ο BS μετατρέπεται σε εξαντλητική αναζήτηση εάν δεν υπάρξει περιορισμός στο μέγεθος της σειράς. Παρομοίως, εάν το μέγεθος της σειράς είναι 1, τότε ο αλγόριθμος BS είναι ισοδύναμος με την SFS (Sequential Forward Selection).

Παρακάτω παρουσιάζεται ο BS για ένα τετραδιάστατο χώρο αναζήτησης και με σειρά μεγέθους 3.



Ο αλγόριθμος BS δεν μπορεί να εγγυηθεί ότι θα βρεθεί το βέλτιστο υποσύνολο χαρακτηριστικών. Στο παραπάνω παράδειγμα το βέλτιστο είναι 2-3-4 (J=9) το οποίο δεν ερευνάται ποτέ. Παρόλα αυτά, με κατάλληλο μέγεθος σειράς ο BS μπορεί να αποφύγει την παγίδευση σε τοπικά ελάχιστα με την διατήρηση λύσεων από διαφορετικές περιοχές στον χώρο αναζήτησης.

3.6.3 Στοχαστικοί Αλγόριθμοι

Οι αλγόριθμοι αυτοί ενσωματώνουν την «τυχαία επιλογή» στην διαδικασία αναζήτησης για να αποφύγουν την παγίδευση σε τοπικά ελάχιστα. Αντιπροσωπευτικοί αλγόριθμοι παρουσιάζονται παρακάτω.

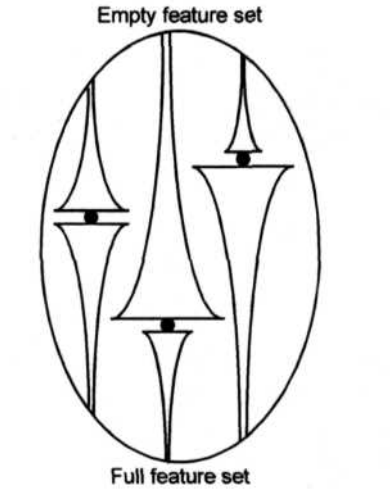
Τυχαία παραγωγή και σειριακή επιλογή (Random Generation Plus Sequential Selection)

Ο αλγόριθμος RGSS αποτελεί μια προσπάθεια για εισαγωγή του «τυχαίου» στους αλγόριθμους SBS και SFS έτσι ώστε να αποφευχθούν τα τοπικά ελάχιστα.

Στην αρχή παράγεται ένα τυχαίο υποσύνολο χαρακτηριστικών.

Μετά εκτελείται ο Sequential Forward Selection και

μετά ο Sequential Backward Selection.



Προσομοιωμένη ανόπτηση (Simulated Annealing)

Η προσομοιωμένη ανόπτηση είναι μια μέθοδος στοχαστικής βελτιστοποίησης που παίρνει το όνομά της από την διαδικασία ανόπτησης που χρησιμοποιείται για την επανακρυστάλλωση των μετάλλων. Κατά την διάρκεια της ανόπτησης το κράμα ψύχεται αργά ώστε να επιτραπεί στα άτομα του να φτάσουν στην κατάσταση ελάχιστης ενέργειας (κρυσταλλική δομή). Εάν το κράμα υποβληθεί σε μια γρήγορη διαδικασία ανόπτησης τότε δεν θα μπορεί να επιτευχθεί η ηρεμία σε όλο το υλικό. Το αποτέλεσμα θα είναι ένα υλικό με περιοχές με κανονική δομή που θα χωρίζονται από σύνορα. Τα σύνορα αυτά θα είναι πιθανές ελαττωματικές γραμμές όπου είναι πολύ πιθανό να συμβούν σπασίματα όταν το υλικό δεχθεί μεγάλη πίεση.

Σύμφωνα με τη στατιστική Boltzmann, σε μια θερμοκρασία T, η πιθανότητα αύξησης της διαφοράς ενέργειας ΔE στο σύστημα δίνεται από την παρακάτω έκφραση.

$$P(\Delta \mathrm{E}) = e^{-\frac{\Delta \mathrm{E}}{\kappa \mathrm{T}}}$$

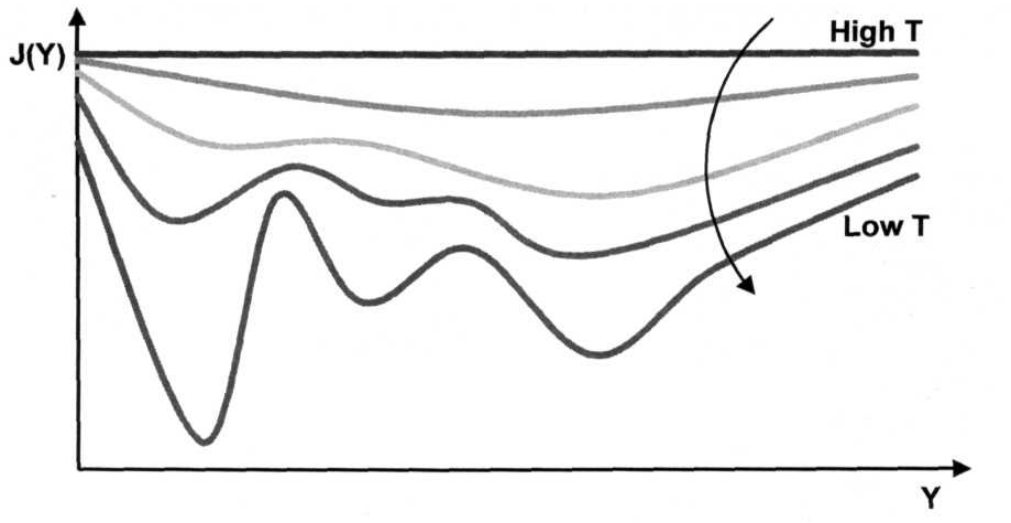
Όπου k είναι η λεγόμενη σταθερά του Boltzmann.

Το γενικό πλάνο που ακολουθείται είναι το εξής:

1. Καθορίζεται ένα αρχικό πρόγραμμα ανόπτησης T(i).
2. Δημιουργείται μια αρχική λύση (τυχαία) Y(0)
3. Όσο το $T(i)>T_{MIN}$
   - o Παράγεται καινούρια λύση Y(i+1) που είναι ένας κοντινότερος συγγενής του Y(i).
   - o Υπολογίζεται η διαφορά $\Delta E = [J(Y(i)) - J(Y(i+1))]$
   - o Εάν ΔE<0 τότε δεχόμαστε την κίνηση από το Y(i) στο Y(i+1)
   - o Αλλιώς δεχόμαστε την κίνηση με πιθανότητα $P = \exp(-\Delta E / T(i))$

Η γενική ιδέα: Όταν βελτιστοποιείται ένα πολύ μεγάλο και σύνθετο σύστημα (ένα σύστημα με πολλούς βαθμούς ελευθερίας), πήγαινε κατηφορικά για να βρεις το ελάχιστο, αλλά καμία φορά πήγαινε και ανηφορικά.

Η επιλογή του annealing schedule είναι πολύ κρίσιμη καθώς εάν το Y είναι πολύ μεγάλο τότε η θερμοκρασία μειώνεται πολύ αργά επιτρέποντας έτσι την μετάβαση σε υψηλότερες ενεργειακές καταστάσεις πιο συχνά. Από την άλλη εάν το Y είναι πολύ μικρό τότε η θερμοκρασία μειώνεται πολύ γρήγορα αυξάνοντας έτσι την πιθανότητα να εγκλωβιστεί ο αλγόριθμος σε τοπικό ελάχιστο. Ένα μοναδικό χαρακτηριστικό του SA είναι η προσαρμοστική φύση του. Σε υψηλή θερμοκρασία ο αλγόριθμος ψάχνει μόνο τα «άσχημα» χαρακτηριστικά της επιφάνειας βελτιστοποίησης , ενώ σε χαμηλές θερμοκρασίες εμφανίζονται με μεγαλύτερη λεπτομέρεια τα χαρακτηριστικά της επιφάνειας.

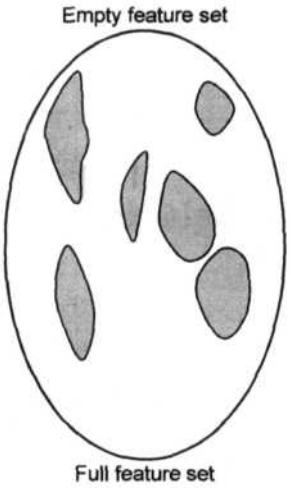


Γενετικοί αλγόριθμοι (Genetic Algorithms)

Οι γενετικοί αλγόριθμοι είναι τεχνικές βελτιστοποίησης που μιμούνται την διαδικασία της επικράτησης του πιο δυνατού. Αρχίζοντας με έναν αρχικό τυχαίο πληθυσμό λύσεων εξελίσσει νέους πληθυσμούς με ζευγάρωμα (διασταύρωση) δύο λύσεων ή μετάλλαξη κάθε μίας λύσης σύμφωνα με μία συνάρτηση καταλληλότητας (αντικειμενική συνάρτηση). Οι καλύτερες λύσεις είναι πιο πιθανό να επιλεγούν για την διαδικασία του ζευγαρώματος ή την διαδικασία της μετάλλαξης και για αυτό τον λόγο φέρουν έναν «γενετικό κώδικα» από γενιά σε γενιά. Για το πρόβλημα της επιλογής χαρακτηριστικών μεμονωμένες λύσεις παρουσιάζονται με έναν δυαδικό αριθμό (1 εάν έχει επιλεχθεί, αλλιώς 0).

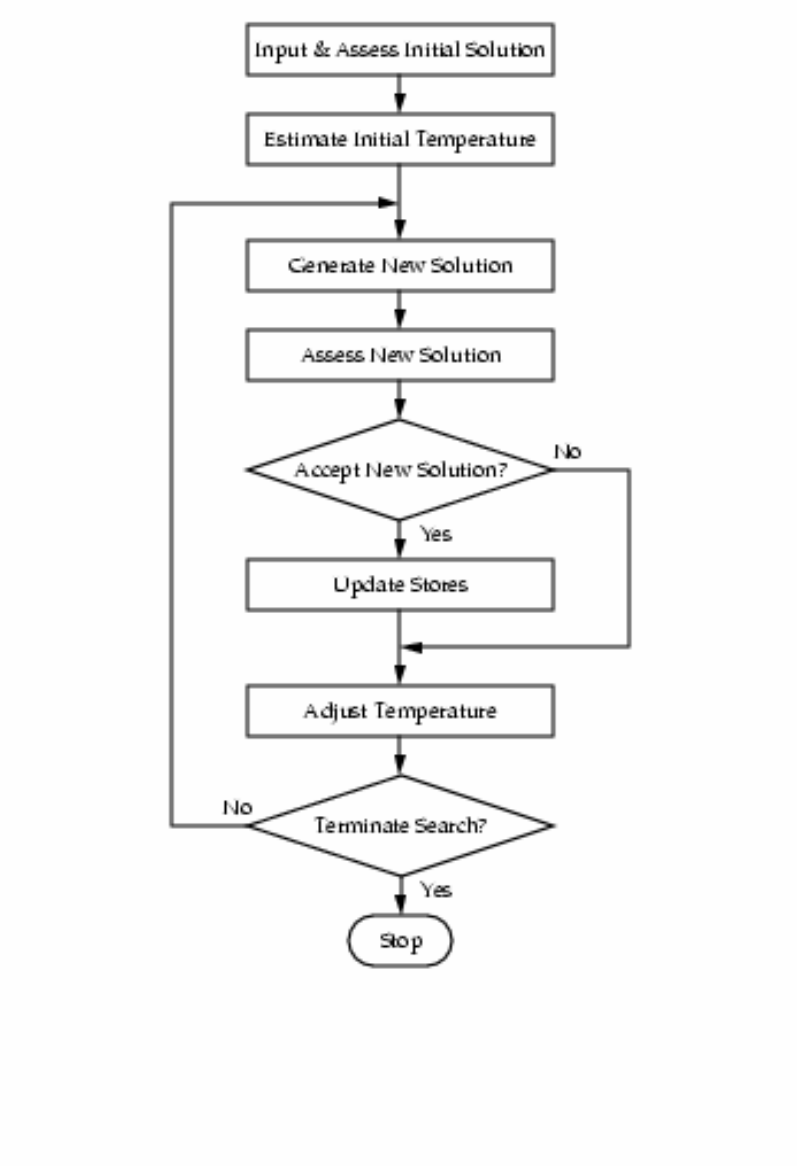
Αλγόριθμος

1. Create an initial random population
2. Evaluate initial population
2. Repeat until convergence (or a number of generations)
   2a. Select the fittest individuals in the population
   2b. Perform crossover on the selected individuals to create offspring
   2c. Perform mutation on the selected individuals
   2d. Create the new population from the old population and the offspring
   2e. Evaluate the new population



ΣΥΝΟΨΗ ΤΩΝ ΣΤΡΑΤΗΓΙΚΩΝ ΑΝΑΖΗΤΗΣΗΣ

	Ακρίβεια	Πολυπλοκότητα	Πλεονεκτήματα	Μειονεκτήματα
Εξαντλητικές	Βρίσκουν την καλύτερη λύση	Εκθετική	Υψηλή ακρίβεια	Υψηλή πολυπλοκότητα
Σειριακές	Καλή αν η μονοτονία είναι εξασφαλισμένη	Τουλάχιστον $O(N^2)$	Απλές και γρήγορες	Αδυνατούν να οπισθοδρομήσουν
Στοχαστικές	Βρίσκουν τη βέλτιστη λύση με σωστές παραμέτρους	Γενικά χαμηλή	Σχεδιασμένες να ξεφεύγουν από τοπικά ελάχιστα	Δυσκολία στην επιλογή καλών παραμέτρων



Η δομή του αλγορίθμου Simulated Annealing

Αναφορές

Justin Doak “An evaluation of feature selection methods and their application to Computer Security” University of California at Davis, Tech Report CSE-92-18

Ricardo Gutierrez-Osuna “Selected Topics in Computer Science” Winter 2002 http://courses.cs.tamu.edu/rgutier/cs790_w02

Arts, E.H.L. and Korst J.H.M. “Simulated Annealing and Boltzmann Machines”, Wiley (Interscience), New York, 1989.

Booker, L.B., Improving Search in Genetic Algorithms, pp 61-73 Genetic Algorithms and Simulated Annealing, (L.Davis, editor) London 1987.

Kirkpatrick, S.,Gerlatt, C.D.Jr., and Vecchi, M. P.,Optimization by Simulated Annealing, IBM Research Report RC 9355,1982.

Laarhoven, P.J.M. van, and Aarts, E. H. L., Simulated Annealing: Theory and Applications, Reidel, Dordrecht, Holland 1987.

Lin, S., Computer Solutions of the Traveling Salesman Problem, Bell Syst. Tech J.44, 2245-2269,1965.

Randelman, R. E., and Grest, G.S., N-City Traveling Salesman Problem – Optimization by Simulated Annelaning, J.Stat. Phys. 45, 885-890,1986

4 ΜΗ ΠΟΛΥΩΝΥΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ

4.1 Εισαγωγή

Έχοντας αναφέρει σε προηγούμενο κεφάλαιο την μέθοδο της εξομοιωμένης ανόπτησης (Simulated Annealing) παρουσιάζεται η έννοια του πλήρους μη-πολυωνυμικού προβλήματος. Ένα πρόβλημα L είναι NP-πλήρες όταν μπορεί να επιλυθεί σε πολυωνυμικό χρόνο αν και μόνο αν όλα τα άλλα NP-πλήρη προβλήματα μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. NP-πλήρη προβλήματα έχουν δημιουργηθεί από όλες τις περιοχές της Πληροφορικής. Μάλιστα στο βιβλίο του Garey-Johnson “Computers and Intractability”: A guide to the theory of NP-Completeness, συγκεντρώθηκαν όλα τα NP-πλήρη προβλήματα που ήταν γνωστά μέχρι το 1979. Τα προβλήματα αυτά ομαδοποιήθηκαν σε 12 κατηγορίες που παρουσιάζονται παρακάτω:

- Θεωρία Γράφων
- Σχεδιασμός δικτύων
- Σύνολα και διαμερισμοί
- Αποθήκευση και ανάκτηση
- Σειριοποίηση και δρομολόγηση
- Μαθηματικός προγραμματισμός
- Άλγεβρα και Θεωρία αριθμών
- Παιχνίδια και Σπαζοκεφαλιές
- Λογική
- Αυτόματα και Γλώσσες
- Βελτιστοποίηση Προγραμμάτων
- Διάφορα

Ένα από τα προβλήματα που παρουσιάστηκαν και τότε ήταν το πρόβλημα του περιοδεύοντος πωλητή.

Διατύπωση Προβλήματος

Δοθέντων ζυγισμένου γράφου G(V,E) και θετικού ακεραίου k, υπάρχει ένας κύκλος που να περνάει από όλες τις κορυφές με άθροισμα των βαρών των ακμών $\leq k$;

4.2 Το πρόβλημα του περιοδεύοντος πωλητή

Σύμφωνα με το πρόβλημα του περιοδεύοντος πωλητή (traveling salesman problem), ένας πωλητής κατά τη διάρκεια της περιοδείας του πρέπει να επισκεφθεί κάποιες πόλεις (N στον αριθμό), που απέχουν μεταξύ τους κάποιες δεδομένες αποστάσεις. Το ερώτημα, λοιπόν, είναι: *"με ποια σειρά πρέπει να επισκεφθεί τις πόλεις αυτές και να επιστρέφει στη δική τον πόλη διανύοντας τη μικρότερη δυνατή συνολική απόσταση·"* Βέβαια, κατά ανάλογο τρόπο μπορούμε να θεωρήσουμε τους απαιτούμενους χρόνους ή το αντίστοιχο οικονομικό κόστος για την κάλυψη των αποστάσεων αυτών. Το πρόβλημα αυτό ανήκει στην κατηγορία των NP-πλήρων προβλημάτων, των οποίων ο χρόνος υπολογισμού για μια ακριβή λύση αυξάνεται με N ως εκθέτη και γίνεται απαγορευτικός όσο το N αυξάνεται.

Προφανώς το πρόβλημα μοντελοποιείται με τη βοήθεια ενός ζυγισμένου απλού γράφου και μπορεί να εξετασθεί θεωρώντας δύο διαφορετικές υποθέσεις: πρώτον, ότι ο πωλητής πρέπει να περάσει μόνο μία φορά από κάθε πόλη, και, δεύτερον, ότι μπορεί να επισκεφθεί κάποια πόλη περισσότερο από μία φορά. Αν ο γράφος είναι Ευκλείδειος (Euclidean), δηλαδή αν ισχύει η ανισοϊσότητα του τριγώνου $dist(u,v) + dist(u,z) \geq dist(u,z)$ για οποιαδήποτε τριάδα ακμών που αποτελούν ένα τρίγωνο, τότε δεν είναι αποτελεσματικό να επισκεφθεί ο πωλητής περισσότερο από μία φορά την ίδια πόλη. Έτσι, στην περίπτωση αυτή το πρόβλημα ταυτίζεται με την εύρεση του Hamiltonian κύκλου με το ελάχιστο βάρος. Αν ο γράφος δεν είναι Ευκλείδειος, τότε η δεύτερη υπόθεση είναι λιγότερο περιοριστική από την πρώτη και μπορεί να οδηγήσει σε καλύτερα αποτελέσματα.

Ας υποθέσουμε ότι χειριζόμαστε Ευκλείδειους γράφους. Ο συνολικός αριθμός Hamiltonian κύκλων σε ένα πλήρη γράφο ισούται με (η-1)!/2. Αυτό προκύπτει εύκολα αν θεωρηθεί ότι από την αρχική πόλη ο πωλητής μπορεί να μεταβεί σε η-1 πόλεις, από εκεί σε η-2, σε n-3, ..., σε 2 πόλεις και τέλος να επιστρέψει στην πόλη του. Αυτές οι επιλογές είναι ανεξάρτητες και έτσι προκύπτει ο όρος (n-1)!. Το αποτέλεσμα αυτό διαιρείται δια 2, γιατί κάθε μονοπάτι προσμετράται δύο φορές. Το πρόβλημα, λοιπόν, θα μπορούσε να επιλυθεί θεωρώντας όλους τους (n-1)!/2 Hamiltonian κύκλους και επιλέγοντας αυτόν με το ελάχιστο βάρος. Βέβαια, η λύση αυτή είναι μόνο θεωρητική επειδή έχει πολυπλοκότητα $O(n!) = O(n^n)$. Έτσι, ακόμη και για μικρές τιμές του ***n*** προκύπτει μία υπολογιστική έκρηξη. Το πρόβλημα του περιοδεύοντος πωλητή, ως περίπτωση του προβλήματος της εύρεσης Hamiltonian κύκλου, ανήκει στην κλάση των δυσχείριστων προβλημάτων.

5 ΕΠΕΞΗΓΗΣΗ SIMULATED ANNEALING

Από τις εφαρμογές του Simulated Annealing είναι αυτή της εύρεσης ενός συντομότερου μονοπατιού ανάμεσα σε πολλούς συνδυασμούς. Η μέθοδος της εξομοιωμένης ανόπτησης ανήκει στους αλγόριθμους που χρησιμοποιούν την έννοια του « τυχαίου » για την αποφυγή τοπικών ελαχίστων. Η θερμοκρασία, όπως εξηγείται και σε προηγούμενο κεφάλαιο, παίζει καθοριστικό ρόλο. Στο βιβλίο “Numerical Recipes” υπάρχει ένα αντιπροσωπευτικό παράδειγμα. Χρειάστηκαν μερικές τροποποιήσεις στον αρχικό κώδικα για να τρέξει. Παρακάτω δείχνουμε αυτές που τελικά χρησιμοποιήθηκαν. Τα ονόματα των συναρτήσεων έμειναν ίδια. Πριν όμως εξηγηθεί η βασική συνάρτηση anneal() θα παρουσιαστούν πρώτα οι συναρτήσεις που χρησιμοποιούνται πριν από αυτήν. Παρόλα αυτά θα δοθεί μια αρχική ερμηνεία στην βασική συνάρτηση έτσι ώστε να είναι πιο κατανοητές οι βοηθητικές συναρτήσεις.

5.1 Αρχική ερμηνεία του τρόπου λειτουργίας της συνάρτησης anneal()

Η συνάρτηση anneal() λειτουργεί ως εξής. Στην αρχή λειτουργίας της η συνάρτηση ορίζει μία αρχική τυχαία διαδρομή μεταξύ όλων των σημείων σε έναν πίνακα iorder[]. Υπολογίζει το αρχικό μήκος του μονοπατιού κάνοντας χρήση της alen(). Έπειτα ενώνει την αρχή με το τέλος του μονοπατιού κλείνοντας με αυτόν τον τρόπο έναν κύκλο. Υπολογίζει και την απόσταση τέλους και αρχής και την προσθέτει μετά στο συνολικό μήκος. Στην συνέχεια επιλέγονται δύο σημεία του πίνακα μονοπατιού iorder[] n[1] και n[2]. Το πρώτο αντιστοιχεί στην αρχή του μονοπατιού και το δεύτερο στο τέλος. Έτσι λοιπόν έχει επιλεχθεί ένα κομμάτι του μονοπατιού. Φυσικά η επιλογή ελέγχεται από μία συνθήκη. Σύμφωνα με αυτή την συνθήκη το κομμάτι που επιλέγεται θα πρέπει να ελέγχεται έτσι ώστε τα σημεία που μένουν έξω από το κομμάτι να μην είναι λιγότερα από τρία. Με αυτό τον τρόπο αποφεύγεται το ενδεχόμενο να επιλεχθεί όλο το μονοπάτι διότι και με την αναστροφή μονοπατιού δεν πρόκειται να αλλάξει τίποτα και το καινούριο μήκος θα είναι το ίδιο με το αρχικό. Αφού επιλεχθούν τα σημεία αρχής και τέλους ο Simulated Annealing προχωρά στο επόμενο βήμα. Σε μία μεταβλητή de εκχωρείται το κόστος της αναστροφής μονοπατιού. Την λειτουργία αυτή καλείται να την φέρει σε πέρας η συνάρτηση revcst(). Η συνάρτηση αυτή δέχεται σαν ορίσματα τις συντεταγμένες, τον πίνακα που περιέχει το μονοπάτι iorder[],τον συνολικό αριθμό των σημείων και τον πίνακα καταστάσεων n[]. Ο πίνακας αυτός είναι εφτά θέσεων από τις οποίες μόνο οι τέσσερις χρησιμοποιούνται. Οι n[1] και n[2] είναι γνωστές. Η revcst βρίσκει το σημείο πριν από την αρχή n[1] , το n[3] και το σημείο μετά το τέλος n[2], το n[4]. Έχοντας αυτά τα σημεία υπολογίζει τους

συνδυασμούς των αποστάσεων του n[1]-n[3], του n[2]-n[4], του n[1]-n[4] και του n[2]-n[3] και επιστρέφει το κόστος της αντιστροφής. Στην συνέχεια και έχοντας την μεταβλητή de εκχωρείται σε μία άλλη μεταβλητή ans η απόφαση για το αν θα γίνει η αντιστροφή. Αυτή την απόφαση καλείται να πάρει η συνάρτηση metrop() η οποία θα επιτρέψει την αλλαγή εάν το de είναι αρνητικό ή εάν ισχύει μία μαθηματική πράξη (ran3(&gljdum) < exp(-de/t)). Σε αυτό το σημείο φαίνεται η δράση της θερμοκρασίας που θα εξηγηθεί παρακάτω. Σε περίπτωση που η μεταβλητή ans είναι 1 τότε θα αυξηθεί μία μεταβλητή που κρατά των αριθμό των επιτυχών βημάτων, θα προσαυξηθεί στο αρχικό μήκος μονοπατιού το κόστος de και έπειτα θα καλεστεί η συνάρτηση που κάνει την αναστροφή. Αλλιώς θα επιλεγεί πάλι καινούριο κομμάτι. Φυσικά και σε αυτό το κομμάτι υπάρχουν συνθήκες τερματισμού και για τις επιτυχείς αλλαγές μονοπατιού αλλά και για τον τερματισμό του αλγορίθμου που θα εξηγηθούν στο αντίστοιχο κεφάλαιο.

5.2 Η Συνάρτηση alen()

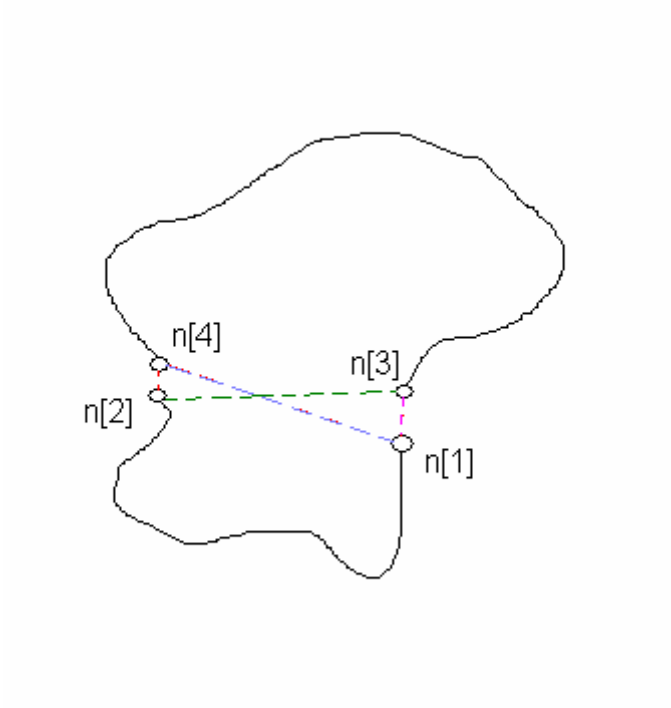
```
1. float alen(float a, float b, float c, float d)
2. {
3. return sqrt(((b)-(a))*((b)-(a))+((d)-(c))*((d)-(c)));
4. }
```

Η συνάρτηση αυτή ουσιαστικά επιστρέφει την απόσταση μεταξύ δύο σημείων με συντεταγμένες χ_1(a,c) και χ_2(b,d). Είναι η γνώριμη ευκλείδεια απόσταση. Η συνάρτηση δέχεται σαν ορίσματα τέσσερις μεταβλητές τύπου float (Βλ 1). Σαν έξοδο η συνάρτηση alen() επιστρέφει την απόσταση των σημείων χρησιμοποιώντας τον τύπο της ευκλείδειας απόστασης $D = \sqrt{(b-a)^2 + (c-d)^2}$ (Βλ 3).

5.3 Η Συνάρτηση revcst()

```
1. float revcst(float x[], float y[], int iorder[], int ncity, int n[])
2. {
3.       float xx[5],yy[5],de;
4.       int j,ii;
5.       n[3]=((n[1] + ncity - 1) % ncity);
6.       n[4]= (n[2] + ncity + 1) % ncity;
7.
8.       for (j=1;j<=4;j++) {
9.                ii=iorder[n[j]];
10.               xx[j]=x[ii];
11.               yy[j]=y[ii];
12.      }
13.      de = -alen(xx[1],xx[3],yy[1],yy[3]);
14.      de -= alen(xx[2],xx[4],yy[2],yy[4]);
15.      de += alen(xx[1],xx[4],yy[1],yy[4]);
16.      de += alen(xx[2],xx[3],yy[2],yy[3]);
17.      return de;
18. }
```

Αυτή η συνάρτηση παίρνει σαν ορίσματα τις συντεταγμένες χ,y τον πίνακα με το μονοπάτι iorder[], τον αριθμό των σημείων και τον πίνακα καταστάσεων n[]. Χρησιμοποιεί τα ορίσματα αυτά για να βρει την τιμή του κόστους για την αντιστροφή. Στην αρχή δηλώνονται δύο πίνακες xx[5] και yy[5] όπου και θα περιέχουν τις συντεταγμένες των σημείων που έχουν επιλεχθεί. Ακόμα δηλώνεται και η μεταβλητή de στην οποία θα εκχωρηθεί το συνολικό κόστος. Στην θέση 3 του πίνακα n[] εκχωρείται η θέση του μονοπατιού πριν από την αρχή του επιλεγμένου κομματιού n[1] (Βλ 5) και στην θέση 4 του n[] η θέση του μονοπατιού μετά το τέλος του επιλεγμένου κομματιού n[2] (Βλ 6). Χρησιμοποιώντας μια επαναληπτική διαδικασία εκχωρούνται στους πίνακες xx[j] και yy[j] οι συντεταγμένες των θέσεων n[1],n[2],n[3] και n[4] από τον iorder[] (Βλ 8-12). Αφού υπολογιστούν τα παραπάνω αρχίζει ο υπολογισμός του κόστους. Παρακάτω ακολουθεί ένα σχεδιάγραμμα για καλύτερη κατανόηση.

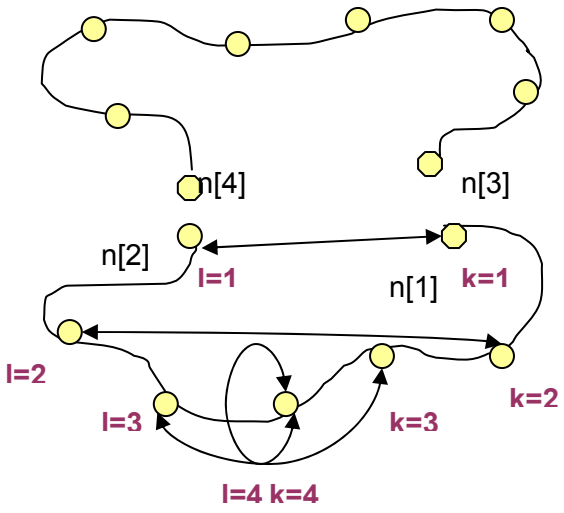


Στην αρχή σαν αρνητικό εκχωρείται η απόσταση του n[1]-n[3]. Έπειτα αφαιρείται και η απόσταση μεταξύ n[2]-n[4]. Κατόπιν προσθέτονται οι αποστάσεις n[1]-n[4] και n[2]-n[3] (Βλ 13-16) . Τέλος επιστρέφεται το τελικό de(Βλ 18) .

5.4 Η συνάρτηση reverse()

```
1. void reverse(int iorder[], int ncity, int n[])
2. {
3.       int nn,j,k,l,itmp;
4.
5.       nn=(1+((n[2]-n[1]+ncity) % ncity))/2;
6.       for (j=1;j<=nn;j++) {
7.
8.              k=((n[1]+j+ncity-1) % ncity);
9.              l=((n[2]-j+ncity+1) % ncity);
10.
11.             itmp=iorder[k];
12.             iorder[k]=iorder[l];
13.             iorder[l]=itmp;
14.
15.      }
16.  }
```

Η συνάρτηση αυτή εκτελεί την λειτουργία της αντιστροφής. Δέχεται σαν ορίσματα τον πίνακα iorder[] που περιέχει τον μονοπάτι, τον αριθμό των σημείων και τον πίνακα καταστάσεων n[]. Στην αρχή σε μία μεταβλητή nn εκχωρείται ο αριθμός των αλλαγών που πρέπει να γίνουν έτσι ώστε να έχει αποτέλεσμα η αναστροφή (Βλ 5). Έτσι έχοντας βρει το nn προχωράμε σε μία επαναληπτική διαδικασία με συνθήκη τερματισμού το nn όπου γίνεται η αναστροφή (Βλ 6). Αρχίζοντας λοιπόν, εκχωρούμε σε μία μεταβλητή k το επόμενο σημείο μετά το n[1] και σε μία μεταβλητή l το προηγούμενο από το n[2] (Βλ 8-9). Παρακάτω ακολουθεί η κλασσική διαδικασία της αλλαγής (swapping) (Βλ 11-13). Ουσιαστικά αρχίζουμε από τα τέλη του κομματιού και κάνουμε αλλαγές πλησιάζοντας προς το κέντρο. Παρακάτω ακολουθεί ένα σχεδιάγραμμα για καλύτερη κατανόηση.



5.5 Η συνάρτηση metrop()

Πρόκειται για τον αλγόριθμο metropolis. Αποτελεί τον “oracle” ή αλλιώς τον σύμβουλο μας. Σαν ορίσματα δέχεται την θερμοκρασία και το κόστος de που έχει υπολογιστεί από την συνάρτηση revcst().

```
1. int metrop(float de, float t)
2. {
3.        float ran3(long *idum);
4.        static long gljdum=1;
5.
6.        return de < 0.0 || ran3(&gljdum) < exp(-de/t);
7. }
```

Στην γραμμή 3 του κώδικα παρατηρούμε πως δηλώνεται μια συνάρτηση με όνομα ran3 και όρισμα idum. Αυτή η συνάρτηση θα παρουσιαστεί παρακάτω. Για χάρη ευκολίας θα αναφέρουμε πως πρόκειται για μα συνάρτηση που είναι γεννήτρια τυχαίων αριθμών από το 0 μέχρι το 1. Παρακάτω δηλώνεται η static long gljdum=1. Πρόκειται για μεταβλητή η οποία αρχικοποιεί την γεννήτρια αριθμών, είναι όπως το λέμε «seed».

Το κυριότερο σημείο είναι αυτό που βρίσκεται στην γραμμή 6. Η συνάρτηση metrop μας επιστρέφει 1 εάν το κόστος de είναι μικρότερο του μηδενός. Να παρατηρήσουμε εδώ πως όταν το κόστος de είναι αρνητικό σημαίνει πως συμφέρει να γίνει η αλλαγή καθώς μειώνεται το μήκος του μονοπατιού. Εάν όμως είναι θετικό τότε υπάρχει πιθανότητα να γίνει η αλλαγή μόνο εάν η τιμή της γεννήτριας τυχαίων αριθμών είναι μικρότερη από την έκφραση exp(-de/t).

Παρατηρούμε εδώ πως σε περίπτωση που η θερμοκρασία στην αρχή είναι πολύ μεγάλη τότε ο metrop επιτρέπει την αλλαγή παρόλο που το κόστος de μπορεί να είναι θετικό! Με ροή όμως του αλγορίθμου θα παρατηρήσουμε στην συνέχεια πως η θερμοκρασία μειώνεται, μειώνοντας έτσι και την πιθανότητα μετάβασης σε υψηλότερη ενεργειακή κατάσταση. Με αυτόν τον τρόπο αποφεύγεται ο εγκλωβισμός σε τοπικά ελάχιστα. Σε περίπτωση που μας επιστραφεί 0 αυτό θα σημαίνει πως reverse δεν θα ακολουθήσει και έτσι θα προχωρήσουμε σε επιλογή καινούριων σημείων.

5.6 Η Συνάρτηση ran3()

Όπως αναφέρεται παραπάνω η συνάρτηση είναι μια γεννήτρια τυχαίων αριθμών από το 0 μέχρι το 1.

```
1. float ran3(long *idum)
2.  {
3.      static int inext,inextp;
4.      static long ma[56];
5.      static int iff=0;
6.      long mj,mk;
7.      int i,ii,k;
8.
9.      if (*idum < 0 || iff == 0) {
10.             iff=1;
11.             mj=MSEED-(*idum < 0 ? -*idum : *idum);
12.             mj %= MBIG;
13.             ma[55]=mj;
14.             mk=1;
15.             for (i=1;i<=54;i++) {
16.                     ii=(21*i) % 55;
17.                     ma[ii]=mk;
18.                     mk=mj-mk;
19.                     if (mk < MZ) mk += MBIG;
20.                     mj=ma[ii];
21.             }
22.             for (k=1;k<=4;k++)
23.                     for (i=1;i<=55;i++) {
24.                             ma[i] -= ma[1+(i+30) % 55];
25.                             if (ma[i] < MZ) ma[i] += MBIG;
26.                     }
27.             inext=0;
28.             inextp=31;
29.             *idum=1;
30.     }
31.     if (++inext == 56) inext=1;
```

```
32.     if (++inextp == 56) inextp=1;
33.     mj=ma[inext]-ma[inextp];
34.     if (mj < MZ) mj += MBIG;
35.     ma[inext]=mj;
36.     return mj*FAC;
37.  }
```

Η παραπάνω γεννήτρια είναι κομμάτι από το βιβλίο “Numerical Recipes”.

5.7 Η συνάρτηση anneal()

Η συνάρτηση anneal() είναι αυτή που τελικά συνοψίζει όλες τις παραπάνω συναρτήσεις για την επίλυση του προβλήματος του συντομότερου μονοπατιού.

```
1 void anneal(float x[], float y[], int iorder[], int ncity)
2 {
3       FILE *finalpath;
4
5       int metrop(float de, float t);
6       float ran3(long *idum);
7       float revcst(float x[], float y[], int iorder[], int ncity, int n[]);
8       void reverse(int iorder[], int ncity, int n[]);
9       int ans,nover,nlimit,i1,i2;
10      int i,j,k,nsucc,nn,idec;
11      static int n[7];
12      long idum;
13      unsigned long iseed;
14      float path,de,t;
15
16      nover=100*ncity;
17      nlimit=10*ncity;
18      path=0.0;
19      t=70.0;
20  /************** calculate initial path cost *************************/
21      for (i=0;i<ncity;i++) {
22              i1=iorder[i];
23              i2=iorder[i+1];
24              path += alen(x[i1],x[i2],y[i1],y[i2]);
```

```
25      }
26
27      i1=iorder[ncity-1];
28      i2=iorder[0];
29      path += alen(x[i1],x[i2],y[i1],y[i2]);
30
31      printf("initial cost of path:%f\n", path);
32
33      finalpath=fopen("finalpath.dat", "w");
34  /************** start the algorithm ***************************/
35      for (j=1;j<=100;j++)
36      {
37              nsucc=0;
38              for (k=1;k<=nover;k++)
39              {
40
41                      do {
42                              n[1]=(int) (ncity*ran3(&idum));
43                              n[2]=(int) ((ncity-1)*ran3(&idum));
44                              if (n[2] >= n[1]) ++n[2];
45                              nn=(1+(n[1]-n[2]+ncity-1) % ncity);
46                      } while (nn<3);
47                      de=revcst(x,y,iorder,ncity,n);
48              ans=metrop(de,t);
49              if (ans) {
50                      ++nsucc;
51                      path += de;
52                      reverse(iorder,ncity,n);
53                                      }
54
55                      if (nsucc >= nlimit) break;
56              }
57              printf("\n %s %10.6f %s %12.6f \n","T =",t,    "Path Length =",path);
58              printf("Successful Moves: %6d\n",nsucc);
59              for (j=0; j<ncity; j++)
60        fprintf(finalpath, "%d\t", iorder[j] );
61              fprintf(finalpath, "%lf\n",path );
```

```
62
63                  t *= TFACTR;
64        if (nsucc == 0)  { fclose(finalpath); return;}
65        }
66          fclose(finalpath);
67  }
```

Στο κομμάτι αυτό θα εξηγήσουμε λεπτομερώς την λειτουργία της «ανόπτησης» καθώς ο παραπάνω κώδικας συγκεντρώνει όλες τις προαναφερθείσες ενέργειες. Η συνάρτηση anneal δέχεται σαν ορίσματα τους πίνακες με τις συντεταγμένες χ[] και y[], τον πίνακα με το μονοπάτι iorder[] και τον αριθμό των σημείων. Για να διατηρήσουμε τα αποτελέσματα μας θα χρησιμοποιήσουμε την εγγραφή σε αρχεία. Για αυτόν τον λόγο δηλώνουμε FILE *finalpath στην γραμμή 3. Οι επόμενες μεταβλητές που δηλώνονται nover και nlimit αφορούν στις συνθήκες τερματισμού επαναληπτικών διαδικασιών. Φυσικά υπάρχει και η μεταβλητή path όπου αρχικοποιείται αλλά και η θερμοκρασία.

Για τον καθορισμό της αρχικής τιμής της θερμοκρασίας θα πρέπει να δείξουμε μεγάλη προσοχή. Επιλέγοντας μικρή θερμοκρασία δεν θα επιτρέψουμε στον αλγόριθμο να κάνει πολλές αλλαγές εγκλωβίζοντας τον έτσι πιθανώς σε τοπικά ελάχιστα. Από την άλλη πλευρά αν επιλεχθεί πολλή μεγάλη θερμοκρασία τότε υπάρχει περίπτωση να γίνουν πολλές μεταβάσεις από περιοχές χαμηλής ενεργειακής κατάστασης σε περιοχές υψηλής ενεργειακής κατάστασης αργοπορώντας έτσι την όλη διαδικασία.

Προχωρώντας, ο αλγόριθμος κάνοντας χρήση του πίνακα με το μονοπάτι βρίσκει το αρχικό μήκος του μονοπατιού κάνοντας χρήση της βοηθητικής συνάρτησης alen(), η οποία επιστρέφει την απόσταση. Τέλος ενώνει την αρχή του μονοπατιού με το τέλος και προσθέτει την απόστασή αυτή στο συνολικό μήκος μονοπατιού (Βλ 21-29). Τελειώνοντας από την επαναληπτική διαδικασία εύρεσης μήκους μονοπατιού και σχηματισμού κύκλου με την ένωση αρχής και τέλους ο αλγόριθμος εκτυπώνει στην οθόνη το αρχικό μήκος (31) και αμέσως μετά ανοίγει το αρχείο finalpath για εγγραφή (33).

Αρχίζοντας, συναντάμε μια συνθήκη. Η συνθήκη αυτή ορίζει απλά την επαναληπτική διαδικασία. Στην συγκεκριμένη περίπτωση 100 φορές. Αυτό δεν σημαίνει πως η μέθοδος θα εκτελεστεί μόνο 100 φορές. Παρακάτω θα υπάρξει εμφωλευμένη επαναληπτική διαδικασία. Σε αυτό το σημείο συναντάμε την μεταβλητή nsucc. Η μεταβλητή αυτή κρατά τον αριθμό των πετυχημένων κινήσεων. Μπαίνοντας στην επόμενη επαναληπτική διαδικασία βλέπουμε πως η συνθήκη είναι διαφορετική (k<=nover), όπου nover είναι ο

αριθμός των σημείων επί 100. Αυτή η συνθήκη χρησιμοποιείται για να κρατηθεί η θερμοκρασία σταθερή για nover αλλαγές και να μην μειωθεί αμέσως μετά τις πρώτες αλλαγές. Αυτό μας δίνει την ευκαιρία για περισσότερες μεταβάσεις μεταξύ ενεργειακών καταστάσεων και αποφυγή έτσι τοπικών ελαχίστων.

Μέσα στην εμφωλευμένη επανάληψη επιλέγονται τα σημεία που καθορίζουν το κομμάτι που επιλέγεται από το συνολικό μονοπάτι. Επιλέγονται δηλαδή η αρχή n[1] και το τέλος n[2]. Φυσικά ελέγχονται και αυτά από συνθήκη. Είναι η μεταβλητή nn που ελέγχει πόσα σημεία μένουν εκτός του επιλεγμένου κομματιού. Αυτό γίνεται έτσι ώστε να αποφευχθεί η επιλογή ολόκληρου του μονοπατιού. Σε μία τέτοια περίπτωση δεν θα αλλάξει το κόστος de οπότε και υπάρχει αυτή η συνθήκη (Βλ 41-46).

Όταν επιλεχθούν κατάλληλα n[1] και n[2], αναλαμβάνει δράση η συνάρτηση revcst() για να μας επιστρέψει την τιμή του κόστους de (Βλ 47). Η συνάρτηση αυτή εξηγείται στο κεφάλαιο 5.1.3.

Όταν επιστραφεί η τιμή του κόστους τότε συμβουλευόμαστε τον metropolis, ο οποίος αποφασίζει για το αν θα γίνει η αναστροφή του κομματιού (Βλ 48). Ο metropolis εξηγείται στο κεφάλαιο 5.1.5. Η τιμή που επιστρέφει ο metropolis εκχωρείται στην μεταβλητή ans. Εάν είναι 1 τότε λαμβάνει χώρα η αναστροφή. Δεν συμβαίνει όμως μόνο αυτό. Πρώτα αυξάνεται ο μετρητής nsucc κατά ένα, προστίθεται το κόστος στο συνολικό μήκος μονοπατιού εκτελείται η συνάρτηση που κάνει την αναστροφή (Βλ 49-53).

Στην περίπτωση που η μεταβλητή ans πάρει την τιμή 0 τότε δεν γίνεται τίποτα και επανερχόμαστε στην επιλογή καινούριου κομματιού χωρίς να βγούμε από την πρώτη επανάληψη (Βλ 39).

Για να βγούμε από την δεύτερη επανάληψη είτε θα πρέπει να ξεπεράσουμε τον επιλεγμένο για τις επιτυχημένες αλλαγές (Βλ 55), είτε να φτάσουμε το όριο nover (Βλ 38).

Στην περίπτωση που ο αλγόριθμος βγει από την δεύτερη επανάληψη εκτυπώνει στην οθόνη την θερμοκρασία , το συνολικό μήκος μονοπατιού και τον αριθμό των πετυχημένων αλλαγών (Βλ 57-58). Έπειτα αναλαμβάνει να εγγράψει σε αρχείο το μονοπάτι και το μήκος του μέχρι εκείνη την στιγμή (Βλ 59-61). Στην επόμενη γραμμή συμβαίνει η μείωση της θερμοκρασίας. Αυτό ισοδυναμεί με μείωση πιθανότητας να συμβεί αλλαγή από μικρή σε μεγαλύτερη ενεργειακή κατάσταση (Βλ 63). Στην περίπτωση που φτάσουμε στο nover και δεν έχει γίνει καμία αλλαγή (δηλαδή nsucc=0) τότε ο αλγόριθμος τερματίζει και μαζί με αυτόν και το αρχείο. Αλλιώς τερματίζει σύμφωνα με την συνθήκη της πρώτης επανάληψης.

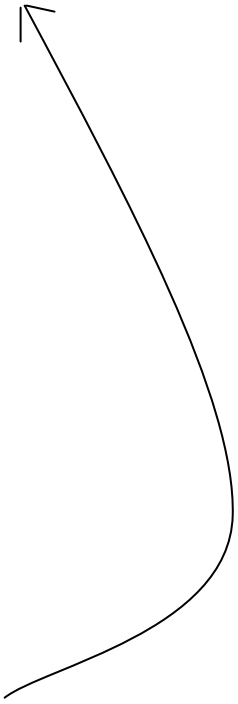
5.8 Παράδειγμα Simulated Annealing με τιμές

Έχουμε τον παρακάτω πίνακα με τις συντεταγμένες, μια στήλη με την αρίθμηση και μία με την αρχική τυχαία διαδρομή.

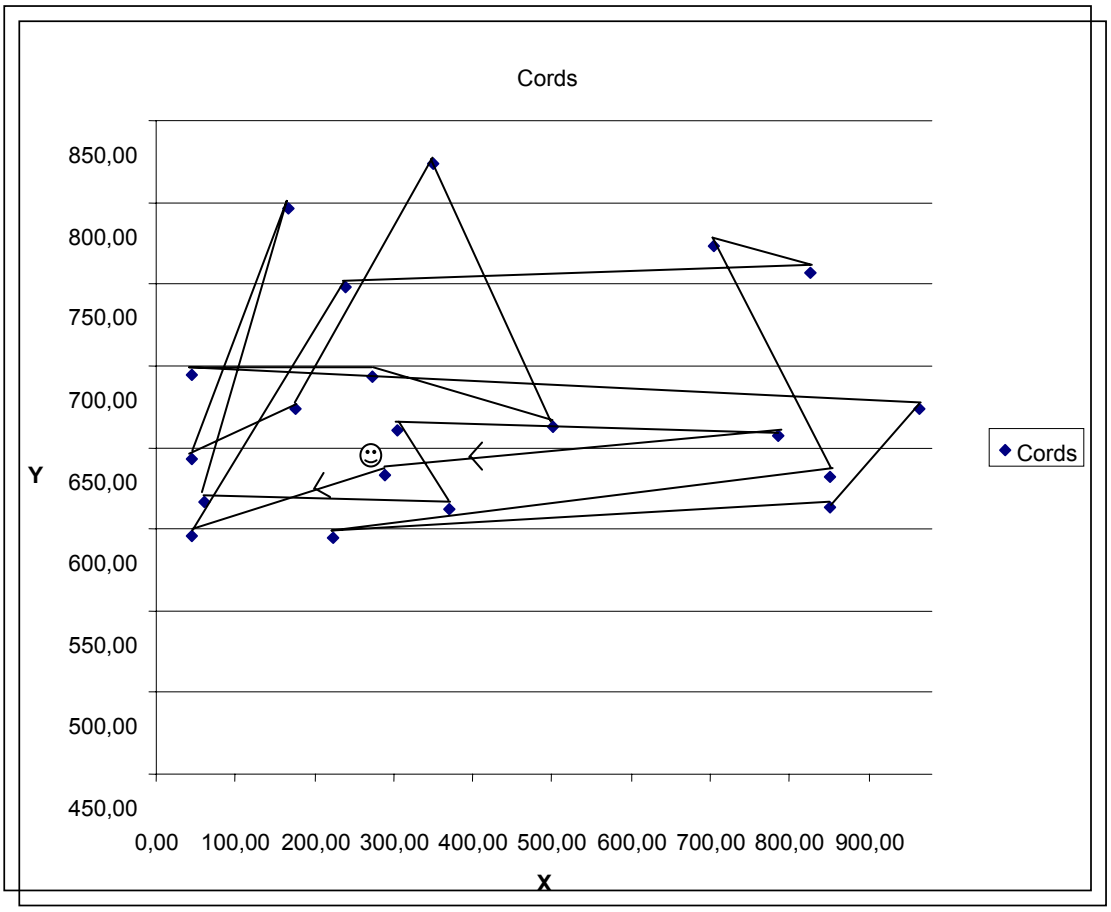
x	y	A/A	iorder
964,25	673,10	0	4
305,67	660,20	1	14
273,15	693,30	2	17
45,54	642,40	3	16
289,41	633,50	4	18
61,80	616,60	5	6
850,45	631,60	6	11
785,41	657,50	7	13
500,77	662,40	8	0
370,70	612,20	9	12
175,61	673,10	10	2
224,38	595,20	11	8
45,54	694,10	12	19
850,45	613,20	13	10
45,54	595,90	14	3
166,82	795,80	15	15
825,40	756,70	16	5
240,11	747,60	17	9
703,60	773,50	18	1
350,00	823,40	19	7

1. Δημιουργείται ένας πίνακας με μια τυχαία αρχική διαδρομή iorder[].
2. Υπολογίζεται το αρχικό μήκος του μονοπατιού.
3. Ενώνουμε το τέλος με την αρχή.

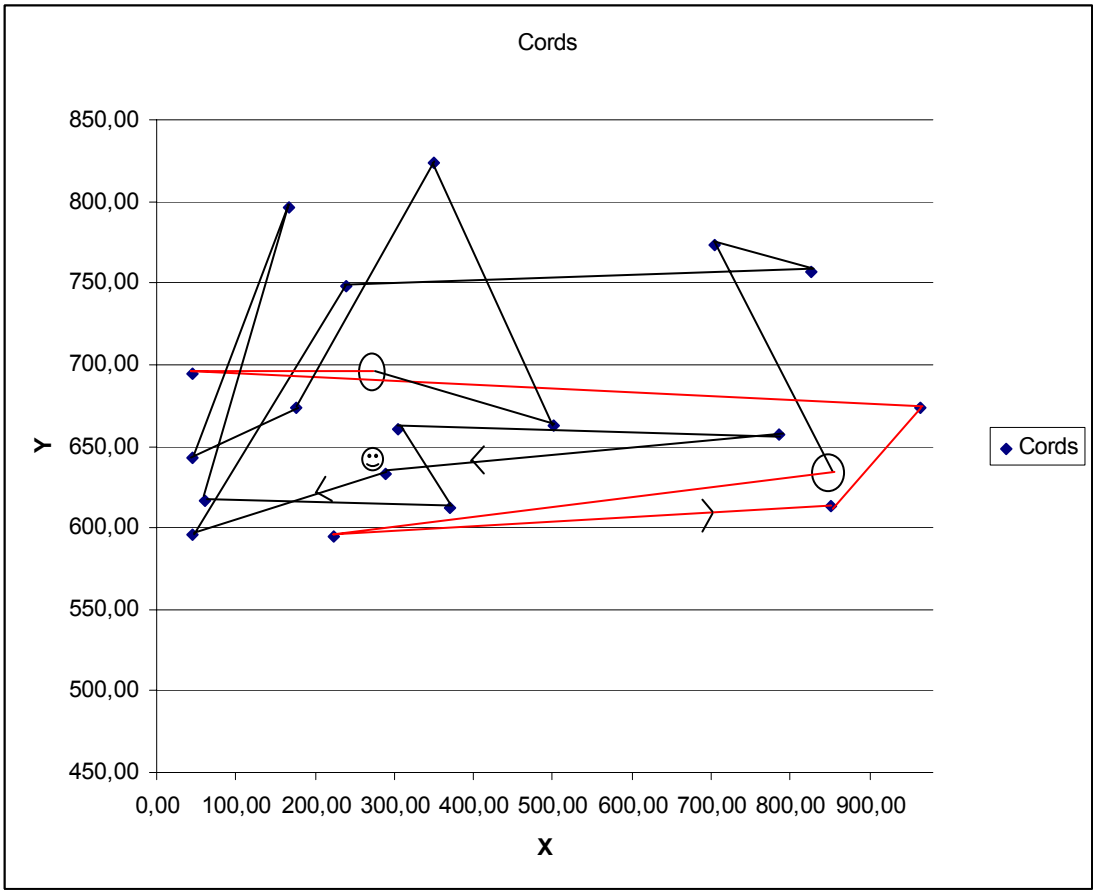
iorder
4
14
17
16
18
6
11
13
0
12
2
8
19
10
3
15
5
9
1
7



Έτσι σχηματίζεται ένας κύκλος με την διαδρομή.



4. path=6021,50021
5. Επιλέγεται ένα σημείο τυχαία από τον πίνακα με το μονοπάτι έστω n[1]=5.
6. Επιλέγεται ένα μονοπάτι από τον πίνακα με το μονοπάτι έστω n[2]=9.
7. Το n[1] είναι η αρχή του κομματιού που επιλέχθηκε και το n[2] το τέλος, επιλέξαμε δηλαδή ένα κομμάτι που περικλείεται μεταξύ του σημείου 5 και 10.
8. Εάν το n[2]> =n[1] τότε αυξάνουμε το n[2] κατά ένα. Άρα n[1]=5 και n[2]=10. Αλλιώς συνεχίζουμε και υπολογίζουμε το nn. Το nn είναι ο αριθμός των σημείων που βρίσκονται έξω από το επιλεγμένο κομμάτι.
9. nn=(1+(n[1]-n[2]+ncity-1) % ncity) όπου ncity ο αριθμός των σημείων, πόλεων στην συγκεκριμένη περίπτωση.
10. Εάν είναι μικρότερο από 3 τότε συνεχίζουμε την αναζήτηση καινούριου κομματιού. Αυτή η συνθήκη χρησιμοποιείται έτσι ώστε να μην επιλεχθεί ολόκληρο το μονοπάτι κάτι που δεν θα είχε επίπτωση στο path. Στην περίπτωση μας το nn είναι 15 οπότε συνεχίζουμε. Έτσι έχουμε το παρακάτω κομμάτι.



11. Καλούμε την συνάρτηση revcst(datax,datay,iorder,ncity,n), η οποία μας επιστέφει το κόστος του να κάνουμε αντιστροφή. Σαν ορίσματα δέχεται τις συντεταγμένες X και Y, τον πίνακα με το μονοπάτι και το n, το οποίο είναι ένας πίνακας 7 θέσεων όπου η πρώτη θέση n[1] αντιπροσωπεύει την αρχή του κομματιού, το n[2] το τέλος, το n[3] το σημείο πριν την αρχή και το n[4] το σημείο μετά το τέλος.

Λειτουργία του revcst

Τα n[3] και n[4] υπολογίζονται ως εξής.

n[3]=(n[1] + ncity - 1) % (ncity);

n[4]=(n[2] + ncity + 1) % (ncity);

Έτσι έχουμε:

n[3]=4

n[4]=11

Μετά σε έναν πίνακα xx[] και έναν πίνακα yy[] εκχωρούνται οι συντεταγμένες των σημείων n[1], n[2], n[3] και n[4]. Έτσι είμαστε έτοιμοι να υπολογίσουμε εκ νέου το κόστος de.

de = - Distance (n[1], n[3]);
de =de - Distance (n[2], n[4]);
de =de + Distance (n[1], n[4]);
de =de + Distance (n[2], n[3]);

Έπειτα επιστρέφουμε το de και τελειώνει διαδικασία του revcst.

Σε μία μεταβλητή ans εκχωρούμε το αποτέλεσμα της συνάρτησης metrop(de,t). Η συνάρτηση αυτή δεν είναι άλλη από τον αλγόριθμο metropolis ο οποίος επιστρέφει 0 ή 1. Στην περίπτωση που επιστρέψει 1 τότε γίνεται η διαδικασία της αντιστροφής. Αλλιώς απλά συνεχίζεται ο κώδικας και πηγαίνουμε ξανά στο βήμα [5]. Ο αλγόριθμος αυτός επιστρέφει 1 σε περίπτωση που το de είναι αρνητικό ή σε περίπτωση που $\mathrm{ran3}(\&\mathrm{gljdum}) < \mathrm{e}^{(-de/t)}$, όπου το αποτέλεσμα της πράξης $\mathrm{ran3}(\&\mathrm{gljdum})$ είναι ένας αριθμός από το 0 μέχρι το 1. Στην περίπτωση μας το de είναι θετικό αλλά η άλλη πράξη είναι αληθής και το ans παίρνει την τιμή 1.

12. Μπαίνουμε στην διαδικασία της αντιστροφής. Εδώ χρησιμοποιούμε μια μεταβλητή nsucc η οποία είναι ουσιαστικά μετρητής αλλαγών. Αυξάνεται η μεταβλητή αυτή

κατά 1 κάθε φορά που θα γίνεται η διαδικασία του reversal. Έτσι και εδώ έχουμε nsucc++, ενημερώνουμε το path προσθέτοντας το καινούριο κόστος. Παρατηρούμε εδώ πως ενώ η αλλαγή αυτή δεν μας συμφέρει καθώς το κόστος είναι θετικό πράγμα που σημαίνει πως θα διανύσουμε παραπάνω δρόμο για να φτάσουμε στον προορισμό μας τελικά γίνεται καθώς ο σύμβουλος μας δηλαδή ο metropolis το επέτρεψε. Στις αρχικές επαναλήψεις η θερμοκρασία είναι πολύ υψηλή και αυτό έχει σαν αποτέλεσμα και uphill διαδρομές. Δηλαδή ξεφεύγουμε έτσι από ένα τυχόν τοπικό ελάχιστο. Αυτή και μια από τις σημαντικές διαφορές του Simulated Annealing με τις υπόλοιπες μεθόδους.

13. Μπαίνουμε στην διαδικασία του path reversal reverse(iorder,ncity,n)

Λειτουργία του Reversal

Η συνάρτηση reverse δέχεται σαν ορίσματα την σειρά του μονοπατιού, τον αριθμό των πόλεων και τον πίνακα με τις καταστάσεις.

Αρχικά στην μεταβλητή nn εκχωρούμε τον αριθμό των πόλεων που πρέπει να αλλαχθούν για να έχει επιτυχία η διαδικασία της αντίστροφης. Εφόσον είναι δια του 2 σημαίνει απλά τον αριθμό των αλλαγών που θα γίνουν.

nn=(1+((n[2]-n[1]+ncity) % ncity))/2;

Άρα nn=3.

Η γενική ιδέα σε αυτό το κομμάτι είναι να αρχίσουμε από τα άκρα του κομματιού και να αντιστρέφουμε τα ζεύγη των πόλεων πλησιάζοντας προς το κέντρο.

Σε μία επανάληψη από το 1 μέχρι **και** το nn εκχωρούμε σε μία μεταβλητή **k** το ένα άκρο του κομματιού και σε μία μεταβλητή **l** το άλλο άκρο.

k =((n[1]+j+ncity-1) % ncity);

l =((n[2]-j+ncity+1) % ncity);

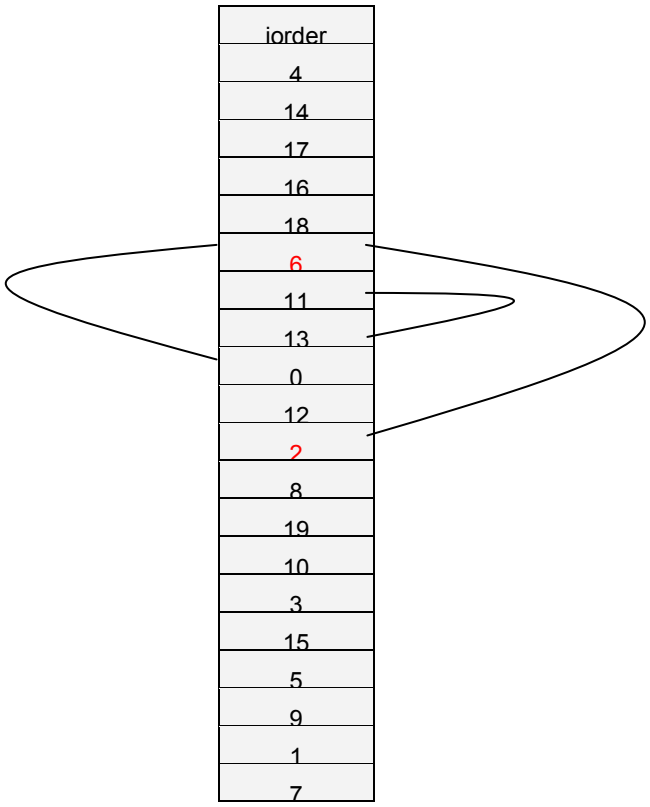
Ακολουθεί η τυπική διαδικασία του reversal.

itmp=iorder[k];

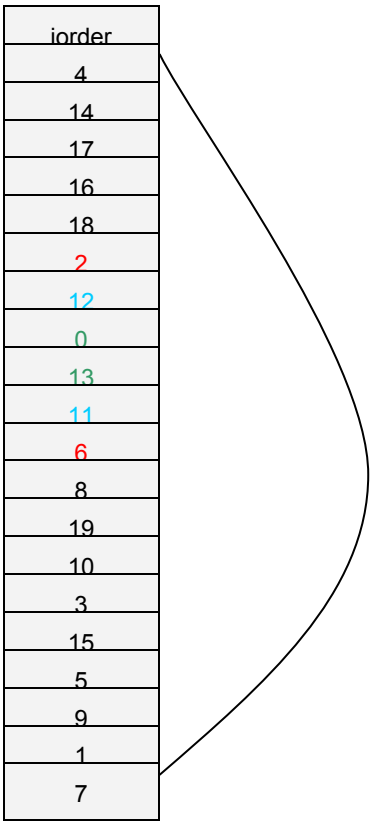
iorder[k]=iorder[l];

iorder[l]=itmp;

14. Τελειώνοντας με την ενέργεια αυτή η κατάσταση έχει ως εξής:

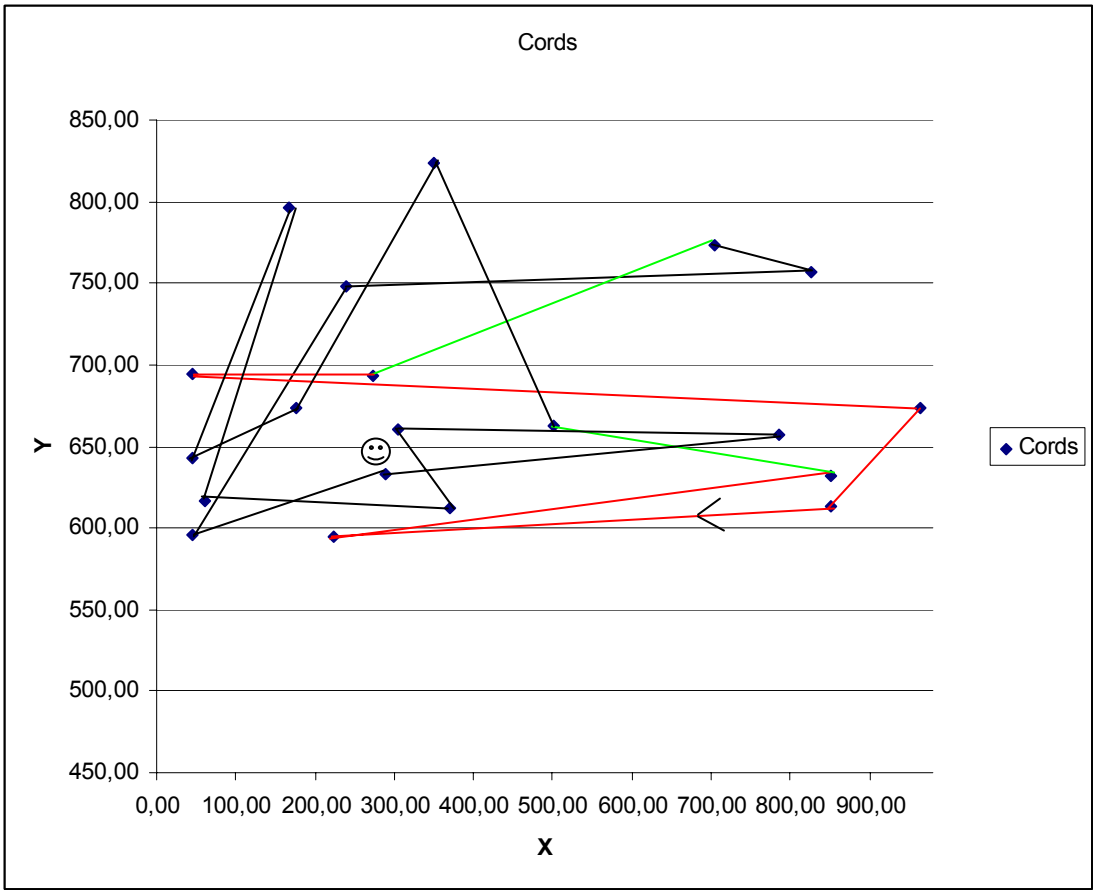


Άρα:



και το path είναι πλέον περίπου 6375!

Το καινούριο path σύμφωνα με τον πίνακα iorder θα είναι:

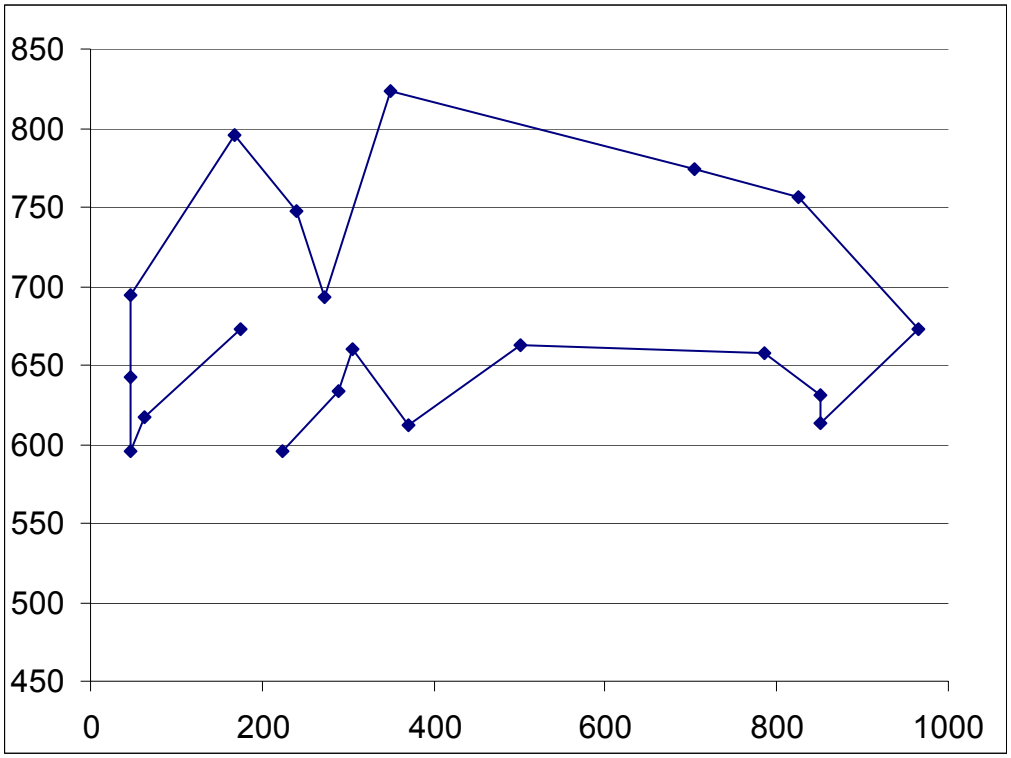


Διαδρομή που επηρεάστηκε από το reversal

Το κομμάτι αντεστραμμένο μπορεί να φαίνεται το ίδιο η φορά όμως έχει αλλάξει

15. Σε περίπτωση που οι αλλαγές ξεπεράσουν έναν αριθμό που σε αυτή την περίπτωση εμείς έχουμε καθορίσει να είναι 10*ncity τότε σταματάμε και μειώνουμε την θερμοκρασία κατά 90%. Αν δεν έχει γίνει καμία αλλαγή και πάλι υπάρχει μία συνθήκη που μας αναγκάζει να σταματήσουμε στις 100*ncity φορές. Αυτό χρησιμεύει πιο πολύ στο τέλος όπου η θερμοκρασία είναι χαμηλή και δεν γίνονται εύκολα οι αλλαγές. Η άλλη συνθήκη αυτή όπου βγαίνουμε από τον κόμβο εάν υπερβούμε έναν αριθμό αλλαγών μας χρησιμεύει όταν είμαστε ακόμα στην αρχή όπου η θερμοκρασία από την μία ή η «συμβολή» του αλγόριθμου metropolis μας επιτρέπει να κάνουμε πολλές αλλαγές.

Το τελικό μονοπάτι που προκύπτει από τον αλγόριθμο Simulated Annealing είναι το παρακάτω.



Το τελικό μονοπάτι που προέκυψε.

Τα όρια των συνθηκών ορίζονται πάντα με βάση κάποια κριτήρια. Ένα από αυτά είναι το μέγεθος του συνόλου δεδομένων που θέλουμε να επεξεργαστούμε. Ένα άλλο θα ήταν η επεξεργαστική ισχύς του συστήματος που θα εφαρμόσουμε το πρόγραμμα. Είναι πάντα στην δική μας επιλογή να ορίσουμε τις συνθήκες αυτές. Οι αλλαγές σε χρόνο που θα προκύψουν μπορεί να είναι αμελητέες σε μικρά σύνολα δεδομένων αλλά θα είναι σίγουρα αισθητές σε μεγαλύτερα! Σκεφθείτε μόνο ένα σύνολο από 100000 δεδομένα. Σκεφτείτε το όριο του 100*ncity δηλαδή 100*100000! Στα τελευταία βήματα του αλγορίθμου όπου οι αλλαγές δεν είναι πολλές θα χρειαστούν δέκα εκατομμύρια επαναλήψεις για να βγούμε από τον κόμβο σε περίπτωση που δεν έχει γίνει καμία αλλαγή!

6 ΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ DBSCAN

Όπως αναφέρεται και σε προηγούμενο κεφάλαιο ο συγκεκριμένος αλγόριθμος συσταδοποίησης χρησιμοποιεί την έννοια της πυκνότητας για την αναγνώριση συστάδων. Συγκεκριμένα, μπορούμε να ξεχωρίσουμε τρία βασικά βήματα, τα οποία ακολουθεί ο αλγόριθμος για να επιτύχει την επιθυμητή συσταδοποίηση. Τα βήματα αυτά θα φανούν καλύτερα όταν εξηγηθεί ο κώδικας. Όπως επιλέχθηκε και παραπάνω θα εξηγηθούν πρώτα οι βοηθητικές συναρτήσεις του DBSCAN.

6.1 Δήλωση Δομών

Στον αλγόριθμο συσταδοποίησης δηλώνονται τρεις δομές, οι οποίες παρουσιάζονται παρακάτω.

```
1  struct InfArray
2        {
3         int cod;
4         int point;
5         double x;
6         double y;
7         vector<int> PointsDDR;
8         AnsiString PointType;
9         int ClusterID;
10        double KNearestDist;
11        };
12
13      struct KnearestInf
14        {
15        int Point;
16        double Dist;
17        };
18
19     struct ClusterArrayInf
20        {
21        vector<int> Clusters;
22        int State;
23         };
```

Η πρώτη δομή είναι η δομή InfArray, η οποία αποτελεί βασικό στοιχείο καθώς εκεί μέσα αποθηκεύονται οι αρχικές πληροφορίες αλλά και τα αποτελέσματα μετά από την επεξεργασία των πληροφοριών αυτών. Στην μεταβλητή cod, η οποία είναι τύπου **int** αποθηκεύεται ο αύξων αριθμός των στοιχείων προς επεξεργασία (Βλ 3). Στην μεταβλητή point, η οποία είναι τύπου **int** αποθηκεύεται η ονομασία του στοιχείου (Βλ 4). Όπως καταλαβαίνουμε από τις δύο παρακάτω μεταβλητές που ανήκουν στην δομή μας πρόκειται για αυτές που αντιπροσωπεύει τα δεδομένα που είναι προς επεξεργασία. Είναι οι χ και y και είναι τύπου **double** (Βλ 5-6). Ακολουθεί μία δήλωση τύπου **vector** με όνομα PointsDDR. Αποτελεί μια δυναμική δομή στην οποία αποθηκεύονται τα σημεία που είναι άμεσα προσεγγίσιμα βάση της πυκνότητας (DirectlyDensityReachable) σε σχέση με ένα άλλο σημείο (Βλ 7). Επειδή το μέγεθος της δομής δεν είναι σε κανένα σημείο γνωστό για αυτόν το λόγο επιλέχθηκε αυτός ο τύπος. Η επόμενη μεταβλητή χρησιμοποιείται για να κρατά τον τύπο του στοιχείου. Αυτό μπορεί να είναι **Core**, **Border** ή **Noise**. Ακολουθεί η μεταβλητή ClusterID που είναι τύπου int όπου αποθηκεύεται η ταυτότητα του στοιχείου που ανήκει σε συστάδα (Βλ 9). Η μεταβλητή KnearestDist είναι τύπου double και κρατά την απόσταση του μικρότερου σημείου από ένα άλλο (Βλ 10).

Η δεύτερη δομή έχει την ονομασία **KnearestInf** όπου και αποθηκεύονται οι πληροφορίες των σημείων που βρίσκονται κοντά σε ένα επιλεγμένο από τον αλγόριθμο σημείο. Συγκεκριμένα, αποθηκεύεται ο κωδικός του σημείου αλλά και το πιο σημαντικό, η απόστασή του (Βλ 13-17).

Η τρίτη και τελευταία δομή περιέχει τις πληροφορίες των συστάδων. Πιο συγκεκριμένα αποθηκεύεται εκεί το μέγεθος συστάδας το οποίο είναι δυναμικό κάθε φορά για αυτό και επιλέχθηκε η μεταβλητή αυτή να είναι τύπου **vector**. Η επόμενη μεταβλητή είναι τύπου **int** και κρατά την κατάσταση (Βλ 19-23).

6.2 Η συνάρτηση SortAuxArray()

Η συνάρτηση αυτή ουσιαστικά ταξινομεί την δομή DataSet.

```
1  void SortAuxArray(int recsize, InfArray *DataSet)
2 {
3
4
5  InfArray* temp=new InfArray[1];
6  for(i=0;i<recsize-1;i++)
7   {
8    for(j=i+1;j<recsize;j++)
9    {
10    if((DataSet[i].KNearestDist) > (DataSet[j].KNearestDist))
11          {
12          //swap places
13         temp[0]=DataSet[i];
14         DataSet[i]=DataSet[j];
15         DataSet[j]=temp[0];
16         }
17   }
18
19
20  }
21  delete [] temp;
22}
```

Η συνάρτηση δέχεται σαν όρισμα τον αριθμό των δεδομένων που βρίσκονται στην βάση και την δομή InfArray (Βλ 1). Στις πρώτες γραμμές δηλώνεται η δομή temp (Βλ 5) και με εμφωλευμένες επαναληπτικές διαδικασίες γίνεται η σύγκριση του ενός στοιχείου με το επόμενο του(Βλ 6-8). Ως μέτρο σύγκρισης χρησιμοποιείται το μέλος της δομής KnearestDist (Βλ 10). Σε περίπτωση που η συνθήκη ταξινόμησης είναι αληθής τότε αναλαμβάνει η διαδικασία της ανταλλαγής (Βλ 13-15). Τέλος, για σωστή διαχείριση διαγράφεται ο πίνακας temp που χρησίμευσε μόνο για την διαδικασία ανταλλαγής.

6.3 Η συνάρτηση GetMax()

```
1   int GetMax(KnearestInf *Knearest, int Knn)
2  {
3
4
5   int i;
6  double maxi=0.0;
7  int pos1=0;
8  maxi=Knearest[0].Dist ;
9  pos1=0;
10
11  for(i=1;i<Knn;i++)
12   {
13   if (Knearest[i].Dist > maxi)
14    {
15    maxi=Knearest[i].Dist;
16    pos1=i;
17    }
18   }
19  return pos1;
20  }
```

Η συνάρτηση δέχεται σαν όρισμα την δομή KnearestInf *Knearest αλλά και την μεταβλητή Knn που είναι το μέγεθός της. Όπως υποδηλώνει και το όνομα της συνάρτησης, η λειτουργία της είναι να επιστρέφει την θέση της μέγιστης απόστασης της δομής που δέχεται σαν όρισμα. Αρχικά μια μεταβλητή maxi αρχικοποιείται με την τιμή μηδέν. Το ίδιο συμβαίνει και με την pos όπου αποθηκεύεται θέση της μεγαλύτερης απόστασης. Αρχίζοντας πάντα (σχεδόν) με μία επαναληπτική διαδικασία από την αρχή μέχρι και την τιμή της μεταβλητής Knn ελέγχουμε απλά εάν η maxi είναι μεγαλύτερη από την τρέχουσα. Εάν ναι, τότε γίνεται αντικατάσταση και έχουμε καινούρια τιμή για την μεταβλητή, αλλιώς συνεχίζουμε το ψάξιμο.

6.4 Η συνάρτηση LoadDataSet()

```
1  void LoadDataset(int recsize, InfArray *DataSet)
2  {
3   int j;
4   Form1->ADOQuery1->Open();
5
6  for(j=0;j<recsize;j++)
7   {
8    DataSet[j].cod=j;
9    DataSet[j].point=Form1->ADOQuery1->Fields->Fields[0]-> Value;
10   DataSet[j].x= Form1->ADOQuery1->Fields->Fields[1]-> Value;;
11   DataSet[j].y= Form1->ADOQuery1->Fields->Fields[2]-> Value;
12   Form1->ADOQuery1->Next();
13   }
14
15
16  Form1->ADOQuery1->Close();
17
18 }
19
```

Οι παραπάνω συναρτήσεις αποτελούν βοηθήματα του DBSCAN. Η συγκεκριμένη, κάνει μια πολύ σημαντική λειτουργία. Επικοινωνεί με την βάση δεδομένων και αντλεί τα στοιχεία που χρειαζόμαστε και τα εκχωρεί στην δομή τύπου InfArray με όνομα DataSet. Παρόλο που είναι μια πολύ σημαντική συνάρτηση είναι πολύ απλή. Έχοντας θέσει ένα ερώτημα SQL στο ADOQuery ,το οποίο αναλαμβάνει να αντλήσει τα επιθυμητά δεδομένα, ενεργοποιούμε το αντικείμενο αυτό και με μία επαναληπτική διαδικασία αντλούμε τα δεδομένα και ταυτόχρονα τα εκχωρούμε στην δομή μας από τα αντίστοιχα πεδία της βάσης. Έπειτα απενεργοποιούμε το αντικείμενο ADOQuery έτσι ώστε να απελευθερωθεί για μετέπειτα χρήση του.

6.5 Η συνάρτηση CalcKnearest()

```
1  void CalcKNearest(int Knn,int recsize, InfArray* DataSet)
2   {
3
4
5   KnearestInf* Knearest=new KnearestInf[Knn];
6
7  int i;
8  int j;
9  int pos;
10  int size;
11  double dist;
12
13  for (i=0;i<recsize;i++)
14    {
15    size=0;
16    for(j=0;j<recsize;j++)
17     {
18      dist=0;
19       if(i!=j)
20       {
21      dist=sqrt(((DataSet[i].x-DataSet[j].x)*(DataSet[i].x-DataSet[j].x))
22        +((DataSet[i].y-DataSet[j].y)*(DataSet[i].y-DataSet[j].y)));
23          if(size<Knn)
24          {
25           Knearest[size].Dist=dist;
26           Knearest[size].Point=j;
27           size++;
28           }
29          else
30           {
31                pos=GetMax(Knearest,Knn);
32              if(dist<Knearest[pos].Dist)
33               {
34               Knearest[pos].Dist=dist;
35               Knearest[pos].Point=j;
```

```
36              }
37            }
38          }
39            }
40      pos=GetMax(Knearest, Knn);
41      dist=sqrt(((DataSet[i].x-DataSet[Knearest[pos].Point].x)*(DataSet[i].x-
42      DataSet[Knearest[pos].Point].x))+((DataSet[i].y-
43      DataSet[Knearest[pos].Point].y)*(DataSet[i].y-
44      DataSet[Knearest[pos].Point].y)));
45      DataSet[i].KNearestDist=dist;
46
47      for(int i=0;i<Knn;i++)
48       {
49       Knearest[i].Point=0;
50       Knearest[i].Dist=0.0;
51       }
52    }
53
54  delete [] Knearest;
55  }
```

Άλλη μια πολύ σημαντική συνάρτηση είναι η CalcKnearest. Πρόκειται για μία συνάρτηση της οποίας η λειτουργία είναι να βρίσκει τους **K** κοντινότερους γείτονες του κάθε σημείου μέσα στο σύνολο μας, που στην προκειμένη περίπτωση είναι η δομή με το όνομα DataSet. Αρχίζοντας, δηλώνεται μία δομή Knearest η οποία έχει μέγεθος **K** το οποίο καθορίζεται από τον χρήστη στην εκκίνηση της διαδικασίας (Βλ 5). Όπως και σε άλλες συναρτήσεις χρησιμοποιείται η έννοια της απόστασης. Χρησιμοποιώντας την ευκλείδεια απόσταση $D = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ σε συνδυασμό με εμφωλευμένες επαναλήψεις βρίσκουμε την απόσταση του ενός σημείου με αυτήν του επόμενου (Βλ 21-22). Εάν το μέγεθος **K** που έχουμε ορίσει δεν έχει ξεπεραστεί (Βλ. 23)τότε καταχωρούμε την απόσταση στην δομή μας και αυξάνουμε τον μετρητή του μεγέθους (Βλ 23-27). Στην περίπτωση που έχει ξεπεραστεί η τιμή της μεταβλητής **Knn** τότε παίρνουμε την θέση του μακρινότερου στοιχείου από την δομή μας Knearest κάνοντας χρήση της συνάρτησης GetMax (Βλ 31). Έχοντας επιστραφεί η τιμή της θέσης τότε αναλαμβάνουμε να συγκρίνουμε την τρέχουσα απόσταση στοιχείου με την μεγαλύτερη που υπάρχει στην δομή μέχρι εκείνη την στιγμή (Βλ 32). Εάν είναι μικρότερη από αυτή που υπάρχει μέσα στην δομή Knearest τότε εκχωρούμε την καινούρια μικρότερη απόσταση στην θέση της δομής που υπήρχε η μεγαλύτερη (Βλ 34-35). Όταν τελειώσουμε με ένα στοιχείο καλούμε

και πάλι την συνάρτηση GetMax έτσι ώστε να βρούμε και πάλι την θέση του στοιχείου έτσι ώστε να υπολογιστεί η απόσταση (Βλ 40-45). Αφού γίνει και αυτό και τελειώσει η επανάληψη για ένα στοιχείο τότε η δομή διαγράφεται για να ξαναχρησιμοποιηθεί στο επόμενο στοιχείο. Η συγκεκριμένη διαδικασία θα πραγματοποιηθεί για όλα τα στοιχεία με όλα τα στοιχεία έτσι ώστε να βρεθούν τα πιο κοντινά σημεία για όλα τα σημεία. Στο τέλος της διαδικασίας διαγράφουμε και την δομή διότι δεν χρησιμοποιείται από πουθενά αλλού.

6.6 Η Συνάρτηση CheckDDRPoints()

```
1  void CheckDDRPoints(double Eps, int MinPts, int recsize, InfArray *DataSet)
2  {
3  int i,k,l;
4   int j;
5   int count;
6   double dist;
7   vector<int> MyCollect;
8  for(i=0;i<recsize;i++)
9     {
10    dist=0;
11    count=0;
12
13             MyCollect.clear() ;
14
15         for(j=0;j<recsize;j++)
16           {
17
18           dist=sqrt(((DataSet[i].x-DataSet[j].x)*(DataSet[i].x-
19         DataSet[j].x))+((DataSet[i].y-DataSet[j].y)*
20               (DataSet[i].y-DataSet[j].y)));
21                if(dist<=Eps)
22                  {
23                   count++;
24                   MyCollect.insert(MyCollect.end(),DataSet[j].point);
25                  }
26
27           }
```

```
28            if(count>= MinPts)
29             {
30             DataSet[i].PointType="Core";
31             DataSet[i].PointsDDR=MyCollect;
32            }
33
34      else
35        {
36         DataSet[i].PointType="Border";
37          }
38       DataSet[i].ClusterID=0;
39     }
40
41
42  }
```

Η συνάρτηση δέχεται σαν ορίσματα την μεταβλητή **Eps** η οποία είναι το μέγεθος της «γειτονιάς», την μεταβλητή **MinPts**, η οποία έχει σαν τιμή τον ελάχιστο αριθμό μέσα σε μια συστάδα, τον συνολικό αριθμό των στοιχείων που βρίσκονται στην δομή και την ίδια την δομή από την οποία και θα αντλούνται τα δεδομένα (Βλ 1). Η λειτουργία της είναι να φτιάχνει συλλογές για κάθε στοιχείο στην δομή μας. Οι συλλογές αυτές περιέχουν άλλα στοιχεία τα οποία βρίσκονται σε ακτίνα μικρότερη από την τιμή **Eps** ως προς το τρέχον σημείο. Αρχίζοντας, δηλώνεται μια δυναμική δομή τύπου **vector** στην οποία θα εκχωρούνται τα σημεία τα οποία θα πληρούν τις προϋποθέσεις (Βλ 7). Και εδώ για να εξεταστούν όλα τα σημεία χρησιμοποιείται διπλή εμφωλευμένη επαναληπτική διαδικασία (Βλ 8,15). Κάθε φορά που θα αλλάζει σημείο για το οποίο θα δημιουργηθεί και μία συλλογή, θα διαγράφονται οι εγγραφές τις προηγούμενης συλλογής (Βλ 13). Αρχίζοντας, επιλέγεται ένα σημείο και μετά επιλέγεται ένα άλλο με την σειρά και βρίσκεται η απόστασή τους (Βλ 18-20). Εάν η απόσταση αυτή είναι μικρότερη από την καθορισμένη από τον χρήστη μεταβλητή **Eps** τότε εισάγεται το σημείο αυτό στην συλλογή και αυξάνεται ένας μετρητής **count** (Βλ 21-24). Όταν εισαχθούν στην συλλογή όλα τα σημεία των οποίων η απόσταση είναι μικρότερη από την δεδομένη **Eps**, τότε ελέγχεται πόσες εισαγωγές έχουν γίνει στην συλλογή (Βλ 28). Εάν αυτές είναι περισσότερες από την μεταβλητή **MinPts** τότε το σημείο για το οποίο έγινε συλλογή χαρακτηρίζεται ως «Core» ή αλλιώς κεντρικό σημείο (Βλ 30), αλλιώς χαρακτηρίζεται ως «Border» (Βλ 36) δηλαδή συνοριακό σημείο και στο μέλος της δομής DataSet **PointsDDR** (Directly Density Reachable Points) εκχωρείται το μέγεθος της συλλογής (Βλ 31). Σε αυτό το σημείο καταλαβαίνουμε πως η συνάρτηση έχει ως λειτουργία να βρίσκει και να καταχωρεί τα σημεία ως Συνοριακά ή ως Πυρήνες αλλά και να βρίσκει τον αριθμό των σημείων που είναι άμεσα συνδεδεμένα με βάση την πυκνότητα. Έπειτα επιλέγεται το επόμενο σημείο και αρχίζει πάλι η αναζήτηση.

6.7 Η συνάρτηση GetDRPoints

```
1   void GetDRPoints( int PointPos, int recsize, InfArray *DataSet)
2   {
3   vector<int> MySet;
4   int i,j,K,m;
5   bool found;
6
7
8     for(i=0;i<DataSet[PointPos].PointsDDR.size();i++)
9       {
10        if(DataSet[PointPos].point!= DataSet[PointPos].PointsDDR[i])
11          {
12            MySet.insert(MySet.end(),DataSet[PointPos].PointsDDR[i]);
13          }
14      }
15
16
17  for(i=0;i<MySet.size();i++)
18    {
19    for(j=0;j<recsize;j++)
20      {
21        if(DataSet[j].point = =MySet[i]  )
22          {
23            for(int K=0;K<DataSet[j].PointsDDR.size();K++)
24                {
25                 found=false;
26                  if(DataSet[j].PointsDDR[K]!=DataSet[PointPos].point)
27                    {
28                      for(m=0;m<MySet.size();m++)
29                          {
30                           if(DataSet[j].PointsDDR[K]= =MySet[m])
31                              {
32                                   found=true;
33                                  break;
34                              }
35                           }
```

```
36                          if(found= =true)
37                          goto a;
38                          else
39                           MySet.insert(MySet.end(),DataSet[j].PointsDDR[K]);
40                        }
41
42  a:              }
43      break;
44            }
45
46     }
47
48
49   }
50    DataSet[PointPos].ClusterID=ClusterID;
51     for(i=0;i<MySet.size();i++)
52       {
53         for(j=0;j<recsize;j++)
54       {
55           if(DataSet[j].point = =MySet[i])
56           {
57             DataSet[j].ClusterID=ClusterID;
58           }
59        }
60      }
61 }
```

Η συνάρτηση δέχεται σαν ορίσματα το επιθυμητό σημείο, το μέγεθος του συνόλου των δεδομένων και την δομή DataSet που τα περιέχει (Βλ 1). Η λειτουργία της συνάρτησης αυτής είναι να βρίσκει τα σημεία που είναι συνδεδεμένα βάση της πυκνότητάς τους και να τα ομαδοποιεί σε συστάδες.

Ο αλγόριθμος αρχίζει με την δήλωση μιας συλλογής τύπου **vector**. Με μια επαναληπτική διαδικασία εισάγονται στην συλλογή MySet τα στοιχεία της συλλογής που έχουν προκύψει από την συνάρτηση CheckDDRPoints για ένα συγκεκριμένο σημείο PointPos που δέχεται η συνάρτηση σαν όρισμα (Βλ 8-14). Μετά ο αλγόριθμος μπαίνει σε μια επαναληπτική διαδικασία με όριο το μέγεθος της συλλογής που έχει γεμίσει πριν λίγο (Βλ 17).

Αμέσως μετά μπαίνει και σε άλλη μια επαναληπτική διαδικασία με όριο το μέγεθος του συνόλου των δεδομένων μας (Βλ 19). Πρόκειται για μια διπλή επαναληπτική διαδικασία. Έτσι συγκρίνονται τα σημεία της συλλογής με όλα τα σημεία του συνόλου. Αρχίζοντας, γίνεται η σύγκριση του στοιχείου της συλλογής του επιλεγμένου στοιχείου **PointPos** με το στοιχείο από ολόκληρο το σύνολο (Βλ 21). Εάν δεν είναι αληθής η σύγκριση τότε επιλέγεται άλλο σημείο από το σύνολο και η σύγκριση ξαναγίνεται. Εάν υπάρχει σημείο ίδιο στην συλλογή τότε ο αλγόριθμος μπαίνει και πάλι σε μια επαναληπτική διαδικασία με όριο το μέγεθος της συλλογής PointsDDR του στοιχείου j (Βλ 23). Εκεί υπάρχει μια μεταβλητή **found** που αρχικοποιείται ως **false** (Βλ 25). Εκεί ο αλγόριθμος

μπαίνει σε ακόμα μια σύγκριση. Συγκρίνεται το K στοιχείο της συλλογής του στοιχείου j από την δομή DataSet με το σημείο point της θέσης PointPos της δομής DataSet (Βλ 26). Εάν είναι διαφορετικά τότε μπαίνουμε σε μια ακόμη for της ποίας όριο **m** είναι το μέγεθος της συλλογής MySet την οποία έχουμε γεμίσει στην αρχή (Βλ 28). Σε αυτή την διαδικασία ελέγχεται εάν το K στοιχείο της συλλογής PointsDDR του στοιχείου j της δομής DataSet είναι ίσο με στοιχείο **m** της συλλογής MySet (Βλ 30). Αν είναι αληθές τότε η μεταβλητή found γίνεται **true** και με την χρήση της εντολής **break** φεύγουμε από την επαναληπτική διαδικασία. Παρακάτω ελέγχουμε την τιμή της μεταβλητής found (Βλ 36). Στην περίπτωση που είναι true τότε κάνοντας χρήση της εντολής goto φεύγουμε από την επανάληψη. Αυτή η διαδικασία θα πραγματοποιηθεί για όλα τα στοιχεία της συλλογής MySet με όλα τα στοιχεία της δομής DataSet, δηλαδή με όλα τα στοιχεία του συνόλου.

Έπειτα, εκχωρούμε την τιμή του ClusterID, που περιέχει την ταυτότητα της συστάδας στην θέση PointPos της δομής DataSet στο μέλος ClusterID (Βλ 50). Να υπενθυμίσουμε πως το PointPos είναι όρισμα που δέχεται η συνάρτηση στην αρχή και βάση αυτής προχωρά δημιουργώντας την συλλογή και κάνοντας αργότερα τους ελέγχους. Τέλος, κάνοντας χρήση και πάλι εμφωλευμένων επαναληπτικών διαδικασιών ελέγχουμε το κάθε στοιχείο της συλλογής MySet με όλα τα στοιχεία του DataSet (Βλ 51-55). Αυτό γίνεται για να εκχωρηθεί η ταυτότητα ClusterID σε κάθε στοιχείο του DataSet που ανήκει στην συλλογή MySet.

Η συνάρτηση θα καλεστεί για όλα τα στοιχεία του συνόλου και για όλα τα στοιχεία θα παραχθούν συλλογές.

6.8 Η Συνάρτηση FindClusters

```
1   void FindClusters(int Point, int recsize,InfArray *DataSet)
2   {
3
4    int i;
5
6          for(i=0;i<recsize;i++)
7           {
8         GetDRPoints(Point,recsize,DataSet);
9             }
10  }
```

Πρόκειται για την συάρτηση που ουσιαστικά το μόνο που κάνει είναι να καλεί την συνάρτηση GetDRPoints.

6.9 Αποθήκευση Αποτελεσμάτων

Για την αποθήκευση των αποτελεσμάτων μπορούν να χρησιμοποιηθούν πολλών ειδών συναρτήσεις. Μπορούμε να κάνουμε εγγραφή σε αρχείο αλλά μπορούμε να κάνουμε και εγγραφή σε πίνακα σε μία βάση δεδομένων. Αυτό μπορεί να γίνει με την χρήση του αντικειμένου ADOCommand και χρήση της γλώσσας SQL.

6.10 Κυρίως Πρόγραμμα

```
1       int MinPts;
2       int i;
3       int j, recsize;
4       int Point;
5       int Knn;
6       double Eps;
7
8       Eps=0.007;
9       MinPts=4;      // minimum number of points
10      Knn=7;     //K nearest neighbours of each point is 7
11
12      recsize=RecordSize(Form1->ADOQuery1);
13      InfArray* DataSet =new InfArray[recsize];
14      LoadDataset(recsize,DataSet);
15
16      CalcKNearest(Knn,recsize, DataSet) ;
17
18      for(i=0;i<recsize;i++)
19       {
20       AverageDist=AverageDist+DataSet[i].KNearestDist;
21       }
22      AverageDist=AverageDist/recsize;
23
24      CheckDDRPoints(Eps,MinPts,recsize,DataSet);
25
26
27      //-------Second Big Iteration----------------------------
28      for(i=0;i<recsize;i++)
29       {
30
31       if(DataSet[i].PointType=="Border" && DataSet[i].ClusterID==0)
32         {
33          DataSet[i].ClusterID=-1;
34         }
```

```
35
36        else if (DataSet[i].ClusterID==0)
37          {
38          ClusterID=ClusterID+1;
39          FindClusters(i,recsize,DataSet);
40          }
41
42        }
43
44      ShowMessage("Clustering Finished");
45      StoreResults(recsize,DataSet);
46      delete [] DataSet;
47      }
```

Είναι το κυρίως πρόγραμμα. Αρχικά δηλώνονται οι κρίσιμες μεταβλητές Eps, MinPts και Knn και παρακάτω εισάγονται τιμές (Βλ 1-10). Στην συνέχεια αντλούμε από μια συνάρτηση, η οποία χρησιμοποιεί ερώτημα από την βάση δεδομένων, τον αριθμό των εγγραφών (Βλ 12). Δηλώνεται η δομή DataSet στην οποία θα εκχωρηθούν τα στοιχεία με τις πληροφορίες τους αμέσως μετά καλώντας την συνάρτηση LoadDataSet (Βλ 13-14). Στην συνέχεια καλείται η συνάρτηση CalcKNearest για να βρεθούν οι Knn κοντινότεροι γείτονες για κάθε σημείο (Βλ 16). Αφού βρεθούν αυτά και εκχωρηθούν στην δομή τότε ο αλγόριθμος βρίσκει τον μέσο όρο των μικρότερων αποστάσεων. Βρίσκεται το άθροισμα των μικρότερων αποστάσεων και διαιρείται με τον συνολικό αριθμό των στοιχείων στην δομή DataSet (Βλ 18-22). Έπειτα καλείται η CheckDDRPoints για να δημιουργηθούν οι συλλογές για κάθε στοιχείο και να χαρακτηριστούν τα στοιχεία ως Core ή Border (Βλ 24). Μετά, με την χρήση επαναληπτικής διαδικασίας ελέγχεται εάν το στοιχείο είναι χαρακτηρισμένο ως Border και έχει ταυτότητα =0 (Βλ 28,31). Σε περίπτωση που αυτό ισχύει τότε το στοιχείο αυτό παίρνει καινούρια ταυτότητα μείον ένα (-1) ή αλλιώς θόρυβος (Βλ 33). Σε διαφορετική περίπτωση, την περίπτωση δηλαδή το στοιχείο να έχει μόνο την ταυτότητα μηδέν, τότε αυξάνεται η ταυτότητα του κατά ένα και καλείται η συνάρτηση FindClusters που καλεί την GetDRPoints για να καταχωρηθεί το στοιχείο εκείνο σε μία συστάδα (Βλ 36-39).

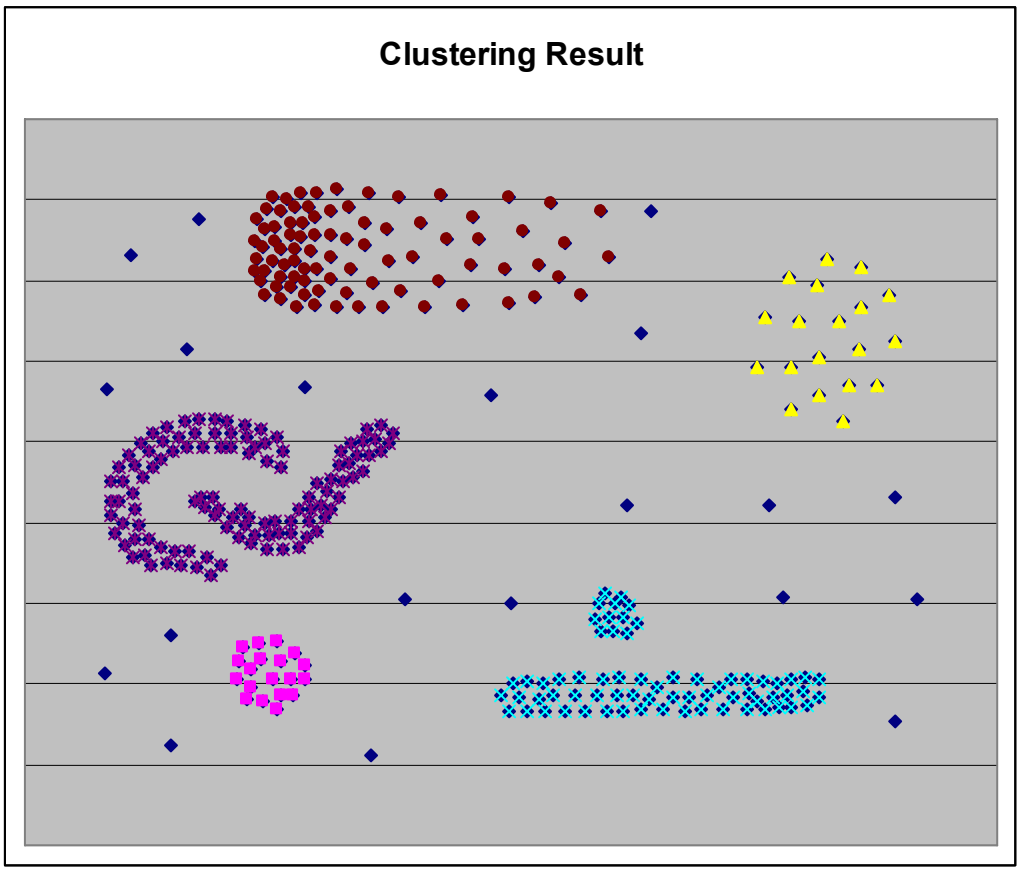
Αυτό θα συνεχιστεί μέχρι όλα τα στοιχεία να έχουν κάποια ταυτότητα, ή αλλιώς να έχουν συσταδοποιηθεί.

Παρακάτω παρουσιάζεται ένα αποτέλεσμα συσταδοποίησης με τις παρακάτω παραμέτρους.

Eps: 0.007

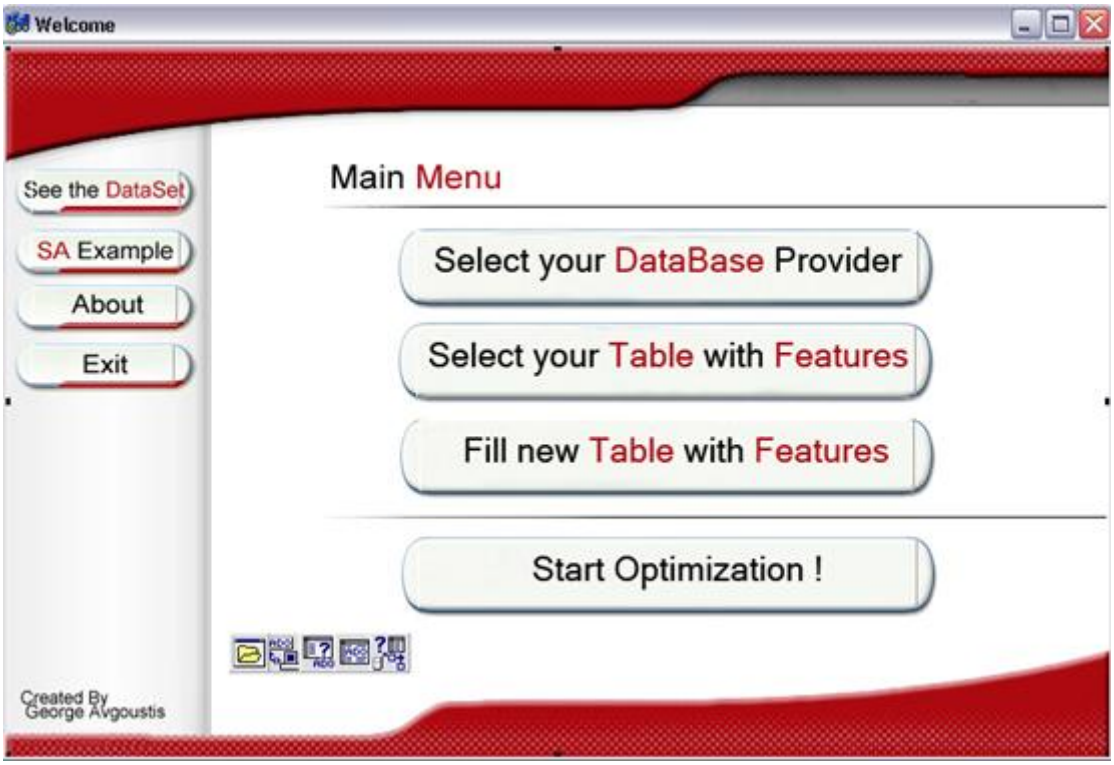
MinPts: 4

Knn: 7



7 ΕΠΕΞΗΓΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

7.1 Επεξήγηση Βασικής Φόρμας



Πρόκειται για την βασική φόρμα. Στο κέντρο της φόρμα υπάρχουν όλα τα κουμπιά που αφορούν στις κύριες λειτουργίες του προγράμματος. Αυτές είναι:

- Επιλογή παροχέα υπηρεσιών
- Επιλογή πίνακα με χαρακτηριστικά γνωρίσματα
- Δημιουργία καινούριου πίνακα με ορίσματα
- Έναρξη βελτιστοποίησης

Στις βοηθητικές λειτουργίες συγκαταλέγονται:

- Προβολή επιλεγμένου πίνακα
- Παράδειγμα Simulated Annealing
- About

Ο χρήστης για να αρχίσει την βελτιστοποίηση θα πρέπει να καθορίσει στην αρχή τον παροχέα υπηρεσίας, ο οποίος είναι ένας από τους παρακάτω.

- Microsoft Access
- Oracle
- SQL Server

Ανάλογα με την επιλογή του εμφανίζονται και οι ανάλογες φόρμες στις οποίες ο χρήστης καλείται να εισάγει πληροφορίες χρήσιμες για την επικοινωνία με την υπηρεσία. Για την επιλογή της Microsoft Access η μοναδική πληροφορία που χρειάζεται για την επικοινωνία με την βάση δεδομένων είναι το μονοπάτι στον δίσκο όπου είναι η βάση. Για αυτήν την περίπτωση χρησιμοποιείται ένα OpenDialogBox από το οποίο θα αντληθεί αυτή η πληροφορία. Στην περίπτωση που επιλεχθεί ο SQL Server τότε ο χρήστης καλείται να εισάγει το όνομα της βάσης με την οποία θέλει να επικοινωνήσει. Τέλος, ο παροχέας Oracle είναι ο πιο απαιτητικός καθώς εκτός από το όνομα της βάσης ο χρήστης καλείται να κάνει login εισάγοντας τα απαραίτητα στοιχεία δηλαδή το όνομα χρήστη και τον κωδικό. Όλες οι παραπάνω πληροφορίες είναι αναγκαίες για την συνέχεια καθώς με βάση αυτών σχηματίζεται μια παράμετρος επικοινωνίας στην οποία αργότερα θα βασιστεί κάθε υπολογισμός ή ενέργεια.

Όταν ο χρήστης επιλέξει τον παροχέα και εισάγει τις συμπληρωματικές αλλά αναγκαίες πληροφορίες τότε είναι σε θέση να επιλέξει πίνακα από την βάση δεδομένων κάνοντας χρήση του αντίστοιχου κουμπιού. Παρέχοντας αυτή την δυνατότητα μπορεί ο καθένας να κάνει χρήση της βοηθητικής λειτουργίας της προβολής πίνακα για να διαπιστώσει εάν πρόκειται για τον σωστό ή ακόμα και να δει οποιονδήποτε άλλο επιθυμεί όπως για παράδειγμα τον πίνακα με τα τελικά αποτελέσματα της βελτιστοποίησης.

Στην περίπτωση που δεν υπάρχει κάποια βάση με χαρακτηριστικά γνωρίσματα που έχουν προκύψει από την επεξεργασία βάσης με χρονοσειρές αλλά υπάρχει ο πίνακας με τις χρονοσειρές τότε υπάρχει η δυνατότητα να δημιουργηθεί πίνακας με χαρακτηριστικά γνωρίσματα, τα οποία υπολογίζονται με βάση μερικούς τύπους.

Έχοντας όλα τα παραπάνω μπορεί πλέον ο χρήστης να προχωρήσει στην διαδικασία της βελτιστοποίησης, της επιλογής δηλαδή των κατάλληλων γνωρισμάτων από μια βάση. Φυσικά τα αποτελέσματα μπορούν να διαφέρουν λόγω της παραμετροποίησης.

Κώδικας Βασικής Φόρμας

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

#include <vcl.h>
#pragma hdrstop
#include <Olectrls.hpp>
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit10.h"
#include "Unit8.h"
#include "Unit11.h"
#include "Unit12.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

#include<math.h>

TMainForm *MainForm;

AnsiString TableName;

//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{

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




void __fastcall TMainForm::SpeedButton1Click(TObject *Sender)
{
//Terminate the application
Application->Terminate();
}
//---------------------------------------------------------------------------


void __fastcall TMainForm::ShowDataSetClick(TObject *Sender)
{
//Control the case of not having selected any table

AnsiString sql="";
```

```
TableName=DataSetSelect->ComboBox1->Text;
if(TableName=="")
MessageBox(NULL,"No Table Selected.. Please Select Table","Error",MB_OK);

// else add new SQL string to gather the data

else
{
if(DataBaseSelect->SelectList->Text=="SQL Server" || DataBaseSelect->SelectList-
>Text=="Microsoft Access" )
      {
      //construct the sql string and put it into the ADOQuery component
      sql="select * from " ;
      sql+=TableName;
      See->Show();
      ADOQuery1->Active=false;
      ADOQuery1->SQL->Clear();
      ADOQuery1->SQL->Text=sql;
      ADOQuery1->Active=true;
      ADOQuery1->Open();
      }
else
    {
    //construct the sql string and put it into the ADOQuery component
    sql="select * from " ;
    sql+=OracleBase->Edit2->Text;
    sql+=".";
    sql+=TableName;
    See->Show();
    ADOQuery1->Active=false;
    ADOQuery1->SQL->Clear();
    ShowMessage(sql);
    ADOQuery1->SQL->Text=sql;
    ADOQuery1->Active=true;
    ADOQuery1->Open();

    }
// Prepare the StringGrid and use an iteration to Show the Data

See->StringGrid1->ColCount=ADOQuery1->FieldCount;
See->StringGrid1->RowCount=ADOQuery1->RecordCount;
See->ProgressBar1->Max=ADOQuery1->FieldCount;
for(int j=0;j<ADOQuery1->FieldCount;j++)
      {

       for(int i=0;i<ADOQuery1->RecordCount;i++)
         {
         See->StringGrid1->Cells[j][i]=ADOQuery1->Fields->Fields[j]->AsString;
         ADOQuery1->Next();
         }
     ADOQuery1->Close();
     ADOQuery1->Open();
     See->ProgressBar1->StepIt();
      }
```

```
   ADOQuery1->Close();
   ADOQuery1->SQL->Clear();

}

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



void __fastcall TMainForm::SelectDataSetClick(TObject *Sender)
{
// Use of TStrings to fill the combobox with the table names of the user only
// set true if you want to show system tables as well
DataSetSelect->ComboBox1->Items->Clear();
TStrings *SL = new TStringList;
if(DataBaseSelect->SelectList->Text=="")
    {
    MessageBox(NULL,"Please Select Database","Error",MB_OK);
    DataBaseSelect->ShowModal();
    }
//use function Get Table Names for all but oracle
 else if(DataBaseSelect->SelectList->Text=="Microsoft Access" || DataBaseSelect->SelectList-
>Text=="SQL Server")
   {
MainForm->ADOConnection1->GetTableNames(SL, false);
DataSetSelect->ComboBox1->Items=SL;
DataSetSelect->ShowModal();
   }

 else
   {

    if(DataBaseSelect->SelectList->Text=="")
    {
    MessageBox(NULL,"Please Select Database","Error",MB_OK);
    DataBaseSelect->ShowModal();
    }
// use of OpenSchema function to gather the names of tables of Oracle Database
   int i,j ;
   Variant a;
   a = VarArrayCreate(OPENARRAY(int, (0, 3)), varVariant);
   a.PutElement(OracleBase->Edit2->Text,1);
   MainForm->ADOConnection1->OpenSchema(siTables, a, EmptyParam, MainForm-
>ADODataSet1)  ;

  if(ADODataSet1->Fields->Fields[2]->Value.IsNull())
     {
     MessageBox(NULL,"No Valid Table Please Select another one","error",MB_OK);
     }
   ShowMessage(MainForm->ADODataSet1->RecordCount);
   for(j=0;j<MainForm->ADODataSet1->RecordCount ;j++)
     {
```

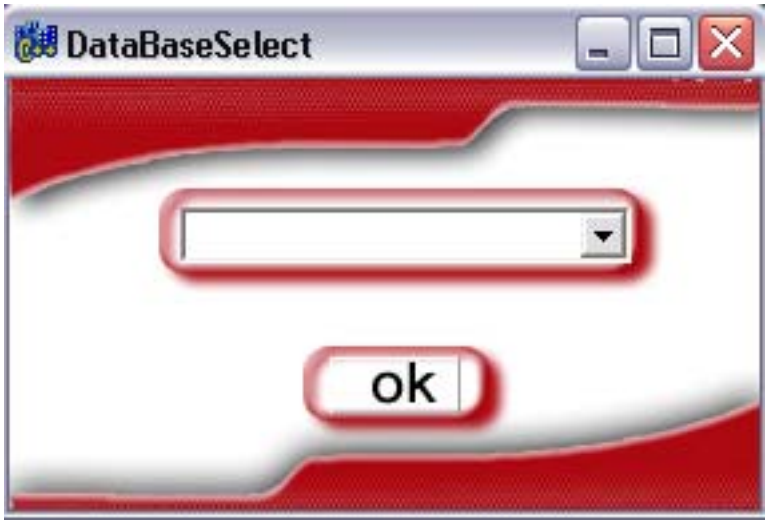
```
     DataSetSelect->ComboBox1->Items->Add(ADODataSet1->Fields->Fields[2]->Value);
     ADODataSet1->Next();
    }

    MainForm->ADODataSet1->Close() ;
    DataSetSelect->ShowModal();
  }


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



void __fastcall TMainForm::SelectDataBaseClick(TObject *Sender)
{
DataBaseSelect->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
//when start disable ADOConnection component
ADOConnection1->Connected=false;

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



void __fastcall TMainForm::ExampleClick(TObject *Sender)
{
// Goto to the Simulated annealing example
SAExample->ShowModal();


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


void __fastcall TMainForm::ShowDataSetMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
// Use this event to update the ShowHint Label for information
ShowHint->Caption="Here you can See the Selected DataSet.";
}
//---------------------------------------------------------------------------




void __fastcall TMainForm::ExampleMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
```

```
ShowHint->Caption="Here you can perform an example of Simulated Annealing";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::BackGroundMouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Welcome to SA-DBSCAN";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectDataBaseMouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to select you DataBase.";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectDataSetMouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to Select your DataSet";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DBSCANConfigreMouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Pre-Configure DBSCAN. You can also configure during the application";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StartItMouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Begin the main function...by clicking Here";
}
//---------------------------------------------------------------------------


void __fastcall TMainForm::StartItClick(TObject *Sender)
{
// Start the optimization
// But first check if there is any database selected in order to gather the name
// of the table to start the optimization

if(DataBaseSelect->SelectList->Text=="")
   {
   MessageBox(NULL,"Please select a DataBase before Continuing...","Error",MB_OK);
   }

// make the appropriate checks for every single case.

if(DataBaseSelect->SelectList->Text=="SQL Server")
    {
```

```
   AnsiString string;
   AnsiString Set;
   Set=DataSetSelect->ComboBox1->Text;
      if(Set=="")
         {
         MessageBox(NULL, "Please Select a valid DataSet","Error",MB_OK);
          SelectDataSet->Click();

         }
   Set=DataSetSelect->ComboBox1->Text;
   MainFunction->Show();
   }

   if(DataBaseSelect->SelectList->Text=="Microsoft Access")
   {
   AnsiString string;
   AnsiString Set;
   Set=DataSetSelect->ComboBox1->Text;
      if(Set=="")
         {
         MessageBox(NULL, "Please Select a valid DataSet","Error",MB_OK);
          SelectDataSet->Click();

         }
   Set=DataSetSelect->ComboBox1->Text;
   MainFunction->Show();
   }
    if(DataBaseSelect->SelectList->Text=="Oracle")
   {
   AnsiString string;
   AnsiString Set;
   Set=DataSetSelect->ComboBox1->Text;
      if(Set=="")
         {
         MessageBox(NULL, "Please Select a valid DataSet","Error",MB_OK);
          SelectDataSet->Click();

         }
   Set=DataSetSelect->ComboBox1->Text;
   MainFunction->Show();
   }


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

void __fastcall TMainForm::SpeedButton1MouseMove(TObject *Sender,
     TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Leave";
}
//---------------------------------------------------------------------------


void __fastcall TMainForm::SpeedButton2MouseMove(TObject *Sender,
```

```
    TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Show Version Info";
}
//---------------------------------------------------------------------------



void __fastcall TMainForm::FillDataSetClick(TObject *Sender)
{
//make the appropriate checks before opening
if(DataBaseSelect->SelectList->Text=="")
    {
    DataBaseSelect->ShowModal();
    MainForm->SelectDataSet->Click();
    }

else
MainForm->SelectDataSet->Click();
FeatureCalc->Show();
}
//---------------------------------------------------------------------------
```

7.2 Επεξήγηση Φόρμας Επιλογής Βάσης



Στην φόρμα αυτή το μόνο που γίνεται η επιλογή της βάσης δεδομένων. Ο χρήστης επιλέγει ανάμεσα σε:

- Oracle
- SQL Server
- Microsoft Access

Εάν ο χρήστης επιλέξει Oracle τότε θα πρέπει να κατασκευαστεί ένα ConnectionString που θα χρησιμοποιηθεί για την επικοινωνία του αντικειμένου ADOConnection με τον παροχέα. Ο κάθε παροχέας ζητά διαφορετικές πληροφορίες. Η Oracle θέλει όνομα βάσης, όνομα χρήστη και Password. Για τον λόγο αυτό έχει κατασκευαστεί μία φόρμα που θα παρουσιαστεί παρακάτω, η οποία δέχεται αυτές τις πληροφορίες που θα συνθέσουν το ConnectionString. Στην περίπτωση που επιλεχθεί η υπηρεσία SQL Server τότε το μόνο που χρειάζεται για να συντεθεί το ConnectionString είναι το όνομα της βάσης. Έτσι παρουσιάζεται στον χρήστη μια φόρμα όπου καλείται να γράψει το όνομα της βάσης. Τέλος στην επιλογή MicrosoftAccess αυτό που χρειάζεται είναι η εύρεση του μονοπατιού της βάσης καθώς η βάση της Access μπορεί να βρίσκεται παντού. Για την συλλογή της πληροφορίας αυτής γίνεται με την χρήση ενός DialogBox.

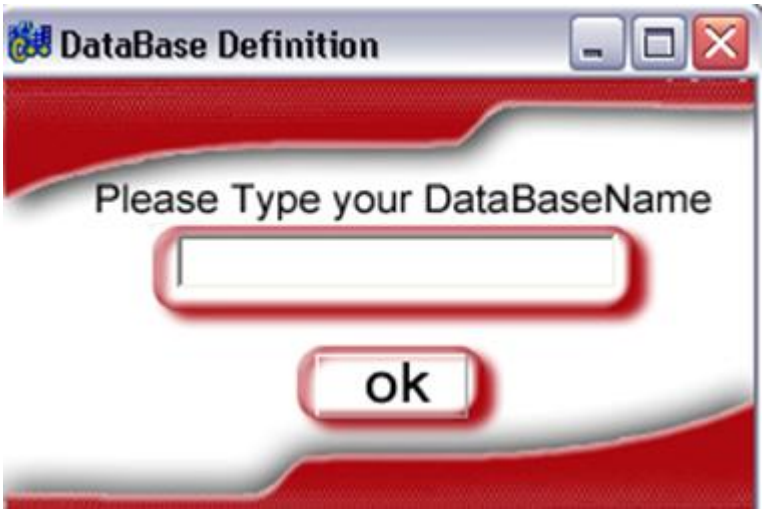
Κώδικας Φόρμας Επιλογής Βάσης

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

#include <vcl.h>
#pragma hdrstop

#include "Unit4.h"
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit7.h"
#include "Unit8.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit10.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDataBaseSelect *DataBaseSelect;
AnsiString s;
//---------------------------------------------------------------------------
__fastcall TDataBaseSelect::TDataBaseSelect(TComponent* Owner)
        : TForm(Owner)
{

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


void __fastcall TDataBaseSelect::ConfirmSelectionClick(TObject *Sender)
{
// Do a DataBase selection between Oracle, SQL Server and Access
// and build a connection string
if(SelectList->Text=="")
  {
  MessageBox(NULL,"Please Select Something","Error",MB_OK);
  }
```

```
//construction of connection string for MS Access with the information gathered

else if(SelectList->Text=="Microsoft Access")
   {
AnsiString file;
MainForm->ADOConnection1->Connected=false;
MainForm->OpenDialog1->Execute();
file=MainForm->OpenDialog1->FileName;            // Every DataBase requires
s="Provider=Microsoft.Jet.OLEDB.4.0;Data Source="; //connection string
s+=file;
s+=";Persist Security Info=False";
MainForm->ADOConnection1->ConnectionString=s;
MainForm->ADOConnection1->Connected=true;
DataBaseSelect->Close();
DataSetSelect->ComboBox1->Items->Clear();
DataSetSelect->ComboBox1->Text="";
s="";

  }

//construction of connection string for oracle with the information gathered
else if(SelectList->Text=="Oracle")
  {

 AnsiString string,DataName,UserName,Password;
 MainForm->ADOConnection1->Connected=false;
 OracleBase->ShowModal();
 DataName=OracleBase->Edit1->Text;
 UserName=OracleBase->Edit2->Text;
 Password=OracleBase->MaskEdit1->Text;
 string="Provider=MSDAORA.1;Password=";
 string+=Password;
 string+=";User ID=";
 string+=UserName;
 string+=";Data Source=";
 string+=DataName;
 string+=";Persist Security Info=True";
 MainForm->ADOConnection1->ConnectionString=string;
 MainForm->ADOConnection1->Connected=true;
 DataBaseSelect->Close();
 DataSetSelect->ComboBox1->Items->Clear();
 DataSetSelect->ComboBox1->Text="";
 string="";
  }


 //construction of connection string for sql server with the information gathered
 if(SelectList->Text=="SQL Server")
    {
      AnsiString DBase;
      DefData->ShowModal();
      DBase=DefData->Edit1->Text;
      MainForm->ADOConnection1->Connected=false;
      s="";
```

```
s="Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial
Catalog=";
        s+=DBase;
        MainForm->ADOConnection1->ConnectionString=s;
        MainForm->ADOConnection1->Connected=true;
        DataSetSelect->ComboBox1->Items->Clear();        //the same goes here as well
        DataSetSelect->ComboBox1->Text="";
        DataBaseSelect->Close();


    }

}
//---------------------------------------------------------------------------
```

7.3 Επεξήγηση Φόρμας Διατύπωσης Ονόματος Βάσης

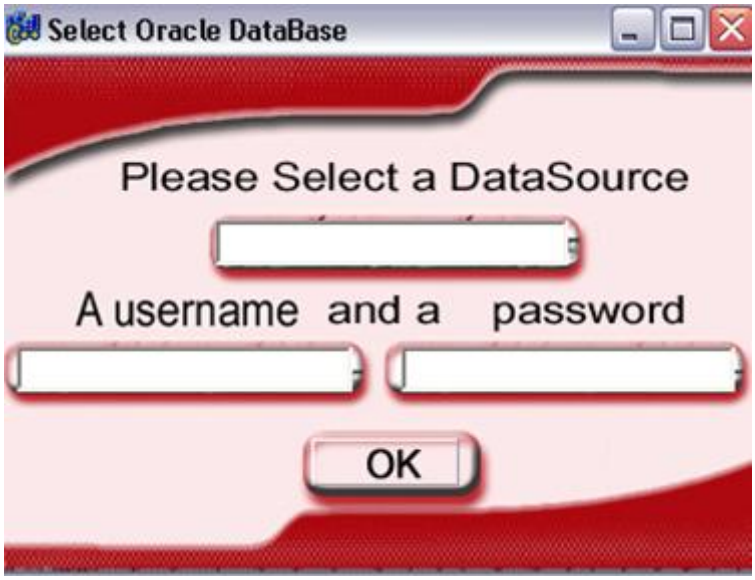


Εδώ απλά ο χρήστης καλείται να εισάγει το όνομα της βάσης για να χρησιμοποιηθεί για την κατασκευή του ConnectionString. Αυτή η φόρμα εμφανίζεται στην περίπτωση που ο χρήστης επιλέξει τις υπηρεσίες του SQL Server.

Κώδικας Φόρμας Διατύπωσης Ονόματος Βάσης

```
#include <vcl.h>
#pragma hdrstop
#include "Unit7.h"
#include "Unit1.h"
#include "Unit10.h"
#include "Unit3.h"
#include "Unit4.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDefData *DefData;
__fastcall TDefData::TDefData(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TDefData::SpeedButton1Click(TObject *Sender)
{
// Just type a database name
if(Edit1->Text=="")
  MessageBox(NULL,"Please Type A DataBase","Error",MB_OK);
else
{
DefData->Close();
}}
```

7.4 Επεξήγηση Φόρμας Εισαγωγής Στοιχείων για Oracle

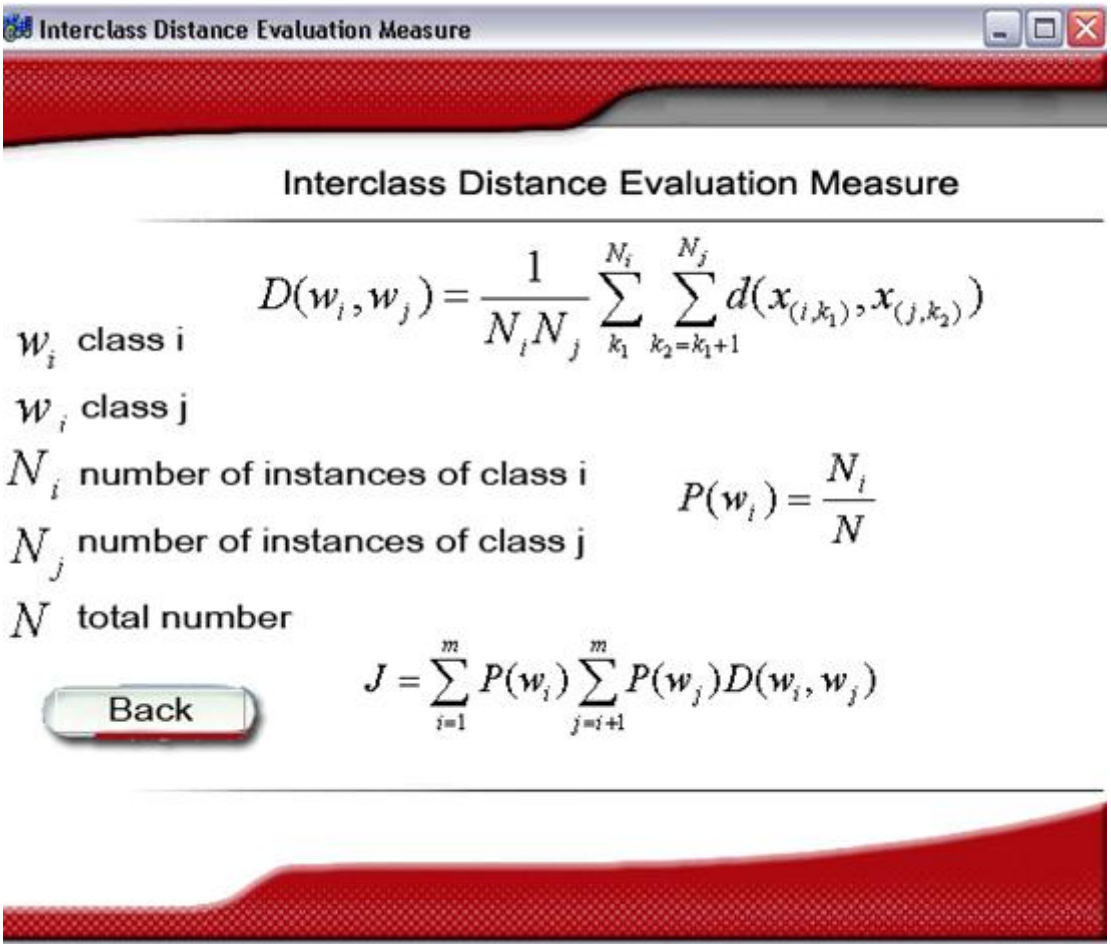


Εδώ απλά ο χρήστης καλείται να εισάγει στοιχεία για να χρησιμοποιηθούν για την κατασκευή του ConnectionString για την επικοινωνία με την Oracle.

Κώδικας φόρμας εισαγωγής στοιχείων για Oracle

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

#include <vcl.h>
#pragma hdrstop
#include "Unit10.h"
#include "Unit1.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit7.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TOracleBase *OracleBase;
//---------------------------------------------------------------------------
__fastcall TOracleBase::TOracleBase(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TOracleBase::SpeedButton1Click(TObject *Sender)
{
if(Edit1->Text=="" || Edit2->Text=="")
      {
     MessageBox(NULL,"Wrong Input!! Please Try
Again","Error",MB_OK|MB_ICONWARNING);
      }
else
   OracleBase->Close();}
```

7.5 Επεξήγηση Φόρμας Ένδειξης τύπου Αξιολόγησης

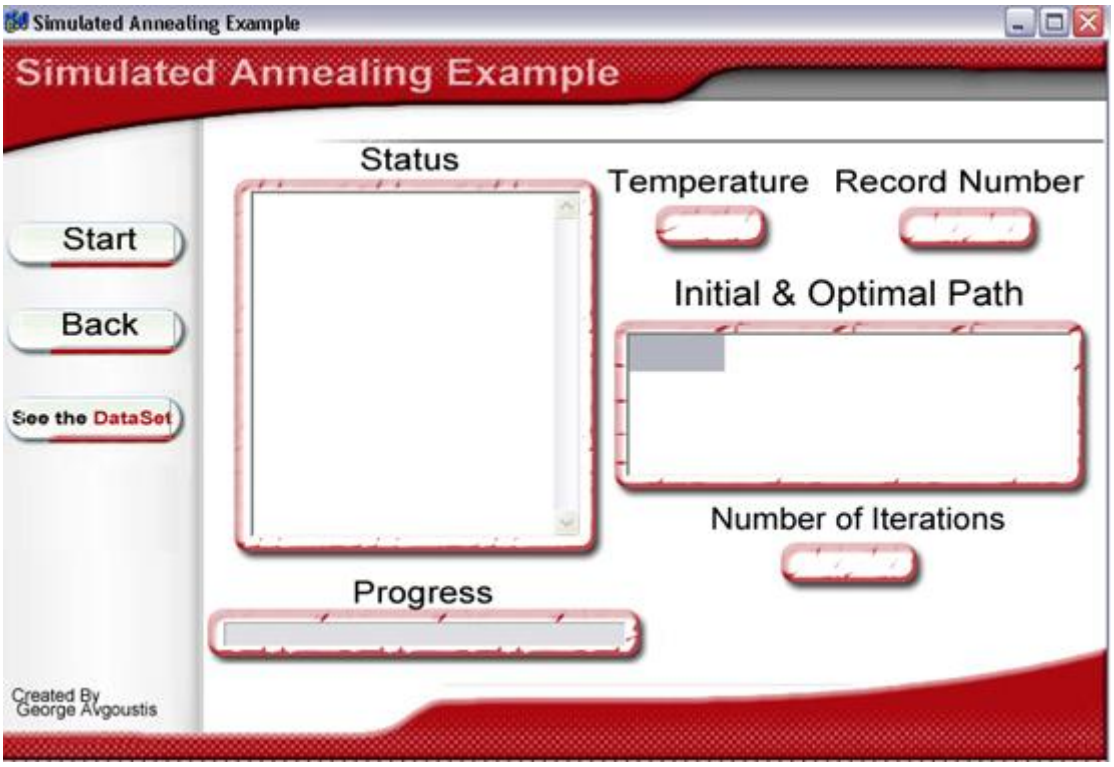


Στην φόρμα αυτή ο χρήστης πληροφορείται για τον τύπο αξιολόγησης ποιότητας συστάδων που χρησιμοποιούμε. Δεν υπάρχει κάποια συγκεκριμένη λειτουργία της φόρμας αυτής. Χρησιμοποιείται μόνο για πληροφοριακό σκοπό.

Κώδικας Φόρμας Ένδειξης τύπου αξιολόγησης

```
#include <vcl.h>
#pragma hdrstop
#include "Unit12.h"
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TShowType *ShowType;
//---------------------------------------------------------------------------
__fastcall TShowType: TShowType(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TShowType::SpeedButton1Click(TObject *Sender)
{ ShowType->Close();}
```

7.6 Επεξήγηση φόρμας παραδείγματος Simulated Annealing



Στην φόρμα αυτή παρουσιάζεται ένα παράδειγμα εύρεσης συντομότερου μονοπατιού. Πρόκειται για μια πλήρη εφαρμογή του αλγορίθμου Simulated Annealing. Στο παράθυρο με ετικέτα Status, το οποίο είναι ένα Memo, ο χρήστης έχει την δυνατότητα να παρακολουθεί την εξέλιξη του προγράμματος. Για την πληροφόρηση του παρουσιάζονται στο Status το μήκος του καινούριου μονοπατιού σε κάθε βήμα, η τιμή της θερμοκρασίας, μιας μεταβλητής που εξηγείται σε παραπάνω κεφάλαιο και οι επιτυχημένες αλλαγές μονοπατιού σε κάθε βήμα. Πάνω δεξιά φαίνονται οι αρχικές τιμές της θερμοκρασίας και ο αριθμός των στοιχείων. Σε έναν πίνακα με ετικέτα Initial & Optimal Path φαίνεται το αρχικό και το τελικό μονοπάτι. Από κάτω ο χρήστης μπορεί δει τον συνολικό αριθμό των επαναλήψεων που χρειάστηκαν για την εύρεση του συντομότερου μονοπατιού. Ο χρήστης έχει την δυνατότητα να παρακολουθήσει την πρόοδο της διαδικασίας βλέποντας το Progress Bar. Ακόμα υπάρχει και η δυνατότητα να απεικονίσουμε τα στοιχεία σε γράφημα, σε άλλη φόρμα που θα παρουσιαστεί παρακάτω.

Κώδικας Φόρμας παραδείγματος Simulated Annealing

```
#include <vcl.h>
#pragma hdrstop
#include "Unit5.h"
#include "Unit1.h"
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include "Unit2.h"
#include "Unit6.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit8.h"
#define TFACTR 0.9
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TSAExample *SAExample;
#define maxsize 20


float x[maxsize];               //x[(0). .(ncity-1)] array used for x coordinates
float y[maxsize];               // y[(0). .(ncity-1)] array used for y coordinates
int iorder[maxsize];          //iorder[(0). .(ncity-1)] specifies the order in which
                                                                  //the cities are visited

int ncity=20;


//--------------ALEN-------------------------------
// Retuns the Distance of two points.
float alen(float a, float b, float c, float d)
{

        return sqrt(((b)-(a))*((b)-(a))+((d)-(c))*((d)-(c)));

}


//---------------------------ANNEAL-----------------------------
// The annealing method

void anneal(float x[], float y[], int iorder[], int ncity)
{
FILE *finalpath;        //File used for output

//-----------------Prologue-------------------------------
SAExample->Memo1->Lines->Add("Simulated Annealing Starting....");
SAExample->Memo1->Lines->Add("Initializing variables...");
```

```
//---------------------------------------------------------
//--------------Variables----------------------------------
        int metrop(float de, float t);
        float ran3(long *idum);
        float revcst(float x[], float y[], int iorder[], int ncity, int n[]);
        void reverse(int iorder[], int ncity, int n[]);
        int ans,nover,nlimit,i1,i2;
        int i,j,k,nsucc,nn,idec;
        static int n[7];            //used for segment informations
        long idum;                  //element n[0] is not used
        unsigned long iseed;
        float path,de,t;

        nover=100*ncity;
        nlimit=10*ncity;
        path=0.0;
        t=70.0;                     // Temperture which is used
SAExample->Label1->Caption=t;               // in the metropolis algorithm
SAExample->Label2->Caption=ncity;
//------------------Comments-----------------------------
SAExample->Memo1->Lines->Add("Done...");
SAExample->Memo1->Lines->Add("Initializing IORDER....");
SAExample->Memo1->Lines->Add("Done...");
SAExample->Memo1->Lines->Add("Calculating Initial Path Cost....");


/************** calculate initial path cost ************************/

        for (i=0;i<ncity;i++)
          {
                i1=iorder[i];
                i2=iorder[i+1];
                path += alen(x[i1],x[i2],y[i1],y[i2]);
        }
          //  connect last element with first element to make a circle

            i1=iorder[ncity-1];
            i2=iorder[0];
            path += alen(x[i1],x[i2],y[i1],y[i2]);

SAExample->Memo1->Lines->Add("Done...");
SAExample->Memo1->Lines->Add("-----INITIAL PATH------");
SAExample->Memo1->Lines->Add(path);

finalpath=fopen("finalpath.dat", "w");    /* open finalpath.dat */

SAExample->Memo1->Lines->Add("Starting...");
SAExample->Memo1->Lines->Add("------INITIAL TEMPERATURE-------");
SAExample->Memo1->Lines->Add(t);
SAExample->Memo1->Lines->Add("**************************");

/************** start the algorithm ************************/

        for (j=1;j<=100;j++)              //first iteration 100 times
        {
```

```
SAExample->ProgressBar1->StepIt();
        nsucc=0;
        for (k=1;k<=nover;k++) //hold T constant for nover=100*ncity
        {                              // reconfigurations
            //pick randomly two elements from iorder[ncity]
            do {
               n[1]=(int) (ncity*ran3(&idum));   //start of segment
               n[2]=(int) ((ncity-1)*ran3(&idum)); //end of segment
               if (n[2] >= n[1]) ++n[2];
               nn=(1+(n[1]-n[2]+ncity-1) % ncity);
               }
        while (nn<3);                  //nn is the number of cities
                        //not on the segment.

//(nn<3) prevents from choosing the whole path because reversal or transport
//of the whole path has no meaning (no change in de).
//For those two elements from iorder array do:

         de=revcst(x,y,iorder,ncity,n);   //calculate cost difference de
            ans=metrop(de,t);     //consult the oracle
                if (ans)
              {          //if answer true then
                    ++nsucc;          //increase nsucc
                    path += de;       //calculate new cost
                  //Enew=Eold+de
                reverse(iorder,ncity,n); //make the path reversal
                    }

            if (nsucc >= nlimit) break;   //if successfull moves
        }                           // more than nlimit break,
                        //to control the maximum
                        //amount of nsucc

        //save the minimum configuration for this T value
        for (i=0; i<ncity; i++)
   fprintf(finalpath, "%d\t", iorder[i] );
        fprintf(finalpath, "%lf\n",path );

        t *= TFACTR;                    //decrease T by Tfactor
        if (nsucc = = 0)            //to avoid many uphill climbing
     { fclose(finalpath);       //if no successfull moves..break
      SAExample->Label3->Caption=j;
      return;
      }

   SAExample->Memo1->Lines->Add("----Path Length----------");
   SAExample->Memo1->Lines->Add(path);
   SAExample->Memo1->Lines->Add("----Successful Moves------");
   SAExample->Memo1->Lines->Add(nsucc);
   SAExample->Memo1->Lines->Add("-----new temperature------");
   SAExample->Memo1->Lines->Add(t);
   SAExample->Memo1->Lines->Add("**************************");
   SAExample->Label3->Caption=j;
  }
    fclose(finalpath);
```

```
        SAExample->Label3->Caption=j;
}


//-------------------REVCST---------------------------------------
// returns the cost of reversing the segment
// n[1]->start , n[2]->end , n[3]->spot before the start,
// n[4]->after the end.
float revcst(float x[], float y[], int iorder[], int ncity, int n[])
{
        float xx[5],yy[5],de;
        int j,ii;
//new conditions used alternatively (if..else  is more costly than +-% operations)

    n[3]=((n[1] + ncity - 1) % ncity);          // Find the city before n[1]
    n[4]= (n[2] + ncity + 1) % ncity;           //Find the city after n[2]


        for (j=1;j<=4;j++) {
                ii=iorder[n[j]];
                xx[j]=x[ii];
                yy[j]=y[ii];
        }
        de = -alen(xx[1],xx[3],yy[1],yy[3]);
        de -= alen(xx[2],xx[4],yy[2],yy[4]);   //calculate the cost
        de += alen(xx[1],xx[4],yy[1],yy[4]);
        de += alen(xx[2],xx[3],yy[2],yy[3]);
        return de;
}
//-------------------REVERSE---------------------------------
// makes the reversal
void reverse(int iorder[], int ncity, int n[])
{
        int nn,j,k,l,itmp;

     nn=(1+((n[2]-n[1]+ncity) % ncity))/2;  //This many cities must be swapped
        for (j=1;j<=nn;j++) {                           // to effect the reversal.

          k=((n[1]+j+ncity-1) % ncity);   //Start at the ends of the segment and
          l=((n[2]-j+ncity+1) % ncity);   //swap pairs of cities, moving toward
                                //the center.
                itmp=iorder[k];
                iorder[k]=iorder[l];
                iorder[l]=itmp;         //swap!

        }
}

//----------------METROPOLIS-----------------------------------
// This is the Oracle that keeps us away from local minima
int metrop(float de, float t)
{
        float ran3(long *idum);
        static long gljdum=1;    //seed for the ran3()
```

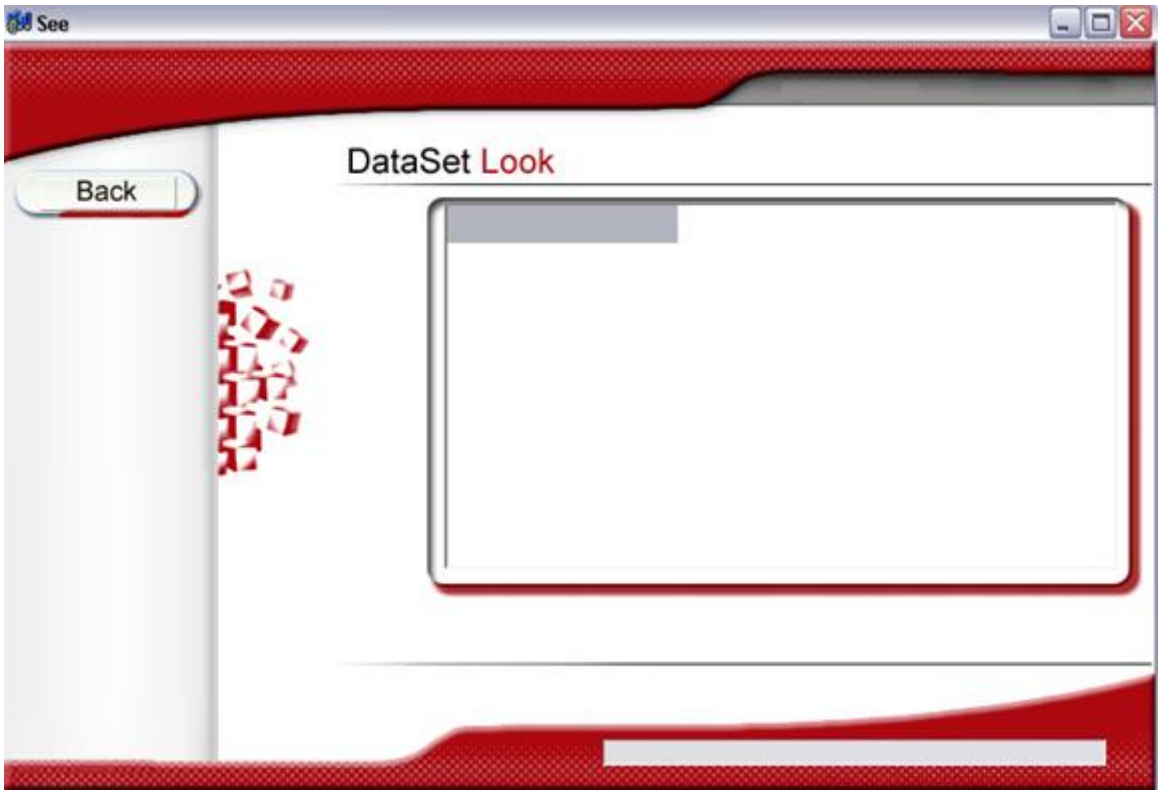
```
// if reversal cost negative do the reversal or do it with
//propability ran3(&gljdum) < exp(-de/t) where t is temperature

        return de < 0.0 || ran3(&gljdum) < exp(-de/t);
}
//--------------RAN3-----------------------------------------
// Random number generator, generates numbers from 0 to 1
float ran3(long *idum)
{
        static int inext,inextp;
        static long ma[56];
        static int iff=0;
        long mj,mk;
        int i,ii,k;

        if (*idum < 0 || iff == 0) {
                iff=1;
                mj=MSEED-(*idum < 0 ? -*idum : *idum);
                mj %= MBIG;
                ma[55]=mj;
                mk=1;
                for (i=1;i<=54;i++) {
                        ii=(21*i) % 55;
                        ma[ii]=mk;
                        mk=mj-mk;
                        if (mk < MZ) mk += MBIG;
                        mj=ma[ii];
                }
                for (k=1;k<=4;k++)
                        for (i=1;i<=55;i++) {
                                ma[i] -= ma[1+(i+30) % 55];
                                if (ma[i] < MZ) ma[i] += MBIG;
                        }
                inext=0;
                inextp=31;
                *idum=1;
        }
        if (++inext == 56) inext=1;
        if (++inextp == 56) inextp=1;
        mj=ma[inext]-ma[inextp];
        if (mj < MZ) mj += MBIG;
        ma[inext]=mj;
        return mj*FAC;
}
//---------------------------------------------------------------------------
__fastcall TSAExample: TSAExample(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TSAExample::GoBackClick(TObject *Sender)
{

Memo1->Lines->Clear();
Label1->Caption="";
```

```
Label2->Caption="";              //Clear all before leaving
Label3->Caption="";
ProgressBar1->Position=0;
for(int i=0;i<ncity;i++)
{
  StringGrid1->Cells[i][0]="";
  StringGrid1->Cells[i][1]="";
}
SAExample->Close();
}
//---------------------------------------------------------------------------


void __fastcall TSAExample::StartClick(TObject *Sender)
{
long idum;
int i=0, j;
Memo1->Lines->Clear();
StringGrid1->ColCount=ncity;
StringGrid1->RowCount=2;


// load data

  FILE *xdata,*ydata , *test, *testran3;
  float costpath;

// File Open for xdata,ydata and test.

  xdata=fopen("x.dat", "r");          /* open x.dat */
  ydata=fopen("y.dat", "r");          /* open y.dat */
  test=fopen("test.dat", "w");        /* open test.dat */

// Loadind Data into X[] and y[] from files

  while ((fscanf(xdata, "%f", &x[i]) !=EOF) && (i<maxsize))
  {
    i++;                              /* reading input data */
  }

  i=0;

  while ((fscanf(ydata, "%f", &y[i]) !=EOF) && (i<maxsize))
  {
    i++;                              /* reading input data */
  }

// Just for testing the x ,y arrays

  for (j=0; j<maxsize; j++)
  {
  fprintf(test, "%6f\t%6f\n", x[j], y[j] );
  }

// Close objects
```

```
  fclose(xdata);
  fclose(ydata);
  fclose(test);

// Initialize iorder[] array that contains the initial path

  for (i=0; i<ncity; i++)
  {
      iorder[i]=i;                          //!!from 0..19
      StringGrid1->Cells[i][0]=i;
  }

// Testing alen function that returns the distance

          costpath = alen(x[1],x[3],y[1],y[3]);

// testing ran3 (Random Number Generator)

          testran3=fopen("testran3.dat", "w");        /* testran3.dat */
          idum = -1;        /* It is the seed for the ran3(), when -1 initializes
                    the function  */

  for (j=0; j<20; j++)
  {
  fprintf(testran3, "%lf\t%d\n", ran3(&idum), (int)(ncity*ran3(&idum)) );
  }

  fclose(testran3);

// Start annealing
  anneal(x, y, iorder, ncity);

// Show Results

    for(j=0;j<ncity;j++)
      StringGrid1->Cells[j][1]=iorder[j];

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

void __fastcall TSAExample::ShowDataSetClick(TObject *Sender)
{
SADataLook->ADOConnection1->Connected=true;     //enable connection
SADataLook->ADOTable1->Active=true;             // enable ADOTable
SADataLook->ADOQuery1->SQL->Clear();            //Clear SQL statement
SADataLook->ADOQuery1->SQL->Add("select * from Data ");   // Add new...
SADataLook->ADOQuery1->Active=true;          // enable Query
SADataLook->ADOQuery1->Open();               // open...
SADataLook->StringGrid1->RowCount=SADataLook->ADOQuery1->RecordCount+1;
SADataLook->StringGrid1->Cells[0][0]="X";     //Set labels
SADataLook->StringGrid1->Cells[1][0]="Y";

// iteration for representing the data on to a stringrid
```

```
 for(int num=1;num<SADataLook->ADOQuery1->FieldCount;num++)
    {

    for(int num2=1; num2<SADataLook->ADOQuery1->RecordCount+1;num2++)
              {

  SADataLook->StringGrid1->Cells[num-1][num2]=SADataLook->ADOQuery1->Fields-
>Fields[num]->Value;
               SADataLook->ADOQuery1->Next();
              }
     SADataLook->ADOQuery1->Close();
     SADataLook->ADOQuery1->Open();
    }
SADataLook->ADOQuery1->Close();
SADataLook->Show();
}

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

7.7 Επεξήγηση Φόρμας Απεικόνισης Πινάκων



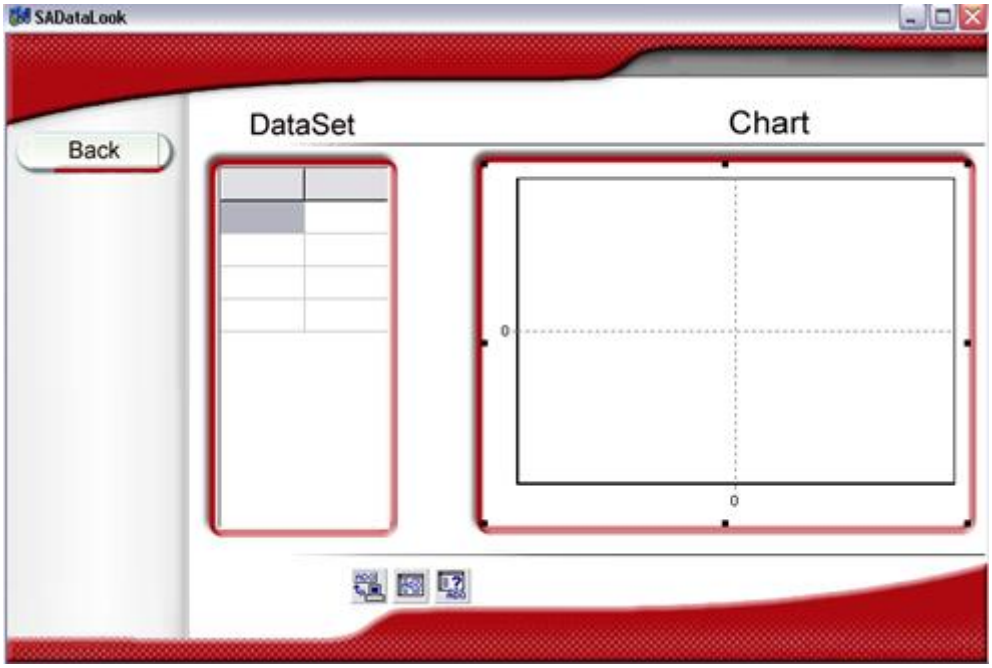
Είναι μια φόρμα που χρησιμοποιείται πληροφοριακά. Ο χρήστης έχοντας επιλέξει έναν πίνακα μπορεί να δει τα περιεχόμενα του σε αυτή την φόρμα.

Κώδικας Φόρμας Απεικόνισης Πινάκων

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

#include <vcl.h>
#pragma hdrstop
#include "Unit2.h"
#include "Unit1.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit8.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TSee *See;
//---------------------------------------------------------------------------
```

```
__fastcall TSee::TSee(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TSee::SpeedButton1Click(TObject *Sender)
{

for(int j=0;j<MainForm->ADOQuery1->FieldCount;j++)
       {

        for(int i=0;i<MainForm->ADOQuery1->RecordCount;i++)
          {
          See->StringGrid1->Cells[j][i]=0;
          }
        }
MainForm->ADOQuery1->SQL->Clear();
MainForm->ADOQuery1->Active=false;

See->Close();
}
//---------------------------------------------------------------------------
void __fastcall TSee::FormCreate(TObject *Sender)
{
if(MainForm->ADOQuery1->Active==false)
 {
 }

else if(MainForm->ADOQuery1->SQL->Text!="" && MainForm->ADOQuery1->Active==false)
 {
ProgressBar1->Max=MainForm->ADOQuery1->FieldCount;
ProgressBar1->Step=1;
ProgressBar1->Smooth=true;
 }
else
  ShowMessage("No sql");
}
{
Results->Close();
}
//---------------------------------------------------------------------------
```

7.8 Επεξήγηση Φόρμας Απεικόνισης Στοιχείων του Παραδείγματος Simulated Annealing

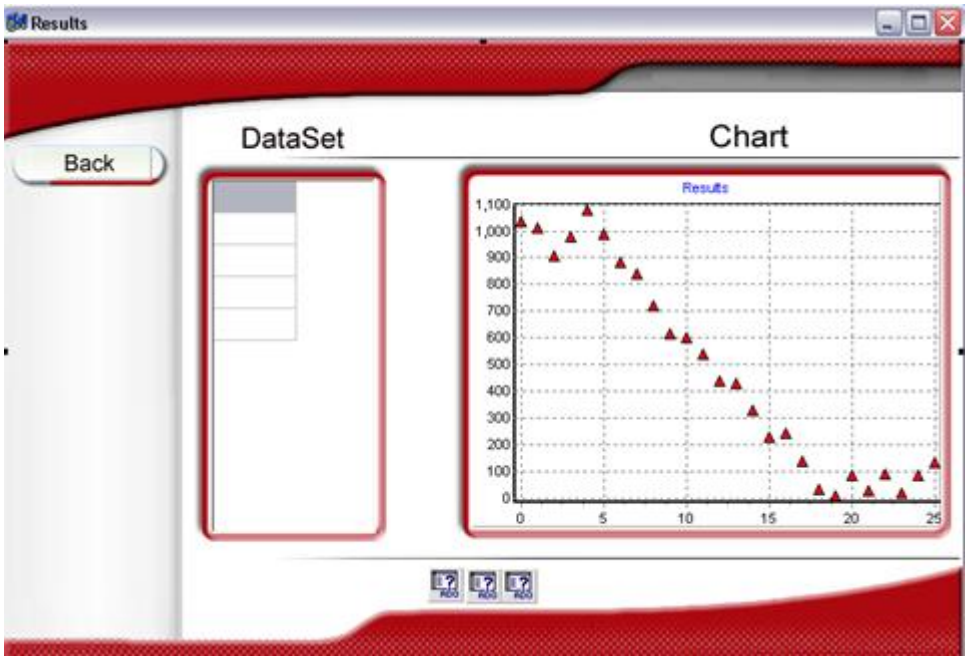


Εδώ παρουσιάζονται τα στοιχεία που επεξεργάζεται ο Simulated Annealing. Αριστερά σε ένα StringGrid παρουσιάζονται τα στοιχεία που επεξεργάστηκαν και δεξιά τα ίδια σε γραφική αναπαράσταση.

Κώδικας Φόρμας Απεικόνισης Στοιχείων του παραδείγματος Simulated Annealing

```
#include <vcl.h>
#pragma hdrstop
#include "Unit6.h"
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit8.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TSADataLook *SADataLook;
//---------------------------------------------------------------------------
__fastcall TSADataLook::TSADataLook(TComponent* Owner)
        : TForm(Owner)
{}

void __fastcall TSADataLook::GoBackClick(TObject *Sender)
{
ADOQuery1->Active=false;
ADOConnection1->Connected=false;
SADataLook->Close();
}
```

7.9 Επεξήγηση Φόρμας Απεικόνισης Χαρακτηριστικών μετά από την βελτιστοποίηση

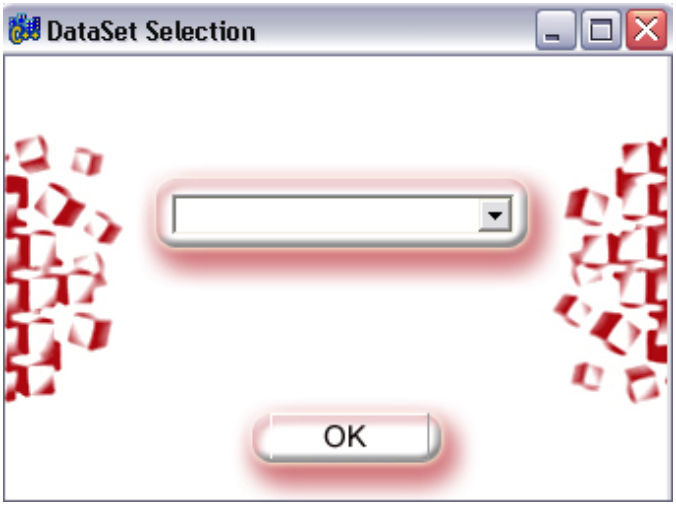


Σε αυτή την φόρμα παρουσιάζονται τα χαρακτηριστικά που επιλέχθηκαν από την διαδικασία της βελτιστοποίησης. Σε περίπτωση που ο χρήστης έχει επιλέξει περισσότερες από δυο διαστάσεις τότε απεικονίζονται μόνο οι πρώτες δύο.

Κώδικας Φόρμας Απεικόνισης Χαρακτηριστικών μετά από την βελτιστοποίηση

```
#include <vcl.h>
#pragma hdrstop
#include "Unit9.h"
#include "Unit1.h"
#include "Unit8.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit7.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TResults *Results;
//---------------------------------------------------------------------------
__fastcall TResults::TResults(TComponent* Owner)
        : TForm(Owner)
{}

void __fastcall TResults::SpeedButton1Click(TObject *Sender)
```

7.10 Επεξήγηση Φόρμας Επιλογής Πίνακα από βάση δεδομένων

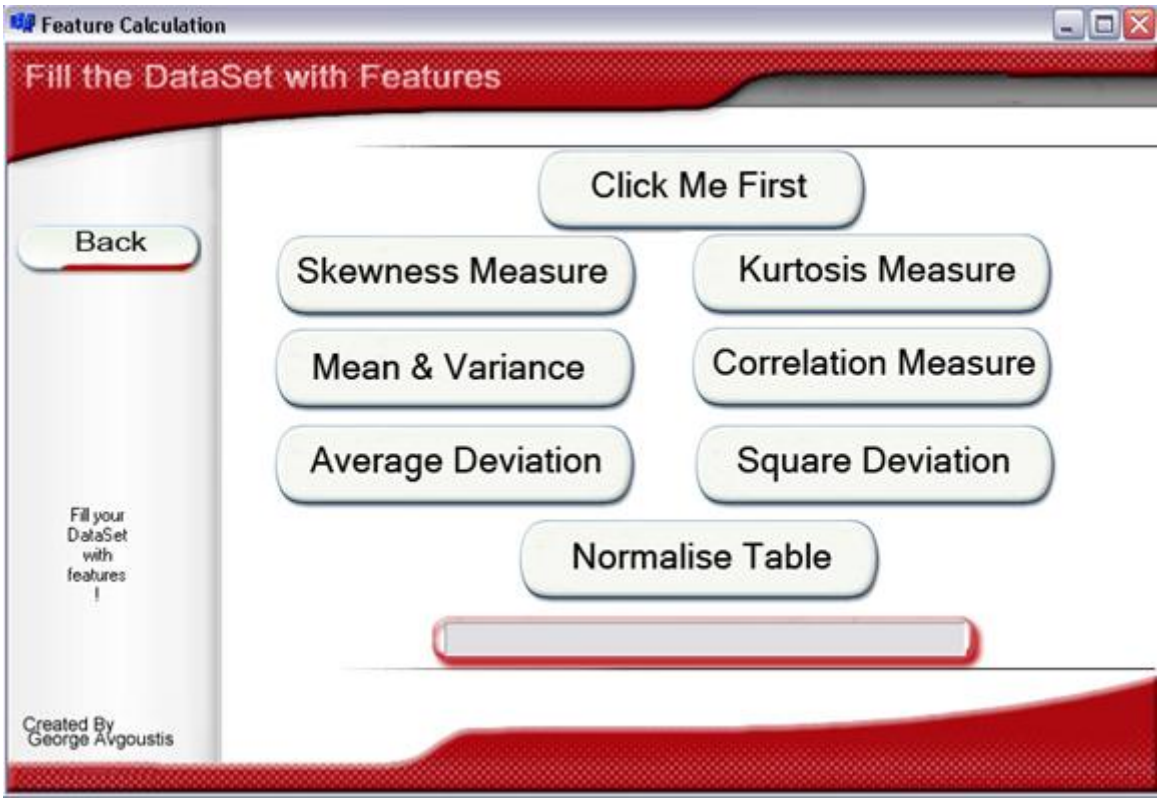


Αποτελεί μια από τις φόρμες η οποία χρησιμοποιείται για την επιλογή πινάκων. Είναι πολύ σημαντική η λειτουργία της καθώς η φόρμα όπου λαμβάνει χώρα η διαδικασία της βελτιστοποίησης χρησιμοποιεί το περιεχόμενο του EditBox, το όνομα δηλαδή του πίνακα. Χωρίς αυτό η διαδικασία δεν μπορεί να αρχίσει.

Κώδικας Φόρμας Επιλογής Πίνακα από βάση δεδομένων

```
#include <vcl.h>
#pragma hdrstop
#include "Unit3.h"
#include "Unit1.h"
#include "Unit2.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit8.h"
#include "Unit10.h"
#include "Unit7.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDataSetSelect *DataSetSelect;
//---------------------------------------------------------------------------
__fastcall TDataSetSelect::TDataSetSelect(TComponent* Owner)
        : TForm(Owner)
{
}
void __fastcall TDataSetSelect::ConfirmSelectionClick(TObject *Sender)
{
// Just Select a DataSet
if (ComboBox1->Text=="")
  MessageBox(NULL,"Please select valid table","Error",MB_OK);
else
{   DataSetSelect->Close(); // Will close only if a DataSet is chosen
}
}
```

7.11 Επεξήγηση Φόρμας Εισαγωγής Γνωρισμάτων σε κενό πίνακα



Σε αυτή την φόρμα υπολογίζονται χαρακτηριστικά τα οποία υπολογίζονται σε περίπτωση που ο χρήστης δεν έχει κάποιον πίνακα με χαρακτηριστικά έτοιμο. Αρχικά θα κληθεί να ενεργοποιήσει το event του κουμπιού ClickMeFirst έτσι ώστε να δημιουργηθούν οι αναγκαίοι πίνακες που θα χρησιμοποιηθούν αργότερα από τις συναρτήσεις υπολογισμού των χαρακτηριστικών. Έπειτα ο χρήστης μπορεί να επιλέξει με οποιαδήποτε σειρά αυτός επιθυμεί να γεμίσει τον πίνακα με χαρακτηριστικά. Τα μέτρα που δεν θα υπολογισθούν σημαίνει πως η συγκεκριμένη στήλη στον πίνακα δεν θα έχει τιμές. Η διαδικασία για την ολοκλήρωση των υπολογισμών είναι αρκετά χρονοβόρα. Αυτό βέβαια εξαρτάται από το μέγεθος της βάσης με χρονοσειρές που χρησιμοποιείται. Στην περίπτωσή μας χρησιμοποιήθηκε μια βάση με 150 χρονοσειρές με 1000 στοιχεία η κάθε μία εξ' αυτών. Τα μέτρα που υπολογίζονται είναι τα παρακάτω:

- Skewness
- Kurtosis
- Mean (μέση τιμή)
- Variance (διακύμανση)
- Correlation (συσχέτιση με κ=1 μέχρι κ=34)
- Average Deviation (μέσος όρος των απολύτων των αποκλίσεων)

- Square Deviation (τετράγωνο της τυπικής απόκλισης)

$$S = \frac{\sum_{i=1}^{N} (x_i - \mu)^3}{n\sigma^3} \qquad S = \frac{\sum_{i=1}^{N} (x_i - \mu)^4}{n\sigma^4} - 3$$

SkewNess Kurtosis

$$S = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Mean

$$r(k) = \frac{\frac{1}{n-k}\sum_{i=k+1}^{n}(x_i x_{i-k} - \bar{x}^{-2})}{\frac{1}{n}\sum_{1}^{N}(x^2{}_{i} - \bar{x}^{-2})} \qquad Q_h = n\sum_{k=1}^{h} r_k^2$$

Corelation

Κώδικας Φόρμας Εισαγωγής Γνωρισμάτων σε κενό πίνακα

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Math.hpp"
#include "Unit11.h"
#include "Unit1.h"
#include "Unit12.h"
#include "Unit8.h"
#include "Unit4.h"
#include "Unit3.h"
#include "Unit10.h"
#include "Unit2.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
#include <math.h>
#include <stdlib.h>
TFeatureCalc *FeatureCalc;

double SquareDeviationMeasure(int column, int recordsize,TObject *Sender)
{
```

```
// Calculate the Square Deviation Measure
//initialize variables
double *timeserie=new double[recordsize]  ;
double SQDev=0;
double meanvalue;
int i,j;
//fill the table with the values of the timeserie from the database
//in order to calculate the square deviation measure for the specific timeserie
for(i=0;i<recordsize;i++)
  {
  timeserie[i]=FeatureCalc->ADOQuery1->Fields->Fields[column]->Value;

  FeatureCalc->ADOQuery1->Next();
  }

  meanvalue=double(Mean(timeserie,recordsize-1));

// Calculate the square deviation Measure
for( j=0;j<recordsize;j++)
   {
SQDev= SQDev + pow((timeserie[j]-meanvalue),2);
   }
delete [] timeserie;
//return the result
return SQDev;
}

double DeviationMeasure(int column, int recordsize,TObject *Sender)
{
// Calculate the Deviation Measure
//initialize variables
double *timeserie=new double[recordsize]  ;
double Dev=0;
double meanvalue;
int i,j;
//fill the table with the values of the timeserie from the database
//in order to calculate the deviation measure for the specific timeserie
for(i=0;i<recordsize;i++)
  {
  timeserie[i]=FeatureCalc->ADOQuery1->Fields->Fields[column]->Value;

  FeatureCalc->ADOQuery1->Next();
  }

  meanvalue=double(Mean(timeserie,recordsize-1));
  //ShowMessage("meanvalue is");
 //  ShowMessage(meanvalue);

// Calculate the deviation Measure
for( j=0;j<recordsize;j++)
   {
Dev= Dev + fabs(timeserie[j]-meanvalue);
   }
delete [] timeserie;
//return the result
```

```
return Dev;
}

double SkewnessMeasure(int column, int recordsize,TObject *Sender)
{
// Calculate the Skewness Measure
//initialize variables
double *timeserie=new double[recordsize]  ;
double Skew=0;
double meanvalue, sdev;
int i,j;
//fill the table with the values of the timeserie from the database
//in order to calculate the skewness measure for the specific timeserie
for(i=0;i<recordsize;i++)
   {
   timeserie[i]=FeatureCalc->ADOQuery1->Fields->Fields[column]->Value;
   FeatureCalc->ADOQuery1->Next();
   }
   meanvalue=double(Mean(timeserie,recordsize-1));
   sdev=double(StdDev(timeserie,recordsize-1));

// Calculate the Skewness Measure
for( j=0;j<recordsize;j++)
    {
Skew= Skew + (pow((timeserie[j]-meanvalue),3)/(recordsize*pow(sdev,3)));
    }
delete [] timeserie;
//return the result
return Skew;
}

double KurtosisMeasure(int column, int recordsize,TObject *Sender)
{
// this function calculates the kurtosis measure of one timeserie at a time
//initialize variables
double *timeserie=new double[recordsize]  ;
double Kurt=0;
double meanvalue, sdev;
int i,j;
// fill the table with the values of the timeserie-one every time
for(i=0;i<recordsize;i++)
   {
   timeserie[i]=FeatureCalc->ADOQuery1->Fields->Fields[column]->Value;
   FeatureCalc->ADOQuery1->Next();
   }
   meanvalue=double(Mean(timeserie,recordsize-1));
   sdev=double(StdDev(timeserie,recordsize-1));

//calculate the kurtosis measure
for( j=0;j<recordsize;j++)
    {
Kurt= Kurt + (pow((timeserie[j]-meanvalue),4)/(recordsize*pow(sdev,4)));
    }
delete [] timeserie;
Kurt=Kurt-3;
```

```
//return the result
return Kurt;
}

double *CorrelationMeasure(double *data,int recordsize,int k )
{
//this function calcultes the corelation between the elements of a timeserie
//initialization of variables
double res1=0,res2=0,finalresult=0;
double mean=Mean(data,recordsize-1);
//the size of the result table is depended on the k factor of corelation
//no need to create a table with the values of a timeserie because it is
// taken through the function arguments
double *result=new double[k];
//calculation of the corelation
for(int j=1;j<=k;j++)
{
 for(int i=0;i<recordsize-j;i++)
  {
  res1+=(data[i]*data[i+j])-pow(mean,2);
  }
 res1=res1/(recordsize-j);

 for(int i=0;i<recordsize;i++)
  {
  res2+=pow(data[i],2)-pow(mean,2);
  }
 res2=res2/recordsize;

finalresult=res1/res2;
result[j-1]=finalresult;
res1=0;res2=0;finalresult=0;
}
//return the result table in order to calculate Q
return result;

}
//---------------------------------------------------------------------------
__fastcall TFeatureCalc::TFeatureCalc(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton5Click(TObject *Sender)
{
FeatureCalc->Close();
}
//---------------------------------------------------------------------------


void __fastcall TFeatureCalc::SpeedButton1Click(TObject *Sender)
{
// this button calcultes the skewness measure by making use of the according
//function above.
```

```
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,j,recsize,fieldcount;
double Skewness;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;

ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;

//calculate the skewness measure for every timeserie in the database and update
//GeorgeFeatures Table
for(i=1;i<fieldcount;i++)
  {
  FeatureCalc->Refresh();
  ProgressBar1->StepIt();
  FeatureCalc->ADOQuery1->Open();
  name=FeatureCalc->ADOQuery1->Fields->Fields[i]->FieldName;
  Skewness=SkewnessMeasure(i, recsize,FeatureCalc->ADOQuery1);
  sql="update GeorgeFeatures set Skew=";
  sql+=Skewness;
  sql+=" where cod=";
  sql+=i;

  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  FeatureCalc->ADOQuery1->Close();

  }


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

void __fastcall TFeatureCalc::SpeedButton2Click(TObject *Sender)
{
// this button calcultes the kurtosis measure by making use of the according
//function above.
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,j,recsize,fieldcount;
double Kurtosis;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
```

```
//calculate the kurtosis measure for every timeserie in the database and update
//GeorgeFeatures Table
for(i=1;i<fieldcount;i++)
   {
   FeatureCalc->Refresh();
   ProgressBar1->StepIt();
   ADOQuery1->Open();
   name=ADOQuery1->Fields->Fields[i]->FieldName;
   Kurtosis=KurtosisMeasure(i, recsize,ADOQuery1);
   sql="update GeorgeFeatures set Kurt=";
   sql+=Kurtosis;
   sql+=" where cod=";
   sql+=i;
   ADOCommand1->CommandText=sql;
   ADOCommand1->Execute();
   ADOQuery1->Close();

   }


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



void __fastcall TFeatureCalc::SpeedButton1MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to calculate the skewness!";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton2MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to calculate the Kurtosis!";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton3MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to calculate the correlation!";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton4MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Click Here to calculate other measures!";
}
void __fastcall TFeatureCalc::Image1MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Fill your DataSet with features !";
```

```
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton5MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Leave";
}
//---------------------------------------------------------------------------



void __fastcall TFeatureCalc::FormCreate(TObject *Sender)
{
// this makes the user to click only one button.
SpeedButton1->Enabled=false;
SpeedButton2->Enabled=false;
SpeedButton3->Enabled=false;
SpeedButton6->Enabled=false;
SpeedButton7->Enabled=false;
SpeedButton8->Enabled=false;
SpeedButton9->Enabled=true;



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

void __fastcall TFeatureCalc::SpeedButton4Click(TObject *Sender)
{
// User must click here at the beggining in order to create two tables.
//GeorgeLink and GeogeFeatures. The first one is used to relate the names of the
//timseries with numbers and the second to fill with the measures.
//check if theu exist and act as ordered
 if(DataSetSelect->ComboBox1->Items->IndexOf("GeorgeLink")==-1)
  {
   ShowMessage("pame gia create");
  FeatureCalc->ADOCommand1->CommandText="create table GEORGELINK (cod int not null
primary key , TIMESERIENAME varchar(60) )";
  FeatureCalc->ADOCommand1->Execute();
  DataSetSelect->ComboBox1->Items->Add("GEORGELINK");
  ShowMessage("ok to create");
  }
else
  {
  FeatureCalc->ADOCommand1->CommandText="delete from GEORGELINK";
  FeatureCalc->ADOCommand1->Execute();
  ShowMessage("ok delete from georgelink");
  }


if(DataSetSelect->ComboBox1->Items->IndexOf("GEORGEFEATURES")==-1)
{
FeatureCalc->ADOCommand1->CommandText="create table GEORGEFEATURES (cod int not
null primary key ,\
```

```
        Kurt float , Skew float , M1 float , M2 float ,\
        AVDEV float , SQDEV float )";
FeatureCalc->ADOCommand1->Execute();
AnsiString sql;
for(int i=1;i<35;i++)
  {
  sql="";
  if(DataBaseSelect->SelectList->Text=="SQL Server" || DataBaseSelect->SelectList-
>Text=="Microsoft Access" )
  {
  sql="ALTER TABLE GEORGEFEATURES ADD CORRELATION_";
  }
  else
  {
  sql="ALTER TABLE ";
  sql+=OracleBase->Edit2->Text;
  sql+=".GEORGEFEATURES ADD CORRELATION_";
  }
  sql+=i;
  sql+=" float ";
  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  }
DataSetSelect->ComboBox1->Items->Add("GEORGEFEATURES");
}
else
{
FeatureCalc->ADOCommand1->CommandText="delete from GEORGEFEATURES";
FeatureCalc->ADOCommand1->Execute();
}

//initialize tables----------------------------------------------------------
// fill george features with the cod only
//use the table name selected from the DataSetSelect form
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,fieldcount;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;
FeatureCalc->ADOQuery1->SQL->Text=sqlsel;
FeatureCalc->ADOQuery1->Open();

fieldcount=FeatureCalc->ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
for(i=1;i<fieldcount;i++)
  {
 FeatureCalc->Repaint();
  ProgressBar1->StepIt();
  FeatureCalc->ADOQuery1->Open();
  name=FeatureCalc->ADOQuery1->Fields->Fields[i]->FieldName;
  sql="insert into GEORGEFEATURES ( cod ) Values ( ";
  sql+=i;
  sql+=" )";
  FeatureCalc->ADOCommand1->CommandText=sql;
```

```
  FeatureCalc->ADOCommand1->Execute();
  sqllink="";
  sqllink="Insert into GEORGELINK ( cod , TIMESERIENAME ) VALUES (";
  sqllink+=i;
  sqllink+=" , '";
  sqllink+=name;
  sqllink+="' )";
  FeatureCalc->ADOCommand1->CommandText=sqllink;
  FeatureCalc->ADOCommand1->Execute();
  FeatureCalc->ADOQuery1->Close();
  }

SpeedButton1->Enabled=true;
SpeedButton2->Enabled=true;
SpeedButton3->Enabled=true;
SpeedButton6->Enabled=true;
SpeedButton7->Enabled=true;
SpeedButton8->Enabled=true;
SpeedButton9->Enabled=true;

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



void __fastcall TFeatureCalc::SpeedButton6MouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Calculate the first 2 moments and put them in the according field";
}
//---------------------------------------------------------------------------



void __fastcall TFeatureCalc::SpeedButton6Click(TObject *Sender)
{
//by clicking this button the first 2 moments of the timeserie are calculated
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,recsize,fieldcount;
double m1,m2,m3,m4,sdev,meanvalue;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
//for every timeserie calculate the moments of the timeserie
//and update georgefeatures table.
for(i=1;i<fieldcount;i++)
  {
  FeatureCalc->Repaint();
  ProgressBar1->StepIt();
```

```
  FeatureCalc->ADOQuery1->Open();
  name=FeatureCalc->ADOQuery1->Fields->Fields[i]->FieldName;
  double *timeserie=new double[recsize];

    for(int t=0;t<recsize;t++)
      {
      timeserie[t]=FeatureCalc->ADOQuery1->Fields->Fields[i]->Value;
      FeatureCalc->ADOQuery1->Next();
      }
    sdev=double(StdDev(timeserie,recsize-1));
    meanvalue=double(Mean(timeserie,recsize-1));
    m1=meanvalue;
    m2=pow(sdev,2);
   sql="update GeorgeFeatures set M1=";
  sql+=m1;
  sql+=",M2=";
  sql+=m2;
  sql+=" where cod=";
  sql+=i;
  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  FeatureCalc->ADOQuery1->Close();
  delete [] timeserie;
  }
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton3Click(TObject *Sender)
{
//calculate the corelation between the elements of the timeserie for every
//k factor, find Q and pdate george features table.
AnsiString sql,sqlsel,datasetname,firstfield;
int i,j,recsize,fieldcount,repeat;
//k factor has values from 1 to 34
int kmax=35;
double *result;
double Q=0;

datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
firstfield=ADOQuery1->Fields->Fields[0]->FieldName;
ADOQuery1->Close();
sqlsel="select * from ";
sqlsel+=datasetname;
sqlsel+="  order by ";
sqlsel+=" ";
sqlsel+=firstfield;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ADOQuery1->Close();
```

```
ProgressBar1->Max=kmax;
ProgressBar1->Step=1;
//for every k factor calculate the corelation
for(i=1;i<kmax;i++)
{
 ADOQuery1->Open();
  ProgressBar1->StepIt();
   // for every timeserie as well
   for(repeat=1;repeat<fieldcount;repeat++)
      {
            ADOQuery1->Open();

      double *timeserie =new double[recsize];
   // fill the table with the values of the timeserie from the database
    for(j=0;j<recsize;j++)
        {
         timeserie[j]=FeatureCalc->ADOQuery1->Fields->Fields[repeat]->Value;
         FeatureCalc->ADOQuery1->Next();
         }
   result=CorrelationMeasure(timeserie,recsize,i);

  //Find Q
  for(int r=0;r<i;r++)
   {
   Q+=pow(result[r],2);
   }
  Q*=recsize;
 // update georgefeatures with the value calculated
  sql="update GEORGEFEATURES set CORRELATION_";
  sql+=i;
  sql+="=";
  sql+=Q;
  sql+=" where cod=";
  sql+=repeat;
  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();

      delete [] timeserie;
      Q=0;
      ADOQuery1->Close();
      }
  Q=0;
  ADOQuery1->Close();
}

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

void __fastcall TFeatureCalc::SpeedButton7Click(TObject *Sender)
{
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,j,recsize,fieldcount;
double deviation,avdeviation;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
```

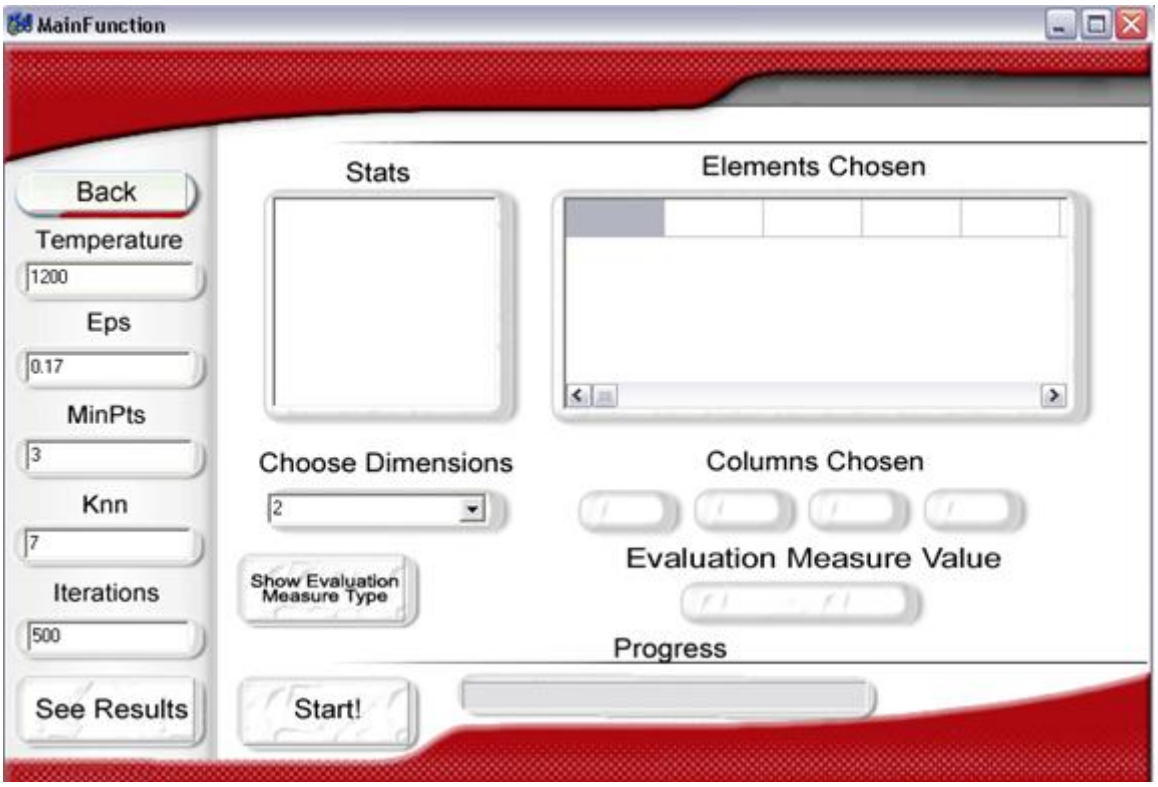
```
sqlsel+=datasetname;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
ADOQuery1->Close();
//calculate the Average Deviation measure for every timeserie in the database and update
//GeorgeFeatures Table
for(i=1;i<fieldcount;i++)
  {
  FeatureCalc->Refresh();
  ProgressBar1->StepIt();
  FeatureCalc->ADOQuery1->Open();
  //hold the name of the timeserie whose elements are being calculated
  name=FeatureCalc->ADOQuery1->Fields->Fields[i]->FieldName;
  deviation=DeviationMeasure(i, recsize,FeatureCalc->ADOQuery1);
  avdeviation=deviation/recsize;
  sql="update GeorgeFeatures set AVDEV=";
  sql+=avdeviation;
  sql+=" where cod=";
  sql+=i;

  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  FeatureCalc->ADOQuery1->Close();

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

void __fastcall TFeatureCalc::SpeedButton7MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Calculate the average deviation";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton8MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Calculate the square deviation";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton9MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
ShowHint->Caption="Normalize your table with features!";
}
//---------------------------------------------------------------------------

void __fastcall TFeatureCalc::SpeedButton8Click(TObject *Sender)
{
```

```
AnsiString sql,sqlsel,sqllink,name,datasetname;
int i,j,recsize,fieldcount;
double sqdeviation;
datasetname=DataSetSelect->ComboBox1->Text;
sqlsel="select * from ";
sqlsel+=datasetname;
ADOQuery1->SQL->Text=sqlsel;
ADOQuery1->Open();
recsize=ADOQuery1->RecordCount;
fieldcount=ADOQuery1->FieldCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
ADOQuery1->Close();
//calculate the square Deviation measure for every timeserie in the database
//and update GeorgeFeatures Table
for(i=1;i<fieldcount;i++)
  {
  FeatureCalc->Refresh();
  ProgressBar1->StepIt();
  FeatureCalc->ADOQuery1->Open();
  name=FeatureCalc->ADOQuery1->Fields->Fields[i]->FieldName;
  sqdeviation=SquareDeviationMeasure(i, recsize,FeatureCalc->ADOQuery1);
  sql="update GeorgeFeatures set SQDEV=";
  sql+=sqdeviation;
  sql+=" where cod=";
  sql+=i;

  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  FeatureCalc->ADOQuery1->Close();

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

void __fastcall TFeatureCalc::SpeedButton9Click(TObject *Sender)
{
//create new table with the normalized values
//first do a check to see if it already exists
//if not create
if(DataSetSelect->ComboBox1->Items->IndexOf("GEORGEFEATURESNORM")==-1)
{
FeatureCalc->ADOCommand1->CommandText="create table GEORGEFEATURESNORM (cod
int not null primary key ,\
        Kurt float , Skew float , M1 float , M2 float ,\
        AVDEV float , SQDEV float )";
FeatureCalc->ADOCommand1->Execute();
AnsiString sql;
for(int i=1;i<35;i++)
  {
  sql="";
  if(DataBaseSelect->SelectList->Text=="SQL Server" || DataBaseSelect->SelectList-
>Text=="Microsoft Access" )
  {
  sql="ALTER TABLE GEORGEFEATURESNORM ADD CORRELATION_";
```

```
  }
  else
  {
  sql="ALTER TABLE ";
  sql+=OracleBase->Edit2->Text;
  sql+=".GEORGEFEATURESNORM ADD CORRELATION_";
  }
  sql+=i;
  sql+=" float ";
  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  }
DataSetSelect->ComboBox1->Items->Add("GEORGEFEATURESNORM");
}
//if exists then delete the records from it
else
{
FeatureCalc->ADOCommand1->CommandText="delete from GEORGEFEATURESNORM";
FeatureCalc->ADOCommand1->Execute();
}
ShowMessage("ok kai to add corellation_x");
//initialize tables-----------------------------------------------------------
// fill george features with the cod only
AnsiString sql,sqlsel,name;
int i,fieldcount,recordcount;
//build this string to gather info about the records anf fields of the table
sqlsel="select * from ";
sqlsel+="GEORGEFEATURES";
FeatureCalc->ADOQuery1->SQL->Text=sqlsel;
FeatureCalc->ADOQuery1->Open();
fieldcount=FeatureCalc->ADOQuery1->FieldCount;
recordcount=FeatureCalc->ADOQuery1->RecordCount;
ProgressBar1->Max=fieldcount;
ProgressBar1->Step=1;
//do only the insertion of cod into the table where the normalized values
//are to be held
for(i=1;i<=recordcount;i++)
  {
  FeatureCalc->Repaint();
  ProgressBar1->StepIt();
  FeatureCalc->ADOQuery1->Open();
  sql="insert into GEORGEFEATURESNORM ( cod ) Values ( ";
  sql+=i;
  sql+=" )";
  FeatureCalc->ADOCommand1->CommandText=sql;
  FeatureCalc->ADOCommand1->Execute();
  }

FeatureCalc->ADOQuery1->Close();
//renew the variables recordcount and fieldcount
ADOQuery1->SQL->Text="select * from GEORGEFEATURES ";
ADOQuery1->Open();
fieldcount=ADOQuery1->FieldCount;
recordcount=ADOQuery1->RecordCount;
AnsiString sql3,sql4,fieldname;
```

```
double normnum,min,max;
int j;

//this is the big iteration for the calculation of the normalized values
//for every column ,that means feature column, do
for( j=1;j<fieldcount;j++)
{
//select the maximum and minimum for every column in order to calculate the
//normalized values
ADOQuery1->Open();
fieldname=ADOQuery1->Fields->Fields[j]->FieldName;
sql3="select max(";
sql3+=fieldname;
sql3+="),min(";
sql3+=fieldname;
sql3+=") from GEORGEFEATURES";
ADOQuery3->SQL->Text=sql3;
ADOQuery3->Open();
ADOQuery1->Open();
//declare new table which will be deleted and re declared each time
double *fieldvalues=new double[recordcount];
//fill the table with the values from the database
for(int r=0;r<recordcount;r++)
    {
    fieldvalues[r]=ADOQuery1->Fields->Fields[j]->Value;
    ADOQuery1->Next();
    }
  ADOQuery1->Close();
//in this iteration the normalized value is being calculated and then
//inserted into the table in the database
for(int t=1;t<=recordcount;t++)
    {
    ADOQuery3->Open();
    max=ADOQuery3->Fields->Fields[0]->Value;
    min=ADOQuery3->Fields->Fields[1]->Value;
    normnum=double((fieldvalues[t-1]-min)/(max-min));
    sql4="update GEORGEFEATURESNORM set ";
    sql4+=fieldname;
    sql4+="=" ;
    sql4+=normnum;
    sql4+=" where cod=";
    sql4+=t;

    ADOCommand1->CommandText=sql4;
    ADOCommand1->Execute();
    ADOQuery3->Close();
    }
delete [] fieldvalues;
}
}
```

7.12 Επεξήγηση Φόρμας Βασικής Διαδικασίας της Βελτιστοποίησης



Πρόκειται για την φόρμα στην οποία γίνεται η όλη επεξεργασία. Εδώ χρησιμοποιούνται όλες οι πληροφορίες για να αξιολογηθούν και με βάση τους υπολογισμούς να επιλεχθεί ο κατάλληλος αριθμός στηλών με χαρακτηριστικά γνωρίσματα. Σε αυτή την φόρμα χρησιμοποιούνται 21 συναρτήσεις που επικοινωνούν σχεδόν όλες μεταξύ τους δεχόμενες ορίσματα και επιστρέφοντας αποτελέσματα χρήσιμα σε άλλες για την διαδικασία της βελτιστοποίησης. Ανοίγοντας αυτήν την φόρμα ο χρήστης έχει επιλέξει τον πίνακα με τα χαρακτηριστικά που προορίζεται για επεξεργασία. Σε περίπτωση που δεν έχει επιλεχθεί κάποιος πίνακας εμφανίζεται και το ανάλογο μήνυμα προειδοποίησης. Αριστερά βρίσκονται τα EditBoxes τα οποία δέχονται την εισαγωγή παραμέτρων από τον χρήστη. Η μεταβλητή Temperature χρησιμοποιείται από τον Simulated Annealing. Η χρήση της εξηγείται σε παραπάνω κεφάλαιο. Οι επόμενες τρεις είναι χρήσιμές για την διαδικασία του Clustering, δηλαδή της συσταδοποίησης. Η μεταβλητή Eps είναι η ακτίνα μέσα στην οποία θα ψάξει ο αλγόριθμος να βρει «συγγενείς». Η μεταβλητή MinPts είναι ο ελάχιστος αριθμός στοιχείων που απαρτίζουν μία συστάδα. Μαζί με την μεταβλητή Knn εξηγούνται καλύτερα σε παραπάνω κεφάλαιο. Τέλος, μπορούμε να καθορίσουμε και τον αριθμό των επαναλήψεων στο EditBox με ονομασία Iterations. Μία ακόμα επιλογή που είναι στο χέρι του χρήστη είναι ο αριθμός των στηλών που θέλει να επιλεχτεί. Αυτός ο αριθμός είναι μεταξύ του 2 και του 4. Στις τρεις και τέσσερις διαστάσεις δεν είναι δυνατή η απεικόνιση σε διάγραμμα οπότε μόνο οι δύο είναι δυνατόν να παρουσιαστούν.

Στον πίνακα με όνομα Stats ο χρήστης μπορεί να παρακολουθήσει την διαδικασία αναγνώρισης των στοιχείων από τον DBSCAN ως σημεία πυρήνες ή συνοριακά σημεία. Δεξιά, παρουσιάζονται τα αποτελέσματα σε ένα StringGrid. Παρουσιάζεται ουσιαστικά ο πίνακας με τα αποτελέσματα, οποίος εγγράφεται και στην βάση δεδομένων του χρήστη στον επιλεγμένο παροχέα. Κάτω από τον πίνακα των αποτελεσμάτων υπάρχουν τέσσερα μικρά κουτιά στα οποία εμφανίζεται ο αριθμός των στηλών που επιλέχθηκε. Εάν είναι λιγότερες από τέσσερις οι στήλες τότε στα υπόλοιπα κουτιά εμφανίζεται ο αριθμός μηδέν. Τέλος κάτω από τις επιλεγμένες στήλες εμφανίζεται και η τιμή του μέτρου αξιολόγησης. Ο τύπος του μέτρου αξιολόγησης εμφανίζεται σε μια φόρμα σε περίπτωση που το επιλέξει ο χρήστης. Το μέτρο αξιολόγησης εξηγείται και σε παραπάνω κεφάλαια.

7.12.1 Σύντομη περιγραφή λειτουργίας βελτιστοποίησης

Για τις ανάγκες της λειτουργίας της βελτιστοποίησης δημιουργούνται στην αρχή δύο πίνακες. Στον πρώτο με την ονομασία **results** εκχωρούνται οι τιμές των επιλεγμένων στηλών, το όνομα της κλάσης που ανήκουν, ο τύπος τους (Border, Core ή Noise), η αρίθμησή τους και το όνομα τους. Ο άλλος πίνακας **maskdata** κρατά τις τιμές της κάθε επανάληψης, την μάσκα που έχει χρησιμοποιηθεί και το μέτρο αξιολόγησης που αφορά στην συγκεκριμένη μάσκα. Πρόκειται για πίνακες που χρησιμοποιούνται καθ' όλη ην διάρκεια της βελτιστοποίησης.

Βασική Περιγραφή

Το πρόγραμμα χρησιμοποιεί έναν πίνακα σειρά ,που στην ουσία είναι σαν μια μάσκα, στον οποίο κρατά τις στήλες που θα χρησιμοποιηθούν από τον DBSCAN, ο οποίος αρχικοποιείται στην αρχή της εκτέλεσης του προγράμματος. Είναι ένας πίνακας που έχει όσες στήλες και ο πίνακας με τα γνωρίσματα. Παίρνει δύο τιμές. Μηδέν για τις στήλες που δεν χρησιμοποιούνται και ένα για αυτές που θα χρησιμοποιηθούν. Κάθε φορά ο Simulated Annealing χρησιμοποιεί την λειτουργία του Reversal για να αλλάξει την θέση των άσσων. Ο DBSCAN εφαρμόζει την λειτουργία της συσταδοποίησης με τις επιλεγμένες στήλες και μετά αναλαμβάνει το μέτρο αξιολόγησης, το οποίο καλείται να πάρει σαν όρισμα τον αριθμό των συστάδων που έχουν βρεθεί από πριν και να βρει την δια-κλασσική απόσταση. Έπειτα γίνεται μία σύγκριση για να διαπιστωθεί εάν η τρέχουσα αξιολόγηση με τις συγκεκριμένες στήλες είναι καλύτερη από την προηγούμενη καλύτερη. Αυτή την απόφαση καλείται να λάβει ο αλγόριθμος **metropolis**. Εάν η τρέχουσα αξιολόγηση είναι καλύτερη τότε εκχωρείται η καινούρια τιμή στην μεταβλητή Jold. Εάν η τρέχουσα αξιολόγηση δεν είναι καλύτερη υπάρχει πιθανότητα να κρατηθεί και πάλι με πιθανότητα $e^{-(-Jnew+Jold)/t}$. Εάν δεν γίνει κάποια αλλαγή τότε αρχίζουμε και πάλι από την αρχή καλώντας τον Simulated Annealing για να κάνει reversal. Κάθε φορά που γίνεται η λειτουργία του reversal γίνεται ένας έλεγχος για να διαπιστωθεί εάν έχει ξαναχρησιμοποιηθεί η μάσκα που προέκυψε. Εάν ναι τότε ξανά-γίνεται το reversal, αλλιώς συνεχίζει η διαδικασία της βελτιστοποίησης.

Κώδικας Φόρμας Βασικής Διαδικασίας Βελτιστοποίησης

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

#include <vcl.h>
#pragma hdrstop
#include "Unit8.h"
#include "Unit1.h"
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <functional>
#include <math.h>
#include <iostream.h>
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit9.h"
#include "Unit12.h"
#define MASK 123459876
#define TFACTR 0.9
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainFunction *MainFunction;
AnsiString results[4][500];
// create structures used for clustering proccess
// struct used for dataset used by DBSCAN
struct InfArray
     {
     int cod;
     int point;
     double x;
     double y;
     double z;
     double r;
     vector<int> PointsDDR;
     AnsiString PointType;
     int ClusterID;
     double KNearestDist;
    };
// the following are used by DBSCAN as well
struct KnearestInf
      {
      int Point;
      double Dist;
```

```
    };

struct ClusterArrayInf
    {
    vector<int> Clusters;
    int State;
    };


//-----------------------RECORD SIZE-----------------------------------------
 int RecordSize(TObject *Sender, AnsiString Name)
{
//this function returns the size of the Dataset
int recsize  ;
// sql string construction
AnsiString sql="select * from ";
sql+=Name;
 MainFunction->ADOQuery1->SQL->Text=sql;
   MainFunction->ADOQuery1->Open();
   recsize = MainFunction->ADOQuery1->RecordCount;
   MainFunction->ADOQuery1->Close();
   return recsize;
}
//---------------------FIELD COUNT-----------------------------------------

int FieldCount(TObject *Sender,AnsiString Name)
{
// this function returns the number of fields of the DataSet
AnsiString sql;
// sql string construction
sql="select * from ";
sql+=Name;
int fieldcount;
  MainFunction->ADOQuery1->SQL->Text=sql;
  MainFunction->ADOQuery1->Open();
    fieldcount=MainFunction->ADOQuery1->FieldCount-1;
      MainFunction->ADOQuery1->Close();
  return fieldcount;

}

//---------------------SORT ARRAY-----------------------------------------
void SortAuxArray(int recsize, InfArray *DataSet)
{
//this routine performs Ascendant sorting the DataSet with respect to KNearestDist
int i,j;


InfArray* temp=new InfArray[1];
for(i=0;i<recsize-1;i++)
    {
    for(j=i+1;j<recsize;j++)
       {
       //do a comparison
       if((DataSet[i].KNearestDist) > (DataSet[j].KNearestDist))
```

```
                {
                //swap places if true
                temp[0]=DataSet[i];
                DataSet[i]=DataSet[j];
                DataSet[j]=temp[0];
                }
        }


    }
delete [] temp;
}
//-------------------------GET MAX-----------------------------------------------
int GetMax(KnearestInf *Knearest, int Knn)
{
//this function returns the position of the maximum distance
// of Knearest - Knn is the size of the structure
int i;
double maxi=0.0;
int pos1=0;

maxi=Knearest[0].Dist ;
pos1=0;

for(i=1;i<Knn;i++)
    {
    //do a comparison
    if (Knearest[i].Dist > maxi)
       {
       maxi=Knearest[i].Dist;   //if true then set maxi=Knearest[i].Dist
       pos1=i;                  //save the position
       }
    }
 return pos1;
}


//-------------------------DATASET LOAD--------------------------------------
void LoadDataset(int recsize, InfArray *DataSet, int spot[],AnsiString Name)
{
// this function fills the DataSet with the chosen columns from the database
int j;
AnsiString sql;

sql="select * from ";
sql+=Name;
MainFunction->ADOQuery1->SQL->Text=sql;
MainFunction->ADOQuery1->Open();

 for(j=0;j<recsize;j++)
     {
     DataSet[j].cod=j;
     DataSet[j].point=MainFunction->ADOQuery1->Fields->Fields[0]-> Value;
     DataSet[j].x= MainFunction->ADOQuery1->Fields->Fields[spot[0]]-> Value;
     DataSet[j].y= MainFunction->ADOQuery1->Fields->Fields[spot[1]]-> Value;
```

```
    //if no third dimension then fill with nulls
    if(spot[2]!=0)
    DataSet[j].z= MainFunction->ADOQuery1->Fields->Fields[spot[2]]-> Value;
    else
    DataSet[j].z=0;
     //if no fourth dimension then fill with nulls
    if(spot[3]!=0)
    DataSet[j].r= MainFunction->ADOQuery1->Fields->Fields[spot[2]]-> Value;
    else
    DataSet[j].r=0;
    MainFunction->ADOQuery1->Next();
    }



MainFunction->ADOQuery1->Close();

}

//-----------------CALCULATE NEAREST-----------------------------------------
void CalcKNearest(int Knn,int recsize, InfArray* DataSet)
{
//this routine finds the K nearest neighbors of each point in the DataSet
//struct KnearestInf Knearest[Knn];
KnearestInf* Knearest=new KnearestInf[Knn];

int i;
int j;
int pos;
int size;
double dist;

for (i=0;i<recsize;i++)
        {
        size=0;
        for(j=0;j<recsize;j++)
          {
           dist=0;
             if(i!=j)
              {
           //calculate the distance between the selected points
           dist=sqrt(((DataSet[i].x-DataSet[j].x)*(DataSet[i].x-DataSet[j].x))+((DataSet[i].y-
DataSet[j].y)*(DataSet[i].y-DataSet[j].y))+((DataSet[i].z-DataSet[j].z)*(DataSet[i].z-
DataSet[j].z))+((DataSet[i].r-DataSet[j].r)*(DataSet[i].r-DataSet[j].r)));
                    if(size<Knn)
                     {
                     //fill the structure as long as size<Knn
                     Knearest[size].Dist=dist;
                     Knearest[size].Point=j;
                     size++;

                     }
                      else
                        {
                         //else start comparing to fill with the near ones
```

```
                    pos=GetMax(Knearest,Knn);

                        if(dist<Knearest[pos].Dist)
                          {
                          Knearest[pos].Dist=dist;
                          Knearest[pos].Point=j;
                          }
                    }
               }
                }
          //hold the position of the one that is the furthest
          pos=GetMax(Knearest, Knn);
          //find the distance  of the furthest and the current
          dist=sqrt(((DataSet[i].x-DataSet[Knearest[pos].Point].x)*(DataSet[i].x-
DataSet[Knearest[pos].Point].x))+((DataSet[i].y-DataSet[Knearest[pos].Point].y)*(DataSet[i].y-
DataSet[Knearest[pos].Point].y))+((DataSet[i].z-DataSet[Knearest[pos].Point].z)*(DataSet[i].z-
DataSet[Knearest[pos].Point].z))+((DataSet[i].r-DataSet[Knearest[pos].Point].r)*(DataSet[i].r-
DataSet[Knearest[pos].Point].r)));
          DataSet[i].KNearestDist=dist;
         //erase data in the structure
          for(int i=0;i<Knn;i++)
            {
            Knearest[i].Point=0;
            Knearest[i].Dist=0.0;
            }
        }


   SortAuxArray(recsize,DataSet);


 delete [] Knearest;
}



//-------------------------CHECK DDRPOINTS-------------------------------------

void CheckDDRPoints(double Eps, int MinPts, int recsize, InfArray *DataSet)
{
//check the direct density reachable points
int i,k,l;
int j;
int count;
double dist;
//declare a vector used for dynamic Set
vector<int> MyCollect;
for(i=0;i<recsize;i++)
{
  dist=0;
  count=0;
  //clear the collection at the beggining
        MyCollect.clear() ;              //new method...
```

```
        for(j=0;j<recsize;j++)
          {
       //find the distance and as long dist<=Eps insert into Collection
          dist=sqrt(((DataSet[i].x-DataSet[j].x)*(DataSet[i].x-DataSet[j].x))+((DataSet[i].y-
DataSet[j].y)*(DataSet[i].y-DataSet[j].y))+((DataSet[i].z-DataSet[j].z)*(DataSet[i].z-
DataSet[j].z))+((DataSet[i].r-DataSet[j].r)*(DataSet[i].r-DataSet[j].r)));
                if(dist<=Eps)   //old statement was dist<Eps
                  {
                   count++;
                   MyCollect.insert(MyCollect.end(),DataSet[j].point);
                  }

          }
     //if MinPts are exceeded the declare the current point as Core
           if(count>= MinPts)
             {
             DataSet[i].PointType="Core";
             DataSet[i].PointsDDR=MyCollect;
             }
 // or Border
  else
    {
    DataSet[i].PointType="Border";
    }
 DataSet[i].ClusterID=0;
}


}

//-------------------------GETDRPOINTS---------------------------------------

void GetDRPoints( int PointPos, int recsize, InfArray *DataSet, int ClusterID)
{
//-----to pointpos einai to point pou exei h findclusters pou exei parei apo thn epanalhpsh ths main
//create a collection
vector<int> MySet;

int i,j,K,m;
bool found;
int Member;
//get density reachable points
//fill the Set with the direct density reachable points found with the previous
//function
for(i=0;i<DataSet[PointPos].PointsDDR.size();i++)
  {
    if(DataSet[PointPos].point!= DataSet[PointPos].PointsDDR[i])
     {
     MySet.insert(MySet.end(),DataSet[PointPos].PointsDDR[i]);
     }

  }


for(i=0;i<MySet.size();i++)
```

```
  {

 for(j=0;j<recsize;j++)
     {


     if(DataSet[j].point==MySet[i]  )
        {

        for(int K=0;K<DataSet[j].PointsDDR.size();K++)
          {
           found=false;

            if(DataSet[j].PointsDDR[K]!=DataSet[PointPos].point)
            {

              for(m=0;m<MySet.size();m++)
               {

                 if(DataSet[j].PointsDDR[K]==MySet[m])
                  {
                 found=true;


                 break;
                  }
               }
             if(found==true)
              goto a;
             else

               MySet.insert(MySet.end(),DataSet[j].PointsDDR[K]);
            }

a:      }
       break;
        }
      }
  }
  DataSet[PointPos].ClusterID=ClusterID;  // Elegxos!~!
  for(i=0;i<MySet.size();i++)
    {

     for(j=0;j<recsize;j++)
     {
       if(DataSet[j].point==MySet[i])
       {
         DataSet[j].ClusterID=ClusterID;
       }

     }
    }

}
```

```
//--------------------FIND CLUSTERS---------------------------------------------

void FindClusters(int Point, int recsize,InfArray *DataSet, int ClusterID)
{
// sto point pernaei to i ths epanalhpshs ths main
int i;

for(i=0;i<recsize;i++)
 {
 GetDRPoints(Point,recsize,DataSet,ClusterID);
 }
}


//-------------------------STORE RESULTS----------------------------------------

void StoreResults(int recsize,InfArray *DataSet)
{
//store the results every time dbscan ends
int i,j;
int cod, point,ClusterID;
float x,y,z,r;
AnsiString PointType;
AnsiString sql;

//empty the previous records of results

   MainFunction->ADOCommand1->CommandText="delete from results";
   MainFunction->ADOCommand1->Prepared=true;
   MainFunction->ADOCommand1->Execute();
//construct the sql string for insertion
     for(i=0;i<recsize;i++)
       {
       sql="";
       sql="insert into results(cod,point,x,y,z,r,ClusterID,PointType) Values (" ;
       cod=DataSet[i].cod;
       point=DataSet[i].point;
       ClusterID=DataSet[i].ClusterID;
       PointType=DataSet[i].PointType;
       x=DataSet[i].x;
       y=DataSet[i].y;
       z=DataSet[i].z;
       r=DataSet[i].r;
       sql+= cod ;
       sql+=",";
       sql+= point;
       sql+=",";
       sql+= x;
       sql+=",";
       sql+= y;
       sql+=",";
       sql+= z;
```

```
        sql+=",";
        sql+= r;
        sql+=",";
        sql+= ClusterID;
        sql+=", '";
        sql+= PointType;
        sql+="' )";
        MainFunction->ADOCommand1->CommandText=sql;
        MainFunction->ADOCommand1->Execute();
        }
//open the results table to show the results to the form
MainFunction->ADOQuery1->SQL->Text="select * from results";
MainFunction->ADOQuery1->Open();
MainFunction->StringGrid1->RowCount=recsize;
for(int k=0;k<recsize;k++)
    {
    MainFunction->StringGrid1->Cells[0][k]=MainFunction->ADOQuery1->Fields->Fields[0]-
>Value;
    MainFunction->StringGrid1->Cells[1][k]=MainFunction->ADOQuery1->Fields->Fields[1]-
>Value;
    MainFunction->StringGrid1->Cells[2][k]=MainFunction->ADOQuery1->Fields->Fields[2]-
>Value;
    MainFunction->StringGrid1->Cells[3][k]=MainFunction->ADOQuery1->Fields->Fields[3]-
>Value;
    MainFunction->StringGrid1->Cells[4][k]=MainFunction->ADOQuery1->Fields->Fields[4]-
>Value;
    MainFunction->StringGrid1->Cells[5][k]=MainFunction->ADOQuery1->Fields->Fields[5]-
>Value;
    MainFunction->StringGrid1->Cells[6][k]=MainFunction->ADOQuery1->Fields->Fields[6]-
>Value;
    MainFunction->StringGrid1->Cells[7][k]=MainFunction->ADOQuery1->Fields->Fields[7]-
>Value;
    MainFunction->ADOQuery1->Next();
    }
MainFunction->ADOQuery1->Close();
MainFunction->ADOQuery1->SQL->Clear();

}
```

```
//------------------------METROPOLIS-------------------------------------------
int metrop(float de, float t,double Jx,double Jo)
{
//the metropolis algorithm which decides if the reversal was effective
//if not returns 0 and the reversal happens again
//if is 1 then reversal is correct and we may continue
        float ran3(long *idum);
        static long gljdum=1;

        return de < 0.0 || ran3(&gljdum) < exp(-(de)/t);
}
//------------------------RAN3-------------------------------------------------
float ran3(long *idum)
{
//function that generates numbers from 0 to 1
//it is a random number generator
        static int inext,inextp;
        static long ma[56];
        static int iff=0;
        long mj,mk;
        int i,ii,k;

        if (*idum < 0 || iff == 0) {
                iff=1;
                mj=MSEED-(*idum < 0 ? -*idum : *idum);
                mj %= MBIG;
                ma[55]=mj;
                mk=1;
                for (i=1;i<=54;i++) {
                        ii=(21*i) % 55;
                        ma[ii]=mk;
                        mk=mj-mk;
                        if (mk < MZ) mk += MBIG;
                        mj=ma[ii];
                }
                for (k=1;k<=4;k++)
                        for (i=1;i<=55;i++) {
                                ma[i] -= ma[1+(i+30) % 55];
                                if (ma[i] < MZ) ma[i] += MBIG;
                        }
                inext=0;
                inextp=31;
                *idum=1;
        }
        if (++inext == 56) inext=1;
        if (++inextp == 56) inextp=1;
        mj=ma[inext]-ma[inextp];
        if (mj < MZ) mj += MBIG;
        ma[inext]=mj;
        return mj*FAC;
}
//-------------------------REVERSE-----------------------------------------
void reverse(int mask[], int fieldcount, int n[])
{
//this function makes the segment reversal taking
```

```
        int nn,j,k,l,itmp;
      //This many cities must be swapped to
      //effect the reversal.
        nn=(1+((n[2]-n[1]+fieldcount) % fieldcount))/2;
        for (j=1;j<=nn;j++) {
       //Start at the ends of the segment and
       //swap pairs of cities, moving toward the center.
                k=((n[1]+j+fieldcount-1) % fieldcount);
                l=((n[2]-j+fieldcount+1) % fieldcount);

                itmp=mask[k];
                mask[k]=mask[l]; // do the swapping!
                mask[l]=itmp;

        }
}
//-------------------------MASK INITIALIZE-----------------------------------
void InitializeMask(int mask[],int fieldcount,int limit,int spot[])
{
//this function initializes the mask
//mask is a table which holds the columns used by the program
//the columns used have value of 1
//others that are not used have value of 0
int i, j, first,second,third,fourth;
//depending on the dimensions chosen initialize with the according number of ones
if(limit= =2)
{
      do {
         first=rand()%(fieldcount);
            second=rand()%(fieldcount);

        } while (first==second);

 for(i=0; i<fieldcount;i++)
            {
             mask[i]=0;
             }

 mask[first]=1;
 mask[second]=1;
 spot[0]=first+1;
 spot[1]=second+1;
 spot[2]=0;
 spot[3]=0;
}

if(limit= =3)
{
      do {
         first=rand()%(fieldcount);
            second=rand()%(fieldcount);
          third=rand()%(fieldcount);
        } while (first==second || first==third || second==third);

 for(i=0; i<fieldcount;i++)
```

```
            {
             mask[i]=0;
             }

 mask[first]=1;
 mask[second]=1;
 mask[third]=1;
 spot[0]= first+1;
 spot[1]=second+1;
 spot[2]=third+1;
 spot[3]=0;
}
if(limit= =4)
{
      do {
         first=rand()%(fieldcount);
             second=rand()%(fieldcount);
         third=rand()%(fieldcount);
         fourth=rand()%(fieldcount);
        } while (first==second || first==third || second==third || first==fourth || third==fourth ||
second==fourth );

 for(i=0; i<fieldcount;i++)
            {
             mask[i]=0;
             }

 mask[first]=1;
 mask[second]=1;
 mask[third]=1;
 mask[fourth]=1;
 spot[0]= first+1;
 spot[1]=second+1;
 spot[2]=third+1;
 spot[3]=fourth+1;
}


}
//------------------------------CHECK DATASET----------------------------------
void CheckDataSet(InfArray *DataSet,int recsize)
{
//those elements that have clusterID -1 are defined as Noise
int i;
for(i=0;i<recsize;i++)
  {
   if(DataSet[i].ClusterID==-1)
     DataSet[i].PointType="Noise";
  }
}
```

```
//------------------------------DBSCAN-----------------------------------------
int dbscan(int recsize, float Eps, int MinPts, int Knn, int spot[], AnsiString TableName)
{
//this functions takes the columns by reading spot. spot is a table that reads
//the mask and holds the columns chosen , or the columns that resulted from
//the mask reversal
double AverageDist=0;
int i,j;
int ClusterID=0;
//declare a new structure which will be erased in the end of the clustering
//proccess
InfArray* DataSet =new InfArray[recsize];
//Load the dataset to begin with the proccess of clustering
LoadDataset(recsize,DataSet,spot, TableName);
CalcKNearest(Knn,recsize,DataSet) ;

for(i=0;i<recsize;i++)
  {
  AverageDist=AverageDist+DataSet[i].KNearestDist;
  }
AverageDist=AverageDist/recsize;
//find the direct density reachable points and build Collections for every
//element on the DataSet
CheckDDRPoints(Eps,MinPts,recsize,DataSet);



for(int i=0;i<recsize;i++)
   {
  MainFunction->Memo1->Lines->Add(DataSet[i].PointType);
   }



for(i=0;i<recsize;i++)
    {
   //if border and ID=0 then it is noise
    if(DataSet[i].PointType=="Border" && DataSet[i].ClusterID==0)
       {
        DataSet[i].ClusterID=-1;
       }

      else if (DataSet[i].ClusterID==0)
        {
        //else find in which cluster it belongs
        ClusterID=ClusterID+1;
        FindClusters(i,recsize,DataSet,ClusterID);
        }


     }
CheckDataSet(DataSet, recsize);
StoreResults(recsize,DataSet);
delete [] DataSet;
return ClusterID;
```

```
}

//------------------------INTERCLASS DISTANCE---------------------------------
double InterClassDistance(int ClusterID)
{
//this function returns the distance between elements in clusters
AnsiString s1,s2,sql;
//in order to do that an sql string must be built
MainFunction->ADOQuery3->SQL->Clear();
MainFunction->ADOQuery3->SQL->Text="SELECT ClusterID, COUNT(*) AS Expr1 FROM
results WHERE (ClusterID <> - 1) GROUP BY ClusterID";
MainFunction->ADOQuery3->Prepared=true;
MainFunction->ADOQuery3->Open();
int clusnum=MainFunction->ADOQuery3->RecordCount;
double Dij;
int i,j,k;
float J=0;
double Pwi, Pwj;
double temp;
int clussum=0;
float *classdissum=new float[50];
double *clusterinfo=new double[clusnum];

for(k=0;k<clusnum;k++)
    {
    clusterinfo[k]=MainFunction->ADOQuery3->Fields->Fields[1]->Value;
    MainFunction->ADOQuery3->Next();
    }
MainFunction->ADOQuery3->Close();

for(int t=0;t<ClusterID;t++)
   clussum+=clusterinfo[t];

for(i=1;i<=ClusterID;i++)
  {
     Pwi=clusterinfo[i-1]/clussum;
 for( j=i+1; j<=ClusterID;j++)
    {
     Pwj=clusterinfo[j-1]/clussum;
   MainFunction->ADOQuery2->SQL->Clear();
   if(DataBaseSelect->SelectList->Text=="Oracle")
    sql= " select sum(sqrt((abs(a.x-b.x)*(abs(a.x-b.x)))+((abs(a.y-b.y))*(abs(a.y-b.y)))))";
   if(DataBaseSelect->SelectList->Text=="SQL Server")
    sql= " select sum(sqrt((abs(a.x-b.x)*(abs(a.x-b.x)))+((abs(a.y-b.y))*(abs(a.y-b.y)))))";
   if(DataBaseSelect->SelectList->Text=="Microsoft Access")
       sql= " select sum(sqr((abs(a.x-b.x)*(abs(a.x-b.x)))+((abs(a.y-b.y))*(abs(a.y-b.y)))))";

    sql+=" from results a, results b where a.ClusterID=" ;
    s1=i;
    s2=j;
    sql+=s1;
    sql+=" and b.ClusterID= ";
    sql+=s2;
    MainFunction->ADOQuery2->SQL->Add(sql);
    MainFunction->ADOQuery2->Prepared=true;
```

```
    MainFunction->ADOQuery2->Open();
   //find the other factors to calculate the final value of the evaluation
   //measure
    classdissum[i]= MainFunction->ADOQuery2->Fields->Fields[0]->Value;
    temp=1/((clusterinfo[i-1])*(clusterinfo[j-1]));
    Dij=classdissum[i]*temp;
    MainFunction->ADOQuery2->Close();

    Pwj=clusterinfo[j-1]/clussum;
    J+=Dij*Pwi*Pwj;

    }

  }
delete [] clusterinfo;
delete [] classdissum;
return J;
}
//------------------------------CHECK MASK-------------------------------------
AnsiString CheckMask(int mask[],int fieldcount,float Jold)
{
//this function makes a check to find if the mask has already being found
//by searching the maskdata table from the database where all the mask are
//held
int answer;
int i;
AnsiString str,sql;
for(i=0;i<fieldcount;i++)
     str+=mask[i];

sql="select Jx from maskdata where mask=";
sql+=" '";
sql+=str;
sql+="'";
MainFunction->ADOQuery1->SQL->Text=sql;
MainFunction->ADOQuery1->Prepared=true;
MainFunction->ADOQuery1->Open();

if (MainFunction->ADOQuery1->RecordCount!=0)
{
answer=1;
MainFunction->ADOQuery1->Prepared=false;
MainFunction->ADOQuery1->Close();
MainFunction->ADOQuery1->SQL->Text="";
}
else answer=0;
return answer;
}
//-----------------------SAVE MASK-----------------------------------------
void SaveMask(TObject *Sender, int mask[], float Jold, int rev,int fieldcount)
{
//this function saves the mask created by the reversal during the annealing
//proccess including the iteration id and the interclass distance evaluation
//measure
AnsiString text;
```

```
AnsiString sql;
for(int i=0;i<fieldcount;i++)
   text+=mask[i];


sql="insert into maskdata ( id, mask , Jx ) Values (";
sql+=rev;
sql+=", '";
sql+=text;
sql+="' ,";
sql+=Jold;
sql+=" )";
MainFunction->ADOCommand1->CommandText=sql;
MainFunction->ADOCommand1->Execute();
}
//----------------------UPDATE SPOT-------------------------------------------
void UpDateSpot(int mask[],int spot[],int fieldcount)
{
//this function reads the mask to find the changes that happened from the
//reversal, and find which columns are to be processed and puts them into
//spot[]
int i=0;
int first;
int second;
int count=0;
int limit=MainFunction->ComboBox1->Text.ToInt();
//according to the dimensions selected
if(limit==2)
  {
  for(i=0;i<fieldcount;i++)
     {
     if(mask[i]==1)
       {
       spot[count]=i+1;
       count++;
       }
     }
   spot[2]=0;
   spot[3]=0;
  }


if(limit==3)
  {
  for(i=0;i<fieldcount;i++)
     {
     if(mask[i]==1)
       {
       spot[count]=i+1;
       count++;
       }
     }
   spot[3]=0;
  }
```

```
if(limit==4)
  {
  for(i=0;i<fieldcount;i++)
     {
     if(mask[i]==1)
       {
       spot[count]=i+1;
       count++;
       }
      }

  }



}
//-------------------------ANNEAL-----------------------------------------------
double anneal(int mask[], int recsize, int fieldcount, double Jold ,int spot[], float t, int repeat ,double
Eps, int MinPts ,int Knn, AnsiString TableName)
{
//the annealing proccess does the reversal finds the clusters through dbscan
//finds the interclass distance and evaluates through this value if the reversal
//is effective. with other words if the columns chosen are good enough
AnsiString test,ans2;
int *endmask=new int [fieldcount];
int rep;
double Jnew;
int ClusterID=0;
int ans;
int i,j,nn;
static int n[6];              //used for segment informations- element n[0] is not used
float de;
double temp;
MainFunction->ProgressBar1->Max=repeat;
MainFunction->ProgressBar1->Step=1;

      //iterations set by the user
         for(rep=1;rep<repeat;rep++)
           {

           MainFunction->ProgressBar1->StepIt();
                    do {
              //pick randomly two elements from mask[fieldcount]
                n[1]=1+rand()%(fieldcount-2);
                          n[2]=1+rand()%(fieldcount-2);
              } while (n[2]<n[1]);

        sp:    //do the reversal
                reverse(mask,fieldcount,n);
              //and check if the new mask that occurred already exists
                ans2=CheckMask(mask,fieldcount,Jold);

               //if not then
               if(ans2==0)
```

```
                {
                t*=TFACTR;
                ClusterID=0;
               //find the new set of columns
                UpDateSpot(mask,spot,fieldcount);
                //find the clusters
              ClusterID=dbscan(recsize,Eps,MinPts,Knn,spot,TableName);
               //calculate the interclass distance
                Jnew=InterClassDistance(ClusterID);
               //if no clusters found then interclass distance is 0
               //if 0 then redo reversal without saving
               if(Jnew==0)
                  goto sp;
          //else save the new mask
          SaveMask(MainFunction->ADOCommand1,mask,Jnew,rep,fieldcount);
          //find the difference with the old best interclass distance
                de=-Jnew+Jold;
           //consult the oracle
                ans=metrop(de,t,Jnew,Jold);
                //if approved then continue and set the new interclass
                //distance as old and save the mask
                   if(ans)
                     {
                     Jold=Jnew;
                     for(int r=0;r<fieldcount;r++)
                     endmask[r]=mask[r];
                     }
                   //else do nothing and continue with new reversal
                   else goto sp;
                }


        }

//after the end of the iterations find the best spot from the table endmask
//in order to fill the results table.
UpDateSpot(endmask,spot,fieldcount);
ClusterID=dbscan(recsize,Eps,MinPts,Knn,spot,TableName);

MainFunction->Label1->Caption=spot[0];
MainFunction->Label2->Caption=spot[1];
MainFunction->Label4->Caption=spot[2];
MainFunction->Label5->Caption=spot[3];
MainFunction->Label3->Caption=Jold;
return Jold;



}

//-----------------------ERASE MASK DATA-------------------------------------
/*void EraseMaskData(TObject *Sender)
{
MainFunction->ADOCommand1->CommandText="delete from maskdata" ;
```

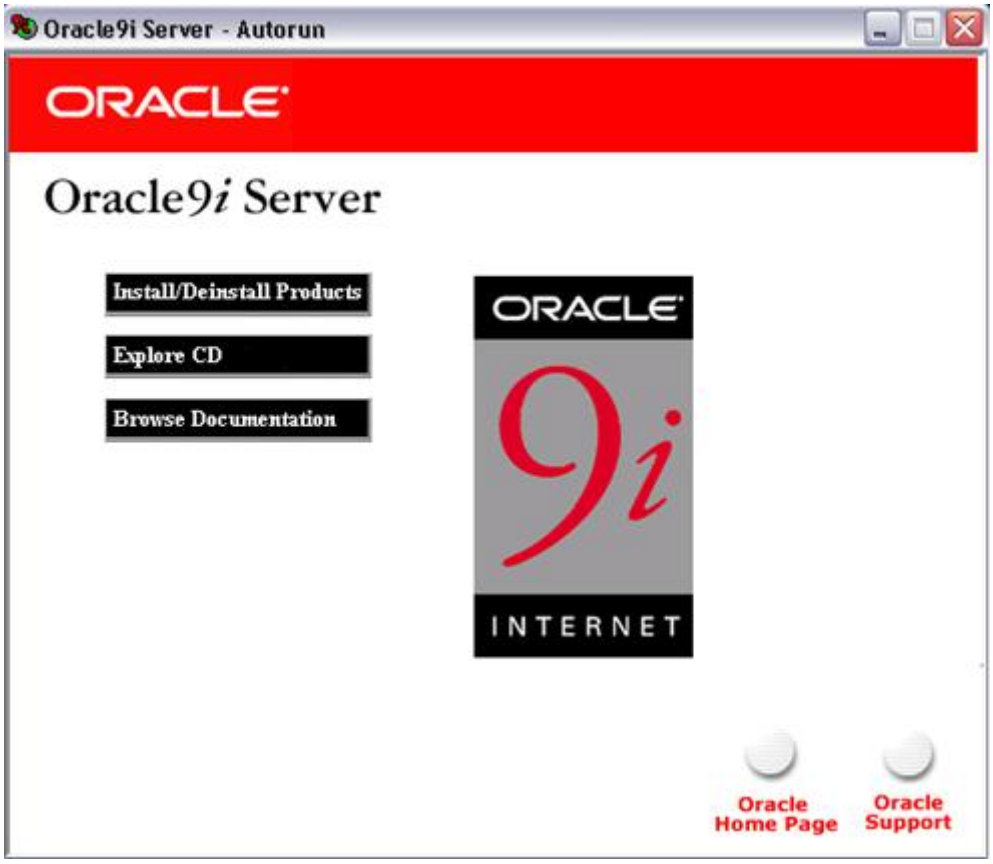
```
MainFunction->ADOCommand1->Execute();
MainFunction->ADOCommand1->CommandText="";
} */

//---------------------------------------------------------------------------
__fastcall TMainFunction::TMainFunction(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TMainFunction::GoBackClick(TObject *Sender)
{MainFunction->Close();
}
//---------------------------------------------------------------------------
void __fastcall TMainFunction::FormCreate(TObject *Sender)
{
SpeedButton1->Enabled=false;
randomize();
}
//---------------------------------------------------------------------------
void __fastcall TMainFunction::StartClick(TObject *Sender)
{
double EndJ;
//------- check for a valid table name.--------------------------------
AnsiString TableName=DataSetSelect->ComboBox1->Text;
if(TableName=="")
   {
   MessageBox(NULL,"Please Select a valid TableName","Error",MB_OK);
   MainForm->SelectDataSet->Click();
   }
TableName=DataSetSelect->ComboBox1->Text;

//----------checking for existence of tables "results" and "maskdata"----------
//--------------these tables are use for output------------------------------
//------------at the end of the program the tables remain in the database------
if(DataSetSelect->ComboBox1->Items->IndexOf("maskdata")!= -1)
    {
    MainFunction->ADOCommand1->CommandText="drop table maskdata";
    MainFunction->ADOCommand1->Execute();
    ADOCommand1->CommandText="Create Table maskdata (id int primary key, mask char(80),
jx float )";
    ADOCommand1->Execute();
    }
else if (DataSetSelect->ComboBox1->Items->IndexOf("maskdata")== -1)
    {
    ADOCommand1->CommandText="Create Table maskdata (id int primary key, mask char(80),
jx float )";
    ADOCommand1->Execute();
    DataSetSelect->ComboBox1->Items->Add("maskdata");
    }

if(DataSetSelect->ComboBox1->Items->IndexOf("results")!= -1 )
    {
    MainFunction->ADOCommand1->CommandText="drop table results";
    MainFunction->ADOCommand1->Execute();
```

```
   MainFunction->ADOCommand1->CommandText="create table results (  cod int primary key ,
point int , x float , y float , z float , r float , ClusterID int, PointType varchar(10)  )"  ;
   MainFunction->ADOCommand1->Execute();

   }
if(DataSetSelect->ComboBox1->Items->IndexOf("results")== -1  )
    {
   MainFunction->ADOCommand1->CommandText="create table results (  cod int primary key ,
point int , x float , y float , z float , r float , ClusterID int, PointType varchar(10)  )"  ;
   MainFunction->ADOCommand1->Execute();
   DataSetSelect->ComboBox1->Items->Add("results");
    }


double Eps;
int MinPts;
float t;
int repeat;
int Knn;
int answer;


repeat=Edit5->Text.ToInt();
t=Edit1->Text.ToInt();
Knn=Edit4->Text.ToInt();
Eps=Edit2->Text.ToDouble();
MinPts=Edit3->Text.ToInt();

int fieldcount=FieldCount(ADOQuery1,TableName);
int recsize=RecordSize(ADOQuery1,TableName);
int *mask=new int [fieldcount];
int *spot=new int [4];
float Jold;
InfArray* DataSet =new InfArray[recsize];

int limit=ComboBox1->Text.ToInt();
int ClusterID=0;
InitializeMask(mask,fieldcount,limit,spot);

ClusterID=dbscan(recsize,Eps,MinPts,Knn,spot, TableName);
Jold=InterClassDistance(ClusterID);

SaveMask(ADOCommand1,mask,Jold,0,fieldcount);

delete [] DataSet;
EndJ=anneal(mask, recsize, fieldcount, Jold ,spot,  t,repeat,Eps,MinPts ,Knn, TableName);
ShowMessage("Finish!!") ;
SpeedButton1->Enabled=true;
delete [] mask;
delete [] spot;
}
//---------------------------------------------------------------------------

void __fastcall TMainFunction::SpeedButton1Click(TObject *Sender)
{
```

```
Results->ADOQuery1->SQL->Text="select x, y, clusterid  from results";
Results->ADOQuery1->Active=true;
Results->ADOQuery1->ExecSQL();
Results->ADOQuery1->Open();
Results->DBChart1->Series[0]->DataSource=Results->ADOQuery1;
Results->DBChart1->Series[0]->YValues->ValueSource='y';
Results->DBChart1->Series[0]->XValues->ValueSource='x';
//Results->DBChart1->Series[0]->XLabelsSource=Results->ADOQuery1->Fields->Fields[2]-
>AsString;

Results->DBChart1->Enabled=true;


Results->ADOQuery2->SQL->Text="select x,y from results";
Results->ADOQuery2->Active=true;
Results->ADOQuery2->ExecSQL();
Results->ADOQuery2->Open();
Results->StringGrid1->ColCount=Results->ADOQuery2->FieldCount;
Results->StringGrid1->RowCount=Results->ADOQuery2->RecordCount;
for(int i=0;i<Results->ADOQuery2->RecordCount;i++)
    {
    Results->StringGrid1->Cells[0][i]=Results->ADOQuery2->Fields->Fields[0]->Value;
    Results->StringGrid1->Cells[1][i]=Results->ADOQuery2->Fields->Fields[1]->Value;
    Results->ADOQuery2->Next();
    }

 Results->ADOQuery2->Close();
Results->ShowModal();
}
//---------------------------------------------------------------------------


void __fastcall TMainFunction::SpeedButton2Click(TObject *Sender)
{
ShowType->ShowModal();
}
//---------------------------------------------------------------------------
```

8 ΠΑΡΑΡΤΗΜΑ Α’

8.1 Εγχειρίδιο εγκατάστασης πλατφόρμας Oracle

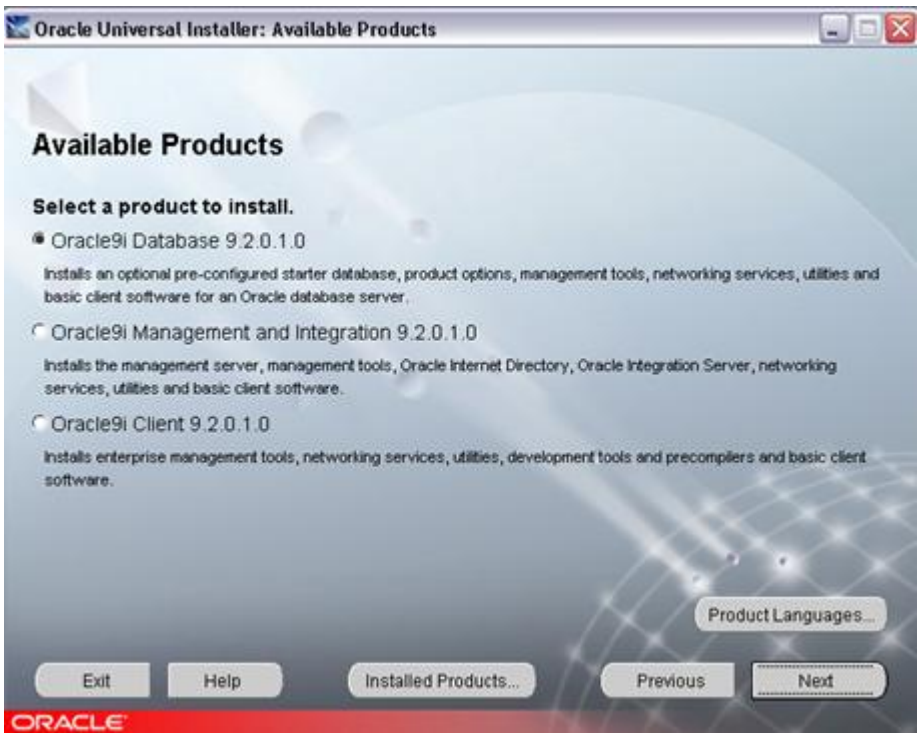


8.2 Εγκατάσταση Oracle

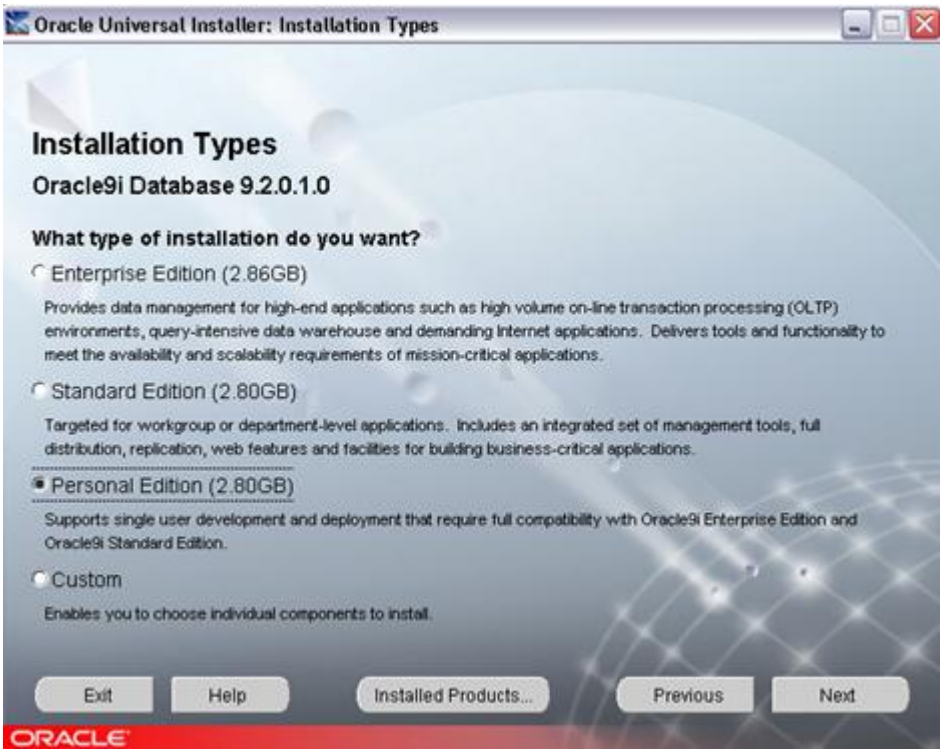
Το πρώτο παράθυρο που συναντάμε κατά την διαδικασία της εγκατάστασης είναι το παρακάτω:

Συνεχίζοντας (Next), Αντικρίζουμε ένα παράθυρο στο οποίο καλούμαστε να καταχωρήσουμε το μονοπάτι στο οποίο θα γίνει η εγκατάσταση. Στο πάνω μέρος βρίσκεται το μονοπάτι στο οποίο θα γίνει η αναζήτηση με την λίστα που περιέχει τα προϊόντα της Oracle. Το μονοπάτι στο οποίο θα γίνει η διαδικασία της εγκατάστασης προκαθορίζεται από το πρόγραμμα και αυτό προτιμάμε.

Προχωρώντας, μας εμφανίζεται η λίστα με τα διαθέσιμα προϊόντα. Από αυτά θα επιλέξουμε την Oracle DataBase 9.2.0.1.0. που είναι η πρώτη επιλογή.

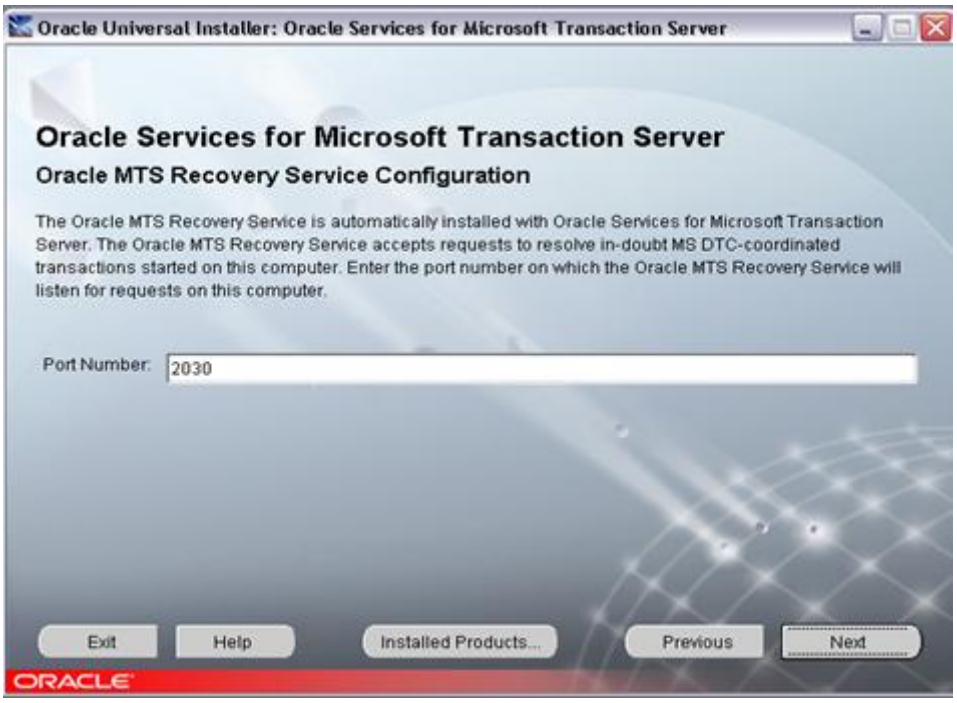


Στην συνέχεια, καλούμαστε να επιλέξουμε τον τύπο της εγκατάστασης. Εδώ παρέχονται τρεις επιλογές για τύπο εγκατάστασης αλλά και μια που καθορίζεται από τον χρήστη. Στην περίπτωσή μας, επιλέχθηκε η personal edition.

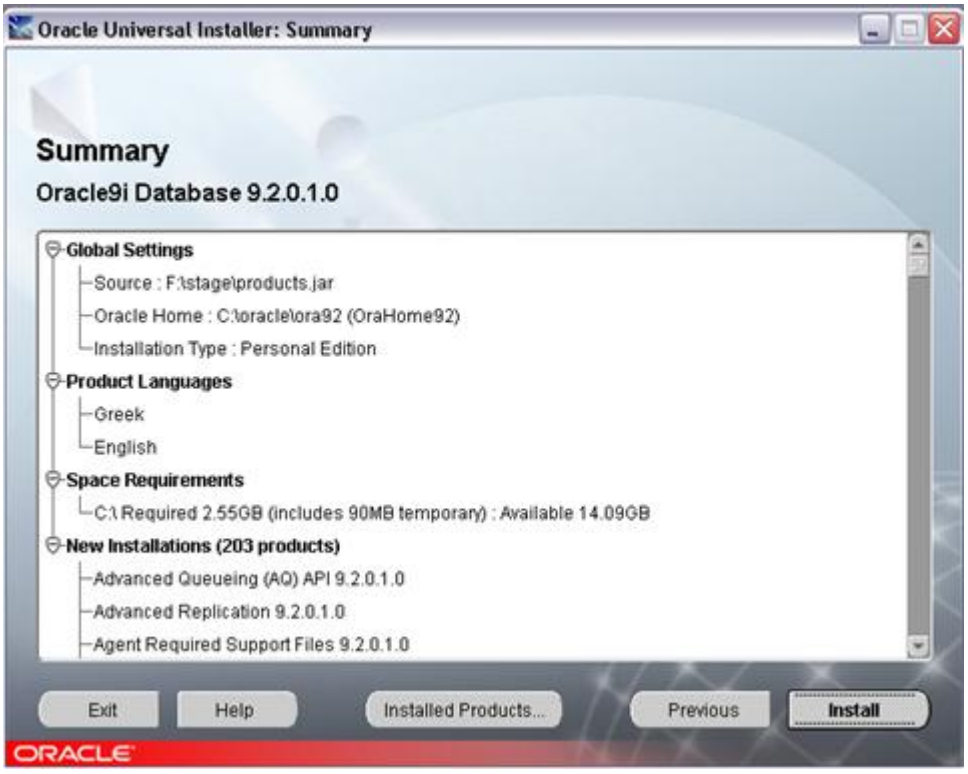


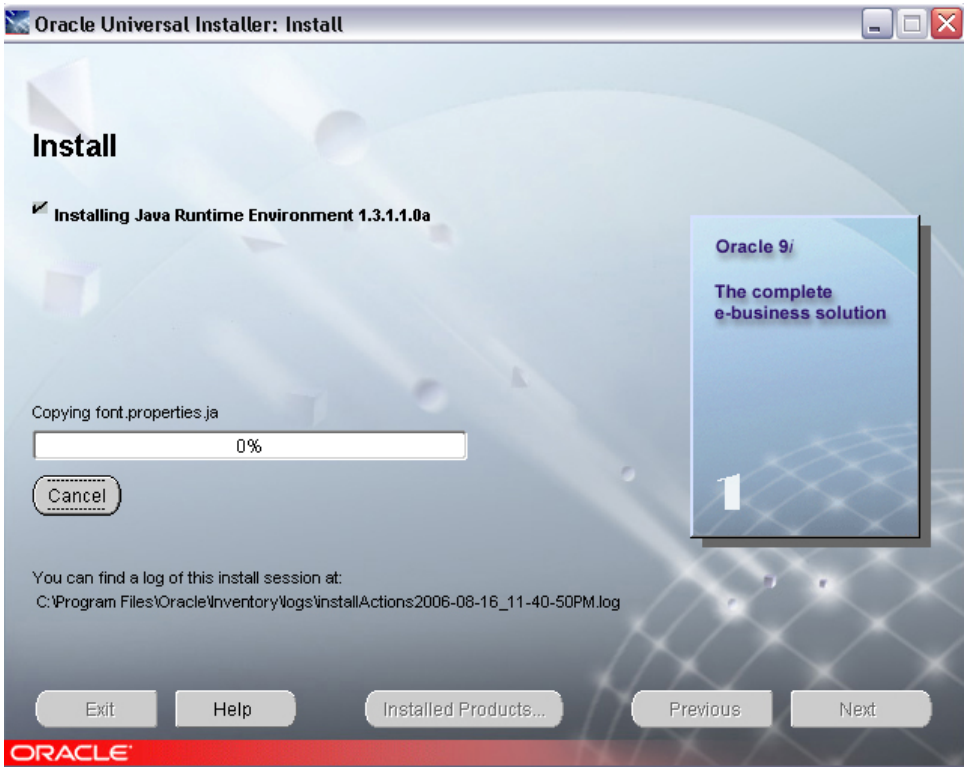
Αφού επιλεχθεί ο τύπος εγκατάστασης έρχεται η σειρά της διαμόρφωσης της βάσης δεδομένων που θα δημιουργηθεί μαζί με την εγκατάσταση της πλατφόρμας. Υπάρχει και η επιλογή για την εγκατάσταση αποκλειστικά και μόνο του προγράμματος και την δημιουργία βάσης δεδομένων μετά την εγκατάσταση. Αυτή είναι και η επιλογή που θα μαρκάρουμε.

Το επόμενο παράθυρο μας πληροφορεί για την πόρτα στην οποία η υπηρεσία αποκατάστασης της Oracle να ακούει τις προσκλήσεις του υπολογιστή. Αποδεχόμαστε την δεδομένη πόρτα και συνεχίζουμε.



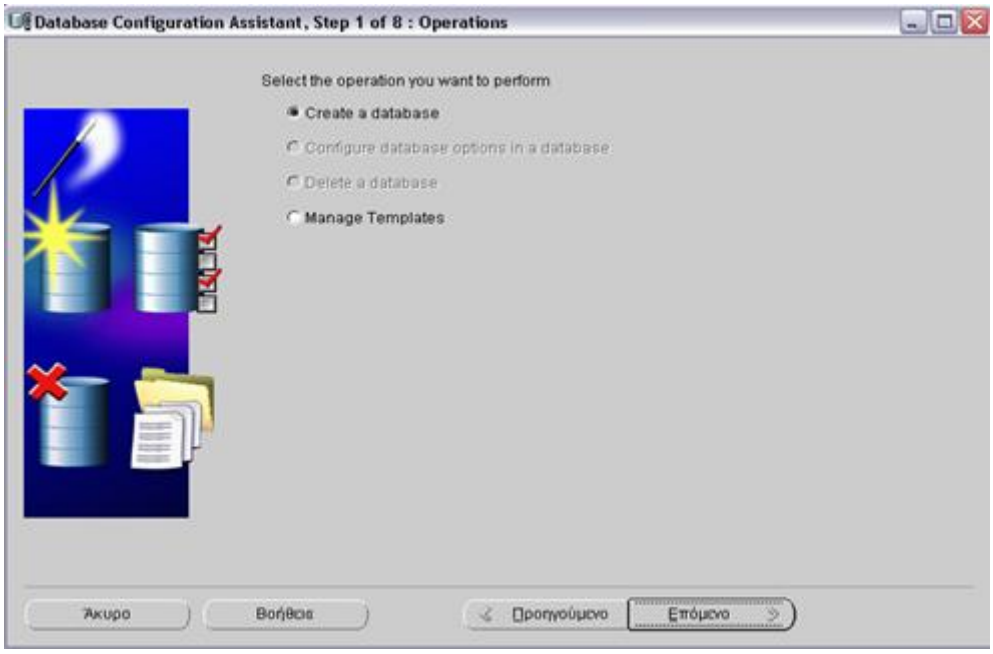
Στα επόμενα παράθυρα απλά γίνεται μια περίληψη αυτών που θα εγκατασταθούν. Παρακάτω εμφανίζονται τα παράθυρα.



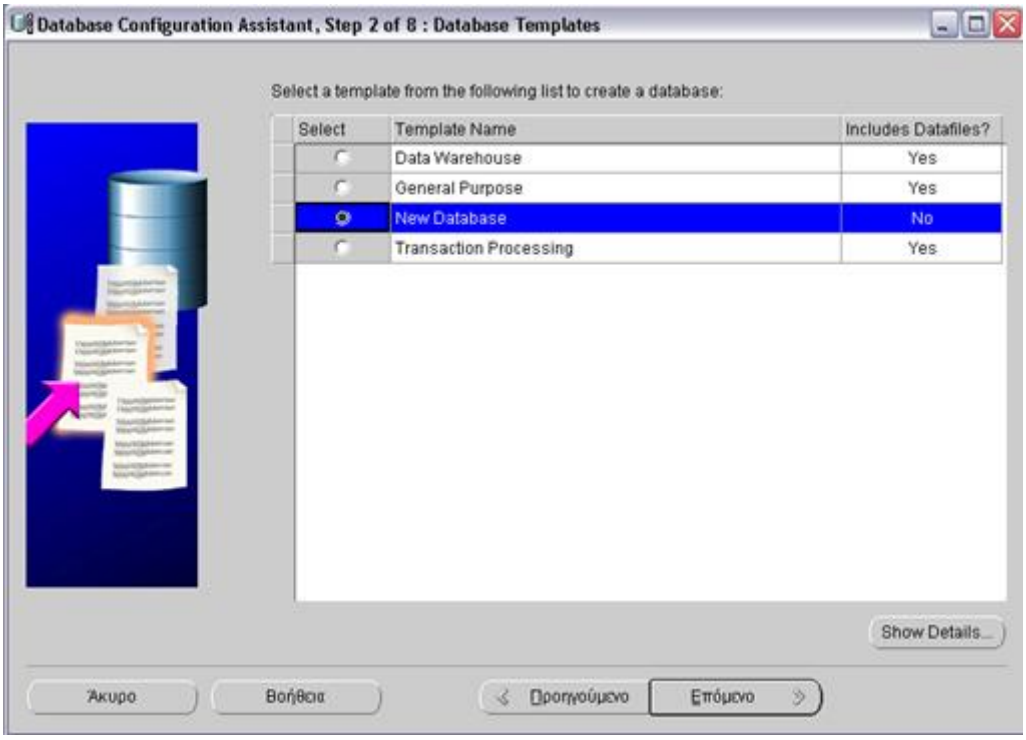

Oracle Universal Installer: Install
Install
Installing Java Runtime Environment 1.3.1.1.0a
Oracle 9i
The complete
e-business solution
Copying font.properties.ja
0%
Cancel
You can find a log of this install session at:
C:\Program Files\Oracle\Inventory\logs\installActions2006-08-16_11-40-50PM.log
Exit
Help
Installed Products...
Previous
Next
ORACLE

8.3 Δημιουργία Βάσης Δεδομένων

Για την δημιουργία της βάσης θα χρειαστούμε την βοήθεια του DataBase Configuration Assistant. Στο πρώτο παράθυρο έχουμε την επιλογή της δημιουργίας βάσης δεδομένων.



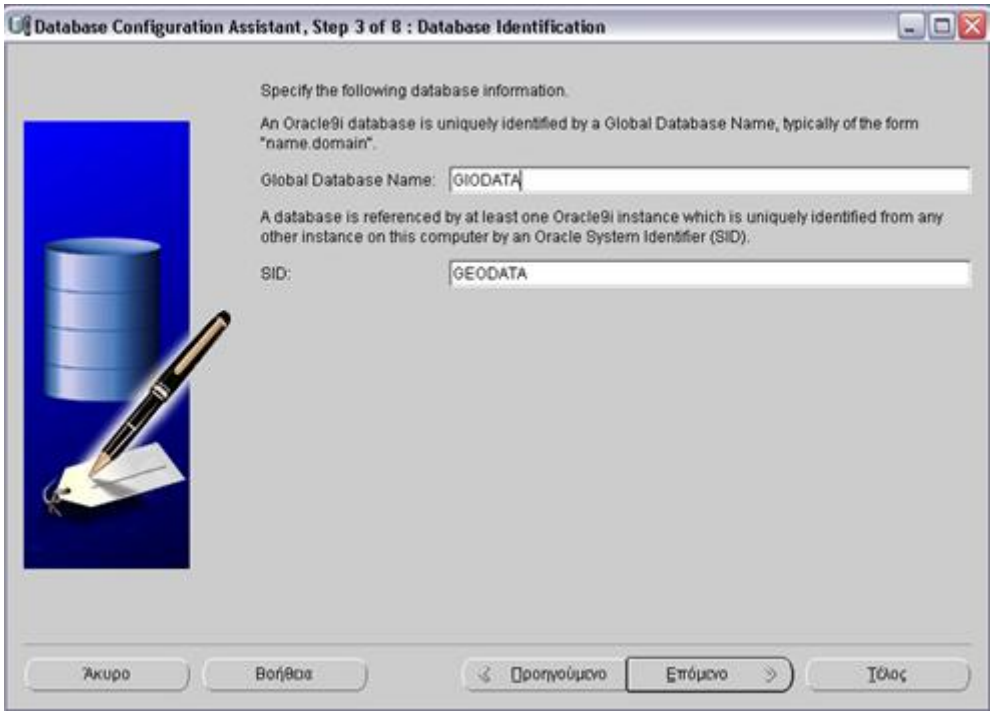
Στην Oracle υπάρχουν μερικές έτοιμες επιλογές για δημιουργία βάσης δεδομένων. Εμείς θα επιλέξουμε New DataBase για να καθορίσουμε εμείς τα περιεχόμενα της.



Στο επόμενο παράθυρο καλούμαστε να δώσουμε ένα μοναδικό όνομα στην βάση δεδομένων αλλά και ένα SID (**S**ystem **ID**entifier). Επιλέχθηκαν τα παρακάτω ονόματα.

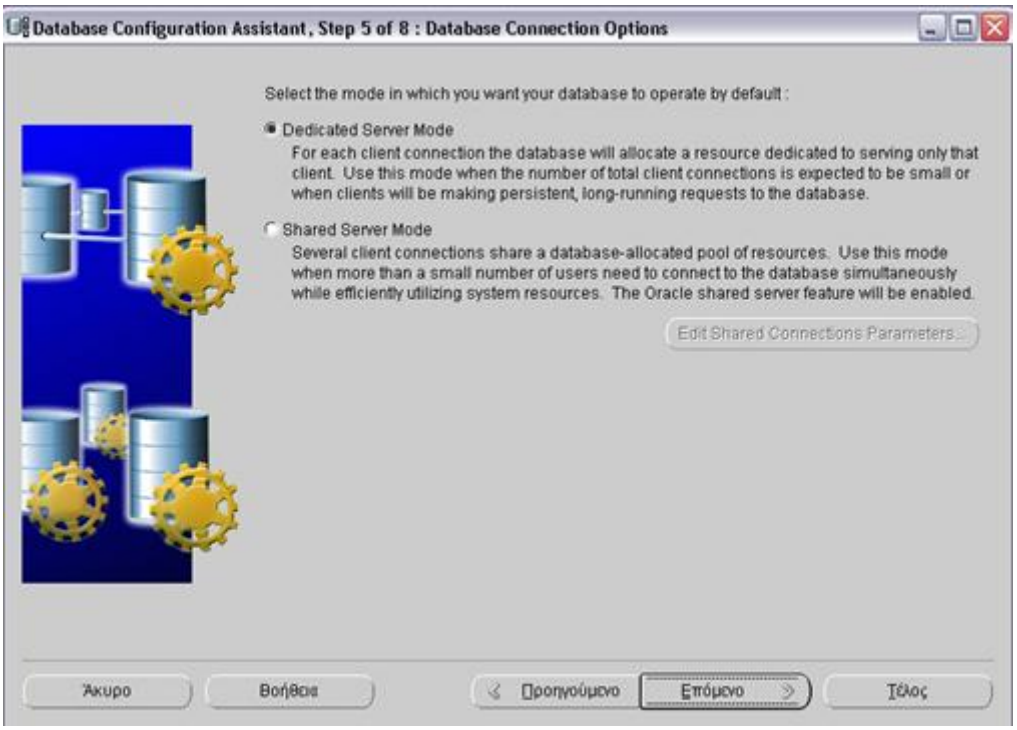
Global Database Name: **GIODATA**

SID: **GEODATA**.

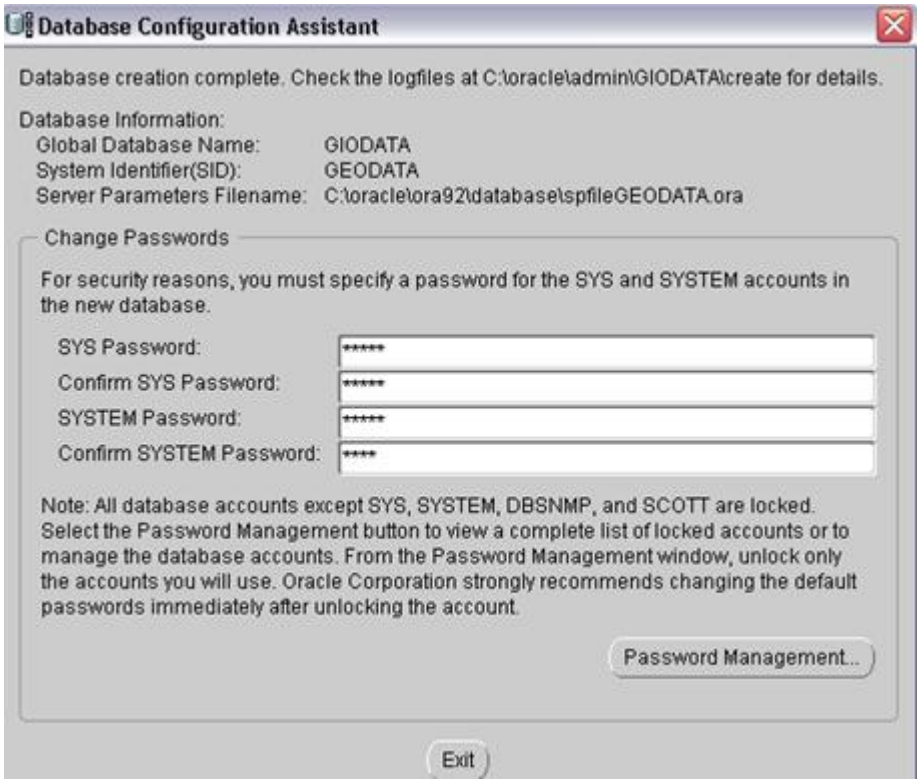


Αμέσως μετά, εμφανίζεται το παράθυρο στο οποίο μπορούμε να επιλέξουμε τα περιεχόμενα της βάσης που θα δημιουργηθεί. Στην περίπτωση μας δεν επιθυμούμε τίποτα από αυτά οπότε δεν επιλέγουμε τίποτα.

Στην συνέχεια μας παρουσιάζεται η επιλογή του τρόπου λειτουργίας της βάσης. Μπορούμε να επιλέξουμε ανάλογα με τους χρήστες που πρόκειται να συνδέονται σε αυτή την βάση και την διάρκεια των συνδέσεων τους. Αναλόγως επιλέγουμε.

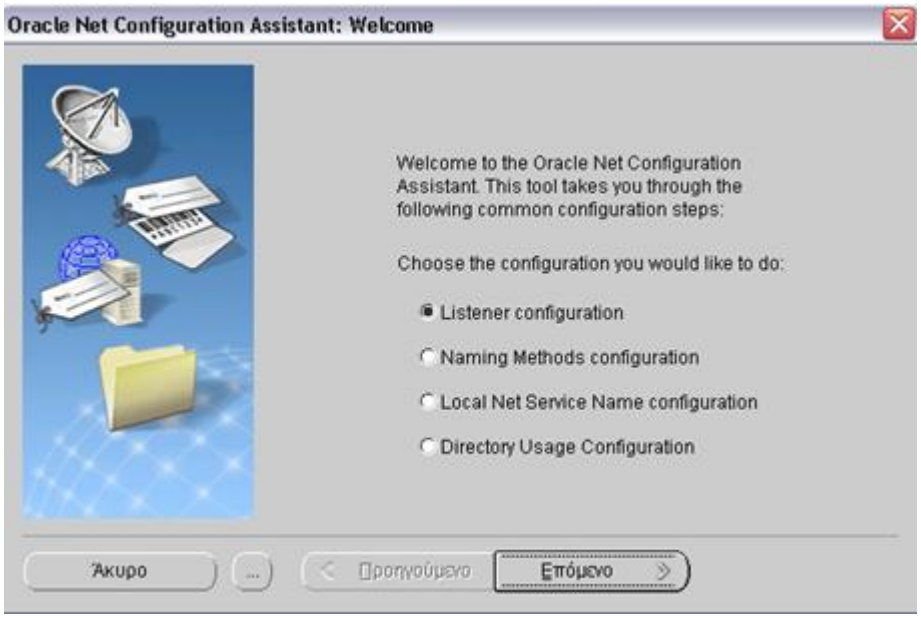


Προσπερνάμε το επόμενα παράθυρα ώσπου να αρχίσει η διαδικασία της δημιουργίας. Τέλος, μετά την δημιουργία θα μας ζητηθούν τα passwords των λογαριασμών sys και system. Και στα δύο επιλέχθηκε το password: **admin**.



8.4 Δημιουργία Listener

Για την επικοινωνία με την βάση δεδομένων απαιτείται η δημιουργία μιας διαδικασίας ακρόασης. Για αυτό θα χρειαστούμε την βοήθεια του Net Configuration Assistant.



Θα προσθέσουμε μια διεργασία ακρόασης.

Επιλέγουμε το όνομα του "ακροατή" και πατάμε επόμενο. Στο επόμενο παράθυρο επιλέγουμε τα είδη των συνδέσεων που θα δέχεται ο ακροατής μας και αργότερα την πόρτα που θα "ακούει". Η πόρτα θα πρέπει να είναι ίδια με αυτή της βάσης μας έτσι ώστε να μπορεί να ακούει τις συνδέσεις που θα πραγματοποιούνται από το πρόγραμμά μας.

8.5 Δημιουργία Χρήστη και εκχώρηση δικαιωμάτων.

Εδώ ανοίγουμε την Enterprise Manager Console και επιλέγουμε login ως standalone. Για την δημιουργία χρήστη θα πάμε στην DataBase Applications και το SQL*Plus WorkSheet. Εκεί θα χρησιμοποιήσουμε την γλώσσα SQL για την δημιουργία χρήστη.

Ο SQL κώδικας για την διαδικασία login είναι ο παρακάτω

```
CONNECT SYSTEM/ADMIN
```

Ο SQL κώδικας για την δημιουργία χρήστη είναι ο παρακάτω

```
CREATE USER GIWRGOS IDENTIFIED BY GIWRGOS_PASS
TEMPORARY TABLESPACE TEMP
DEFAULT TABLESPACE USERS
```

Ο SQL κώδικας για την εκχώρηση δικαιωμάτων είναι ο παρακάτω

```
GRANT RESOURCE, CONNECT , UNLIMITED TABLESPACE TO GIWRGOS;
```

Εκτελώντας τις εντολές διαδοχικά δημιουργείται ο χρήστης και μετά γίνεται η εκχώρηση των δικαιωμάτων.