ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη μεθόδων λήψης απόφασης με πολλαπλά κριτήρια μέσω ερωτημάτων κορυφογραμμής.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΤΣΑΜΗ ΒΑΣΙΛΙΚΗΣ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2013

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη μεθόδων λήψης απόφασης με πολλαπλά κριτήρια μέσω ερωτημάτων κορυφογραμμής.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΤΣΑΜΗ ΒΑΣΙΛΙΚΗΣ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Μαρτίου.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Κώστας Κοντογιάννης Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2013

(Υπογραφή)

.....

ΤΣΑΜΗ ΒΑΣΙΛΙΚΗ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η μελέτη της λήψης αποφάσεων με πολλαπλά κριτήρια σε βάσεις δεδομένων με τη βοήθεια του τελεστή κορυφογραμμής (skyline). Το skyline ενός συνόλου σημείων ορίζεται ως τα σημεία εκείνα που δεν κυριαρχούνται από κανένα άλλο, λαμβάνοντας υπόψη ένα σύνολο από τις διαστάσεις του. Έχουν προταθεί αρκετοί αλγόριθμοι υπολογισμού του skyline. Το βασικό πρόβλημα είναι ότι ένας skyline αλγόριθμος πρέπει να είναι υπολογιστικά αποτελεσματικός και να έχει μικρό κόστος εισόδου εξόδου (I/O). Ιδιαίτερο ενδιαφέρον παρουσιάζει επίσης η περίπτωση που το skyline μιας βάσης δεδομένων είναι μεγαλύτερο από τη διαθέσιμη μνήμη του συστήματος, φαινόμενο αρκετά συχνό λόγω της ραγδαίας αύξησης του μεγέθους των βάσεων δεδομένων τα τελευταία χρόνια.

Στην παρούσα διπλωματική, μελετήθηκαν και υλοποιήθηκαν δύο αλγόριθμοι για τον υπολογισμό του skyline στην εξωτερική μνήμη. Ο ένας αλγόριθμος είναι ο Sort and Limit Skyline Algorithm (SALSA) που αξιοποιεί την ιδέα της ταξινόμησης των δεδομένων εισόδου, έτσι ώστε να περιοριστεί άμεσα ο αριθμός των πλειάδων που θα διαβαστεί και θα λάβει μέρος σε ελέγχους κυριαρχίας (dominance checks). Ταξινομώντας τα σημεία με τη βοήθεια συμμετρικών συναρτήσεων, ο αριθμός των πλειάδων που πρέπει να διαβαστούν ελαχιστοποιείται. Ο δεύτερος αλγόριθμος είναι ο Object Space Partitioning (OSP), που βασίζεται στην ιδέα της οργάνωσης των skyline σημείων σε ένα δέντρο, που ορίζει έναν αναδρομικό κατακερματισμό του χώρου (space partitioning). Με τη βοήθεια αυτού του skyline δέντρου κάθε υποψήφιο skyline σημείο χρειάζεται να ελεγχθεί για κυριαρχία μόνο με ένα μικρό σύνολο των ήδη υπάρχοντων skyline σημείων, το οποίο μειώνεται, όσο αυξάνεται η διάσταση των σημείων.

Καλούμενοι να λύσουμε το πρόβλημα της εύρεσης του skyline σε περιορισμένη κύρια μνήμη οι παραπάνω αλγόριθμοι υλοποιήθηκαν ως external αλγόριθμοι, χρησιμοποιώντας για την αποθήκευση των δεδομένων, αρχεία όπου τα δεδομένα αποθηκεύονται ανά μπλοκ (BlockFiles).

Λέξεις Κλειδιά:<<skyline, dominance checks, OSP, SALSA, space partitioning, blockfile>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The scope of this thesis was to study the application of the skyline operator in multi-criteria decision making. A skyline query retrieves from a given data set, a set of tuples that are not dominated by any other tuples with respect to a set of dimensions. Numerous algorithms have been proposed for the skyline computation. The main challenge is to have a computationally efficient skyline algorithm with a low I/O cost. Considerable attention has been given to the fact that the skyline of a database grows larger than the available system memory, a common problem with today's rapidly enlarging databases.

In this thesis, we study and develop two algorithms in order to tackle the problem of skyline computation in external memory. The Sort and Limit Skyline algorithm (SALSA) exploits the idea of presorting the input data so as to effectively limit the number of tuples to be read and compared. By sorting the data with symmetric monotonic functions, the number of tuples to be read is minimized. The second algorithm is the Object Space Partitioning algorithm (OSP), which is based on the idea of organizing the skyline points in a tree which defines a recursive space partitioning. With the help of this tree, each candidate skyline object only needs to be compared for dominance with a small subset of the existing skyline points, which decreases with the dimensionality of the problem.

In order to solve the skyline computation problem on bounded memory, the aforementioned algorithms were developed as external ones. As a result, files, where data can be stored as blocks of memory, were used (blockfiles).

Keywords:<<skyline, dominated, OSP, SALSA, space partitioning, blockfile>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 *Εισαγωγή*

1.1 Πολυδιάστατα δεδομένα

Οι Βάσεις Δεδομένων έχουν εξελιχθεί κατά πολύ από την αρχική τους μορφή, χωρίς ωστόσο να έχουν διαφοροποιηθεί ως προς τις ουσιαστικές λειτουργίες τους, δηλαδή την αποθήκευση και ανάκτηση πληροφοριών. Μία από τις σημαντικότερες εξελίξεις που τις αφορούν είναι η υποστήριξη πολυδιάστατων δεδομένων και η αποδοτική αντιμετώπιση πληθώρας ερωτημάτων που τα αφορούν. Η ανάγκη για την υποστήριξη τέτοιου τύπου δεδομένων ήταν απόρροια της ανάπτυξης αντίστοιχων εφαρμογών που τα χρησιμοποιούν, όπως είναι τα Γεωγραφικά Συστήματα Πληροφοριών (Geographic Information Systems ή GIS) και εφαρμογές Πολυμέσων (Multimedia applications). Δεν είναι τυχαίο άλλωστε ότι συχνά γίνονται αναφορές για Χωρικές / Χώρο-Χρονικές Βάσεις Δεδομένων (Spatial / Spatio – Temporal Databases) και Βάσεις Δεδομένων Πολυμέσων (Multimedia Databases), προκειμένου να υποδηλώσουμε τόσο την υποστήριξη των πολυδιάστατων δεδομένων ως εγγενείς τύπους δεδομένων όσο και των αντίστοιχων ερωτημάτων που πραγματοποιούνται επ' αυτών και ποικίλουν ανά εφαρμογή. Ο λόγος που γίνεται διαχωρισμός μεταξύ των πολυδιάστατων και μονοδιάστατων δεδομένων προκύπτει από τις διαφορετικές έννοιες που αντιπροσωπεύουν και από το πώς μπορούμε να τις διαχειριστούμε. Ένας από τους βασικότερους λόγους είναι ότι στα πολυδιάστατα δεδομένα δεν υπάρχει αυστηρός τρόπος διάταξης, όπως συμβαίνει με τα μονοδιάστατα. Για παράδειγμα, ενώ μπορούμε να διατάξουμε άτομα ως προς το ύψος τους ή το βάρος τους, δεν μπορούμε να τα διατάξουμε ως προς την τοποθεσία της πόλης από την οποία κατάγονται (ακόμα και αν η πληροφορία αυτή αναπαρίσταται με γεωγραφικό μήκος και πλάτος). Ένα άλλο χαρακτηριστικό των πολυδιάστατων δεδομένων είναι ότι, επειδή δεν υπάρχει η έννοια της διάταξης, μας ενδιαφέρει συνήθως να βρίσκουμε δεδομένα τα οποία είναι παρόμοια ως προς κάποιο δοθέν δεδομένο και δεν ψάχνουμε για ακριβή ομοιότητα. Αντιπροσωπευτικά τέτοια ερωτήματα είναι το ερώτημα των «k-πλησιέστερων γειτόνων» (k-nearest neighbor) και τα top-k ερωτήματα, με τα οποία αναζητούμε τα k δεδομένα που είναι πιο κοντά / όμοια με ένα δοθέν αντικείμενο. Μάλιστα, τα ερωτήματα αυτού του τύπου είναι τόσο σημαντικά, ώστε υπάρχει συγκεκριμένος κλάδος της Πληροφορικής (Ανάκτηση Πληροφορίας – Information Retrieval) που ασχολείται κατεξοχήν με αυτά. Τα πολυδιάστατα δεδομένα αντιμετωπίζουν, επίσης, και

προβλήματα αποθηκευτικού χώρου, λόγω των πολλών διαστάσεων που έχουν. Οι υψηλές διαστάσεις, οι οποίες είναι σύνηθες φαινόμενο σε συγκεκριμένες περιπτώσεις, όπως σε Βάσεις Δεδομένων Πολυμέσων ή σε εφαρμογές Βίο-πληροφορικής (Bioinformatics) δημιουργούν αυτομάτως την ανάγκη για ύπαρξη μεγαλύτερου αποθηκευτικού χώρου. Ένα από τα σημαντικότερα μειονεκτήματα της ύπαρξης πολλών διαστάσεων είναι ότι τα δεδομένα απομακρύνονται πολύ το ένα από το άλλο, το οποίο είναι γνωστό και ως κατάρα της διάστασης των δεδομένων (curse of dimensionality), που ορίστηκε από τον R. Bellman το 1961. Αυτό έχει ως συνέπεια σε πολύ υψηλές διαστάσεις η έννοια της ομοιότητας να καταρρέει, καθώς όλα τα σημεία απέχουν σημαντικά πολύ μεταξύ τους. Το ζήτημα αυτό είναι ιδιαίτερα σημαντικό, καθώς δημιουργεί προβλήματα και στην κατασκευή αποδοτικών μηχανισμών ευρετηρίου (indexing mechanisms). Τα προβλήματα αυτά, σε συνδυασμό με την υπολογιστική πολυπλοκότητα που έχουν οι πράξεις επάνω στα πολυδιάστατα δεδομένα, κάνουν εμφανή την ανάγκη για αποδοτικούς μηχανισμούς, υποστηριζόμενους από επαρκή επεξεργαστική ισχύ. Από τους πιο διαδεδομένους τρόπους για την αύξηση της διαθέσιμης επεξεργαστικής ισχύος ενός συστήματος είναι η χρήση συμπλέγματος (clusters) ή παράλληλων υπολογιστών. Οι τεχνικές αυτές οδηγούν σε συστήματα γνωστά και ως «Παράλληλες» ή «Κατανεμημένες Βάσεις Δεδομένων» (Parallel / Distributed Databases). Στις περιπτώσεις αυτές, σημαντικό ρόλο στη συνολική απόδοση του συστήματος παίζει και ο συντονισμός των επιμέρους μονάδων, εκτός από την εσωτερική λειτουργία του καθενός μεμονωμένα. Για παράδειγμα, ένα κατανεμημένο σύστημα το οποίο κατευθύνει όλες τις ερωτήσεις προς ένα και μοναδικό από τα υποσυστήματα του, αφήνοντας όλα τα υπόλοιπα σε αδράνεια είναι ισοδύναμο με ένα σύστημα που αποτελείται από μια και μόνο υπολογιστική μονάδα. Αντιθέτως, ένα ισορροπημένο σύστημα, που κατανέμει ισοδύναμα την εργασία μεταξύ των υποσυστημάτων του είναι σαφώς προτιμότερο, αφού θα απαντήσει γρηγορότερα. Βέβαια, καθώς ο συντονισμός μεταξύ των επιμέρους υποσυστημάτων του , απαιτεί την μεταξύ τους επικοινωνία, η οποία επηρεάζει (επιβραδύνει) την απόκριση του συστήματος, η άσκοπη χρήση ενός κατανεμημένου συστήματος Βάσεων Δεδομένων θα μειώσει κι άλλο την τελική του απόδοση.

Λόγω λοιπόν της ύπαρξης των πολυδιάστατων δεδομένων έχουν μελετηθεί πολλές μέθοδοι για τη λήψη απόφασης με πολλαπλά κριτήρια. Οι μέθοδοι αυτές επιτρέπουν τη συγκριση διαφορετικών σεναρίων ή επιλογών με βάση πολλαπλά κριτήρια, συχνά αλληλοσυγκρουόμενα, έτσι ώστε να μπορεί να ληφθεί μια συνετή απόφαση.

Κατ’ επέκταση, στα σημερινά συστήματα δεδομένων είναι πολύ σημαντικό να μπορούν να εξαχθούν κάποιες πλειάδες από τη βάση δεδομένων για την υποστήριξη της λήψης αποφάσεων με πολλαπλά κριτήρια. Κάτι τέτοιο έχει μεγάλη εφαρμογή σε πολλά πεδία εφαρμογών όπου οι τελικοί χρήστες ενδιαφέρονται κυρίως για τις πιο σημαντικές απαντήσεις σε ερωτήματα (queries) σε ένα πιθανά

τεράστιο χώρο απαντήσεων. Με αυτό το σκεπτικό θεωρούμε τη βάση δεδομένων ως ένα μεγάλο σύνολο πολυδιάστατων σημείων αποθηκευμένων στο δίσκο και στόχος είναι οι χρήστες να μπορούν να βρουν γρήγορα το πιο επιθυμητό σημείο.

Μια ιδιαιτέρα χρήσιμη και διαδεδομένη κατηγορία ερωτημάτων που τίθεται σε πολυδιάστατα δεδομένα είναι τα λεγόμενα ερωτήματα κορυφογραμμής (skyline queries). Τα ερωτήματα κορυφογραμμής αναζητούν εκείνες τις πλειάδες μιας σχέσης για τις οποίες δεν υπάρχει καμία άλλη πλειάδα που είναι καλύτερη από αυτές σε όλες τις διαστάσεις, ή όπως συνηθίζεται να λέγεται δεν υπάρχει καμία άλλη πλειάδα που να κυριαρχεί (dominates) πάνω στις πρώτες.

1.2 Αντικείμενο Διπλωματικής

Στόχος λοιπόν, για την παρούσα διπλωματική ήταν η υλοποίηση και η πειραματική μελέτη μεθόδων λήψης απόφασης με πολλαπλά κριτήρια, σε βάσεις δεδομένων, μέσω ερωτημάτων κορυφογραμμής. Για το λόγο αυτό, έπρεπε να μελετηθεί εκτενώς το θεωρητικό υπόβαθρο σχετικά με τα ερωτήματα κορυφογραμμής, αλλά και των υπαρχουσών μεθόδων λήψης απόφασης. Στη συνέχεια, έπρεπε να υλοποιηθούν δύο από τις μεθόδους αυτές λαμβάνοντας υπόψη την περίπτωση που τα δεδομένα υπερβαίνουν τη διαθέσιμη μνήμη του συστήματος. Τελευταίο στάδιο**,** ήταν η λήψη πειραματικών αποτελεσμάτων και η σύγκριση της απόδοσης των επιλεγμένων μεθόδων.

1.3 Οργάνωση κειμένου

Το κείμενο αποτελείται από 9 κεφάλαια που καλύπτουν πλήρως την ανάπτυξη της εργασίας.

Στο **Κεφάλαιο 2**, παρουσιάζεται το θεωρητικό υπόβαθρο σχετικά με τα ερωτήματα κορυφογραμμής σε βάσεις δεδομένων. Στο **Κεφάλαιο 3**, γίνεται μια εκτενής αναφορά στην υπάρχουσα βιβλιογραφία σχετικά με τις διαφορετικές προσεγγίσεις και τους αλγόριθμους που χρησιμοποιούνται για την εύρεση της κορυφογραμμής. Στα **Κεφάλαια 4 και 5**, αναλύονται οι δύο αλγόριθμοι για την εύρεση της κορυφογραμμής, στους οποίους βασίστηκε η διπλωματική. Στο **Κεφάλαιο 6**, παρατίθενται τα πειράματα που έγιναν σε διαφορετικές βάσεις δεδομένων, με υλοποίηση των αλγορίθμων που παρουσιάστηκαν στα προηγούμενα κεφάλαια. Επίσης, παρατίθενται τα αποτελέσματα αυτών των πειραμάτων, ενώ εξάγονται και συμπεράσματα, σχετικά με την απόδοση των αλγορίθμων αυτών. Στο **Κεφάλαιο 7**, αναλύονται οι λεπτομέρειες υλοποίησης. Αρχικά, περιγράφεται ο κώδικας που υλοποιήθηκε, και στη συνέχεια γίνεται μια σύντομη αναφορά στη πλατφόρμα και τα προγραμματιστικά εργαλεία που επιλέχθηκαν. Στο **Κεφάλαιο 8**, γίνεται μια σύντομη σύνοψη και αναφέρονται οι μελλοντικές επεκτάσεις της διπλωματικής. Τέλος, στο **Κεφάλαιο 9** δίνεται η βιβλιογραφία και

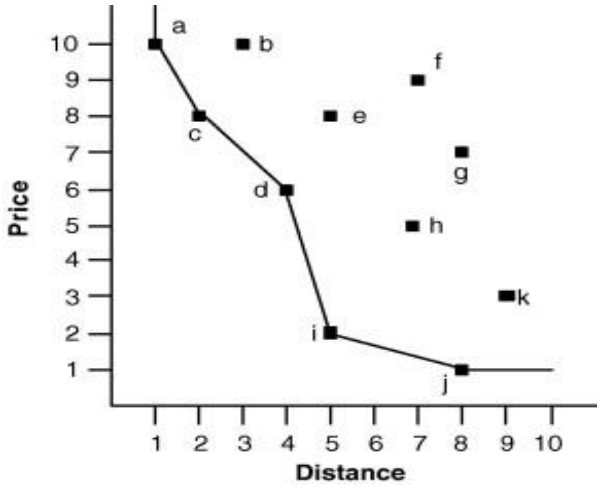
γενικότερα οι πηγές από τις οποίες αντλήθηκαν οι απαραίτητες πληροφορίες για τη συγγραφή της διπλωματικής.

2 *Ερωτήματα κορυφογραμμής*

2.1 Ερωτήματα κορυφογραμμής-Σύνταξη και σημασιολογία

Η σημερινή υποδομή των πληροφοριακών συστημάτων παρέχει στους καταναλωτές της ένα τεράστιο πλήθος πληροφορίας. Οι χρήστες θέλουν να λαμβάνουν αποφάσεις, οι οποίες εξαρτώνται συνήθως από πολύ μεγάλα σύνολα δεδομένων. Μάλιστα, είναι αρκετά συχνό φαινόμενο το να θέλουν να βελτιώσουν τις επιλογές τους αποφασίζοντας με γνώμονα παραπάνω από ένα χαρακτηριστικό των δεδομένων. Εδώ κάνουν την εμφάνιση τους τα ερωτήματα κορυφογραμμής.
Τα ερωτήματα κορυφογραμμής (skyline queries) είναι ένας από τους πολλούς τύπους ερωτημάτων που μπορούν να πραγματοποιηθούν πάνω σε πολυδιάστατα δεδομένα. Τα ερωτήματα αυτά έχουν προσελκύσει το ενδιαφέρον μεγάλου μέρους της επιστημονικής κοινότητας από διάφορους τομείς (Βάσεις Δεδομένων, Υπολογιστική Γεωμετρία, Συστήματα Λήψης Αποφάσεων κ.ά.), λόγω της ιδιαίτερης χρησιμότητας που παρουσιάζουν.

Το skyline ενός συνόλου S αποτελείται από τα σημεία εκείνα τα οποία δεν κυριαρχούνται από κανένα άλλο. Ένα παράδειγμα που χρησιμοποιείται συχνά στη βιβλιογραφία για να περιγράψει τη χρηστικότητα των skyline queries είναι το ακόλουθο: Έστω ότι έχουμε στη διάθεσή μας ένα σύνολο από δεδομένα, που αποτελούνται από σημεία στο δισδιάστατο χώρο. Το κάθε σημείο αντιπροσωπεύει ένα ξενοδοχείο για το οποίο έχουμε (ως πληροφορία) την τιμή ενός δωματίου του και την απόστασή του από τη θάλασσα. Έχοντας τις πληροφορίες αυτές, μπορούμε να απεικονίσουμε τα ξενοδοχεία ως σημεία στο επίπεδο, όπως φαίνεται στην παρακάτω εικόνα.



Ένα ταξιδιωτικό γραφείο είναι μια ακόμα εφαρμογή για την οποία η λειτουργία κορυφογραμμής θα ήταν χρήσιμη. Είναι σαφές, ότι πολλές άλλες εφαρμογές, στον τομέα της λήψης αποφάσεων, μπορούν να βρεθούν, π.χ. η εύρεση καλών πωλητών με χαμηλές αποδοχές. Μια λειτουργία κορυφογραμμής μπορεί να είναι εξίσου χρήσιμη και για την απεικόνιση βάσεων δεδομένων (database visualization). Η κορυφογραμμή του Manhattan, για παράδειγμα, μπορεί να υπολογιστεί ως το σύνολο των κτιρίων που είναι ψηλά και κοντά στον ποταμό Hudson. Με άλλα λόγια, ένα κτίριο κυριαρχεί ενός άλλου αν είναι ψηλότερο, πιο κοντά στον ποταμό Hudson και έχει την ίδια συντεταγμένη x. Αυτό φαίνεται στην παρακάτω εικόνα.

Στο παράδειγμα με τα ξενοδοχεία, το skyline (κορυφογραμμή) ενός ερωτήματος στο συγκεκριμένο σύνολο δεδομένων πρέπει να επιστρέψει όλα τα ξενοδοχεία για τα οποία δεν υπάρχει κάποιο άλλο που να είναι πιο κοντά στη θάλασσα με καλύτερη τιμή. Το ξενοδοχείο α κυριαρχεί το b δεδομένου ότι έχει μικρότερη απόσταση από την παραλία και μικρότερη επίσης τιμή. Επιπλέον, το ξενοδοχείο c κυριαρχεί το e, καθώς αν και τα δύο στοιχίζουν το ίδιο, η απόσταση του c από την παραλία είναι μικρότερη από αυτή του e. Τέλος, αν και το ξενοδοχείο j είναι πολύ μακριά από την παραλία, έχει τη μικρότερη τιμή δωματίου, γι’ αυτό και ανήκει στο τελικό skyline.

Τα skyline ερωτήματα (skyline queries) πρωτοεμφανίστηκαν με την ονομασία «πρόβλημα μέγιστων ανυσμάτων» (maximal vector problem) στις εργασίες των Kung κ.ά. [KLP75], που αντιμετώπιζαν το πρόβλημα στα πλαίσια της Υπολογιστικής Γεωμετρίας. Σημαντικές δουλειές επάνω στο αντικείμενο συνέχισαν να γίνονται από την πρώτη εμφάνιση του προβλήματος κι έπειτα, οι οποίες αποσκοπούσαν κυρίως στην εύρεση αποδοτικών αλγορίθμων υπολογισμού των σημείων που ανήκουν στο skyline, όπως αυτή του Godfrey κ.α. [GGC+02]. Ωστόσο, η πρώτη εφαρμογή των skyline queries στα πλαίσια των Βάσεων Δεδομένων είναι αυτή των Borzsonyi κ.ά. [BKS01], όπου γίνεται λόγος για τον τελεστή κορυφογραμμής (skyline operator). Στη συγκεκριμένη τους εργασία παρουσιάζονται δύο διαφορετικοί αλγόριθμοι υπολογισμού του skyline των δεδομένων που βρίσκονται αποθηκευμένα σε μία Βάση Δεδομένων, προτείνοντας, επιπλέον, μια επέκταση της SQL με τον τελεστή SKYLINE OF, ώστε οι υπολογισμοί να πραγματοποιούνται κατά τη διάρκεια της ανάκτησης των δεδομένων και όχι αργότερα.

Η σύνταξη των Skyline ερωτημάτων (skyline queries), όπως ορίστηκε από τους Borzsonyi κ.ά. [BKS01]είναι η εξής:

SELECT…FROM…
WHERE
GROUP BY…HAVING…
SKYLINE OF [DISTINCT] d_1 [MIN|MAX|DIFF],…, d_m [MIN|MAX|DIFF]
ORDER BY…

Τα $d_1,\ldots,d_m$,δηλώνουν τις διαστάσεις της κορυφογραμμής (skyline), για παράδειγμα την τιμή ενός δωματίου, την απόσταση του από τη θάλασσα κ.α. Τα MIN, MAX και DIFF δηλώνουν αν η τιμή σ' αυτήν τη διάσταση πρέπει να ελαχιστοποιηθεί, να μεγιστοποιηθεί ή απλά να είναι διαφορετική. Στο προηγούμενο παράδειγμα με τα ξενοδοχεία, η τιμή ενός δωματίου θέλουμε να ελαχιστοποιηθεί (MIN επισημείωση), ενώ η αξιολόγηση του θέλουμε να μεγιστοποιηθεί (MAX επισημείωση). Αντίθετα, στο παράδειγμα μας με τον υπολογισμό της κορυφογραμμής των κτιρίων του Manhattan, μπορούμε να δούμε δύο κτίρια παρόλο που έχουν διαφορετικές συντεταγμένες για τη διάσταση x, και γι αυτόν το λόγο και τα δύο μπορεί να ανήκουν στο skyline. Σαν αποτέλεσμα, η διάσταση x αναφέρεται στη SKYLINE OF πρόταση του ερωτήματος με μια DIFF επισημείωση. Το προαιρετικό DISTINCT καθορίζει πως θα διαχειριστούμε τα διπλότυπα.

Η σημασιολογία του SKYLINE OF είναι ξεκάθαρη. Η SKYLINE OF πρόταση εκτελείται μετά το SELECT … FROM … WHERE GROUP BY … HAVING … μέρος του ερωτήματος, αλλά πριν από την ORDER BY πρόταση και είναι πιθανό να ακολουθούν και άλλες προτάσεις μετά από αυτό. Το SKYLINE OF επιλέγει τις ενδιαφέρουσες πλειάδες, δηλάδή τις πλειάδες που δεν κυριαρχούνται από καμμία άλλη.

Επεκτείνονστας τον παραπάνω ορισμό, η πλειάδα p= (p_1,..., p_k, p_{k+1},...., p_l, p_{l+1},..., p_m, p_{m+1},..., p_n) κυριαρχεί της q= (q_1,..., q_k, q_{k+1},...., q_l, q_{l+1},..., q_m, q_{m+1},..., q_n) για ένα skyline ερώτημα "SKYLINEOF d_1 MIN,…, d_k MIN, d_{k+1} MAX,…, d_1 MAX, d_{l+1} DIFF,…, d_m DIFF", αν ισχύουν οι παρακάτω τρεις συνθήκες:

- $p(d_i) \leq q(d_i)$, για όλα τα i=1,…,k
- $p(d_i) \geq q(d_i)$, για όλα τα i =(k+1),…,l
- $p(d_i) = q(d_i)$, για όλα τα i=(l+1),…,m

Στην παρακάτω είκονα παρουσιάζονται παραδείγματα χρήσης του SKYLINE OF query.

```
SELECT *
FROM Hotels
WHERE city = 'Nassau'
SKYLINE OF price MIN, distance MIN;
```

Query 1

```
SELECT *
FROM Buildings
WHERE city = 'New York'
SKYLINE OF distance MIN, height MAX,
          x DIFF;
```

Query 2

```
SELECT e.name, e.salary, sum(s.volume) as volume
FROM Emp e, Sales s
WHERE e.id = s.repr AND s.year = 1999
GROUP BY e.name, e.salary
SKYLINE OF e.salary MIN, volume MAX;
```

Query 3

```
SELECT name, distance,
       (CASE WHEN price ≤ 50 THEN 'cheap'
             WHEN price > 50 THEN 'exp') AS pcat
FROM Hotels
WHERE city = 'Nassau'
SKYLINE OF pcat MIN, distance MIN;
```

Query 4

Αν $p_i = q_i$ για όλα τα i=1,..,m, τότε τα p και q δεν μπορούν να συγκριθούν και μπορεί και τα δύο να ανήκουν στο skyline, αν δεν υπάρχει πουθενά DISTINCT. Με το DISTINCT διατηρείται ένα από τα p και q. Οι τιμές των γνωρισμάτων $d_{m+1},...,d_n$ δεν έχουν καμία σημασία για τον υπολογισμό του skyline, χωρίς να σημαίνει όμως ότι δεν ανήκουν στο skyline.

Ο υπολογισμός ενός μονοδιάστατου skyline δεν παρουσιάζει κανένα ενδιαφέρον, καθώς είναι ισοδύναμο με του υπολογισμό του MIN, MAX ή DIFF. Ο υπολογισμός του skyline είναι επίσης εύκολος αν το SKYLINE OF περιλαμβάνει μόνο δύο διαστάσεις. Ένας δισδιάστατος skyline μπορεί να υπολογιστεί ταξινομώντας τα δεδομένα. Αν τα δεδομένα είναι τοπολογικά ταξινομημένα, ο έλεγχος για το αν μια πλειάδα είναι κομμάτι της κορυφογραμμής έχει πολύ χαμηλό κόστος, αφού χρειάζεται να ελέγχουμε κάθε φορά μια πλειάδα μόνο με την προηγούμενη της.

Αν η ταξινόμηση χρειάζεται δύο ή περισσότερα περάσματα, γιατί τα δεδομένα δε χωρούν στην κύρια μνήμη, τότε οι πλειάδες μπορούν να περιοριστούν στην αρχή κάθε περάσματος και στη φάση συγχώνευσης κατά την ταξινόμηση. Εξαλείφοντας πλειάδες κατά τη διάρκεια της εκτέλεσης κάνουμε

τα περάσματα μικρότερα και περιορίζουμε αρκετά τις προσβάσεις εισόδου/εξόδου (I/O) στο δίσκο. Η απαλειφή των πλειάδων από τόσο νωρίς είναι ανάλογη με τη χρήση του συναθροίσματος από νωρίς , που χρησιμοποιείται για να βελτιώσει την απόδοση των group-by πράξεων.

2.2 Ορισμός skyline και κυριαρχίας

Το skyline ενός συνόλου σημείων ορίζεται ως τα σημεία που δεν κυριαρχούνται (not dominated) από κανένα άλλο σημείο. Ένα σημείο p κυριαρχεί ενός άλλου σημείου q, αν το p δεν είναι χειρότερο σε καμία διάσταση (από αυτές που μας ενδιαφέρουν) από το q και είναι καλύτερο σε τουλάχιστον μία.
Ας θεωρήσουμε έναν d-διάστατο χώρο, όπου το D_i δηλώνει το πεδίο τιμών της i-οστής διάστασης $i \in \{1,...,d\}$. Κάθε πεδίο τιμών D_i είναι ολικά ταξινομημένο με την < προτεραιότητα στις τιμές του πεδίου τιμών. Θεωρούμε τις τιμές $u, v \in D_i$, όπου το u είναι προτιμότερο από το v, αν και μόνο αν u<v. Το πεδίο τιμών P περιέχει N=|P| d-διάστατα σημεία. Κάθε σημείο $p \in P$ στον χώρο $D = D_1 \times D_2 \times ....D_d$,αναπαριστάται από τις συντεταγμένες $p = (p^1, p^2, ..., p^d)$.

Ορισμός 1:Κυριαρχία (Dominance)

Έστω, $p_1, p_2 \in P$ και $i, j \in \{1,...,d\}$. Ένα σημείο p_1 κυριαρχεί ένα άλλο σημείο p_2 και συμβολίζεται ως $p_1 \succ p_2$, αν και μόνο αν: i) για όλες τις διαστάσεις p_1^i είναι περισσότερο ή ισοδύναμα προτιμότερο από το p_1^i, δηλαδή $\forall i : p_1^i < p_2^i$ και ii) σε τουλάχιστον μια διάσταση έστω j, το p_1^j είναι προτιμότερο από το p_2^j.

Ένα σημείο που δεν κυριαρχείται από κανένα άλλο στο σύνολο δεδομένων ονομάζεται σημείο του skyline. Διαισθητικά, δεν προτιμούμε ένα μη-skyline σημείο από ένα skyline για κάθε συνάρτηση ταξινόμησης που είναι μονότονη σε κάθε διάσταση. Με άλλα λόγια το skyline περιέχει το κορυφαίο 1-σημείο για κάθε συνάρτηση ταξινόμησης και εναλλακτικά για ένα δεδομένο skyline σημείο υπάρχει πάντα μια συνάρτηση για την οποία το σημείο αυτό είναι το κορυφαίο 1.

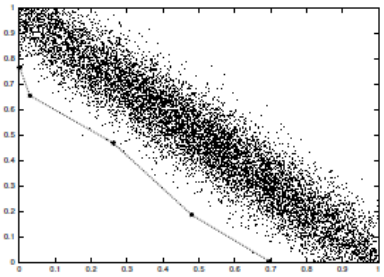
Ορισμός 2: Skyline

Το skyline του P, που δηλώνεται ως SL(P) είναι το σύνολο των σημείων του P που δεν κυριαρχούνται από κανένα άλλο σημείο του P δηλαδή, $SL(P) = \{p_1 \in P \mid \nexists p_2 \in P : p_2 \prec p_1\}$.

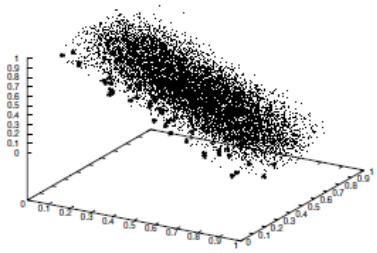
Ένα skyline ενός συνόλου ιδιοτήτων δεν μπορεί να υπολογιστεί από τα skyline των υποσυνόλων των γνωρισμάτων του. Αυτό συμβαίνει, γιατί γενικά (skyline of $a_1, a_2, ..., a_k \cup$ skyline of $a_{k+1}, ..., a_n) \subseteq$ (skyline of $a_1, ..., a_n$).

Ο τελεστής skyline είναι ολιστικός, με την έννοια του ολιστικού αθροίσματος τελεστών. Αυτό υπονοεί ότι τον skyline τελεστή δεν τον ενδιαφέρει γενικά η σειρά των επιλογών.

Στα παρακάτω σχήματα απεικονίζονται το skyline δεδομένων στο δισδιάστατο και στον τρισδιάστατο χώρο αντίστοιχα.



(a) 2-dimensional



(b) 3-dimensional

2.2.1 Ιδιότητες της κυριαρχίας

Δοθέντος ενός συνόλου S σημείων και n συναρτήσεων ταξινόμησης F={f_1,f_2,…,f_n}, μπορεί να οριστεί μία μερική διάταξη των στοιχείων του S σύμφωνα με τον ορισμό της κυριαρχίας, όπου το p κυριαρχεί του q ως προς f, αν και μόνο αν:

$$p \succ_f q :\Leftrightarrow (i)\forall f \in F : f(p) \geq f(q), \kappa\alpha\iota$$
$$(ii)\exists f \in F : f(p) > f(q).$$

Δηλαδή, σύμφωνα με τον ορισμό, για να κυριαρχεί το p του q, θα πρέπει να είναι τόσο καλό όσο το q για όλες τις συναρτήσεις ταξινόμησης και καλύτερο τουλάχιστον σε μία. Αν για δύο αντικείμενα δεν ισχύει $p \succ q$, αλλά ούτε $q \succ p$, τότε λέμε ότι τα σημεία αυτά είναι μη συγκρίσιμα.

Η σχέση κυριαρχίας ικανοποιεί τις εξής ιδιότητες:

- Μη αυτοπαθής: p $\not\succ$ p, ∀p∈S.
- Αντι-συμμετρική: Αν ισχύει $p \succ q$, τότε q $\not\succ$ p.
- Μεταβατική: Αν $p \succ q$ και $q \succ v$, τότε ισχύει $p \succ v$.

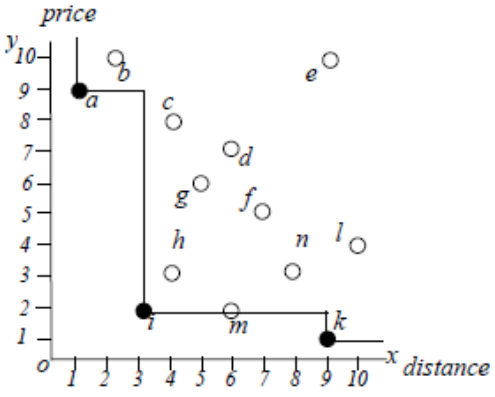
Οι παραπάνω ιδιότητες αποδεικνύονται πολύ εύκολα από τον ορισμό της κυριαρχίας.

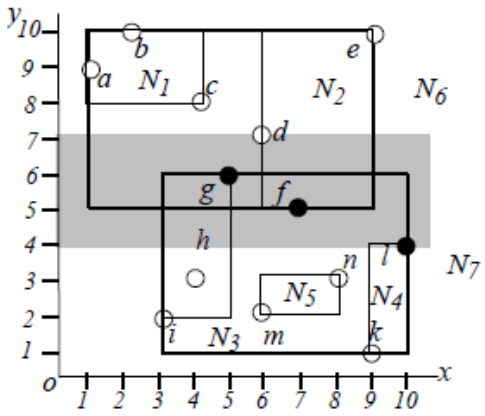
2.3 *Skyline παραλλαγές*

Στην ενότητα αυτή θα παρουσιαστούν κάποιες παραλλαγές των skyline queries, που παρουσιάστηκαν αρχικά στην εργασία των Παπαδιάς κ.α.[PTF+05].

2.3.1 Περιορισμένα skyline ερωτήματα (Constrained Skyline queries)

Όπως παρουσιάστηκε στην εργασία των Παπαδιάς κ.α. [PTF+05], δοθέντος ενός συνόλου από περιορισμούς, ένα constrained skyline ερώτημα επιστρέφει τα πιο ενδιαφέροντα σημεία στο χώρο των δεδομένων που ορίζεται από τους περιορισμούς. Συνήθως, κάθε περιορισμός εκφράζεται σαν ένα πεδίο κατα μήκος μιας διάστασης και η σύνδεση όλων των περιορισμών σχηματίζει ένα υπέρ-ορθογώνιο στον d-διάστατο χώρο των γνωρισμάτων. Ας θεωρήσουμε λοιπόν και πάλι το παράδειγμα για την κατάλληλη επιλογή ξενοδοχείου, λαμβάνοντας υπόψη το σύνολο δεδομένων που φαίνεται στην παρακάτω αριστερή εικόνα. Τώρα θεωρούμε ότι ένας χρήστης ενδιαφέρεται μόνο για τα ξενοδοχεία εκείνα των οποίων η τιμή (άξονας y) είναι μέσα στο εύρος 4-7. Σ'αυτήν την περίπτωση το skyline περιέχει μόνο τα σημεία g, f, και l, όπως φαίνεται παρακάτω δεξιά, και όχι τα σημεία a, i, k, που θα είχαν επιλέγει διαφορετικά. Τα ξενοδοχεία g, f, και l, λοιπόν, είναι τα πιο ενδιαφέροντα στο συγκεκριμένο εύρος τιμών.





2.3.2 Δυναμικό skyline (Dynamic skyline queries)

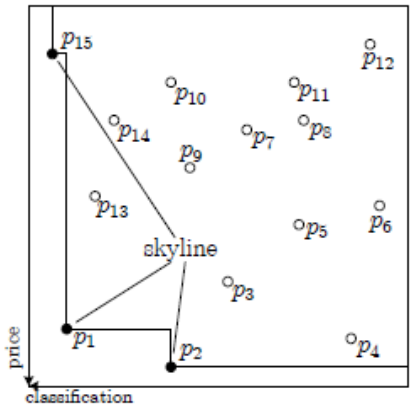
Σύμφωνα, με την εργασία αρχικά των Παπαδιάς κ.α.[PTF+05] και κατόπιν των Σαχαρίδης κ.α [SBS08], δεδομένου ενός query σημείου q , όχι απαραίτητα μέσα στο σύνολο δεδομένων P, το δυναμικό skyline ερώτημα (query) βρίσκει όλα τα σημεία στο P που δεν κυριαρχούνται δυναμικά από κάποιο άλλο σημείο στο P. Ένα σημείο κυριαρχεί δυναμικά ένα άλλο έχοντας ως αναφορά το q ,αν έχει τιμές πιο

κοντά στο q σε όλες τις διαστάσεις και έχει τιμές αυστηρά πιο κοντά στο q σε τουλάχιστον μια διάσταση.

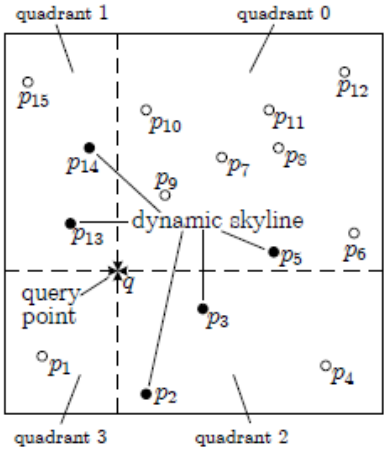
Τα δυναμικά skyline είναι χρήσιμα σε μια ποικιλία καταστάσεων, όπου οι προτιμήσεις ορίζονται σε σχέση με ένα υπόδειγμα. Επίσης, χρησιμεύουν και σαν ένας βασικός φραγμός/όριο για πιο περίπλοκα queries.

Ένα δυναμικό skyline μπορεί να μετατραπεί σε ένα στατικό skyline query υποκύπτοντας σε αλλαγές σε όλες τις συντεταγμένες των σημείων. Συγκεκριμένα, δοθέντος ενός query q, ένα σημείο p μετασχηματίζεται σ'ένα σημείο p', έτσι ώστε η i-ιοστή συντεταγμένη του p' να υπολογίζεται ως $p'^i = | p^i - q^i |$. Άρα, οποιαδήποτε μέθοδος έχει σχεδιαστεί για ένα υποτυπώδες skyline query μπορεί να εφαρμοστεί και σε μια δυναμική περίπτωση. Αξίζει να σημειώσουμε ότι όλες αυτές οι μέθοδοι έχουν σχεδιαστεί ώστε να βρίσκουν λύση για ένα μόνο στιγμιότυπο, δεδομένου ενός μοναδικού συνόλου δεδομένων. Οπότε, στη δυναμική περίπτωση όπου υπάρχουν πολλαπλά στιγμιότυπα-ένα για κάθε query-ο αλγόριθμος πρέπει να εφαρμοστεί από την αρχή κάθε φορά, εξετάζοντας όλα τα σημεία για κυριαρχία.

Ας θεωρήσουμε έναν πίνακα που περιέχει δεδομένα για ξενοδοχεία, με γνωρίσματα το όνομα του ξενοδοχείου (Name), την τιμή των δωματίων (Price) και την κατηγορία του ξενοδοχείου (Classification). Αν λάβουμε υπόψη μόνο την τιμή, ένα ξενοδοχείο είναι προτιμότερο, αν είναι φθηνό. Ομοίως, αν λάβουμε υπόψη την κατηγορία του, ένα ξενοδοχείο περισσότερων αστέρων είναι προτιμότερο. Στην παρακάτω εικόνα φαίνονται 15 ξενοδοχεία που έχουν σχεδιαστεί σαν σημεία στο δισδιάστατο χώρο, όπου η διάσταση x είναι η κατηγορία του ξενοδοχείου και η διάσταση y είναι η τιμή του. Τα βέλη στους άξονες δείχνουν την κατεύθυνση της προτίμησης σε κάθε διάσταση. Παρατηρούμε ότι δεν υπάρχει κάποιο ξενοδοχείο που να κυριαρχεί τα p_1, p_2 και p_{15} και άρα ανήκουν στο skyline. Απ' την άλλη το p_3 δεν μπορεί να είναι στο skyline γιατί κυριαρχείται και από το p_1, αλλά και από p_2. Μάλιστα, όλα τα ξενοδοχεία που βρίσκονται στη δεξιά πλευρά της γραμμής που συνδέει τα skyline ξενοδοχεία, κυριαρχούνται από κάποιο άλλο.



Με βάση τώρα το παραπάνω παράδειγμα, ας υποθέσουμε ότι ένας χρήστης ψάχνει να βρει τα ξενοδοχεία εκείνα που ανταποκρίνονται στα χρήματα που θέλει να διαθέσει και στα πρότυπα του. Για το λόγο αυτό καθορίζει το ιδανικό του ξενοδοχείο «q» και θέλει να βρει όλα τα ξενοδοχεία που είναι παρεμεφερή αυτού, όσον αφορά την τιμή και την κατηγορία. Στην παρακάτω εικόνα φαίνεται το query σημείο q και το δυναμικό skyline με βάση αυτό. Το δυναμικό ερώτημα κορυφογραμμής με βάση το q επιστρέφει, λοιπόν, τα ξενοδοχεία $p_2, p_3, p_5, p_{13}, p_{14}$, όπως φαίνεται και στην εικόνα. Παρατηρούμε τώρα ότι το p_9 δεν είναι στο skyline, γιατί το p_{13} είναι πιο κοντά στο q σε όλες τις διαστάσεις.



2.3.2.1 Ορισμοί Δυναμικού skyline

Σύμφωνα με τους ορισμούς της κυριαρχίας και του skyline, το πιο επιθυμητό σημείο είναι η αρχή των αξόνων $o = (0,...,0)$, υποθέτοντας βέβαια ότι υπάρχει μέσα στο P, αφού κυριαρχεί όλα τα άλλα σημεία. Παρολαυτά, σε πολλές περιπτώσεις το πιο επιθυμητό σημείο μπορεί να είναι ένα σημείο q που έχει οριστεί από το χρήστη. Σ'αυτή την περίπτωση, χρειάζεται να εκφράσουμε τις ιδέες της προτίμησης και της κυριαρχίας σχετικά με το q.

Δεδομένου ενός σημείου $q = (q^1, q^2, ..., q^d) \in D$ (όχι απαραίτητα μέσα στο P) η τιμή $u \in D_i$ για μερικά i είναι προτιμότερη από την τιμή $v \in D_i$ αν και μόνο αν $|u - q^i| \leq |v - q^i|$. Βασιζόμενοι σ' αυτήν την ιδέα της προτίμησης ,έχουμε τους παρακάτω ορισμούς [SBS08].

Ορισμός: Δυναμική Κυριαρχία (Dynamic Dominance)

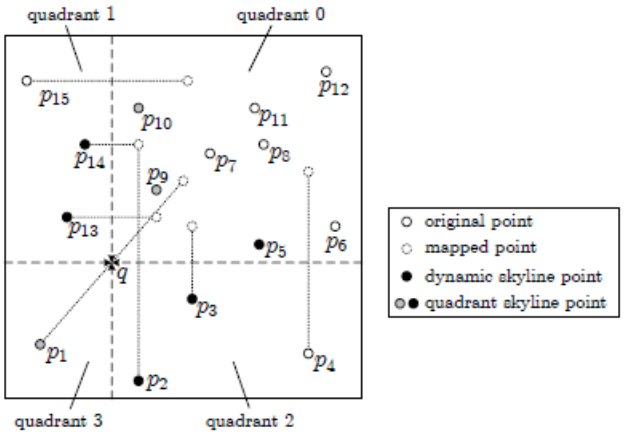
Έστω $p_1 \prec p_2 \in P$, $i, j \in \{1, ..., d\}$ και $q \in D$. Δοθέντος ενός query q, ένα σημείο p1 κυριαρχεί δυναμικά έχοντας ως αναφορά το q, ένα άλλο σημείο p2 και συμβολίζεται ως $p_1 \prec_q p_2$, αν και μόνο αν, i) για όλες τις διαστάσεις το p_1^i είναι περισσότερο ή εξίσου προτιμότερο από το p_2^i, έχοντας ως αναφορά το q, δηλαδή $\forall i : |p_1^i - q^i| \leq |p_2^i - q^i|$ και ii) σε τουλάχιστον μια διάσταση έστω j, το p_1^j είναι αυστηρά προτιμότερο από το p_2^j έχοντας σαν αναφορά το q δηλαδή, $\exists j : |p_1^j - q^j| \leq |p_2^j - q^j|$.

Ορισμός: Δυναμικό Skyline (Dynamic Skyline)

Δοθέντος ενός σημείου query $q \in D$ το δυναμικό skyline του P έχοντας σαν αναφορά το q, που συμβολίζεται ως DSL(P,q), είναι το σύνολο των σημείων του P που δεν κυριαρχούνται δυναμικά από κανένα άλλο σημείο στο P έχοντας ως αναφορά το q, δηλαδή $SL(P) = \{p_1 \in P \mid \nexists p_2 \in P : p_2 \prec_q p_1\}$.

Τα δυναμικά πανομοιότυπα της ιδέας του dominance και του skyline στην ουσία αναφέρονται στις γενικές ιδέες που εφαρμόζονται στο μετασχηματισμένο σύνολο δεδομένων P' που προκύπτει αντιστοιχώντας κάθε σημείο $p = (p^1, p^2, ..., p^d) \in P$ στο σημείο p'=$(|p^1 - q^1|, ..., |p^d - q^d|)$, δοθέντος ενός query σημείου q. Κάθε σημείο p , όπου $p^j - q^j < 0$ για τουλάχιστον μια διάσταση έχει αντιστοιχηθεί μ'ένα σημείο p' στο άνω δεξί τεταρτημόριο με αναφορά το q, όπως φαίνεται στην παρακάτω εικόνα. Η αντιστοίχιση φαίνεται με μια διακεκομμένη γραμμή και το σημείο που έχει αντιστοιχηθεί φαίνεται με έναν διακεκομμένο κύκλο. Το query σημείο q χωρίζει τον χώρο D σε 4 τεταρτημόρια (2^d ορθομόρια στην d-διάστατη περίπτωση), που ορίζονται έτσι ώστε να είναι ψηλότερα ή χαμηλότερα από από το q^j για κάθε διάσταση j. Ένα ορθομόριο (orthant) μπορεί να αναγνωριστεί από έναν αριθμό γραμμένο σε δυαδική μορφή που περιέχει d bits, όπου το j-όστο bit είναι 0 ή 1, αν η j-οστή διάσταση του ορθομορίου περιλαμβάνει τιμές όχι μικρότερες ή μικρότερες αντίστοιχα από το q^j.

Η ένωση όλων των orthant skyline με αναφορά το q είναι ένα υπερσύνολο του δυναμικού skyline με αναφορά το q.



2.3.2.2 Orthant skyline

Δοθέντος ενός σημείου query $q \in D$ το ο-οστό orthant skyline του P έχοντας ως αναφορά το q, όπου $o \in \{0,...,2^{d-1}\}$, συμβολίζεται ως OSL(P,q,o) και είναι το σύνολο των σημείων του P που ανήκει στο ο-οστό orthant και δεν κυριαρχείται δυναμικά από κανένα άλλο σημείο στο συγκεκριμένο orthant έχοντας ως αναφορά το q.

Σημείωση: ένα orthant skyline σημείο μπορεί να κυριαρχείται από ένα skyline σημείο ενός άλλου orthant και άρα το πρώτο δεν μπορεί να ανήκει στο δυναμικό skyline.

2.3.3 Συναρτήσεις ιεράρχησης/ταξινόμησης (scoring functions)

Η έκφραση των ερωτημάτων με βάση τις προτιμήσεις στο πλαίσιο των σχεσιακών βάσεων δεδομένων μπορεί να πραγματοποιηθεί πέραν από τα skyline queries και με τη βοήθεια συναρτήσεων ιεράρχησης. Η βασική ιδέα είναι να οριστεί μια συνάρτηση πάνω σε μια σχέση και να προκύψει ένα σκορ (score) για κάθε πλειάδα (tuple). Ένα query που εκφράζει μια προτίμηση επιστρέφει τα tuples ιεραρχικά ταξινομημένα ανάλογα με τα score τους.

Ένα σκορ για ένα tuple ορίζεται σαν ο μέσος βαρύτητας πάνω στις τιμές των γνωρισμάτων του: $v_1 * a_1 + ... + v_n * a_n$, όπου $a_1, a_2, ..., a_n$ είναι οι τιμές των ιδιοτήτων $A_1,..., A_n$ και $v_1,...., v_n$ είναι τα βάρη που αναθέτει ο χρήστης στις ιδιότητες. Αυτή η προσέγγιση δεν προϋποθέτει πολλές πληροφορίες από έναν χρήστη, επίσης περιορίζει κατά πολύ τους τύπους προτίμησης που μπορεί να εκφραστούν: εφαρμόζεται μόνο σε αριθμητικές τιμές.

Το κύριο πρόβλημα με αυτήν την προσέγγιση είναι ότι το σκορ ενός tuple αυξάνεται μονότονα και ομοιόμορφα με την τιμή μιας ιδιότητας.

2.3.3.1 Skyline ενάντι Ιεράρχησης

Το skyline μιας σχέσης στην ουσία αναπαριστά τα καλύτερα tuples της σχέσης, τις Pareto βέλτιστες λύσεις, με σεβασμό στα skyline κριτήρια. Ένας άλλος τρόπος για να βρούμε τα καλύτερα tuples είναι να ιεραρχήσουμε/ταξινομήσουμε κάθε tuple με σεβασμό στα κριτήρια ενός και στη συνέχεια να επιλέξουμε εκείνα τα tuples με το καλύτερο σκορ. Το τελευταίο μπορεί να γίνει αποδοτικά σ'ένα σχεσιακό σκηνικό. Με μια διάσχιση του πίνακα μπορούμε να βαθμολογήσουμε τα tuples και να συλλέξουμε τα tuples με το καλύτερο σκορ. Είναι γνωστό ότι το skyline αναπαριστά το όριο πάνω στα tuples της σχέσης με το καλύτερο σκορ.

Το skyline είναι το ελάχιστο άνω όριο των μεγίστων των μονότονων συναρτήσεων ιεράρχησης (scoring functions). Αυτό σημαίνει ότι το skyline μπορεί να χρησιμοποιηθεί έναντι της ιεράρχησης ή ότι μπορεί να χρησιμοποιηθεί σε συνδυασμό με την ιεράρχηση.

Το skyline είναι συνήθως αρκετά μικρότερο από τον πίνακα που εξετάζουμε, οπότε θα ήταν πιο αποδοτικό να δοκιμάζαμε αρκετά queries προτίμησης πάνω στο ίδιο σύνολο δεδομένων. Κατά δεύτερον, καθώς το να ορίσουμε τις προτιμήσεις μας σ'ένα query προτίμησης μπορεί να είναι αρκετά δύσκολο, ενώ το να εκφράσουμε ένα skyline query είναι σχετικά εύκολο οι χρήστες μπορεί να βρουν τα skyline queries αρκετά αποδοτικά. Το skyline απαντά με σεβασμό στην πρόθεση των χρηστών κατά έναν τρόπο αφού περιλαμβάνει τα καλύτερα queries με σεβασμό σε οποιαδήποτε προτίμηση. Οπότε θα υπάρχουν και μερικές επιλογές ανάμεσα στα skyline που δε θα ενδιαφέρουν τον χρήστη. Παρολαυτά εμφανίζονται και όλες οι καλές επιλογές σύμφωνα με τις αυστηρές προτιμήσεις του χρήστη. Τέλος υπάρχουν ενδιαφέρουσες επιλογές που έχουν αυστηρή διαισθητική απήχηση και εμφανίζονται στο skyline, αλλά που είναι εξαιρετικά δύσκολο να βρεθούν με ιεράρχηση.

2.3.3.2 Συναρτήσεις ταξινόμησης/ιεράρχησης (Scoring Functions)

Η έννοια του scoring function παρουσιάστηκε από τους Chomicki κ.α [CGG+02], όπως φαίνεται από τα παρακάτω.

Έστω οι ιδιότητες $a_1, a_2, ..., a_k$ του σχήματος R, είναι τα skyline κριτήρια με σεβασμό στο "max". Έστω ότι τα πεδία τιμών των a_i είναι πραγματικοί αριθμοί και έστω **R** η σχέση του σχήματος R που αναπαριστά ένα δεδομένο στιγμιότυπο.

Ορισμός: Ορίζουμε μια μονότονη συνάρτηση ταξινόμησης S με σεβασμό στο R σαν μια συνάρτηση που παίρνει σαν είσοδο tuples του R και τα αντιστοιχεί σε ένα εύρος πραγματικών αριθμών. Η S αποτελείται από k μονότονες αύξουσες συναρτήσεις $f_1, ..., f_k$. Για κάθε tuple $t \in R$, $S(t) = \sum_{i=1}^{k} f_i(t[a_i])$.

Λήμμα: Κάθε tuple που έχει το καλύτερο σκορ (score) στο R με βάση κάποια μονότονη συνάρτηση ιεράρχησης (scoring function) S με βάση το R πρέπει να βρίσκεται στο skyline.

Ορισμός: Ορίζουμε μια θετική γραμμική συνάρτηση ιεράρχησης (**scoring function**) W σαν μια συνάρτηση πάνω στα tuple του πίνακα R της μορφής : $W(t) = \sum_{i=1}^{k} w_i t[a_i]$, στην οποία όλα τα w_i είναι θετικές, πραγματικές σταθερές.

Θεώρημα: Είναι πιθανό να υπάρχει ένα skyline tuple στο R το οποίο για κάθε θετική, γραμμική συνάρτηση ιεράρχησης (scoring function), δεν έχει τη μέγιστη βαθμολογία με βάση τη συνάρτηση αυτή πάνω στον πίνακα R.

Θεώρημα: Το skyline περιέχει όλα μόνο τα tuples που έχουν τα μεγαλύτερα αποτελέσματα σε μονότονες συναρτήσεις ιεράρχησης (scoring function).

Θεώρημα: Κάθε ολική ταξινόμηση των tuples του R με βάση οποιαδήποτε συνάρτηση ταξινόμησης (ταξινομημένα από το μεγαλύτερο στο μικρότερο) είναι μια τοπολογική ταξινόμηση με σεβασμό στην μερική σχέση skyline κυριαρχίας.

Αν S(r) <S(t) είναι πιθανό να ισχύει $r \prec t$, αλλά σίγουρα ισχύει $t \not\prec r$. Αν S(r) = S(t), τότε $r \prec t$ και $t \preceq r$ (δηλαδή είναι ισοδύναμα με βάση την προβολή τους πάνω στις ιδιότητες του skyline) ή τα r και t είναι μη συγκρίσιμα με σεβασμό στην κυριαρχία. Ένα skyline tuple $t \in R$ είναι ένα tuple τέτοιο ώστε για $r \in R$ να ισχύει: $t \prec r$

Θεωρούμε την ολική ταξινόμηση στο R που παρέχεται από το SQL order by. Αυτή η ολική σειρά είναι μια τοπολογική ταξινόμηση με σεβασμό στην κυριαρχία.

select * from R

order by a_1 desc,…. a_k . desc;

Θεώρημα: Κάθε φωλιασμένη ταξινόμηση του R πάνω στις ιδιότητες του skyline (ταξινόμηση κατά φθίνουσα σειρά) είναι μια τοπολογική ταξινόμηση με σεβασμό στην μερική διάταξη της κυριαρχίας.

Οι συναρτήσεις ταξινόμησης παρουσιάστηκαν και από τους Bartolini κ.α [BCP08], όπως φαίνεται παρακάτω.

Η ιδέα του προ-φιλτραρίσματος ενός συνόλου δεδομένων ονομάζεται scoring. Το scoring αναφέρεται στη μετατροπή ενός σημείου, που είναι ένα πολυδιάστατο αντικείμενο, σ' ένα μονοδιάστατο αντικείμενο, που ονομάζεται score. Αυτή η μετατροπή μπορεί να γίνει με διάφορους τρόπους. Το κοινό στοιχείο σε όλους τρόπους είναι να εφαρμοστεί μια μονότονη αύξουσα συνάρτηση σε όλες τις διαστάσεις του σημείου.

Ας θεωρήσουμε έναν m-διάστατο χώρο $A=[0,1]^m$ τον οποίο ονομάζουμε χώρο απαντήσεων/απόκρισης. Για ένα δοθέν query Q, κάθε αντικείμενο o_i αναπαριστάται μονοσήμαντα στον A από ένα σημείο $s_i=(s_{i,1},....s_{i,m})$, του οποίου οι τιμές των συντεταγμένων είναι τα m μερικά αποτελέσματα για τα m υπο-ερωτήματα.

Ορισμός:(Scoring Function) Μια scoring function στο A είναι μια οποιαδήποτε συνάρτηση $S:A\to[0,1]$, που αναθέτει σε κάθε σημείο $s_i\in A$ μια τιμή $s_i=S(s_i)$, που ονομάζεται το συνολικό αποτέλεσμα του o_i.

Ορισμός: (Μονοτονία των scoring συναρτήσεων) Μια scoring συνάρτηση S, m τάξης είναι μονότονη αν $s_{j,q}\le s_{i,q}$ για όλα τα q σημαίνει ότι $s_j=(s_{j,1},....s_{j,m})\le s_i=(s_{i,1},....s_{i,m})$.

Στην πραγματικότητα οι πιο συχνά χρησιμοποιούμενες scoring συναρτήσεις όπως οι min, max, avg κτλ. , δεν είναι μόνο μονότονες αλλά αυστηρά μονότονες, δηλαδή $s_{j,q}<s_{i,q}$ για όλα τα q σημαίνει ότι $s_j<s_i$.

Στη μονοτονία και στην αυστηρή μονοτονία μπορεί να δοθεί μια απλή αλλά χρήσιμη γεωμετρική ερμηνεία. Για αυτό ας θεωρήσουμε το σημείο-στόχο στο A που ορίζεται ως $1=(1,...,1)$, και αντιστοιχεί στην καλύτερη δυνατή εκτίμηση για όλα τα υπο-ερωτήματα και έστω R_i, το άνω ορθογώνιο που έχει τα σημεία s_i και 1 σαν απέναντι κορυφές. Ονομάζουμε R_i το άνω ορθογώνιο του o_i.

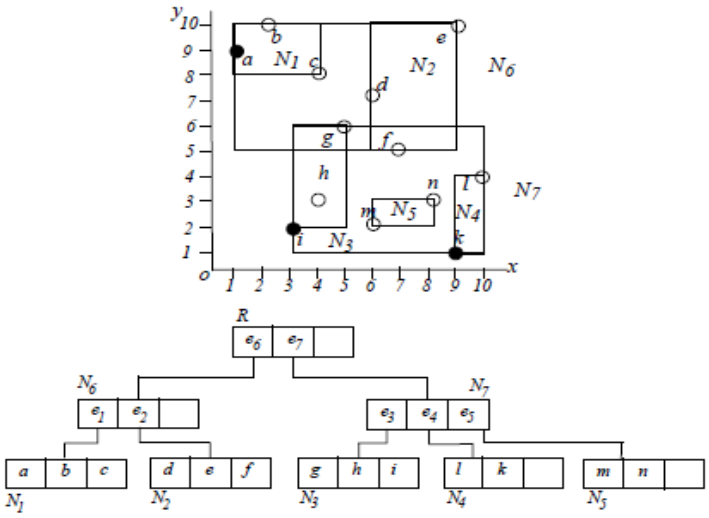
Παρατήρηση: Αν η S είναι (αυστηρά) μονότονη και s_j είναι ένα σημείο του άνω-ορθογωνίου R_i του o_i, τότε $S(s_i)\le S(s_j)$ $(S(s_i)<S(s_j))$.

Σα συμπέρασμα, αν υπάρχουν τουλάχιστον k σημεία του C στο άνω-ορθογώνιο του o_i, τότε δεν υπάρχει αυστηρά μονότονη scoring συνάρτηση που να κάνει το o_i ένα από τα k σημεία με το μεγαλύτερο αποτέλεσμα στο C. Επίσης, αφού οι δεσμοί μπορούν να σπάσουν αυθαίρετα, το o_i μπορεί να παραλειφθεί με ασφάλεια στην περίπτωση των μονότονων scoring συναρτήσεων. Δηλαδή:

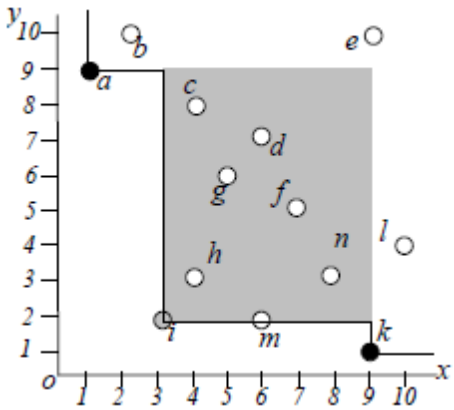
Ορισμός: (k πιθανή καλύτερη αντιστοίχιση) Ένα αντικείμενο o_i για το οποίο υπάρχουν k-1 σημεία του C στο άνω-ορθογώνιο του ονομάζεται k πιθανή καλύτερη αντιστοίχιση. Μια πιθανή 1 καλύτερη αντιστοίχιση ονομάζεται απλά μια πιθανή καλύτερη αντιστοίχιση.

2.3.4 Απαριθμημένα και K-κυριαρχούντα ερωτήματα (Enumerating and K-dominating queries)

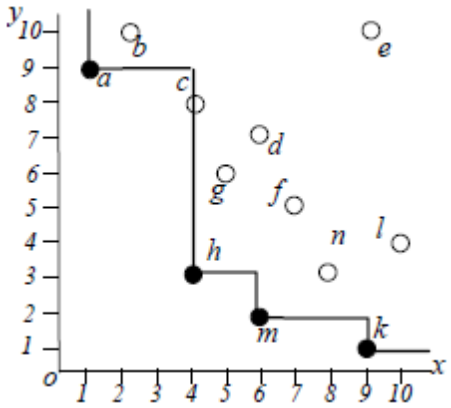
Τα απαριθμημένα ερωτήματα επιστρέφουν για κάθε skyline σημείο p, τον αριθμό των σημείων που κυριαρχούνται από το p. Αυτή η πληροφορία μπορεί να είναι απαραίτητη σε μερικές εφαρμογές καθώς παρέχει κάποιο μέτρο σύγκρισης για τα skyline σημεία. Στο προηγούμενο παράδειγμα με τα ξενοδοχεία, για παράδειγμα το ξενοδοχείο i μπορεί να είναι πιο ενδιαφέρον από τα άλλα skyline σημεία καθώς κυριαχεί 9 ξενοδοχεία σε σχέση με τα a, k που κυριαρχούν μόνο δύο άλλα ξενοδοχεία. Ας ονομάσουμε num(p) τον αριθμό των σημείων που κυριαρχούνται από το σημείο p. Μια προσεγγίση για να επεξεργαστούμε τέτοια ερωτήματα περιλαμβάνει δύο στάδια: (i) πρώτα υπολογίζουμε το skyline, (ii) για κάθε skyline σημείο p εφαρμόζουμε ένα query παράθυρο στο R-δέντρο των δεδομένων και μετράμε τον αριθμό των σημείων num(p) που βρίσκονται μέσα στην περιοχή που κυριαρχεί το p. Παρατηρούμε ότι αφού όλα τα σημεία κυριαρχούνται, όλοι οι κόμβοι του R-δέντρου, που φαίνεται στην παρακάτω εικόνα, θα διασχισθούν από κάποιο ερώτημα. Επιπλέον, λόγω του μεγάλου μεγέθους των περιοχών κυριαρχίας πολυάριθμοι κόμβοι του R-δέντρου θα διασχισθούν από αρκετά παράθυρα ερωτημάτων. Για να αποφύγουμε τις πολλαπλές επισκέψεις ενός κόμβου αντιστρέφουμε τη διαδικασία, δηλαδή διατρέχουμε το αρχείο των δεδομένων και για κάθε σημείο εφαρμόζουμε ένα ερώτημα για το R-δέντρο στην κύρια μνήμη για να βρούμε τις περιοχές κυριαρχίας που περιέχει. Οι αντίστοιχοι μετρητές num(p) των skyline σημείων αυξάνονται μετά αναλόγως.



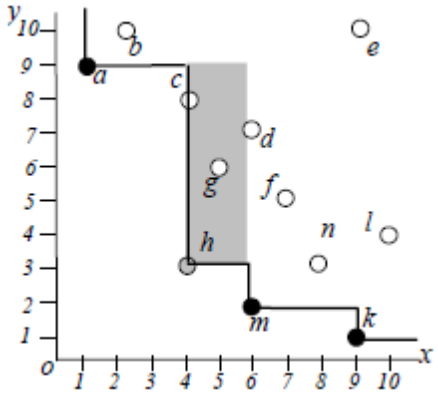
Μια ενδιαφέρουσα παραλλαγή του προβλήματος είναι το K-κυριαρχόν ερώτημα (K-dominating query), που βρίσκει τα K σημεία που κυριαρχούν το μεγαλύτερο αριθμό από τα άλλα. Το ερώτημα αυτό δεν είναι skyline ερώτημα καθώς το αποτέλεσμα δεν περιέχει απαραίτητα skyline σημεία. Αν για παράδειγμα K=3, η έξοδος πρέπει να περιλαμβάνει τα ξενοδοχεία i, h και m, αφού num(i)=9, num(h)=7 και num(m)=5. Για να πάρουμε το αποτέλεσμα εφαρμόζουμε πρώτα ένα ερώτημα απαρίθμησης (enumerating query) που επιστρέφει τα skyline σημεία και τον αριθμό των σημείων που κυριαρχούν αυτά. Αυτή η πληροφορία για τα K=3σημεία εισέρχεται σε μία λίστα που ταξινομείται σύμφωνα με το num(p), δηλαδή η λίστα είναι: list = <i,9>, <a,2>, <k,2>. Το πρώτο σημείο της λίστας, δηλαδή το i είναι το πρώτο απότελεσμα του 3-dominating query. Όποιοδήποτε άλλο σημείο του αποτελέσματος θα πρέπει να βρίσκεται στην περιοχή που κυριαρχεί το i, αλλά όχι στην περιοχή που κυριαρχεί το a ή το k, δηλαδή στη σκιασμένη περιοχή της παρακάτω εικόνας. Διαφορετικά θα κυριαρχεί λιγότερα σημεία από οτι το a ή το k.



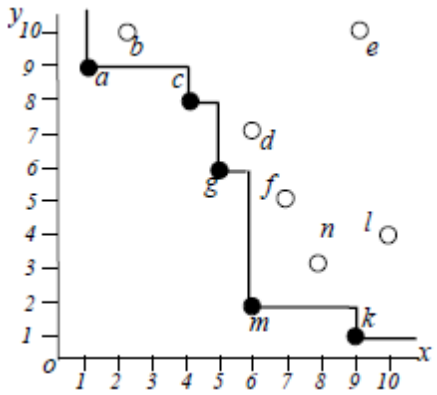
Για να μπορέσουμε να βρούμε τα υποψήφια σημεία εφαρμόζουμε ένα τοπικό skyline ερώτημα S' σ'αυτήν την περιοχή αφού αφαιρέσουμε το i από το S και το βγάζουμε στην έξοδο. Το S' περιέχει τα σημεία h και m. Το νέο skyline $S_1 = (S - \{i\}) \cup S'$ φαίνεται στην παρακάτω εικόνα.



Αφού τα h και m δεν κυριαρχούνται μεταξύ τους μπορεί το καθένα να κυριαρχεί το πολύ 7 σημεία, που σημαίνει οτι είναι υποψήφιοι του 3-dominating query. Για να μπορέσουμε να βρούμε τον ακριβή αριθμό των σημείων που κυριαρχούνται, εφαρμόζουμε ένα ερωτήμα-παράθυρο στα δεδομένα του R-δέντρου χρησιμοποιώντας τις κυριαρχούσες περιοχές του h και του m, σαν ερωτήματα-παράθυρα. Μετά από αυτό το βήμα, τα <h,7> και <m,5> αντικαθιστούν τα προηγούμενα υποψήφια σημεία <a,2> και <k,2> στη λίστα. Το σημείο h είναι το δεύτερο αποτέλεσμα του 3-dominating query και άρα η έξοδος του δίνεται στο χρήστη. Στη συνέχεια, η διαδικασία επαναλαμβάνεται για τα σημεία που ανήκουν στην περιοχή κυριαρχίας του h, αλλά όχι στις περιοχές κυριαρχίας των άλλων σημείων στο S_1, δηλαδή στη σκιασμένη περιοχή της παρακάτω εικόνας.



Το καινούριο skyline $S_2 = (S_1 - \{h\}) \cup \{c, g\}$ φαίνεται παρακάτω.



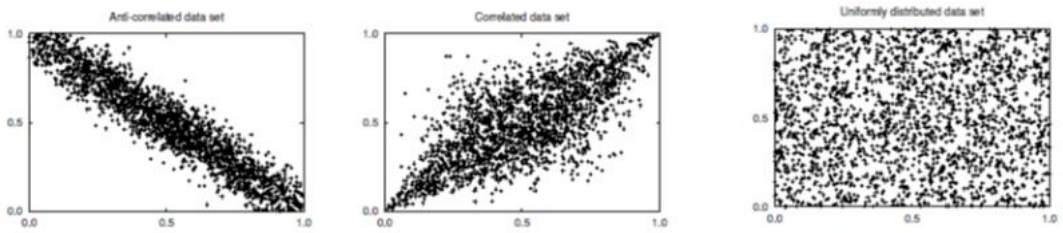
Τα σημεία c και g μπορεί να κυριαρχούν το πολύ 5 σημεία το καθένα, που σημαίνει οτι δεν μπορούν να υπερισχύσουν του m, άρα το ερώτημα τερματίζει με <i,9>, <h,7>, <m,5> σαν τελικό αποτέλεσμα.

2.4 Επίδραση κατανομής δεδομένων στο skyline

Ο αριθμός των αποτελεσμάτων του skyline ενός συνόλου σημείων δεν εξαρτάται μόνο από το αρχικό πλήθος των δεδομένων. Εξαρτάται κυρίως από τη συσχέτιση που έχουν οι τιμές των δεδομένων στις διαστάσεις τους. Τα δεδομένα με βάση το κριτήριο αυτό χωρίζονται στις παρακάτω τρεις κατηγορίες:

- **Independent:** στην κατανομή αυτή οι τιμές των διαστάσεων των δεδομένων παράγονται τυχαία. Με άλλα λόγια δεν υπάρχει καμία συσχέτιση ανάμεσα στις τιμές που έχει ένα σημείο στις επιμέρους διαστάσεις του.
- **Correlated:** εδώ οι τιμές που έχουν τα σημεία στις διαστάσεις τους είναι θετικά συσχετισμένες. Δηλαδή, σημεία που είναι καλά σε μία διάσταση είναι καλά και στις υπόλοιπες διαστάσεις και το αντίθετο. Συνήθως, όταν τα δεδομένα παράγονται με αυτή την κατανομή, τα σημεία που ανήκουν στο skyline είναι πολύ λίγα.
- **Anti-correlated:** οι τιμές των διαστάσεων των σημείων είναι αρνητικά συσχετισμένες. Δηλαδή, τα σημεία που είναι καλά σε μία διάσταση τείνουν να μην είναι καλά στις υπόλοιπες. Όταν τα δεδομένα παράγονται με αυτή την κατανομή, τότε τα σημεία που ανήκουν στο skyline είναι πάρα πολλά.

Η κατανομή των δεδομένων με βάση τις παραπάνω περιπτώσεις φαίνεται στις παρακάτω εικόνες.



(a) Anti-correlated (b) Correlated (c) Uniformly distributed

3 *Αλγόριθμοι εύρεσης κορυφογραμμής*

Ο τελεστής skyline επιστρέφει από ένα σύνολο με πολυδιάστατα σημεία/αντικείμενα ένα υποσύνολο από τα σημεία που υπερτερούν και δεν κυριαρχούνται από τα άλλα. Αυτή η λειτουργία είναι πολύ χρήσιμη για την πολύ-αντικειμενική (πολύ-σημειακή) ανάλυση μεγάλων συνόλων δεδομένων. Παρόλο που έχει προταθεί ένας μεγάλος αριθμός από μεθόδους για την εύρεση της κορυφογραμμής (skyline), η πλειονότητα αυτών εστιάζει στη μείωση του κόστους εισόδου/εξόδου. Σε χώρους πολλών διαστάσεων, όμως, το πρόβλημα μπορεί να γίνει πολύ εύκολα δέσμιο της κεντρικής μονάδας επεξεργασίας (cpu) λόγω του μεγάλου αριθμού υπολογισμών που απαιτούνται για σύγκριση δεδομένων με τα τρέχοντα σημεία του skyline καθώς ανατρέχουμε τη βάση δεδομένων.

Οι πρώτοι αλγόριθμοι που προτάθηκαν, είναι ο divide &conquer (**D&C**) [BKS01] και ο Block nested loop (**BNL**) [BKS01]. Ο D&C διαιρεί το σύνολο των δεδομένων σε επιμέρους τμήματα έτσι ώστε κάθε τμήμα (partition) να χωράει στη μνήμη. Στη συνέχεια, υπολογίζεται ένα κομμάτι της κορυφογραμμής των σημείων σε κάθε τμήμα χρησιμοποιώντας τον αλγόριθμο στην κύρια μνήμη και το τελικό skyline προκύπτει από την συγχώνευση των επιμέρους skyline. Η μέση απόδοση του D&C μειώνεται καθώς η διάσταση (dimensionality) των πλειάδων αυξάνεται. Αυτό συμβαίνει, γιατί καθώς αυξάνεται η διάσταση, τα σημεία που δεν ανήκουν στο skyline έχουν μεγαλύτερη πιθανότητα να ανήκουν στο τοπικό skyline του partition τους και τα μεγέθη των τοπικών αυτών skyline αυξάνουν σημαντικά. Ο BNL διατρέχει το σύνολο των δεδομένων χρησιμοποιώντας έναν buffer, όπου διατηρούνται τα σημεία που δεν κυριαρχούνται από άλλα. Αν ο buffer γεμίσει τότε ελευθερώνεται χώρος σώζοντας μερικά

σημεία σε ένα προσωρινό αρχείο. Μετά το πρώτο πέρασμα τα αντικείμενα που μπήκαν στον buffer πριν να γραφτεί κάποιο σημείο στο προσωρινό αρχείο, είναι βέβαιο οτι ανήκουν στο skyline. Τα υπόλοιπα παραμένουν στον buffer και το προσωρινό αρχείο χρησιμοποιείται σαν είσοδος για μια καινούρια εκτέλεση του BNL. Ο αλγόριθμος αυτός μπορεί να χρειαστεί ένα μεγάλο αριθμό περασμάτων/εκτελέσεων μέχρι να υπολογιστεί το συνολικό skyline.

Μια βελτίωση του BNL είναι ο Sort Filter Skyline (**SFS)** [CGG02], που ταξινομεί πρώτα το σύνολο των δεδομένων τοπολογικά με τη βοήθεια μιας μονότονης συνάρτησης. Η ταξινόμηση, μας εγγυάται ότι τα σημεία δεν μπορούν να κυριαρχούνται από αυτά που ακολουθούν. Σαν αποτέλεσμα κάθε σημείο που μπαίνει στον buffer μπορεί αμέσως να δηλωθεί σαν κομμάτι του skyline. Ο αριθμός των περασμάτων πάνω στα δεδομένα είναι τότε ίσος με το μέγεθος του skyline πάνω στο μέγεθος του buffer. Μια βελτιστοποιημένη παραλλαγή του SFS είναι ο Linear Elimination Sort for Skyline (**LESS**). Ο LESS [GSG06], χρησιμοποιεί ένα μικρό buffer, που ονομάζεται elimination-filter παράθυρο (EF window) στο αρχικό πέρασμα της ρουτίνας εξωτερικής ταξινόμησης του SFS. Το EF παράθυρο διατηρεί ένα μικρό σύνολο σημείων που χρησιμοποιούνται για να μπορούν να διαγραφούν νωρίς όσα σημεία κυριαρχούνται από αυτά. Επιπλέον ο LESS συνδυάζει το τελευταίο πέρασμα της εξωτερικής ταξινόμησης του SFS με τη διαδικασία φιλτραρίσματος του SFS. Στους SFS και LESS όλα τα σημεία πρέπει να εξεταστούν/σαρωθούν τουλάχιστον μία φορά μετά την ταξινόμηση. Ο Sort and Limit skyline αλγόριθμος (**SALSA**) [BCP08] προσπαθεί να αποφύγει το σκανάρισμα ολόκληρου του συνόλου των ταξινομημένων σημείων. Στην αρχή, με μία βέλτιστη συνάρτηση ταξινόμησης ταξινομεί τα σημεία σύμφωνα με την έλαχιστη τιμή των συντεταγμένων τους. Στη συνέχεια κατά τη διάρκεια της διαδικασίας φιλτραρίσματος αυτή η μέθοδος ελέγχει αν όλα τα σημεία στο σύνολο δεδομένων που έχει απομείνει κυριαρχούνται από ένα σημείο που ονομάζεται stop point. Το stop point μπορεί να καθοριστεί σε O(1) χρόνο από τα δεδομένα που έχουν διαβαστεί μέχρι στιγμής. Παρολαυτά, η απόδοση αυτής της μεθόδου επηρεάζεται σημαντικά από την κατανομή των δεδομένων και από την αύξηση της διάστασης τους, καθώς σε προβλήματα υψηλών διαστάσεων η ικανότητα του stop point να διαγράφει δεδομένα είναι περιορισμένη. Γενικά, όλες οι μέθοδοι που βασίζονται στην ταξινόμηση έχουν το πρόβλημα οτι απαιτείται μέγάλος αριθμός υπολογισμών κατά το στάδιο του φιλτραρίσματος, καθώς κάθε σημείο που διαβάζεται πρέπει να συγκρίθεί με τα skyline σημεία στον buffer.

Οι αλγόριθμοι που αναφέρθηκαν παραπάνω δεν απαιτούν τα δεδομένα που λαμβάνουν στην είσοδο να έχουν δεικτοδοτηθεί. Υπάρχουν όμως διάφορες τεχνικές που εκμεταλλεύονται δείκτες (ευρετήρια) σε δεδομένα για να επιταχύνουν τα skyline ερωτήματα. Στην αρχή προτάθηκαν κάποιοι απλοί αλγόριθμοι που χρησιμοποιούν B-δέντρα και R-δέντρα για τον υπολογισμό του skyline. Στη συνέχεια αναπτύχθηκαν δύο προοδευτικές μέθοδοι επεξεργασίας, ο **Bitmap** και ο **Index**. Ο Bitmap [TE01]

κωδικοποιεί όλα τα δεδομένα σε μία bitmap δομή έτσι ώστε το skyline να μπορεί να αναγνωριστεί γρήγορα με μια bitwise and πράξη. Ο Index [TE01] χωρίζει το σύνολο των δεδομένων σε λίστες, δεικτοδοτεί κάθε λίστα με ένα B-δέντρο και χρησιμοποιεί το δέντρο αυτό για να βρεί τα skyline σημεία σε κάθε λίστα, τα οποία συνενώνονται στη συνέχεια αποτελώντας το συνολικό skyline. Σε μεταγενέστερη εργασία παρατηρήθηκε οτι το πιο κοντινό γειτονικό σημείο (nearest neighbor-NN) σε σχέση μ'αυτό που εξετάζεται πρέπει να βρίσκεται μέσα στο skyline. Ο αλγόριθμος **NN** [KRR02] χρησιμοποιεί ένα R-δέντρο για να βρει το nearest neighbor και στη συνέχεια τεμαχίζει τα εναπομείνοντα δεδομένα σε επικαλυπτόμενα τμήματα με βάση το NN. Κατόπιν, βρίσκονται επαναληπτικά οι επόμενοι πιο κοντινοί γείτονες σε κάθε κομμάτι. Απαιτούνται πολλαπλές διασχίσεις του R-δέντρου για να απομακρυνθούν τα διπλότυπα από τις επικαλυπτόμενες περιοχές. Ο Branch and Bound αλγόριθμος (**BBS**) [PTF+03] αποφεύγει αυτές τις παγίδες δίνοντας προτεραιότητα σε προσβάσεις σε κόμβους του R-δέντρου που κυριαρχούνται μερικώς. Ο αλγόριθμος αυτός φαίνεται να είναι I/O βέλτιστος και ανώτερος του NN. Αργότερα προτάθηκε ένας ακόμα αλγόριθμος,ο ZSearch [LZL+07], με βάση ένα **ZBtree** που κωδικοποιεί και συγκεντρώνει όλα τα σημεία με τη βοήθεια μιας Z-τάξης καμπύλης που είναι συμβατή με τη σχέση κυριαρχίας. Οι τεχνικές που βασίζονται στη δεικτοδότηση έχουν συγκεκριμένους περιορισμούς που τις κάνουν χρήσιμες μόνο σε ειδικές περιπτώσεις.

Με στόχο τη βελτίωση της απόδοσης των skyline αλγορίθμων που βασίζονται στην ταξινόμηση προτάθηκε μια δυναμικά δεικτοδοτούμενη τεχνική, ο αλγόριθμος **OSP** (object-space-partitioning). Ο OSP [ZMC09] χωρίζει επαναλαμβανόμενα το χώρο σε 2^d ξεχωριστά τμήματα (partition) έχοντας ως αναφορά ένα skyline σημείο (object) και διευκολύνει την εύρεση του skyline προοδευτικά σε σύνολα δεδομένων πολλών διαστάσεων. Χρησιμοποιώντας αυτό το σχέδιο η μέθοδος οργανώνει τα τρέχοντα skyline σημεία σε ένα δέντρο αναζήτησης, που διευκολύνει τον αποτελεσματικό υπολογισμό του skyline. Κάθε τρέχον σημείο συγκρίνεται με μόνο ένα μικρό αριθμό από τα τρέχοντα skyline σημεία, χρησιμοποιώντας το δέντρο σαν οδηγό αναζήτησης. Τα επιμέρους τμήματα κωδικοποιούνται χρησιμοποιώντας bitmaps και χρησιμοποιείται ένα left–child/right-sibling δέντρο για να τα οργανώσουμε. Το πλεονέκτημα αυτού του δέντρου είναι ότι επιτρέπει μια αποτελεσματική breadth-first αναζήτηση, ενώ η κωδικοποίηση επιτρέπει να γίνονται γρήγορες συγκρίσεις bit προς bit. Ο αλγόριθμος αυτός είναι αρκετές τάξεις πιο γρήγορος από τους υπόλοιπους, μειώνοντας τις απαιτούμενες συγκρίσεις κατά τη διάρκεια εύρεσης του skyline.

Πρόσφατα, κάποιες μελέτες προχώρησαν παραπέρα την εκτίμηση του skyline για ολικώς ταξινομημένα αριθμητικά πεδία και θεώρησαν μερικώς ταξινομημένα πεδία περιλαμβάνοντας κατηγορηματικές και ονομαστικές διαστάσεις. Οι περισσότερες από αυτές υιοθετούν ένα μηχανισμό μερικής προς συνολική αντιστοίχιση και στη συνέχεια εφαρμόζουν κάποιες υπάρχουσες μεθόδους ολικής ταξινόμησης, που

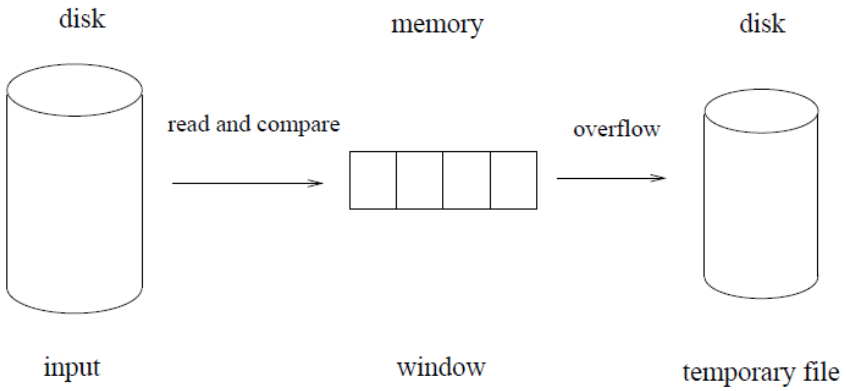
παρολαυτά πάσχουν από το περίπλοκο και μεγάλο μέγεθος των μερικώς ταξινομημένων πεδίων. Τέλος, ο Lattice skyline (**LS**) [MPJ07] αλγόριθμος χρησιμοποιεί μια δικτυωτή δομή (δομή πλέγματος) για να απαντηθούν skyline ερωτήματα με διαστάσεις που προέρχονται από μικρής πληθικότητας πεδία. Η μέθοδος αυτή γίνεται αναποτελεσματική όταν ο αριθμός ή το μέγεθος των πεδίων είναι μεγάλα και δεν εφαρμόζεται όταν υπάρχει ένα πεδίο υψηλής διαστασιμότητας.

Τέλος, υπάρχουν και αλγόριθμοι, οι οποίοι βασίζονται στο κατανεμημένο μοντέλο πρόσβασης.

Κάποιοι από τους παραπάνω αλγόριθμους εξετάζονται αναλυτικά παρακάτω.

3.1 Block Nested Loops Αλγόριθμος (BNL)

Η λειτουργία του BNL, που παρουσιάζεται από τους Borzsonyi κ.α [BKS01], είναι η πιο απλή αλλά και η πιο μη-αποδοτική, καθώς ελέγχει κάθε ένα από τα δεδομένα σε σχέση με όλα τα υπόλοιπα. Η προσέγγιση αυτή μπορεί να χρησιμοποιηθεί και αφού έχει πραγματοποιηθεί η ανάκτηση των δεδομένων, και άρα μπορεί να υλοποιηθεί ως επιπλέον επίπεδο, πάνω σε ένα Σύστημα Διαχείρισης ΒΔ (ΣΔΒΔ). Ο BNL διατηρεί στη μνήμη ένα τμήμα του συνολικού πλήθους των σημείων του skyline, ένα παράθυρο (window), ενώ τα υπόλοιπα διατηρούνται σε κάποιο προσωρινό αρχείο.



Όταν εξετάζεται ένα νέο σημείο p, υπάρχουν οι εξής περιπτώσεις:

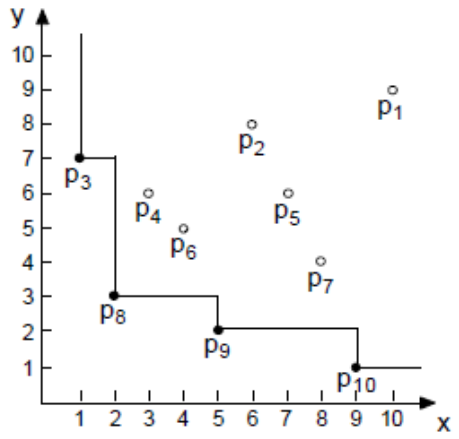
1) Το σημείο p να κυριαρχείται από κάποιο άλλο, το οποίο υπάρχει αυτή τη στιγμή στη μνήμη. Τότε, το σημείο αυτό απαλείφεται, δεν θα εξεταστεί σε επόμενες επαναλήψεις και δε θα εμφανιστεί στην έξοδο.

2) Το σημείο p να κυριαρχεί άλλα σημεία που βρίσκονται ήδη στη μνήμη. Τότε τα σημεία αυτά, απομακρύνονται από τη μνήμη και δε θα εξεταστούν σε επόμενες επαναλήψεις. Το σημείο p προστίθεται στη μνήμη.

3) Το σημείο p να ανήκει στο skyline, συγκρινόμενο με όλα τα σημεία που βρίσκονται στη μνήμη. Αν υπάρχει χώρος, το σημείο p προστίθεται στη μνήμη, διαφορετικά γράφεται σε ένα προσωρινό αρχείο, το οποίο θα υποστεί επεξεργασία σε επόμενες επαναλήψεις.

Στο τέλος της κάθε επανάληψης, μπορούμε να δώσουμε στην έξοδο όλα εκείνα τα σημεία τα οποία βρίσκονται στη μνήμη (στο παράθυρο) και ελέγχθηκαν ως προς όλα τα άλλα σημεία του συνόλου. Τα υπόλοιπα δεδομένα που βρίσκονται στη μνήμη μπορούν να δοθούν στην έξοδο (εφόσον εξακολουθούν να είναι στη μνήμη), στην επόμενη επανάληψη του αλγορίθμου. Μόλις ελεγχθούν όλα τα σημεία από την αρχική είσοδο, αλλάζουμε την είσοδο του αλγορίθμου στο προσωρινό αρχείο που δημιουργήθηκε και πηγαίνουμε στην επόμενη επανάληψη, όπου εκτελείται ο ίδιος αλγόριθμος.

Id	Coordinates
p_1	$(10, 9)$
p_2	$(6, 8)$
p_3	$(1, 7)$
p_4	$(3, 6)$
p_5	$(7, 6)$
p_6	$(4, 5)$
p_7	$(8, 4)$
p_8	$(2, 3)$
p_9	$(5, 2)$
p_{10}	$(9, 1)$



Ας θεωρήσουμε ότι θέλουμε να υπολογίσουμε το skyline για το σύνολο δεδομένων της παραπάνω εικόνας με τη βοήθεια του BNL και με μέγεθος παραθύρου 3. Το σύνολο δεδομένων διαβάζεται με τη σειρά που φαίνεται στην παραπάνω εικόνα. Αρχικά, το p_1 εισέρχεται στο παράθυρο που είναι άδειο. Στη συνέχεια διαβάζεται το επόμενο σημείο p_2. Αν το p_1 δεν κυριαρχεί το p_2, το p_1 απομακρύνεται από το παράθυρο και εισέρχεται σ' αυτό το p_2. Ομοίως, αφού διαβαστεί το p_3, μόνο το p_3 μένει στο παράθυρο. Κατόπιν, το p_4 εισέρχεται στο παράθυρο καθώς δεν κυριαρχείται από κανένα από τα σημεία του παραθύρου (δηλαδή, από το p_3). Μετά από αυτό επεξεργαζόμαστε με τον ίδιο τρόπο τα p_5 και p_6 και το p_6 εισέρχεται στο παράθυρο. Όταν διαβαστεί το p_7 ,εισέρχεται στο προσωρινό αρχείο, καθώς δεν κυριαρχείται από κανένα από τα δεδομένα στο παράθυρο και το παράθυρο είναι γεμάτο. Η χρονική

ένδειξη (timestamp) του προσωρινού αρχείου είναι 7. Στο τέλος αυτής της επανάληψης στο παράθυρο υπάρχουν τα σημεία p_3, p_8 και p_9, ενώ στο προσωρινό αρχείο βρίσκονται τα σημεία p_7 και p_{10}. Τα σημεία του παραθύρου που διαβάστηκαν πριν από τη χρονική ένδειξη (timestamp) του προσωρινού αρχείου βγαίνουν στην έξοδο σαν σημεία της κορυφογραμμής. Στην προκειμένη περίπτωση λοιπόν, βγαίνει στην έξοδο το p_3. Στη συνέχεια, ο BNL συνεχίζει και επεξεργάζεται το προσωρινό αρχείο με τον ίδιο τρόπο μέχρι να μην παραμείνουν καθόλου σημεία στο προσωρινό αρχείο. Το skyline αποτέλεσμα είναι$\{ p_3, p_8, p_9, p_{10} \}$.

Σε γενικές γραμμές τα μειονεκτήματα του BNL είναι τα εξής:

- Η εξάρτηση του από την κύρια μνήμη, αφού στην περίπτωση που η μνήμη είναι αρκετά μικρή θα χρειαστούν πολλές επαναλήψεις για να υπολογιστεί το τελικό skyline.
- Η ανεπάρκεια του στην περίπτωση on-line επεξεργασίας, αφού το αρχείο με τα δεδομένα πρέπει να διαβαστεί ολόκληρο πριν επιστραφεί το πρώτο skyline σημείο.
- Δεν υπάρχει πρόβλεψη για πρόωρο τερματισμό.

Στην καλύτερη περίπτωση που το skyline χωράει στο παράθυρο ο αλγόριθμος τερματίζει μετά από μία ή δύο επαναλήψεις και η πολυπλοκότητα του είναι: $O(n)$. Στη χειρότερη περίπτωση η πολυπλοκότητα είναι της τάξης $O(n^2)$.

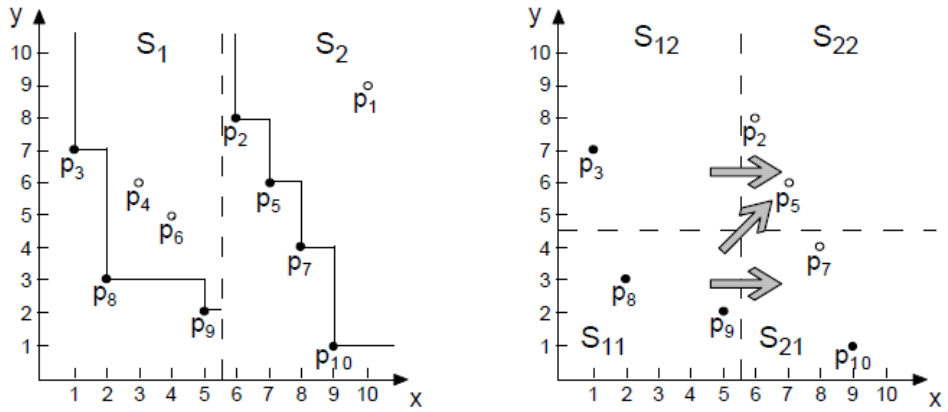
Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται παρακάτω :

M	Είσοδος της skyline διαδικασίας. Ένα σύνολο d-διάστατων σημείων.
R	Έξοδος της skyline διαδικασίας. Ένα σύνολο d-διάστατων σημείων.
T	Προσωρινό αρχείο. Ένα σύνολο d-διάστατων σημείων.
S	Κύρια μνήμη. Ένα σύνολο d-διάστατων σημείων.
$p \prec q$	Το σημείο p κυριαρχείται από το σημείο q.

```
functionSkylineBNL(M)
begin
//Initialization
R:=∅, T:=∅, S:= ∅
CountIn:=∅, CountOut:=∅
//Scanning the database repeatedly
while ¬EOF(M) do begin
 foreach p∈S do //propagate points that have been compared to all
    if TimeStamp(p):=CountIn then save(R,p),release(p)
    load(M,p),TimeStamp(p):=Countout
    CountIn:=CountIn+1
 foreach q∈S\{p} do begin //compare it to all points in memory
    if p≻q then release(p),break
    if p≺q then release(q)
end
if ¬memoryAvailable then begin  //write it to tempfile if necessary
    save(T,p),release(p)
CountOut:=CountOut+1
end
```

3.2 Divide & Conquer Αλγόριθμος(D&C)

Η προσέγγιση του D&C, που προτάθηκε από τουςBorzsonyi κ.α [BKS01], διαιρεί το σύνολο των δεδομένων σε επιμέρους τμήματα έτσι ώστε κάθε τμήμα να χωράει στη μνήμη. Στη συνέχεια, υπολογίζεται ένα κομμάτι της κορυφογραμμής των σημείων σε κάθε τμήμα χρησιμοποιώντας έναν αλγόριθμο στην κύρια μνήμη και η τελική κορυφογραμμή προκύπτει από την συγχώνευση των επιμέρους κορυφογραμμών. Η παρακάτω είκονα δείχνει ένα παράδειγμα για το πώς λειτουργεί ο D&C χρησιμοποιώντας το σύνολο των δεδομένων, που παρουσιάστηκε παραπάνω.



(a) Divide step. (b) Merge step.

Ο D&C υπολογίζει πρώτα τη μέση τιμή όλων των δεδομένων στη διάσταση x και χωρίζει το σύνολο δεδομένων σε 2 τμήματα, S_1 και S_2. Το S_1 περιέχει όλα τα σημεία των οποίων οι τιμές στη διάσταση x είναι μικρότερες από τη μέση τιμή. Το S_2 περιέχει όλα τα υπόλοιπα. Έπειτα, οι κορυφογραμμές των S_1 και S_2 υπολογίζονται αντίστοιχα. Αυτό γίνεται με το να διαμερίζουμε επιμέρους κάθε κομμάτι μέχρι αυτό να περιέχει ένα μόνο σημείο. Σ' αυτήν την περίπτωση, είναι εύκολο να υπολογίσουμε το skyline. Το βήμα αυτό του D&C ονομάζεται βήμα διαίρεσης (divide). Τα τοπικά skyline του S_1 και S_2 φαίνονται στην παραπάνω εικόνα (α).

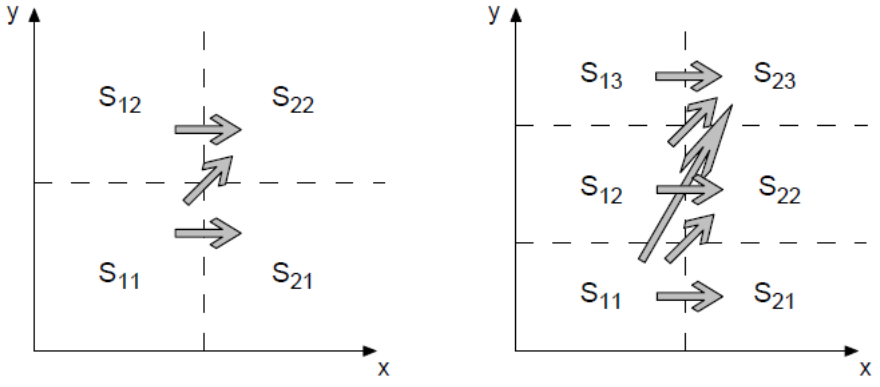
Αφού υπολογιστούν όλα τα τοπικά skyline ο D&C εξαλείφει όλα τα σημεία στο τοπικό skyline του S_2 τα οποία κυριαρχούνται από τα τοπικά σημεία skyline του S_1, έτσι ώστε να πάρουμε το ολικό skyline (βήμα συγχώνευσης-merge). Για να γίνει η εξάλειψη αποτελεσματικά, τα τοπικά skyline σημεία του S_1 και S_2 διαμερίζονται περαιτέρω σε 2 τμήματα με βάση τη μέση τιμή των τοπικών skyline σημείων του S_1 στη διάσταση y αντίστοιχα. Τα αποτελέσματα αυτής της περαιτέρω διαμέρισης φαίνονται στην παραπάνω εικόνα (b). Τα σημεία στο S_{21} έχουν μικρότερες συντεταγμένες στη διάσταση y από τα σημεία στο S_{12}. Σαν αποτέλεσμα, η σύγκριση του S_{12} με το S_{21} δε χρειάζεται να γίνει. Από την άλλη,

τα σημεία στο S_{22} έχουν μεγαλύτερες συντεταγμένες και στις 2 διαστάσεις (x και y) από αυτά των σημείων στο S_{11}. Δηλαδή, όλα τα σημεία του S_{22} κυριαρχούνται από οποιοδήποτε σημείο του S_{11}, οπότε και απορρίπτονται αμέσως χωρίς καμμία σύγκριση. Τώρα ο D&C χρειάζεται να συγκρίνει μόνο το S_{21} με το S_{11} για να απορριφθούν τα σημεία του S_{21} που δεν ανήκουν στην κορυφογραμμή. Το p_7 απορρίπτεται. Άρα το συνολικό skyline είναι το : $\{p_3, p_8, p_9, p_{10}\}$

Η πολυπλοκότητα του αλγόριθμου D&C για d-διάστατα σημεία είναι $O(n \log_2 n)$ για d=2,3 και $O(n(\log_2 n)^{d-2})$ για $d \geq 4$.

Αν τα δεδομένα δε χωράνε στη μνήμη ο D&C απαιτεί να διαβάσουμε και να γράψουμε το σύνολο δεδομένων αρκετές φορές κατά τη διάρκεια της διαδικασίας κατάτμησης (divide) με αποτέλεσμα να προκαλεί σημαντικό I/O κόστος. Για να βελτιωθεί η απόδοση του D&C όταν η μνήμη είναι περιορισμένη προτάθηκε ο M-way D&C αλγόριθμος. Η βασική ιδέα του M-way D&C είναι ότι κατά τη διάρκεια της διαδικασίας διαμερισμού τα δεδομένα χωρίζονται σε m τμήματα αντί για 2 έτσι ώστε κάθε τμήμα να χωράει στη μνήμη. Η ίδια ιδέα εφαρμόζεται και στο βήμα συγχώνευσης έτσι ώστε κάθε τμήμα να απασχολεί το πολύ τη μισή από τη διαθέσιμη κύρια μνήμη. Έτσι δεν υπάρχει κάποιο επιπλέον I/O κόστος κατά τη διάρκεια κάθε σύγκρισης.

Στην παρακάτω εικόνα γίνεται η σύγκριση μεταξύ του D&C όταν η κύρια μνήμη μπορεί να χωρέσει όλα τα σημεία και του 3-way D&C στην αντίθετη περίπτωση.



(a) DC. (b) 3-way DC.

Σε γενικές γραμμές τα μειονεκτήματα του D&C είναι τα εξής:

- Είναι αποτελεσματικό μόνο για μικρά σύνολα δεδομένων. Για πολύ μεγάλα σύνολα δεδομένων το κόστος του αριθμού πρόσβασεων εισόδου/εξόδου (I/O) είναι τεράστιο.
- Δεν είναι κατάλληλο για online επεξεργασία, καθώς η έξοδος δεν μπορεί να δοθεί αν δεν ολοκληρωθεί το partitioning του χώρου.

Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται παρακάτω :

```
function SkylineBasic(M,Dimension)
begin
if |M|=1 then return M
Pivot:=Median{ m_Dimension |m∈M}
(P1, P2):=Partition(M,Dimension,Pivot)
S1:=SkylineBasic(P1,Dimension)
S2:=SkylineBasic(P2,Dimension)
Return S1 ∪MergeBasic(S1,S2,Dimension)
end
function MergeBasic(S1,S2,Dimension)
begin
if S1={p} then R:={q∈ S2 | p ≺ q }
else if S2={q} then begin
     R:=S2
     foreach p ∈ S1 do if p ≺ q then R:=0
end else begin
     Pivot:=Median{ p_Dimension-1 |p∈ S1 }//partition both sets
```

```
        (S1,1 , S1,2 ):=Partition(S1 ,Dimension-1,Pivot)

        (S2,1 , S2,2 ):=Partition(S2 ,Dimension-1,Pivot)

        R1 :=MergeBasic(S1,1 ,S2,1 ,Dimension)//compare adjacent parts

        R2 :=MergeBasic(S1,2 ,S2,2 ,Dimension)

        R3 :=MergeBasic(S1,1 ,R2 ,Dimension-1)//comparediagonallyR:= R1
 ∪ R2
end

return R

end
```

3.3 Bitmap Αλγόριθμος

Ο αλγόριθμος αυτός, που παρουσιάστηκε από τους Tan κ.α.[TEO01], πήρε αυτήν την ονομασία, επειδή χρησιμοποιεί ως είσοδο μία αναπαράσταση των δεδομένων ως ακολουθία από bit. Υποθέτουμε ότι τα δεδομένα είναι αποθηκευμένα ανά σημεία d διαστάσεων.

Έστω ένα σημείο $p = (p[1], p[2], ...p[d])$, όπου d είναι η διάσταση. Κάθε συντεταγμένη $p[i]$, $(1 \le i \le d)$, αναπαρίσταται ως ένα διάνυσμα των m_i -bit, όπου m_i είναι ο αριθμός των διαφορετικών τιμών της i-οστής διάστασης , στην οποία τα $(m_i - rank(p_i) + 1)$ πιο σημαντικά bits είναι 1 και τα άλλα είναι 0. Μετά τη μετατροπή κάθε σημείο αντιστοιχεί σ'ένα m-bit διάνυσμα, όπου m= $\sum_{i=1}^{d} m_i$.

Σαν παράδειγμα ο παρακάτω πίνακας δείχνει τα αντίστοιχα bit ανύσματα για τα σημεία που παρουσιάστηκαν παραπάνω.

Id	Coordinates	BitVectors
p_1	(10,9)	(1000000000,1100000000)
p_2	(6,8)	(1111100000,1110000000)
p_3	(1,7)	(1111111111,1111000000)
p_4	(3,6)	(1111111100,1111100000)
p_5	(7,6)	(1111000000,1111100000)
p_6	(4,5)	(1111111000,1111110000)
p_7	(8,4)	(1110000000,1111111000)
p_8	(2,3)	(1111111110,1111111100)
p_9	(5,2)	(1111110000,1111111110)
p_{10}	(9,1)	(1100000000,1111111111)

Αφού η τιμή του $p_{10}[1]$ στη 1η διάσταση είναι 9, μετατρέπεται στο διάνυσμα 1100000000. Ομοίως όλα τα bits του αντίστοιχου διανύσματος του $p_{10}[2]$ είναι 1, γιατί το $p_{10}[2]$ είναι η μικρότερη τιμή στη δεύτερη διάσταση. Αφού μετατρέψουμε όλα τα σημεία σε bitmaps, μπορούμε να βρούμε για κάθε σημείο αν ανήκει στο skyline κάνοντας bitwise πράξεις με τα bitmaps.

Συγκεκριμένα, δοθέντος ενός σημείου $p = (p[1], p[2], ...p[d])$ ο αλγόριθμος bitmap δημιουργεί ανύσματα των d bit $b_1, b_2, ...b_d$, όπου b_i $(1 \le i \le d)$ και αντιπαραβάλλει τα αντίστοιχα $rank(p_i)$ bits κάθε σημείου. Τα 1 στο αποτέλεσμα του $b_1 \& b_2 \& ... \& b_d$ δείχνουν τα σημεία που κυριαρχούν το p. Αφού υπάρχει ένα μόνο 1 στο αποτέλεσμα, το αντίστοιχο σημείο είναι ένα skyline σημείο.

Για παράδειγμα, τα αντίστοιχα b_1, b_2 bit διανύσματα του p_7 είναι 0111111110 και 0000001111 αντίστοιχα. Το αποτέλεσμα του $b_1 \& b_2$ είναι 0000001110 δείχνει ότι το p_7 κυριαρχείται από το p_8 και το p_9. Σαν αποτέλεσμα, το p_7 δεν είναι skyline σημείο. Απ'την άλλη για το p_8, το $b_1 \& b_2$ είναι 0010000100&0000000100, το οποίο έχει ένα μόνο 1. Άρα το p_8 ανήκει στο skyline.

Για να πάρουμε το συνολικό skyline, ο Bitmap επαναλαμβάνει την ίδια διαδικασία για κάθε σημείο.

3.4 Index Αλγόριθμος

Ο αλγόριθμος αυτός, που προτάθηκε από τους Tan κ.α.[TEO01].

Δοθέντος ενός συνόλου P από d-διάστατα σημεία, ο αλγόριθμος Index οργανώνει το P σε ένα B^+ δέντρο από δείκτες. Ένα σημείο $p = (p[1], p[2], ...p[d])$ ανατίθεται στον i-οστό δείκτη του B^+ δέντρου (

$1 \le i \le d$), αν και μόνο αν, το $p[i]$ είναι η μικρότερη συντεταγμένη από όλες τις συντεταγμένες του p (minValue). Το κλειδί κάθε B^+ δέντρου είναι η ελάχιστη συντεταγμένη κάθε σημείου. Τα σημεία του ίδιου δείκτη του B^+ δέντρου διατηρούνται σε μια δέσμη.

Για να υπολογίσουμε το skyline, διατηρείται η μέγιστη τιμή (maxValue) όλων των συντεταγμένων των υπάρχοντων skyline σημείων. Ο αλγόριθμος Index εξετάζει επαναληπτικά κάθε δείκτη του B^+ δέντρου και επεξεργάζεται τη δεσμίδα (batch) που έχει την μικρότερη minValue. Η minValue αυτής της δεσμίδας συγκρίνεται με την maxValue. Αν η maxValue είναι μικρότερη ή ίση της minValue θα υπάρχει κάποιο skyline σημείο στη μέχρι στιγμής λίστα κορυφογραμμής που θα κυριαρχεί τα σημεία της δεσμίδας και όλα τα σημεία που δεν έχουμε ακόμα επεξεργαστεί στον ίδιο δείκτη του B^+ δέντρου. Άρα, η δεσμίδα και όλα τα σημεία του δείκτη του B^+ δέντρου που δεν έχουν ακόμα επεξεργαστεί , μπορούν να διαγραφούν. Αλλιώς, υπολογίζεται ένα skyline μέσα στη δέσμη που επεξεργαζόμαστε. Στη συνέχεια τα εναπομείναντα σημεία συγκρίνονται με τα skyline σημεία που έχουν υπολογιστεί μέχρι στιγμής. Αν κάποιο σημείο που εξετάζουμε κυριαρχείται από κάποιο skyline σημείο, το σημείο αυτόδιαγράφεται, αλλιώς εισέρχεται στο skyline σαν νέο skyline σημείο. Μόλις βρεθεί ένα καινούριο skyline σημείο η maxValue ανανεώνεται. Αφού επεξεργαστούμε όλες τις δεσμίδες του B^+ δέντρου ο αλγόριθμος Index επιστρέφει την τελική skyline λίστα.

Ας θεωρήσουμε το σύνολο δεδομένων που εξετάσαμε παραπάνω. Όλες οι δεσμίδες των 2 B^+ δέντρων κατά αύξουσα σειρά της minValue φαίνονται στο παρακάτω σχήμα. Αρχικά, ο αλγόριθμος επεξεργάζεται τις δεσμίδες με minValue=1 μία προς μία. Αφού τα σημεία p_3 και p_{10} δεν κυριαρχούνται από κάποιο από τα τρέχοντα skyline σημεία εισέρχονται στο skyline. Μετά από αυτό η maxValue γίνεται 9. Ομοίως τα σημεία p_8 και p_9 προστίθενται στο skyline αφού επεξεργαστούμε τις δέσμες στις οποίες ανήκουν. Στην περίπτωση αυτή η maxValue δεν αλλάζει. Στη συνέχεια διαβάζεται το p_4, το οποίο και διαγράφεται αφού κυριαρχείται από κάποιο από τα skyline σημεία (το p_8). Ο αλγόριθμος συνεχίζει και εξετάζει τις υπόλοιπες δεσμίδες κατά αύξουσα σειρά της minValue. Όταν διαβαστεί το p_1 διαγράφεται χωρίς να συγκριθεί με κάποιο από τα τρέχοντα skyline σημεία, καθώς η minValue του είναι ίση με την maxValue(δηλαδή 9). Αφού εξεταστούν όλα τα σημεία ο αλγόριθμος τερματίζει και βγάζει σαν έξοδο την skyline λίστα $\{p_3, p_{10}, p_8, p_9\}$.

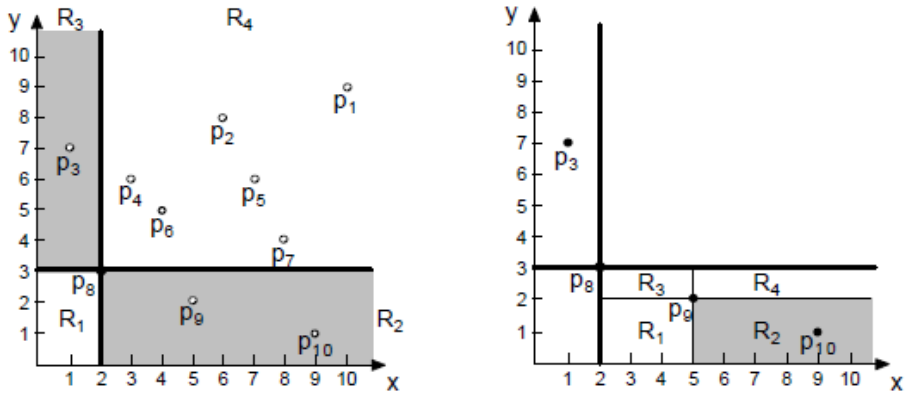
Index 1		Index 2	
$minValue$	batch	$minValue$	batch
1	$p_3(1,7)$	1	$p_{10}(9,1)$
2	$p_8(2,3)$	2	$p_9(5,2)$
3	$p_4(3,6)$	4	$p_7(8,4)$
4	$p_6(4,5)$	6	$p_5(7,6)$
6	$p_2(6,8)$	9	$p_1(10,9)$

3.5 Nearest Neighbor Αλγόριθμος(NN)

Ο NN, που παρουσιάστηκε από του Παπαδιάς κ.α.[PTF+05] βασίζεται στην παρακάτω θεμελιώδη παρατήρηση:

Δοθέντος ενός συνόλου δεδομένων και μιας συνάρτησης απόστασης f, ο κοντινότερος γείτονας της αρχής των αξόνων είναι ένα skyline σημείο.

Βασιζόμενοι σ'αυτήν την παρατήρηση για να υπολογίσουμε το skyline ενός δοθέντος συνόλου δεδομένων, ο NN βρίσκει τον πιο κοντινό γείτονα p της αρχής των αξόνων. Στη συνέχεια με βάση το p χωρίζει το χώρο σε 3 τμήματα, όπως φαίνεται στην παρακάτω εικόνα.



(a) The first skyline point. (b) All the skyline points.

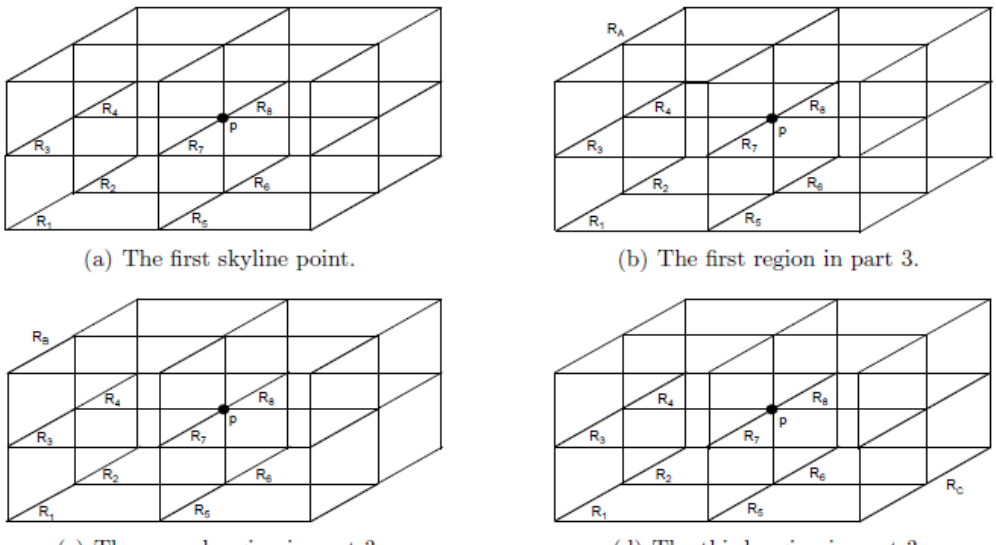
Τμήμα 1: Το άνω ορθογώνιο, έχοντας την αρχή των αξόνων, ως την κάτω αριστερά γωνία και το p στην πάνω δεξιά γωνία (R_1). Το p κυριαρχείται από το σημείο σ'αυτήν την περιοχή. Σύμφωνα με την παραπάνω παρατήρηση, το τμήμα αυτό είναι άδειο καθώς κανένα σημείο δεν κυριαρχεί το p.

Τμήμα 2: Το άνω ορθογώνιο με το p ως την κάτω αριστερά γωνία του και την πάνω δεξιά γωνία του χώρου σαν πάνω δεξιά γωνία (R_4). Όλα τα σημεία στο τμήμα αυτό κυριαρχούνται από το p . Γι'αυτό και αυτά τα σημεία μπορούν να διαγραφούν.

Τμήμα 3: Οι άλλες περιοχές (R_2 και R_3). Η ιδιότητα αυτών των περιοχών είναι ότι τα τοπικά skyline σημεία τους ανήκουν στο συνολικό skyline καθώς το p δεν κυριαρχεί κανένα σημείο σ'αυτές τις περιοχές. Σαν αποτέλεσμα ο NN εφαρμόζεται επαναληπτικά σ'αυτές τις περιοχές μέχρι να εκτιμηθεί όλος ο χώρος των δεδομένων.

Θεωρούμε το σύνολο δεδομένων της παραπάνω εικόνας (α) ως παράδειγμα. Αφού βρεθεί ο πιο κοντινός γείτονας της αρχής των αξόνων p_8 , ο χώρος των δεδομένων χωρίζεται σε 4 τμήματα, R_1, R_2, R_3, R_4 . Όπως αναφέρθηκε και παραπάνω ο NN χρειάζεται μόνο τις R_2 και R_3 . Για κάθε περιοχή, ο NN βρίσκει επαναληπτικά τον πιο κοντινό γείτονα της κάτω αριστερής γωνίας του και τη χωρίζει σε υπο-περιοχές. Στην R_3 , αφού υπάρχει μόνο ένα σημείο, το p_3 , το σημείο αυτό επιστρέφεται ως skyline σημείο και σταματάει η αναζήτηση. Ομοίως στην R_2 , το p_9 βρίσκεται ως το πιο κοντινό γειτονικό σημείο και με βάση αυτό το R_2 , χωρίζεται σε τμήματα. Αφού βρεθεί και το τελευταίο σημείο p_{10} ως το πιο κοντινό γειτονικό σημείο της υπό-περιοχής του, ο NN τερματίζει και βγάζει σαν έξοδο όλα πιο κοντινά γειτονικά σημεία ως το skyline.

Γενικά, για πολυδιάστατο χώρο οι περιοχές στο τμήμα 3 μπορεί να επικαλύπτονται. Η παρακάτω εικόνα δείχνει ένα παράδειγμα του NN στον 3-διάστατο χώρο.



(a) The first skyline point.

(b) The first region in part 3.

(c) The second region in part 3.

(d) The third region in part 3.

Αφού βρεθεί το πρώτο skyline p , ο NN διερευνά περαιτέρω τις περιοχές στο τμήμα 3. Όπως φαίνεται στην εικόνα δημιουργούνται και επεξεργάζονται 3 περιοχές, οι R_a , R_b και R_c . Αυτές οι περιοχές επικαλύπτονται,κάτι το οποίο οδηγεί διπλή αναζήτηση και αποτελέσματα. Για να αποφευχθούν αυτές οι διπλές αναζητήσεις ο NN εκμεταλλεύεται τις ακόλουθες μεθόδους για τον πολυδιάστατο χώρο, που προτάθηκαν από τους Kossmann κ.α. [KRR02].

Laisser-faire: Ο NN διατηρεί έναν συνολικό πίνακα, όπου αποθηκεύονται τα skyline σημεία που έχουν υπολογιστεί μέχρι στιγμής. Όταν βρεθεί ένα skyline σημείο συγκρίνεται με τον πίνακα πρώτα. Αν δεν υπάρχει στον πίνακα, βγαίνει στην έξοδο και εισέρχεται στον πίνακα. Αλλιώς αγνοείται καθώς έχει ανακαλυφθεί νωρίτερα. Παρολαυτά, αυτό δεν περιορίζει τις διπλές αναζητήσεις στις επικαλυπτόμενες περιοχές.

Propagate: Μόλις βρεθεί ένα skyline σημείο p , ο NN εξετάζει όλες τις περιοχές που θα επεξεργαστούν και ξαναχωρίζει τις περιοχές που περιέχουν το p , έχοντας ως αναφορά το p . Παρόλο που αυτή η μέθοδος δεν επιστρέφει το ίδιο skyline πάνω από 2 φορές, απαιτεί περισσότερους υπολογισμούς αφού κάθε skyline σημείο p ξαναδιαμερίζει όλες τις περιοχές που περιέχουν το p .

Merge: Η βασική ιδέα αυτής της μεθόδου είναι να συγχωνεύσουμε μερικές περιοχές που θα επεξεργαστούμε έτσι ώστε να περιοριστεί ο συνολικός αριθμός των NN ερωτημάτων. Παρόμοια με την propagate και αυτή η μέθοδος έχει μεγαλύτερο CPU κόστος καθώς είναι πιο δύσκολο να βρεθούν οι «καλές» περιοχές για συγχώνευση.

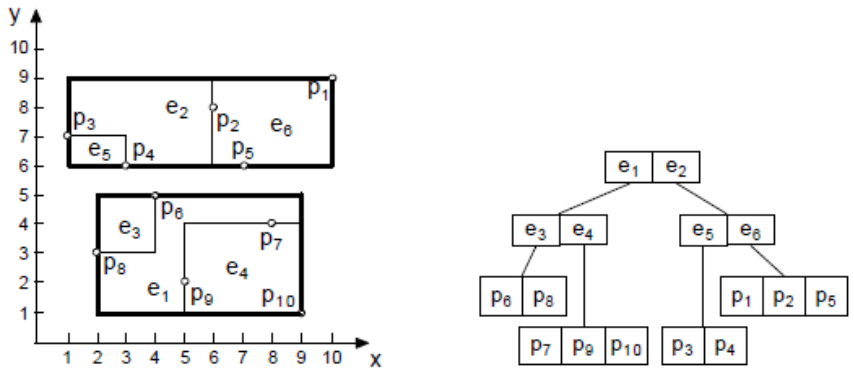
Fine-grained Partitioning: Ο αρχικός NN αλγόριθμος διαμερίζει τον χώρο σε d περιοχές αφού βρεθεί ένα skyline σημείο. Για να περιοριστούν οι επικαλυπτόμενες περιοχές, η μέθοδος αυτή δημιουργεί 2^d μη επικαλυπτόμενες περιοχές κάθε φορά. Ένα παράδειγμα φαίνεται στην παραπάνω εικόνα, όπου ο χώρος έχει χωριστεί σε 8 τμήματα. Οι περιοχές $R_2 - R_7$ ανήκουν στο τμήμα 3. Σαν αποτέλεσμα ο NN αλγόριθμος καλείται συνέχεια σ'αυτές τις περιοχές. Παρόλο που αυτή η μέθοδος αποφεύγει διπλότυπες αναζητήσεις και αποτελέσματα μπορεί να βγάλει λάθος αποτέλεσμα, π.χ. το skyline που βρίσκεται σε μια περιοχή μπορεί να κυριαρχείται από αλλά σ'άλλες περιοχές. Γι'αυτό και αν βρεθεί ένα skyline σημείο ελέγχεται σε σχέση με όλα τα skyline σημεία που έχουν βρεθεί μέχρι στιγμής ώστε να αποφευχθούν λάθη.

Ο NN παρουσιάζει το εξής μειονέκτημα: Σε περίπτωση συνόλων δεδομένων υψηλών διαστάσεων, η ανεξερεύνητη περιοχή αυξάνει σημαντικά.

3.6 Branch and Bound Skyline Αλγόριθμος(BBS)

Όπως και ο NN, ο αλγόριθμος BBS, που προτάθηκε από του Παπαδιάς κ.α.[PTF+05] βασίζεται στην αναζήτηση του skyline στην κοντινότερη γειτονική περιοχή. Εδώ, τα δεδομένα ταξινομούνται με βάση ένα R-δέντρο. Ο BBS είναι I/O βέλτιστος καθώς ο αριθμός των κόμβων του R δέντρου, που διατρέχονται ελαχιστοποιείται. Για να βρει το skyline ο BBS διατρέχει όλο το δέντρο εξαντλώντας τον κόμβο που είναι πιο κοντά στη ρίζα όλων των κόμβων που δεν έχει ακόμα επισκεφθεί. Γι' αυτόν το λόγο ο BBS χρησιμοποιεί ένα σωρό που το κλειδί για κάθε είσοδο του είναι η ελάχιστη απόσταση από τη ρίζα. Η ελάχιστη αυτή απόσταση είναι το άθροισμα των συντεταγμένων της κάτω αριστεράς γωνίας του. Στην αρχή όλα τα παιδιά του κόμβου-ρίζα του R-δέντρου εισέρχονται στο σωρό. Σε κάθε επανάληψη η κορυφαία είσοδος e αφαιρείται από το σωρό και εξετάζεται σε σχέση με το τρέχον skyline που έχει υπολογιστεί μέχρι στιγμής. Αν το e κυριαρχείται από κάποιο σημείο του skyline διαγράφεται. Αλλιώς, το e είτε επεκτείνεται ή βγαίνει στην έξοδο σαν skyline. Αν το e είναι ένας κόμβος του R-δέντρου επεκτείνεται εισάγοντας όλα τα παιδιά του που δεν κυριαρχούνται από τα τρέχοντα skyline σημεία στο σωρό. Αν το e είναι ένα σημείο, βγαίνει στην έξοδο ως skyline σημείο. Ο BBS τερματίζει όταν αδειάσει ο σωρός. Για να επιταχυνθεί η διαδικασία εξέτασης του e, το τρέχον skyline διατηρείται στη μνήμη σε μορφή ενός R-δέντρου.

Θεωρούμε το σύνολο δεδομένων, που φαίνεται στην παρακάτω εικόνα και το αντίστοιχο R-δέντρο του.



(a) Dataset. (b) *R*-tree.

Για να υπολογίσουμε το skyline, ο BBS εισάγει πρώτα τα e_1 και e_2 στο σωρό. Επειδή, το e_1 είναι πιο κοντά στη ρίζα από το e_2 επεκτείνεται με τα e_3 και e_4. Η επόμενη είσοδος είναι το e_3 και τα σημεία του

p_8 και p_6 εισέρχονται στο σωρό μετά από επεξεργασία. Στη συνέχεια, επεξεργαζόμαστε το σημείο p_8. Επειδή δεν κυριαρχείται από κανένα skyline σημείο, βγαίνει στην έξοδο σαν skyline σημείο. Ομοίως, το e_4 επεκτείνεται και τα σημεία του εισέρχονται στο σωρό. Όταν το e_2 επεκταθεί, το παιδί του e_6 κυριαρχείται από το p_8 και διαγράφεται. Για τον ίδιο λόγο, αφού επεξεργαστούμε το e_5 μόνο το σημείο p_3 εισέρχεται στο σωρό. Στη συνέχεια, ο BBS εξακολουθεί να εξετάζει τα σημεία που έχουν απομείνει στο σωρό ένα προς ένα. Μόνο τα σημεία που δεν κυριαρχούνται από τα τρέχοντα skyline σημεία βγαίνουν στην έξοδο σαν skyline σημεία. Τέλος το skyline είναι το εξής: $\{p_3, p_{10}, p_8, p_9\}$.

Action	Heap	Skyline
initializing	$\langle e_1, 3\rangle$, $\langle e_2, 7\rangle$	$\varnothing$
expend e_1	$\langle e_3, 5\rangle$, $\langle e_4, 6\rangle$, $\langle e_2, 7\rangle$	$\varnothing$
expend e_3	$\langle \mathbf{p_8}, \mathbf{5}\rangle$, $\langle e_4, 6\rangle$, $\langle e_2, 7\rangle$, $\langle p_6, 9\rangle$	p_8
expend e_4	$\langle \mathbf{p_9}, \mathbf{7}\rangle$, $\langle e_2, 7\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9
expend e_2	$\langle e_5, 7\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9
expend e_5	$\langle \mathbf{p_3}, \mathbf{8}\rangle$, $\langle p_6, 9\rangle$, $\langle p_{10}, 10\rangle$, $\langle p_7, 12\rangle$	p_8, p_9, p_3
examine p_6, p_{10}, p_7	$\varnothing$	p_8, p_9, p_3, p_{10}

Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται παρακάτω:

```
Algorithm BBS
S=∅ //list of skyline points
Insert all entries of the root R in the heap
while heap not empty
remove top entry e
if e is dominated by some point in S, discard e
else //e is not dominated
        if e is an intermediate entry
            for each child ei of e
                if ei is not dominated by some point in S
                    insert ei into heap
                else //e is data point
                    insert ei into S
end while
end BBS
```

3.7 Sort First SkylineΑλγόριθμος (SFS)

Ο SFS, που παρουσιάστηκε από τους Chomicki κ.α.[CGG+02] είναι μια παραλλαγή του BNL. Για να βελτιωθεί ο BNL, ο SFS εισάγει την τιμή entropy E(p) για κάθε σημείο $p = (p[1], p[2], ..., p[d])$

$E(p) = \sum_{i=1}^{d} \ln(p'[i] + 1)$, όπου $p'[i]$ είναι η κανονικοποιημένη τιμή του $p[i]$. Δοθέντων δύο σημείων p και q το p δεν μπορεί να κυριαρχεί το q αν το E(p) είναι μεγαλύτερο ή ίσο με το E(q). Βασιζόμενος σ'αυτήν την παρατήρηση ο SFS ταξινομεί όλα τα σημεία με μια μη-φθίνουσα σειρά των τιμών της συνάρτησης entropy. Μετά απ'αυτό ο αλγόριθμος επεξεργάζεται το ταξινομημένο σύνολο δεδομένων με τον ίδιο τρόπο όπως ο BNL. Πιο συγκεκριμένα ο αλγόριθμος ακολουθεί τα εξής:

Πρώτα ταξινομούμε τον πίνακα μας. Σε μια σχεσιακή μηχανή, αυτό μπορεί να γίνει καλώντας μια εξωτερική ρουτίνα ταξινόμησης. Τότε ένας χώρος αποθήκευσης διανέμεται σαν ένα παράθυρο στο οποίο θα μπουν οι skyline πλειάδες (tuples) που θα βρεθούν. Στη συνέχεια ξεκινάει ένα πέρασμα ενός κέρσορα πάνω στις ταξινομημένες πλειάδες. Η τρέχουσα πλειάδα ελέγχεται σε σχέση με όλες τις άλλες που έχουν μπει στο παράθυρο. Αν η τρέχουσα πλειάδα κυριαρχείται από κάποια άλλη του παράθυρου τότε μπορούμε να τη διαγράψουμε. Δεν μπορεί να ανήκει στο skyline. Αλλιώς, η τρέχουσα πλειάδα δεν μπορεί να συγκριθεί με καμία από τις πλειάδες του παραθύρου. Είναι και αυτή άρα μία skyline πλειάδα. Σημειώνουμε ότι ήταν αρκετό που συγκρίναμε το τρέχον tuple μόνο με τα tuples στο παράθυρο και όχι μ'όλα τα tuples που προηγούνται αυτού. Αυτό συμβαίνει γιατί αν κάποιο από τα προηγούμενα tuples διαγράφηκαν είναι γιατί κυριαρχούνταν ήδη από κάποιο άλλο tuple του παραθύρου. Επειδή η κυριαρχία είναι μεταβατική, αρκούν μόνο οι συγκρίσεις μεταξύ των tuples του παραθύρου. Στην περίπτωση που το τρέχον tuple δεν κυριαρχείται, αν υπάρχει ελεύθερος χώρος στο παράθυρο τοποθετείται στο παράθυρο. Θα μπορούσαμε επίσης να δώσουμε ταυτόχρονα το tuple κατευθείαν στην έξοδο, αν θέλαμε, μιας και ξέρουμε ότι είναι skyline. Ο αλγόριθμος παίρνει το επόμενο tuple από τα δεδομένα και επαναλαμβάνει την ίδια διαδικασία.

Μπορεί να τύχει το τρέχον tuple να είναι στο skyline, αλλά να μην υπάρχει χώρος για να προστεθεί στο παράθυρο. Σ'αυτήν την περίπτωση ο αλγόριθμος αλλάζει λειτουργία. Αυτό το tuple γράφεται τώρα σ' ένα προσωρινό αρχείο. Δεν μπορούμε πλέον να είμαστε σίγουροι ότι ένα τέτοιο tuple είναι skyline. Έχει συγκριθεί με τα tuples του παραθύρου, και δεν κυριαρχείται από αυτά, αλλά δεν έχει συγκριθεί με τα άλλα tuples στο προσωρινό αρχείο που προηγούνται αυτού. Αν ο αλγόριθμος εξαντλήσει όλα τα tuples της εισόδου και κανένα tuple δεν έχει γραφεί στο προσωρινό αρχείο, τότε μπορεί να τερματίσει. Όλα τα

skyline tuples έχουν βρεθεί. Αλλιώς ο αλγόριθμος επαναλαμβάνει ένα ακόμα πέρασμα έχοντας σαν είσοδο το προσωρινό αρχείο αυτήν τη φορά. Κάποια στιγμή, μετά από κάποιο πέρασμα δε θα έχει γραφεί κανένα προσωρινό αρχείο και ο αλγόριθμος μπορεί να τερματίσει.

Αν το παράθυρο είναι αρκετά μεγάλο και μπορούν να τοποθετηθούν εκεί όλα τα skyline tuples ο SFS τερματίζει μετά από ένα μόνο πέρασμα. Αλλιώς, θα χρειαστούν πολλαπλά περάσματα. Είναι σημαντικό να σημειώσουμε ότι τα περάσματα του SFS δεν είναι περάσματα/εκτελέσεις με την κλασσική έννοια, όπως με την εξωτερική ταξινόμηση (external sorting). Δεν είναι κάθε πέρασμα πάνω σε όλον τον πίνακα, με τον ίδιο αριθμό εισόδων/εξόδων κάθε φορά.Πολλά tuples μπορεί να απορριφθούν/διαγραφούν κατά την διάρκεια ενός περάσματος και επομένως το επόμενο πέρασμα θα είναι πάνω σε λιγότερες σελίδες.

Στην καλύτερη περίπτωση η πολυπλοκότητα του SFS είναι: $O(n\lg n + kn)$, ενώ στη χειρότερη: $O(kn^2)$, όπου k και n είναι η διάσταση και ο συνολικός αριθμός των δεδομένων αντίστοιχα

Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται παρακάτω:

```
unfinished= TRUE

while(unfinished)
|
|T=open_cursor(Heap)
|
|unfinished=FALSE
|
|while(next_tuple(T,t))
|   |
|   |   if("t is not dominated")
|   |   |
|   |   |   if("window is full")
|   |   |   |
|   |   |   |       unfinished=TRUE
|   |   |   |
|   |   |   |       break
|   |   |   |
|   |   |   |      else
|   |   |   |
|   |   |   |       "Add to window."
|
|if(unfinished)
|
|       S=open_new_file(SecondPass)
|
|       write(S,t)
|
|       while(next_tuple(T,t))
|           |
|           |   if("t is not dominated")
|           |   |
|           |   |       write(S,t)
|           |   |
|           |   free(Heap)
|           |   |
|           |   close(S)
|           |   |
|           |   Heap= SecondPass
|           |   |
|           |   "Write window tuples to output."
|           |   |
|           |   "Clearwindow."
|           |
```

Λαμβάνοντας σαν παράδειγμα το προηγούμενο σύνολο δεδομένων, οι αντίστοιχες κανονικοποιημένες συντεταγμένες και η τιμή της entropy φαίνονται στον παρακάτω πίνακα.

Έστω, ότι το παράθυρο έχει μέγεθος 3 και αφού ταξινομήσουμε τα σημεία με βάση την τιμή της entropy η σειρά με την οποία θα τα επεξεργαστούμε είναι: $p_8, p_3, p_9, p_{10}, p_4, p_6, p_7, p_5, p_2$ και p_1. Τα 3 πρώτα σημεία προφανώς δεν κυριαρχούν το ένα το άλλο. Οπότε όλα εισέρχονται στο παράθυρο, άρα το παράθυρο είναι τώρα γεμάτο. Στη συνέχεια το p_{10} επεξεργάζεται και γράφεται στο προσωρινό αρχείο καθώς δεν κυριαρχείται από κανένα από τα σημεία του παραθύρου. Αφού εξετάσουμε όλα τα σημεία βλέπουμε ότι όλα απορρίπτονται καθώς όλα κυριαρχούνται από κάποιο από τα p_8, p_3, p_9. Έτσι τελειώνει και η πρώτη επανάληψη του αλγόριθμου. Στην αρχή της επόμενης επανάληψης, όλα τα σημεία του παραθύρου βγαίνουν στην έξοδο ως skyline σημεία καθώς τα είχαμε επεξεργαστεί πριν τη δημιουργία του προσωρινού αρχείου. Αφού το p_{10} είναι το μόνο σημείο στο προσωρινό αρχείο, ο SFS τερματίζει αφού το επεξεργαστεί.

Id	Coordinates	Normalized Coordinates	Entropy Value
p_1	$(10, 9)$	$(1.0, 0.9)$	1.34
p_2	$(6, 8)$	$(0.6, 0.8)$	1.06
p_3	$(1, 7)$	$(0.1, 0.7)$	0.63
p_4	$(3, 6)$	$(0.3, 0.6)$	0.73
p_5	$(7, 6)$	$(0.7, 0.6)$	1.00
p_6	$(4, 5)$	$(0.4, 0.5)$	0.74
p_7	$(8, 4)$	$(0.8, 0.4)$	0.92
p_8	$(2, 3)$	$(0.2, 0.3)$	0.44
p_9	$(5, 2)$	$(0.5, 0.2)$	0.59
p_{10}	$(9, 1)$	$(0.9, 0.1)$	0.74

Σε σχέση με τον BNL ο SFS έχει τα ακόλουθα πλεονεκτήματα:

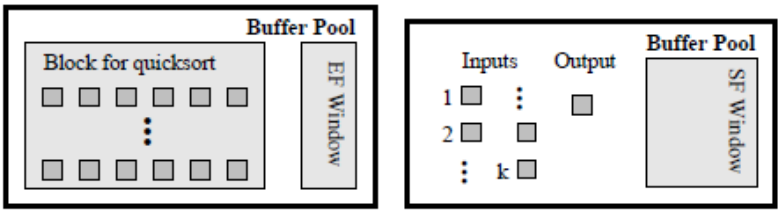
1. Μειώνεται ο αριθμός των συγκρίσεων ανάμεσα στα δεδομένα. Αφού το σημείο με τη μικρότερη τιμή της entropy εκτιμάται πρώτα, το skyline σημείο μπορεί να βρεθεί νωρίτερα. Άρα, ο αριθμός των συγκρίσεων ανάμεσα στα σημεία και στα σημεία του παραθύρου που δεν ανήκουν στο skyline μειώνεται.
2. Ο SFS είναι ένας προοδευτικός αλγόριθμος, ενώ ο BNL όχι. Όταν ένα σημείο p τοποθετείται στο παράθυρο είναι βέβαιο ότι ανήκει στο skyline, γιατί όλα τα σημεία που δεν έχουμε επεξεργαστεί ακόμα δεν κυριαρχούν το p γιατί έχουν μεγαλύτερες τιμές για το E(p).

3.8 Linear Elimination Sort for Skyline Αλγόριθμος (LESS)

Ο αλγόριθμος LESS, που προτάθηκε από τους Godfreyκ.α. [GSG06], βελτιώνει τον SFS ενσωματώνοντας την εξωτερική διαδικασία ταξινόμησης στον υπολογισμό του skyline. Όπως και ο SFS, ο LESS ταξινομεί τα δεδομένα σύμφωνα με τις τιμές της λογαριθμικής συνάρτησης (entropy) και στη συνέχεια υπολογίζει το skyline με τον ίδιο τρόπο όπως και ο BNL. Για να μπορέσουν να περιοριστούν τα σημεία αποτελεσματικά ο LESS κάνει τις 2 ακόλουθες μεγάλες αλλαγές κατά τη διάρκεια της εξωτερικής διαδικασίας ταξινόμησης και συγχώνευσης.

1. Διατηρεί ένα παράθυρο ως φίλτρο εξάλειψης (EF) στο πέρασμα 0 της εξωτερικής διαδικασίας ταξινόμησης και συγχώνευσης για να περιορίσει κάποια από τα σημεία που δεν ανήκουν στο skyline.
2. Συνδυάζει το τελικό πέρασμα της εξωτερικής διαδικασίας ταξινόμησης και συγχώνευσης με τη διαδικασία που ακολουθεί ο BNL για την αναζήτηση του skyline.

Συγκεκριμένα, ένα μικρό EF παράθυρο διατηρείται στο πέρασμα 0 της εξωτερικής διαδικασίας ταξινόμησης. Αντίγραφα των σημείων με την καλύτερη τιμή entropy διατηρούνται στο παράθυρο. Όταν διαβάζεται ένα μπλοκ δεδομένων, τα δεδομένα αυτά συγκρίνονται με αυτά του EF παραθύρου πρώτα. Τα σημεία που κυριαρχούνται από αυτά του EF παραθύρου απομακρύνονται, όπως και τα σημεία του EF παραθύρου που κυριαρχούνται από κάποιο από τα καινούρια σημεία. Στη συνέχεια τα σημεία στο EF παράθυρο αντικαθίστανται από αυτά που έχουν τις καλύτερες τιμές της entropy από όλα τα καινούρια σημεία και από τα σημεία του EF παραθύρου.



(a) in pass zero (b) in pass f

Τα περάσματα συγχώνευσης της εξωτερικής διαδικασίας ταξινόμησης και συγχώνευσης του LESS είναι τα ίδια μ'αυτά της τυπικής εξωτερικής ταξινόμησης και συγχώνευσης, εκτός από το τελικό πέρασμα συγχώνευσης, που συνδυάζεται με την αρχική διαδικασία εξέτασης του skyline του BNL. Στο τελικό πέρασμα, το παράθυρο φιλτραρίσματος του skyline (SF), μέσα στο οποίο αποθηκεύονται τα υποψήφια σημεία του skyline, διατηρείται. Εκτός από το να ταξινομηθούν όλα τα σημεία που επεξεργάζονται στο

τελικό πέρασμα συγκρίνονται με τα σημεία του SF παραθύρου. Όπως και στον BNL, αν το καινούριο σημείο κυριαρχείται από κάποιο στο SF παράθυρο, διαγράφεται αυτομάτως. Αν τα σημεία στο SF παράθυρο κυριαρχούνται από τα καινούρια σημεία, διαγράφονται και αυτά. Αν το νέο σημείο δε διαγραφεί , εισέρχεται στο SF παράθυρο σαν καινούριο υποψήφιο skyline. Όταν το παράθυρο SF γεμίσει δημιουργείται ένα προσωρινό αρχείο. Αυτή η ενσωμάτωση του τελικού περάσματος συγχώνευσης και της διαδικασίας αναζήτησης του skyline βοηθάει στο να γλιτώσουμε ένα πέρασμα πάνω στα δεδομένα.

Σε σχέση με τον SFS, ο LESS αποδίδει καλύτερα γιατί: (1) τα δεδομένα που χρειάζεται να επεξεργαστούμε μετά το πέρασμα 0 στον LESS είναι λιγότερα από αυτά στον SFS, το οποίο μπορεί να έχει και ως αποτέλεσμα να χρειαστούν περισσότερα περάσματα στον SFS για να ολοκληρωθεί η ταξινόμηση, και (2) ο LESS μας εξοικονομεί τουλάχιστον ένα πέρασμα αφού συνδυάζει το τελικό πέρασμα συγχώνευσης με τη διαδικασία εξέτασης του skyline. Έχει αποδειχθεί ότι ο μέσος χρόνος εκτέλεσης του LESS είναι $O(_{dn})$, όπου d και n είναι η διάσταση και ο συνολικός αριθμός των δεδομένων αντίστοιχα.

4 *OSP Skyline αλγόριθμοι*

Ο υπολογισμός του skyline σε πολυδιάστατους χώρους αποτελεί μια πρόκληση και οι έλεγχοι κυριαρχίας ανά ζεύγη που είναι αρκετά έντονοι για την cpu αποτελούν τον κύριο παράγοντα κόστους.

Ο αλγόριθμος OSP (object-space-partitioning), που υλοποιήθηκε στην παρούσα διπλωματική, προτάθηκε από τους Zhang κ.α [ZMC09]. Ο OSP χωρίζει επαναλαμβανόμενα τον χώρο σε 2^d ξεχωριστά τμήματα έχοντας ως αναφορά ένα skyline σημείο/αντικείμενο και διευκολύνει την εύρεση του skyline προοδευτικά σε σύνολα δεδομένων πολλών διαστάσεων. Χρησιμοποιώντας αυτό το σχέδιο η μέθοδος οργανώνει τα τρέχοντα skyline σημεία σε ένα δέντρο αναζήτησης, που διευκολύνει τον αποτελεσματικό υπολογισμό του skyline. Κάθε τρέχον σημείο συγκρίνεται με μόνο ένα μικρό αριθμό από τα τρέχοντα skyline σημεία , χρησιμοποιώντας το δέντρο σαν οδηγό αναζήτησης. Κάνουμε μια θεωρητική ανάλυση, που υπολογίζει τον αναμενόμενο αριθμό των συγκρίσεων που πρέπει να

πραγματοποιηθούν ώστε να αποφασιστεί αν ένα σημείο είναι στο skyline. Στη συνέχεια κωδικοποιούμε τα τμήματα χρησιμοποιώντας bitmaps και χρησιμοποιούμε ένα left –child/right-sibling δέντρο για να τα οργανώσουμε. Το πλεονέκτημα αυτού του δέντρου είναι ότι επιτρέπει μια αποτελεσματική breadth-first αναζήτηση, ενώ η κωδικοποίηση επιτρέπει να γίνονται γρήγορες συγκρίσεις bit προς bit. Οι πειραματικές αξιολογήσεις που πραγματοποιήθηκαν από τους Zhang κ.α[ZMC09] δείχνουν ότι ο αλγόριθμος είναι αρκετές τάξεις πιο γρήγορος από τους υπόλοιπους, μειώνοντας τις απαιτούμενες συγκρίσεις κατά τη διάρκεια εύρεσης του skyline.

Ο σκοπός ήταν να βελτιωθεί η απόδοση των αλγορίθμων εύρεσης skyline που βασίζονται στην ταξινόμηση. Οι αλγόριθμοι αυτοί πρώτα ταξινομούν τα δεδομένα ώστε κανένα σημείο να μην κυριαρχεί κανένα άλλο που βρίσκεται πριν από αυτό στην σειρά. Στη συνέχεια, καθώς διατρέχουν το ταξινομημένο αρχείο συγκρίνουν κάθε σημείο με τα σημεία του skyline που έχουν βρεθεί μέχρι στιγμής και είναι αποθηκευμένα στη μνήμη. Εάν το τρέχον σημείο δεν κυριαρχείται από τα προηγούμενα skyline σημεία, εισέρχεται στο τρέχον skyline. Το σημείο καθυστέρησης αυτής της μεθόδου είναι η σύγκριση κάθε σημείου με το τρέχον skyline, που μπορεί να γίνει τόσο μεγάλο όσο και η διαθέσιμη μνήμη.

Ο σκοπός, λοιπόν ήταν, να μειωθεί το υπολογιστικό κόστος των ελέγχων αν το τρέχον σημείο που διαβάζεται είναι στο skyline κατά τη διάρκεια της αναζήτησης.

Η σημειογραφία που θα χρησιμοποιηθεί στη συνέχεια,φαίνεται στον παρακάτω πίνακα.

Σύμβολο	Ερμηνεία
O, o	Σύνολο αντικειμένων,αντικείμενο
d	Διάσταση
$H_i^{o+}(H_i^{o-})$	Άνω (Κάτω) υπό-χώρος με αναφορά το αντικ. ο και τη διάσταση i
$A_o(V)$	d-bitδιάσταση του partitionVμε γνώμονα το αντικείμενο ο
$L_o(o')$	το partitionπου βρίσκεται το o' με γνώμονα το αντικ. ο
$D_o(o')$	τα partitionπου κυριαρχούν ή είναι ισοδύναμα του $L_o(o')$
$U_o(o')$	τα partition που κυριαρχούνται από το $L_o(o')$
m	το βάθος του partitioningδέντρου
β	μέγιστος αριθμός αντικειμένων στα φύλλα του partitionδέντρου

4.1 Object-based Space Partitioning (OSP)

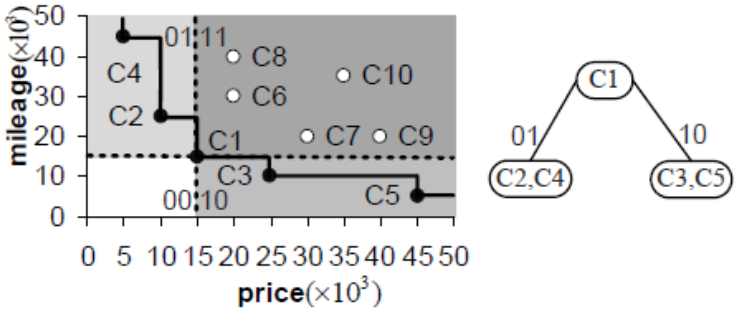
Ας θεωρήσουμε ένα σύνολο αντικειμένων O στον d-διάστατο χώρο R^d. Δεδομένου ενός σημείου $o=\{o_1,o_2,...,o_d\}$ στο O για κάθε διάσταση $i\in[1,d]$, το R^d μπορεί να χωριστεί σε 2 ημιχώρους, τον άνω ημιχώρο (superior halfspace) H_i^{o+} και τον κάτω ημιχώρο (inferior halfspace) H_i^{o-}, από το υπέρ-επίπεδο $R_i=o_i$, τέτοιο ώστε $\forall o'\in H_i^{o+}, o_{i'}<o_i$ και $\forall o'\in H_i^{o-}, o_{i'}\geq o_i$. Γι'αυτό, ο χώρος R^d μπορεί να χωριστεί σε 2^d τμήματα χρησιμοποιώντας το ο και κάθε σημείο $o'\in O\setminus o$ πρέπει να βρίσκεται αποκλειστικά σ'ένα από αυτά.

Ορισμός 4.1: Δοθέντος ενός skyline σημείου ο στο R^d, που ονομάζεται αναφορά, το ο χωρίζει το R^d σε 2^d ξεχωριστά τμήματα από dυπέρ-επίπεδα $\{R_1,R_2,...,R_d\}$, όπου $R_i=o_i$ και η διεύθυνση κάθε τμήματος V είναι ένας αριθμός d bit $A_o(V)$ τέτοιος ώστε για κάθε διάσταση i $A_o(V)[i]=0$, αν και μόνο αν, $V\subset H_i^{o+}$; $A_o(V)[i]=1. V\subset H_i^{o-}$. Επιπλέον, το τμήμα που περιέχει ένα αντικείμενο $o'\in O\setminus o$ ονομάζεται locating partition του ο' έχοντας ως αναφορά το ο και συμβολίζεται ως $L_o(o')$.

Με βάση τον ορισμό 4.1 και έχοντας σαν αναφορά ένα skyline αντικείμενο ο, μπορούμε να ορίσουμε ένα διαχωρισμό του χώρου σε 2^d περιοχές οι οποίες δε συμπίπτουν και να τους δώσουμε διευθύνσεις από 0 έως 2^d-1. Πρακτικά, η διεύθυνση του τμήματος ενός αντικειμένου ο' με σεβασμό στο ο μπορεί να υπολογιστεί ευθέως συγκρίνοντας τις συντεταγμένες του μ'αυτές του ο. Επειδή το ο είναι ένα skyline σημείο το τμήμα με διεύθυνση 00…0 πρέπει να είναι άδειο. Επίσης όλα τα αντικείμενα που βρίσκονται στο τμήμα με διεύθυνση 11…1 πρέπει να κυριαρχούνται από το ο, οπότε δεν είναι skyline σημεία. Τέλος, τα σημεία σε όλα τα άλλα τμήματα δεν μπορούν να συγκριθούν με το σημείο αναφοράς ο. Γι' αυτό, στην πράξη χρειάζεται να θεωρούμε μόνο τα τμήματα με διευθύνσεις $[1,....,2^d-2]$.

Αυτά τα 2^d-2 τμήματα οργανώνονται από ένα δέντρο, όπως φαίνεται στην παρακάτω εικόνα, με βάση τα παρακάτω δεδομένα. Σ' αυτό το δισδιάστατο παράδειγμα που τα γνωρίσματα είναι η τιμή (price) και τα χιλιόμετρα(mileage), σε μια βάση δεδομένων μεταχειρισμένων αυτοκινήτων το skyline αντικείμενο αναφοράς είναι το C_1. Τα αντικείμενα C_6-C_{10} κυριαρχούνται από το C_1, γι' αυτό και διαγράφονται. Τα αντικείμενα C_2 και C_4 δεν μπορούν να συγκριθούν με το C_1 και μπαίνουν στο κομμάτι (partition) με

διεύθυνση 01, ενώ τα αντικείμενα C_3 και C_5, που ούτε πάλι μπορούν να συγκριθούν με το C_1 και μπαίνουν στο partition με διεύθυνση 10.



Ορισμός 4.2: Αν τα τμήματα V και W ανήκουν στο ίδιο OSPS με βάση το ο, το V κυριαρχεί το W, και συμβολίζεται ως $V \succ W$ αν και μόνο αν $\forall i \in [1,d]$ $A_o(V)[i] \le A_o(W)[i] \wedge \exists j \in [1,d], A_o(V)[j] < A_o(W)[j]$, αλλιώς το V δεν κυριαρχεί το W και συμβολίζεται ως $V \not\succ W$ Επίσης τα VκαιW είναι ασύγκριτα αν και μόνο αν $V \not\succ W \wedge W \not\succ V$.

Θεώρημα 4.1: Δεδομένων των $o, o', o'' \in O$ αν $o' \succ o''$ τότε $L_o(o') \succeq L_o(o'')$.

Λήμμα 4.1: $V \succ W$ αν και μόνο αν i) $A_o(V) < A_o(W)$ και ii) $(A_o(V) \mid A_o(W)) = A_o(W)$.

Συμπέρασμα: Το V είναι μη συγκρίσιμο με το W, αν και μόνο αν, $(A_o(V) \mid A_o(W)) > \max\{A_o(V), A_o(W)\}$.

Λήμμα 4.2: Αν V και W είναι μη συγκρίσιμα τότε $\forall v \in V, \forall w \in W$,τα v ,w είναι μη συγκρίσιμα.

Το λήμμα 4.2 υπονοεί ότι οι έλεγχοι κυριαρχίας ανά ζεύγος ανάμεσα σε μη συγκρίσιμα τμήματα μπορεί να αγνοηθούν. Αντίστροφα, κάθε αντικείμενο ο' μπορεί να μην κυριαρχείται από μερικά αντικείμενα στα κυρίαρχα τμήματα του $L_o(o')$ και οι έλεγχοι κυριαρχίας ανά ζεύγος σε σχέση με το ο' για αντικείμενα σ'αυτά είναι απαραίτητοι, αν θέλουμε να ελέγξουμε αν το ο' είναι skyline σημείο.

Ορισμός 4.3: Το σύνολο τμημάτων που κυριαρχεί $D_o(o')$ και το σύνολο τμημάτων που κυριαρχείται $U_o(o')$ ενός σημείου ο' $\in O \setminus o$ με σεβασμό στο ο ορίζονται ως εξής:

$$\mathrm{D}_o(o') = \{V \in \mathcal{S} \mid V \succeq \mathcal{L}_o(o')\}$$
$$\mathrm{U}_o(o') = \{V \in \mathcal{S} \mid \mathcal{L}_o(o') \succ V\}$$

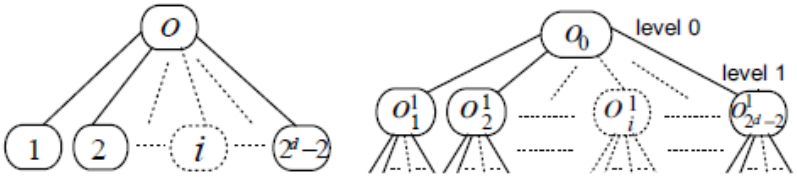
Ας υποθέσουμε ότι το αντικείμενο ο' δεν είναι χειρότερο από το ο σε k διαστάσεις ή ισοδύναμα ότι το $A_o(o')$ (ή $L_o(o')$) διατηρεί k bits. Τότε $D_o(o') = 2^{k-1}$ και $|U_o(o')| = 2^{d-k} - 1$. Αν οργανώσουμε τα skyline σημεία που έχουν βρεθεί μέχρι στιγμής στον OSP με βάση το ο, τότε για να καθορίσουμε αν ένα υποψήφιο $o_c \in O$ που μελετάται αυτήν τη στιγμή είναι στο skyline, είναι αρκετό αυτό να περάσει όλους τους ελέγχους κυριαρχίας με skyline αντικείμενα στα τμήματα του $D_o(o_c)$.

Τα skyline αντικείμενα στα τμήματα του $D_o(o_c)$ αναμένεται να είναι πολύ λιγότερα από τον αριθμό των skyline σημείων που έχουν βρεθεί μέχρι στιγμής, γι' αυτό και αυτός ο αλγόριθμος αναμένεται να μειώσει αρκετά τους υπολογισμούς σε σχέση με άλλες τεχνικές για την εύρεση του skyline που συγκρίνουν το εκάστοτε αντικείμενο με ολόκληρο το skyline σύνολο στον αποθηκευτικό χώρο.

4.1.1 Recursive Object-based Space Partitioning

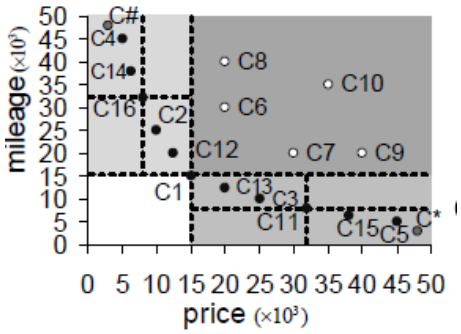
Ο ορισμός 4.1 μας δίνει το βασικό σχέδιο για την τμηματοποίηση (partitioning) με βάση τα αντικείμενα για να χωρίσουμε το χώρο R^d σε 2^d μη-επικαλυπτόμενα τμήματα με βάση το αντικείμενο αναφοράς ο. Οποιοδήποτε αντικείμενο $o' \in O$ μπορεί μετά να αναγνωριστεί ότι βρίσκεται σ'ένα από αυτά τα τμήματα. Τα $2^d - 2$ θεωρούμενα τμήματα ταξινομούνται με τη βοήθεια δεικτών από ένα δέντρο τμημάτων. Μάλιστα, αυτό το OSP σχέδιο με σεβασμό σ'ένα skyline αντικείμενο μπορεί να εφαρμοστεί αναδρομικά σε κάθε τμήμα και όλες οι ιδιότητες και οι απόρροιες που παρουσιάζονται παραπάνω διατηρούν για το καθένα επακόλουθη τμηματοποίηση (partitioning). Τα τμήματα (partition) μπορούν να οργανωθούνε με βάση ένα ιεραρχικό partition δέντρο. Συγκεκριμένα, κάθε εσωτερικός κόμβος περιέχει ένα skyline αντικείμενο αναφοράς, το οποίο υποδιαιρεί το τμήμα που αναπαριστάται από αυτόν τον κόμβο. Για κάθε αντικείμενο ο', μπορούμε να το συγκρίνουμε επαναλαμβανόμενα με το αντικείμενο αναφοράς στο τμήμα που ανήκει σε κάθε επίπεδο το δέντρου και εν-τέλει να βρούμε το κομμάτι με το καλύτερο επίπεδο στο οποίο ανήκει αυτό. Κατά τη διάρκεια της διάσχισης του δέντρου, για κάθε κόμβο, που αντιστοιχεί σ'ένα αντικείμενο αναφοράς ο, χρειάζεται να ελέγχουμε αν το ο' περνάει όλους τους ελέγχους κυριαρχίας πάνω στα τμήματα που το κυριαρχούν $D_o(o')$. Αν το ο' δεν απορριφθεί κατά τη διάρκεια αυτής της διαδικασίας και φτάσει τελικά στο κομμάτι (partition) κάποιου φύλλου του δέντρου, μπορούμε να είμαστε σίγουροι ότι το ο' ανήκει στο skyline του Ο. Αυτό το τμήμα (partition) ονομάζεται skyline δέντρο και το χρησιμοποιούμε ώστε να δείχνει ιεραρχικά όλα τα αντικείμενα του skyline.

Ένα παράδειγμα skyline δέντρου καθώς και ενός αναδρομικού φαίνεται παρακάτω.

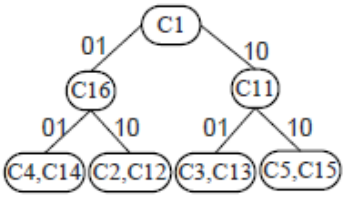


(a) OSP Skyline Tree (b) ROSP Skyline Tree

Σαν παράδειγμα αυτού του αναδρομικού OSP σχεδίου ας θεωρήσουμε τα σημεία που φαίνονται στην παρακάτω εικόνα.



Ας υποθέσουμε ότι το αντικείμενο C_1 χρησιμοποιείται στο διαμερισμό στο επίπεδο-0. Τα τμήματα (partitions) 01 και 10 που χωρίζονται από το C_1, διαμερίζονται ξανά χρησιμοποιώντας τα skyline αντικείμενα C_{16} και C_{11} αντίστοιχα. Το αντίστοιχο ROSP skyline δέντρο φαίνεται παρακάτω.



Το υποψήφιο αντικείμενο C^* συγκρίνεται πρώτα με το C_1 και βρίσκεται ότι ανήκει στο τμήμα (partition) 10. Στη συνέχεια συγκρίνεται με το C_{11} και μπαίνει στο υπό-τμήμα του 10 που περιλαμβάνει τα C_5 και C_{15}. Αφού συγκριθεί με αυτά τα αντικείμενα, συμπεραίνουμε ότι είναι ένα καινούριο skyline αντικείμενο και μπαίνει σ' αυτό το partition. Ομοίως, το $C^{\#}$ συγκρίνεται με τα C_1, C_{16} και τέλος με τα αντικείμενα, C_4 και C_{14}, στα φύλλα του δέντρου. Σ' αυτό το δισδιάστατο παράδειγμα ένα υποψήφιο skyline αντικείμενο (object) συγκρίνεται μόνο μ' ένα αντικείμενο σε κάθε επίπεδο του δέντρου, μέχρι να φτάσει στο τελικό (φύλλο) partition, όπου και συγκρίνεται με όλα τα αντικείμενα που βρίσκονται εκεί. Σε περιπτώσεις περισσότερων διαστάσεων, ένα αντικείμενο o' μπορεί να χρειάζεται να συγκριθεί

με πολλά partitions ανά επίπεδο, καθώς το σύνολο των τμημάτων (partition) $D_o(o')$, που το κυριαρχεί έχει μέγεθος μεγαλύτερο του ένα. Αυτό σημαίνει ότι διασχίζονται γενικά πολλά μονοπάτια του skyline δέντρου.

4.1.2 Εκτιμήσεις-Συμπεράσματα

Για απλότητα θεωρούμε ότι έχουμε έναν μεγάλο αριθμό σημείων στο d-διάστατο χώρο R^d, ομοιομόρφα και ανεξάρτητα κατανεμημένα (UI). Αυτό υπονοεί οτι ο χώρος διαμερίζεται από ένα skyline σημείο ο που είναι πάνω ή κοντά στην κύρια διαγώνιο. Κάθε δεικτοδοτούμενο partition στο OSP partition δέντρο περιέχει τον ίδιο αριθμό skyline σημείων. Με άλλα λόγια ένα καινούριο υποψήφιο σημείο o_c θα τοποθετηθεί σε ένα από αυτά τα partition με την ίδια πιθανότητα $1/2^d$ -2. Με βάση το θεώρημα 4.1, είναι αρκετό το o_c να περάσει όλους τους ελέγχους κυριαρχίας με σημεία στα partition στο $D_o(o_c)$, ώστε να επιβεβαιωθεί αν το o_c βρίσκεται στο skyline. Το αναμενόμενο ποσοστό των υπάρχοντων skyline σημείων που θα συγκριθεί με το τωρινό υποψηφίο o_c, δίνεται από το παρακάτω θεώρημα.

Θεώρημα 4.2: Με την υπόθεση οτι έχουμε ένα UI σύνολο σημείων και με τη βοήθεια ενός OSP δέντρου με βάση ένα skyline σημείο ο, το μέσο ποσοστό συγκρίσεων R(d) για ένα υποψήφιο skyline σημείο o_c, είναι $O(\frac{3^d - 2^{d+1} + 1}{(2^d - 2)^2}) < O((\frac{3}{4})^d)$.

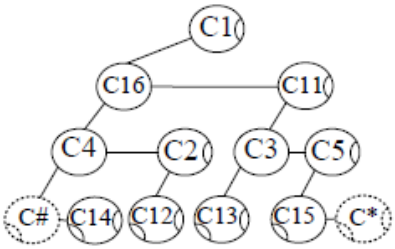
Επίσης διαπιστώνουμε οτι:

Θεώρημα 4.3: Με την υπόθεση οτι έχουμε ένα UI σύνολο σημείων και χρησιμοποιώντας ένα ROSP skyline δέντρο με ύψος m, το μέσο ποσοστό ελέγχων κυριαρχίας που γίνονται μεταξύ δύο σημείων, R(*), για ένα υποψήφιο skyline είναι: $O(R(d)^m) < O((\frac{3}{4})^{dm})$.

Γενικά, βλέπουμε οτι η επιλογή των σημείων αναφοράς για το διαμερισμό του χώρου επηρέαζει σημαντικά τη δομή του δέντρου και την απόδοση. Στη χειρότερη περίπτωση, το ROSP δέντρο εκφυλίζεται σε μια λίστα και τότε η στρατηγική των ελέγχων κυριαρχίας είναι όμοια με αυτήπου χρησιμοποιούν οι άλλοι αλγόριθμοι (όπως π.χ. ο LESS). Για να ξεπεραστεί αυτό το πρόβλημα χρησιμοποιούμε μια τεχνική που μαντεύει τυχαία το σημείο αναφοράς για κάθε διαμερισμό του χώρου (partitioning). Παράλληλα, χρησιμοποιούμε ένα Left Child/Right Sibling δομή δέντρου ενσωματώνοντας το ROSP δέντρο, που διευκολύνει στο να έχουμε αποτελεσματικούς ελέγχους κυριαρχίας.

4.2 Left-Child/Right-Sibling Skyline Tree

Όπως φαίνεται και στο παρακάτω σχήμα, τα skyline αντικείμενα μπορούν να τοποθετηθούν με τη χρήση δείκτη σε ένα left-child/right-sibling δέντρο με βάση τις διευθύνσεις των τμημάτων τους με αναφορά στο ROSP σχέδιο. Αυτή η εφαρμογή εξυπηρετεί 2 σκοπούς: i) υψηλή αποδοτικότητα χώρου , αφού πολλά τμήματα είναι άδεια και πρέπει να δεικτοποιηθούν και ii) αποτελεσματική σειριακή πρόσβαση των τμημάτων και των αντικειμένων κατά πλάτος.



Ορισμός 4.4: Ένας κόμβος e σ' ένα LCRS (left-child/right-sibling) δέντρο είναι ένα σύνολο 4 πραγμάτων <o,pa,child,sb>, όπου ο είναι το skyline αντικείμενο (το οποίο μπορεί υποδιαιρεί αυτή τη διαμέριση του κόμβου, αν ο κόμβος δεν είναι φύλλο), pa είναι η διεύθυνση της διαμέρισης (partition) που αναπαριστάται από τον κόμβο , το child είναι ένας δείκτης που δείχνει στο πιο αριστερό υπο-τμήμα του ο που δεν είναι άδειο, sb είναι ο δείκτης στο δεξί γειτονικό υπό-τμήμα του ο που δεν είναι άδειο.

Προφανώς, η ρίζα του δέντρου είναι το πρώτο αντικείμενο αναφοράς που διαμερίζει το R^d , οπότε το pa του πρέπει να είναι 0 και το sb του null (κενό). Οι γειτονικοί κόμβοι ταξινομούνται με αύξουσα διεύθυνση διαμέρισης pa.

Για κάθε κόμβο e στο LCRS δέντρο, όλα τα αντικείμενα του υπόδεντρου-παιδιού του μαζί με το αντικείμενο του κόμβου του e.o είναι στο ίδιο κομμάτι και μοιράζονται την ίδια διεύθυνση έχοντας ως αναφορά το αντικείμενο του κόμβου-γονιού(parent) $\hat{o}$. Ας θεωρήσουμε ένα υποψήφιο skyline αντικείμενο ο με διεύθυνση $A_{\hat{o}}(o)$. Αν $(A_{\hat{o}}(o) \mid e.pa) = e.pa$,είναι εμφανές ότι η διαμέριση (partition) που αναπαριστάται από τον κόμβο e είναι στο $D_{\hat{o}}(o)$ και $e.pa \le A_{\hat{o}}(o)$. Γι'αυτόν το λόγο τα αντικείμενα στο υπό-δέντρο του e συμπεριλαμβανομένου του αντικειμένου αναφοράς e.o πρέπει να συγκριθούν με το υποψήφιο ο για να ανιχνεύσουμε αν το ο κυριαρχείται από κάποιο απ'αυτά.

Αλλιώς, ο κόμβος e θα προσπεραστεί και όλα τα αντικείμενα στο υπό-δέντρο παιδί (child) του μαζί με το αντικείμενο του κόμβου του e.o μπορούν να αγνοηθούν. Γι'αυτόν το λόγο οι έλεγχοι κυριαρχίας για κάθε υποψήφιο ο μπορούν εύκολα να ενσωματωθούν βασιζόμενοι σε μια προδιατεταγμένη διάσχιση του

δέντρου LCRS. Η διαδικασία σχηματισμού του LCRS skyline δέντρου μπορεί εύκολα να προσαρμοστεί και σ'ένα LCRS δέντρο διαμέρισης, αν μερικοί κόμβοι –παιδιά e, εκτός του αντικειμένου αναφοράς e.o, διατηρούν όλα τα αντικείμενα τους στο αντίστοιχο κομμάτι (partition).

Παρακάτω παρουσιάζεται ο αλγόριθμος 1, όπου φαίνεται πως μπορούμε να διαχειριστούμε τα skyline αντικείμενα δυναμικά με την βοήθεια ενός LCRS δέντρου.

Algorithm 1:PreOrderDominate(e,o,pa,isL)

```
Input: e:LCRS tree node; o:candidate;pa:L(o)'s address
       isL: whether e is on o's locating path
Output: whether o is dominated by some skyline point
begin
  if(e.pa|pa)=pa then //check dominating node
    Pa:= A_{e.o}(o) //compute A_{e.o}(o)
   if Pa=2^d-1 then //o is in the partition 11..1
    return true  //e.o dominates o
    inSL:=isL∧(e.pa=pa) //whether o definitely is in e
   if e.child∧e.child.pa≤Pa then
   if PreOrderDominate(e.child,o,Pa,inSL) then
   return true //o is dominated by a descendant of e
    else if inSL then  //insert o as e.child
     e.child:=<o,Pa,null,e.child>
   return false
  if e.sb∧(e.sb.pa≤pa) then
   if PreOrderDominate(e.sb,o,pa,isL)then
      return true //o is dominated by a sibling of e
  else if isL∧(e.pa<pa) then  //insert o as e.sb
    e.sb:=<o,pa,null,e.sb>
  return false
end
```

Κάθε κόμβος e του δέντρου διασχίζεται και συγκρίνεται με το υποψήφιο ο αν η διεύθυνση διαμέρισης του e.pa έχοντας ως αναφορά το αντικείμενο του κόμβου-γονέα ικανοποιεί δύο συνθήκες: (i) $e.pa \leq A_{\hat{o}}(o))$ και (ii) $(A_{\hat{o}}(o) \mid e.pa) = e.pa$. Γι'αυτόν το λόγο για ένα υποψήφιο αντικείμενο ο, κάνουμε μια προδιατεταγμένη διάσχιση του δέντρου, όπως περιγράφεται στον αλγόριθμο 1. Αν το υπό-δέντρο του e ή ένας από τους γειτονικούς κόμβους του e χρειάζεται να διασχιστεί, εξαρτάται από τις 2 αυτές συνθήκες. Συγκεκριμένα, για κάθε κόμβο e που επισκεπτόμαστε υπάρχουν 2 περιπτώσεις για το υποψήφιο ο με την διεύθυνση του τοπικού τμήματος e.pa με σεβασμό στον κόμβο-γονέα $\hat{o}$.

- **Περίπτωση 1:** $(e.pa \mid pa) = pa$. Σ'αυτήν την περίπτωση ο κόμβος e είναι ο κυρίαρχος κόμβος για το υποψήφιο ο. Ο αλγόριθμος πρέπει πρώτα να ελέγξει αν το αντικείμενο του κόμβου e.o κυριαρχεί το ο υπολογίζοντας τη διεύθυνση της διαμέρισης του ο P_a με σεβασμό στο e.o. Αν το ο δε διαγραφεί, τότε ο αλγόριθμος εκτελείται αναδρομικά για το e.child. Εδώ η μέθοδος ελέγχει πρώτα αν ο κόμβος e βρίσκεται στο τρέχον τμήμα του ο, όπως δείχνει η μεταβλητή inSL μαζί με την isL. Η isL δείχνει αν ο επισκεπτόμενος κόμβος e είναι στο τρέχον μονοπάτι που κληρονομείται από την επαναλαμβανόμενη κλήση του γονέα. Αν isL=false εκτελούμε ελέγχους κυριαρχίας για το ο με το e και το e.child, αλλά το ο δεν μπορεί να εισέλθει στο υπό-δέντρο του e. Αλλιώς, δηλαδή αν isL=true, το e είναι στο τρέχον κομμάτι του ο και το e.child.pa είναι μικρότερο από το o.pa, ο αλγόριθμος καλείται αναδρομικά για το e.child και το ο θα εισέλθει πιθανά σ' αυτό το υπό-δέντρο. Παρολαυτά, αν το e δεν έχει παιδί με e.chil.pa<o.pa, τότε το ο μπορεί να εισέλθει στο δέντρο σαν το πιο αριστερό παιδί του e και συνδέεται με αυτό το παιδί μ'έναν γειτονικό δείκτη.
- **Περίπτωση 2:** $(e.pa \mid pa) \prec pa$. Σ'αυτήν την περίπτωση ο κόμβος e παραλείπεται και ο έλεγχος περνά στον γείτονα του e.sb, αν το e.sb υπάρχει και ικανοποιεί τη συνθήκη i). Αλλιώς ο γειτονικός κόμβος δημιουργείται δυναμικά και το ο εισέρχεται εκεί.

Αυτή η προδιατεταγμένη διάσχιση δεν τερματίζεται αν το υποψήφιο ο ή κυριαρχείται από το αντικείμενο ή εισέρχεται στο e σαν νέο skyline αντικείμενο. Η ειδική μεταβλητή isL χρησιμοποιείται για να δείξει αν το υποψήφιο ο είναι σ'έναν κόμβο και έχει την ίδια διεύθυνση όπως το ο με σεβασμό στον κόμβο-γονέα (parent). Αν ισχύει αυτό, το υποψήφιο αντικείμενο μπορεί να μπει κάτω από αυτόν τον κόμβο. Αλλιώς ο κόμβος παραλείπεται.

Στην εργασία των Zhang κ.α [ZMC09], προτάθηκαν κάποιοι αλγόριθμοι για τον υπολογισμό του skyline, οι οποίοι βασίζονται στην PreOrderDominate μέθοδο.

Η πρώτη μέθοδος OSPSOnSortingFirst (αλγόριθμος 2) είναι μια εφαρμογή της διαδικασίας επέκτασης του LCRS δέντρου. Ακολουθεί το παράδειγμα των skyline αλγορίθμων που βασίζονται στην ταξινόμηση. Τα δεδομένα ταξινομούνται τοπολογικά έτσι ώστε κανένα αντικείμενο να μην κυριαρχείται από αυτά που βρίσκονται πριν από αυτό στο ταξινομημένο σύνολο δεδομένων. Στη συνέχεια, πρέπει να θέσουμε ένα skyline αντικείμενο σαν ρίζα του δέντρου, οπότε βάζουμε το πρώτο αντικείμενο για να είμαστε σίγουροι ότι ανήκει στο skyline. Για να αυξήσουμε την πιθανότητα να έχουμε ισορροπημένες διαμερίσεις 0-επιπέδου, αντί του να επιλέξουμε το πρώτο αντικείμενο σαν αναφορά, μπορούμε να επιλέξουμε ένα τυχαίο skyline σημείο διατρέχοντας τα δεδομένα μια φορά. Στη συνέχεια ο OSPSOnSortingFirst διασχίζει τα υπόλοιπα αντικείμενα και εφαρμόζει την διαδικασία PreOrderDominate με pa=0 και isL=true. Τελικά έχουμε το SL σαν αποτέλεσμα ως το skyline δέντρο.

```
Algorithm 2:OSPSOnSortingFirst(O)
Input:O:dataset;
Output:SL: LCRSSkylineTree
begin
  SortObyatopologicalmonotonefunction F
  s_first :=a skyline object in O
  SL:=<s_first,0,null,null> //initialize SL
  foreach o ∈O\s_first do  //check all other objects in O
    PreOrderDominate(SL,o,0,true)
return SL
end
```

Η δεύτερη μέθοδος OSPSOnPartitiongFirst (αλγόριθμος 3) δε βασίζεται στην ταξινόμηση, αλλά προσπαθεί να χωρίσει το σύνολο δεδομένων σε τμήματα (partition) και να λύσει ανεξάρτητα προβλήματα. Η κύρια ιδέα βασίζεται στο να διαμερίσουμε δυναμικά το σύνολο δεδομένων O, καθώς μεγαλώνει το LCRS δέντρο. Ο αλγόριθμος 3 που είναι ο ψευδοκώδικας της δεύτερης μας μεθόδου είναι μια προσαρμοσμένη έκδοση της PreOrderDominate.

Ξέρουμε ότι τα αντικείμενα σε μια διαμέριση με μεγαλύτερη διεύθυνση δεν μπορούν να κυριαρχήσουν αντικείμενα σε διαμερίσεις με μικρότερες διευθύνσεις. Οπότε αν μεγαλώσουμε το δέντρο ξεκινώντας από τις διαμερίσεις με τις μικρότερες διευθύνσεις, μπορούμε εύκολα να διαγράψουμε τις διαμερίσεις που κυριαρχούνται από αυτές, όσο προχωρά η διαδικασία. Οι λεπτομέρειες της μεθόδου είναι οι ακόλουθες. Πρώτα, χωρίζουμε αναδρομικά το πιο αριστερό παιδί (child) της διαμέρισης SO ενός δυναμικά ορισμένου LCRS δέντρου διαμέρισης (partition tree) μέχρι να περιλαμβάνει το πολύ το μέγιστο δυνατό αριθμό αντικειμένων. Στη συνέχεια υπολογίζουμε το τοπικό skyline δέντρο SL για τα αντικείμενα του SO. Το SL φιλτράρει όλα τα τμήματα $U_o(SO)$ που κυριαρχούνται από αυτό ανάμεσα στους τρέχοντες γειτονικούς κόμβους. Τέλος, καλούμε αναδρομικά τον αλγόριθμο θέτοντας το SO σα γείτονα (sibling) ή αν δεν υπάρχουν γειτονικοί κόμβοι πηγαίνουμε προς τα πίσω στο αναδρομικό

δέντρο στην επόμενη διαμέριση (partition) στη σειρά. Καθώς διαμερίζουμε το SO ο αλγόριθμος διαγράφει όλα τα αντικείμενα που κυριαρχούνται από την αναφορά r. Σημειώνουμε ότι το αντικείμενο αναφοράς r στη διαμέριση του SO πρέπει να είναι ένα αντικείμενο skyline. Για να βρούμε ένα τέτοιο αντικείμενο αρκεί να κάνουμε την αναζήτηση μέσα στη διαμέριση, αφού τα αντικείμενα εκεί που κυριαρχούνται από άλλες διαμερίσεις έπρεπε να είχαν φιλτραριστεί νωρίτερα.

```
Algortihm 3:OSPSOnPartitioningFirst(SO)
Input: SO:LCRS Partition Tree
Output: SL:LCRS Skyline Tree
begin
  if SO=null∨|SO.O|=0 then return null
  return OSPSOnSortingFirst(SO.O)
 r:=a skyline object in SO.O
 SL:=<r,SO.pa,null,null>
 partition SO w.r.t r,prune all r's dominating objects
 SL.child:=OSPSOnPartitoningFirst(SO.child)
 FilterDominatedPartitions(SO.sb,SL)
 SO:=SO.sb
 SL.sb:=OSPSOnPartitoningFirst(SO)
 return SL
end
```

Αν η χωρητική ικανότητα β του κόμβου-φύλλου γίνει 1, οι διαμερίσεις (partition) υποδιαιρούνται αναδρομικά μέχρι να περιέχουν ένα μόνο αντικείμενο και αυτό το αντικείμενο είναι σίγουρα ένα skyline αντικείμενο. Γενικά, μικρότερες τιμές για το β έχουν σαν αποτέλεσμα διαμερίσεις (partition) με μεγαλύτερο βάθος, το οποίο βοηθά στο να διαγραφούν περισσότερα αντικείμενα και περιορίζει το κόστος σε σχέση με την ταξινόμηση. Απ'την άλλη ένα μικρό β συνεπάγεται και περισσότερες κατατμήσεις (partitioning) που μπορεί να αυξήσουν το I/O κόστος.

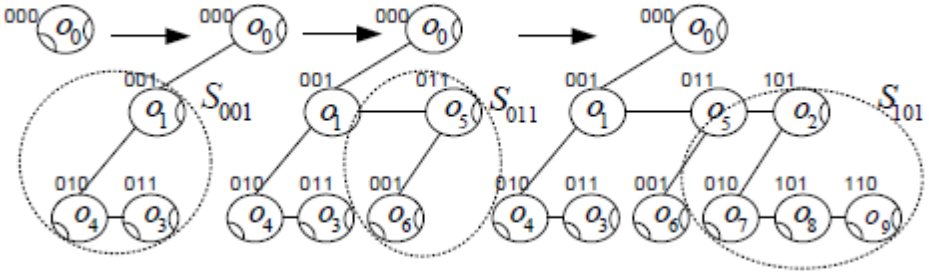
Η διαδικασία φιλτραρίσματος υλοποιείται από τον αλγόριθμο 4. Αν το εξεταζόμενο γειτονικό τμήμα SO κυριαρχείται από τον κόμβο-ρίζα του SL, τα αντικείμενα σ'αυτό θα φιλτραριστούν από το SL χρησιμοποιώντας την PreOrderDominate με isL=false, αφού χρειάζεται να κάνουμε ελέγχους κυριαρχίας μόνο πάνω στο SL , αλλά τα αντικείμενα που απομένουν μπορεί να μην εισέλθουν στο SL. Η μέθοδος εξετάζει αναδρομικά τα γειτονικά τμήματα του SO στη σειρά, ελέγχοντας όλα τα

κυριαρχούνται τμήματα. Αν ένα τμήμα είναι άδειο, θα απορριφθεί ώστε να αποφύγουμε πλεονάζοντες κόμβους στο δέντρο κατάτμησης. Μετά την ολοκλήρωση της διαδικασίας φιλτραρίσματος, τα αντικείμενα που κυριαρχούνται από κάποια skyline αντικείμενα στο SL θα διαγραφούν σε όλα τα τμήματα/κομμάτια που κυριαρχούνται που είναι γείτονες του SO.

Algorithm 4: FilterDominatedPartitions(SO,SL)

```
Input: SO:LCRS partition; SL: skyline tree pre-sibling of SO
Output: SO:LCRS partition filtered using SL
begin
    if SO=null then return
    if(SO.pa|SL.pa)=SO.pa then  //SL dominates SO
        foreach o∈SO.O do
            if PreOrderDominate(SL,o,SO.pa,false)then
                remove o from SO.O
    FilterDominatedPartitions(SO.sb,SL)
    if |SO.O|=0 then
        delete SO from LCRS partition tree; SO:=SO.sb
end
```

Για να γίνουν τα παραπάνω πιο κατανοητά ας θεωρήσουμε ένα τρισδιάστατο σύνολο δεδομένων O={ $o_0, o_1, ..., o_9$ } και ας υποθέσουμε οτι το β είναι 3. Πρώτα ο αλγόριθμος μας θα επιλέξει ένα σημείο r από το O. Έστω οτι r= o_0 . Με βάση το r, ο αλγόριθμος χωρίζει το σύνολο των δεδομένων σε τρία υποσύνολα : P001= { o_1, o_3, o_4 } ,P011= { o_5, o_6 }, P101= { o_2, o_7, o_8, o_9 } με διευθύνσεις 001, 011 και 101 αντίστοιχα. Θα επεξεργαστούμε πρώτα το υποσύνολο (partition) P001 και επειδή το μέγεθος του δεν είναι μεγαλύτερο από το β, το skyline δέντρο του θα υπολογιστεί με βάση τον αλγόριθμο 2 OSPSOnSortingFirst. Υποθέτουμε οτι το skyline δέντρο του S_{001} αποτελείται από τα o_1, o_3, o_4 , όπως φαίνεται στην παρακάτω εικόνα.



Στο επόμενο βήμα φιλτράρουμε τα γειτονικά partition, P101 και P101, του P001, χρησιμοποιώντας το S_{001} , αφού και τα δύο κυριαρχούνται από το partition P001. Ας υποθέσουμε οτι κανένα από τα σημεία/αντικείμενα σ' αυτά δεν μπορούν να διαγραφούν. Στο επόμενο βήμα, συνεχίζουμε επεξεργαζόμενοι το γειτονικό κόμβο P001 χρησιμοποιώντας την ίδια διαδικασία και έτσι επιστρέφεται το τοπικό skyline δέντρο S_{011} , που περιέχει τα o_5, o_6 . Το δέντρο αυτό συνδέεται ως γειτονικό του S_{001} . Στη συνέχεια εφαρμόζεται η διαδικασία φιλτραρίσματος, αλλά δε χρειάζεται να κάνει κάτι, καθώς το P011 δεν κυριαρχεί το P101. Στο τελευταίο βήμα επεξεργαζόμαστε το partition P101. Αφού περιέχει παραπάνω από β σημεία, σπάει σε 3 μικρότερα κομμάτια χρησιμοποιώντας το o_2 . Τα partition αυτά περιέχουν τα σημεία o_7, o_8, o_9 αντίστοιχα. Αφού επεξεργαστεί τα τρία αυτά ο αλγόριθμος θα υπαναχωρήσει προς τη ρίζα και θα τεραματίσει. Η παραπάνω εικόνα δείχνει το σταδιακό σχηματισμό του skyline δέντρου.

Γενικά, ο OSPSOnPartitioningFirst είναι γρηγορότερος από τον OSPSOnSortingFirst καθώς αποφεύγει την ταξινόμηση και εξετάζει επίσης τα σημεία προσπαθώντας να διαφράψει τα σημεία μόλις βρεθούν skyline σημεία στα κυριαρχούντα (dominating) τους partition.

4.2.1 OSP αλγόριθμοι με χρήση της τεχνικής SALSA

Εδώ θα συζητήσουμε, το πως οι προτεινόμενοι αλγόριθμοι μπορούν να ενσωματώσουν μονοκόμματα τη τεχνική βελτιστοποίησης που χρησιμοποιεί ο SALSA (Sort and Limit Skyline Algorithm) για πρόωρο τερματισμό, αλλά και θα αναλύσουμε τις απαιτήσεις τους σε χώρο και τον αναμενόμενο χρόνο επεξεργασίας.

Στο LCRS skyline δέντρο μπορούμε να διατηρήσουμε δύο επιπρόσθετες τιμές minMax από όλα τα skyline αντικείμενα (i) στο υπό-δέντρο με ρίζα το e και (ii) στο γειτονικό (sibling) υπόδεντρο του e, με τον συμβολισμό T_e και $T_{e.sb}$ αντίστοιχα. Άρα η συνθήκη τερματισμού του SALSA μπορεί να εφαρμοστεί σε κάθε κόμβο χρησιμοποιώντας το T_e και το $T_{e.sb}$. Αν η συνθήκη τερματισμού του T_e ικανοποιείται όταν επισκεπτόμαστε τον κόμβο e, οι έλεγχοι κυριαρχίας στο (i) μπορούν να παραλειφθούν χωρίς κίνδυνο να χάσουμε κάποιο skyline σημείο. Το ίδιο μπορεί να συμβεί κάνοντας συγκρίσεις στο γειτονικό υπόδεντρο του e, χρησιμοποιώντας το $T_{e.sb}$. Εξετάζοντας τη συνθήκη τερματισμού στον κόμβο ρίζα του LCRS δέντρου ο OSPSOnSortingFirst μπορεί να τερματίσει το ταζύτερο δυνατό. Ο OSPSOnPartitioningFirst εφαρμόζει την ίδια ιδέα σε κάθε κομμάτι (partition) και όταν συγκρίνει partition που κυριαρχούν το ένα το άλλο.

Κατά τη διάρκεια της αναζήτησης, μόνο το τωρινό LCRS skyline δέντρο SL πρέπει να διατηρείται στη μνήμη των δύο αλγόριθμων. Επίσης, το βάθος της αναδρομικής κλήσης στους ελέγχους κυριαρχίας που γίνονται με βάση τα partition περιορίζεται από το μέγιστο μήκος του μονοπατιού, το οποίο περιορίζεται από το μέγεθος του skyline r. Άρα η πολυπλοκότητα του χώρου στους αλγόριθμους είναι O(r). Για ένα πρωταρχικό υποψήφιο σημείο o_c, αν δεν είναι skyline σημείο, θα πρέπει να κυριαρχείται από ένα skyine σημείο από τα $D_o(o_c)$ σε κάθε επίπεδο του LCRS skyline δέντρου SL. Συνεπώς, ο αναμενόμενος αριθμός των ελέγχων κυριαρχίας γι' αυτό είναι O(R(*)|SL) σε indepedent δεδομένα. Στη χειρότερη περίπτωση, όπου το SL υποβαθμίζεται σε μια συνδεδεμένη λίστα, η πολυπλοκότητα του χρόνου των παραπάνω μεθόδων είναι ίδια με αυτή των προηγούμενων προσεγγίσεων, δηλαδή O(nr), όπου n είναι το μέγεθος του συνόλου δεδομένων. Παρόλαυτα περιμένουμε ο αριθμός των έλεγχων κυριαρχίας να είναι αρκετά μικρότερος, δηλαδή O(nR(*)logr), αν το δέντρο είναι ισορροπημένο.

4.3 OSP σε περίπτωση περιορισμένης μνήμης

Κοιτώντας πιο προσεχτικά την OSPSOnPartitiongFirst καταλαβαίνουμε ότι κατά τη διάρκεια της εκτέλεσης του αλγορίθμου αυτού μπορούμε να βρούμε και να αφαιρέσουμε από τη μνήμη το κομμάτι του υπολογισμένου τοπικού skyline δέντρου που έχει ήδη χρησιμοποιηθεί για να φιλτράρει αντικείμενα στα τμήματα που κυριαρχεί. Στην περίπτωση που το skyline δέντρο μεγαλώνει περισσότερο από τη διαθέσιμη μνήμη προτάθηκε ο αλγόριθμος 5.

Εδώ χρησιμοποιείται ένα προσωρινό αρχείο T για τη βαθμιαία συλλογή των κόμβων του τοπικού skyline δέντρου που απορρίπτονται από τη μνήμη. Αν το εξεταζόμενο τμήμα δε χωράει στη μνήμη ξαναδιαιρείται χρησιμοποιώντας ένα από τα skyline αντικείμενα του και ο αλγόριθμος εφαρμόζεται αναδρομικά στο πιο αριστερό υπό-τμήμα του (child). Αλλιώς, η OSPSOnPartitioningFirst εφαρμόζεται για να επιστρέψει το skyline δέντρο SL περιλαμβάνοντας μόνο το εξεταζόμενο τμήμα SO. Προφανώς, το SL είναι ένα υπό-δέντρο του συνολικού skyline δέντρου και μπορεί να γραφτεί κατευθείαν στο αρχείο T. Ο αλγόριθμος ανιχνεύει κινούμενος προς τα πίσω το μονοπάτι που οδηγεί στο SL και χρησιμοποιεί το SL για να φιλτράρει όλα τα τμήματα που είναι γείτονες (sibling) οποιουδήποτε κόμβου σ'αυτό το μονοπάτι. Η κύρια διαφορά από την OSPSOnPartitioningFirst είναι αυτή η στρατηγική φιλτραρίσματος. Η OSPSOnPartitioningFirst φιλτράρει μόνο τα κυρίαρχα τμήματα ανάμεσα στους γειτονικούς κόμβους του επεξεργαζόμενου τμήματος SO. Εδώ, το SL φιλτράρει πρώτα όλα τα κυριαρχούντα γειτονικά τμήματα. Στη συνέχεια το αντικείμενο του γονιού-κόμβου και το αντικείμενο αναφοράς που χωρίζει το τμήμα –γονιό τοποθετούνται στο SL. Αυτό το SL δέντρο που μεγαλώνει δυναμικά φιλτράρει όλα τα τμήματα, που κυριαρχούνται στο επίπεδο-γονιό και ελέγχει προοδευτικά όλα τα τμήματα που κυριαρχούνται από κόμβους στο μονοπάτι που ενώνει τη ρίζα με το SL. Στη συνέχεια το SL απομακρύνεται από τη μνήμη για να κάνει χώρο για το επόμενο τμήμα αφού τα skyline αντικείμενα στο SL έχουν ήδη χρησιμοποιηθεί για φιλτράρουν κάθε πιθανό αντικείμενο στο O που μπορούν να κυριαρχήσουν. Έχοντας τελειώσει με το SL, ο αλγόριθμος θα συνεχίσει να επεξεργάζεται αναδρομικά τον επόμενο γείτονα (sibling partition) του τρέχοντος τμήματος (partition) ή θα πηγαίνει προς τα πίσω στον γείτονα του γονιού (parent) του. Όταν όλα τα τμήματα έχουν επεξεργαστεί το τελικό skyline δέντρο θα έχει συλλεχθεί σ'ένα skyline αρχείο T.

Algorithm 5: OSPSOnOverflowingMemory(SO)

```
Input: SO:LCRSPTree
Output: T:file where skyline is written
begin
  if SO=null then return
  if |SO.O|≤βthen
    Cur:=SO, Parent:=SO.Parent, SO:=SO.sb
     Cur.sb:=null
     SL:=OSPSOnSortingFirst(Cur)
     output SL to file
     FilterDominatedPartitions(SO,SL)
     while Parent do
      SL:=<Parent.o,Parent.pa,SL,null>
      FilterDominatedPartitions(Parent.sb,SL)
      Parent:=Parent.parent
    remove SL from memory
  else
    r:=a skyline object in SO.O
    partition SO w.r.t r, prune all r's dominating objects in SO and
    output r to temporary file T
    OSPSOnOverflowingMemory(SO.child)
   OSPSOnOverflowingMemory(SO)
end
```

5 *Αλγόριθμος SALSA*

5.1 Sort and Limit Skyline Αλγόριθμος(SALSA)

Ο αλγόριθμος SaLsa, που υλοποιήθηκε στην παρούσα διπλωματική, δημιουργήθηκε από τους Bartolini κ.α [BCP08], με στόχο τον περιορισμό των σημείων που πρέπει να διαβαστούν για την εύρεση του skyline. Ο SaLsa, λοιπόν,διαφέρει από τους άλλους ακολουθιακούς (generic) αλγορίθμους για την εύρεση του skyline στο ότι μειώνει-περιορίζει συνεχόμενα τον αριθμό των σημείων πάνω στα οποία γίνονται οι έλεγχοι κυριαρχίας (dominance tests). Ο σχεδιασμός του SaLsa βασίζεται σε 2 βασικές ιδέες: πρώτον, ένα βήμα-στάδιο ταξινόμησης των δεδομένων εισόδου και δεύτερον, την παρατήρηση ότι με την κατάλληλη επιλογή μιας συνάρτησης ταξινόμησης είναι πράγματι εφικτό να υπολογίσουμε το skyline ελέγχοντας μόνο ένα μικρό κομμάτι της ταξινομημένης εισόδου.

Η ταξινόμηση των δεδομένων στον SaLsa χρησιμοποιείται σαν μέσο για να σταματήσουμε να «φέρνουμε» δεδομένα από την είσοδο. Δηλαδή, ο SaLsa βασίζεται στις συναρτήσεις ταξινόμησης που μπορούν να εγγυηθούν ότι τα σημεία μετά από ένα συγκεκριμένο στάδιο στην είσοδο κυριαρχούνται από κάποιο σημείο που έχει ήδη διαβαστεί, το οποίο και ονομάζουμε σημείο τερματισμού (stop point).

Αυτή η δυνατότητα του SaLsa τον κάνει ιδανικό υποψήφιο για την εύρεση του skyline όταν τα δεδομένα διαχειρίζονται από ένα σύστημα που δεν υποστηρίζει skyline queries και άρα το κόστος επικοινωνίας μπορεί να γίνει ένας βασικός παράγοντας κόστους. Ο SaLsa ερευνά την ικανότητα του server στο να ταξινομήσει πλειάδες και στη συνέχεια υπολογίζει σωστά το skyline διαβάζοντας μόνο ένα κομμάτι των δεδομένων εισόδου.

Αφού ο SaLsa βασίζεται όπως και ο SFS στην ιδέα της ταξινόμησης των δεδομένων προτού τα επεξεργαστούμε, διατηρεί και μερικά από τα πλεονεκτήματα του SFS, όπως την απλοποιημένη διαχείριση του παραθύρου, την αυξανόμενη παράδοση αποτελεσμάτων και του βέλτιστου αριθμού εκτέλεσης της φάσης φιλτραρίσματος.

Ο SaLsa παίρνει σαν είσοδο μια ακολουθία δεδομένων, η οποία έχει ταξινομηθεί από μια μονότονη συνάρτηση, δηλαδή μια συνάρτηση για την οποία αν ισχύει ότι M(p)≤M(p_i) υπονοείται ότι $p_i \prec p$. Η επιλογή της M δεν επηρεάζει την ορθότητα του αλγορίθμου ,αλλά μόνο την απόδοση του .

Ο αλγόριθμος επεξηγείται αναλυτικά παρακάτω. Ως u δηλώνουμε το κομμάτι του r που σε οποιαδήποτε δεδομένη στιγμή δεν έχει επεξεργαστεί ακόμα από τον SaLsa. Κάθε φορά που ένα καινούριο σημείο p διαβάζεται από το u, το p συγκρίνεται με το τρέχον skyline S. Αυτό μπορεί να πυροδοτήσει την ανανέωση του stop point p_{stop}. Στο βήμα 5, ο SaLsa ελέγχει αν υπάρχουν αρκετά δεδομένα-ενδείξεις ώστε να συμπεράνουμε ότι κανένα σημείο στο u δεν μπορεί να είναι μέρος του skyline, που σημαίνει ότι

όλα τα σημεία στο u κυριαρχούνται από το p_{stop} ($p_{stop} \succ u$). Οπότε ο αλγόριθμος μπορεί να τερματίσει σωστά επιστρέφοντας το S σαν αποτέλεσμα.

Algorithm 1. SaLsa[M]

Input: Input stream r sorted using a monotone function M

Output: the skyline S(r) of r

```
1. S ← ∅, stop ← false, p_stop ← undefined ,u ← r
2. while not stop ∧ u ≠ ∅ do
3.     p ← get next point from u, u ← u\{p}
4.     if S ≻ p then S ← S ∪ {p}, update p_stop
5.     if p_stop ≻ u then stop ← true
6. return S
```

Δύο παράγοντες είναι αυτοί που καθορίζουν εν τέλει την απόδοση του SaLsa: η επιλογή της συνάρτησης ταξινόμησης και η στρατηγική για την επιλογή του stop point.

5.1.1 Επιλογή του σημείου τερματισμού

Έστω ότι κατά την εκτέλεση του SaLsa το τελευταίο σημείο p που διαβάζεται έχει τιμή M(p)=l . Λέμε ότι η M είναι στο επίπεδο l αφού διαβάσουμε το P και δηλώνουμε ως u(M,l)το σύνολο των μη αναγνωσμένων σημείων σ'αυτό το στάδιο της εκτέλεσης. Σημειώνουμε ότι M(p_i)>=l ισχύει για κάθε $p_i \in u(M,l)$. Για να σταματήσουμε με ασφάλεια την εκτέλεση ,πρέπει να εγγυηθούμε ότι $p_{stop} \succ p_i$ για κάθε τέτοιο σημείο. Αυτό γίνεται στον SaLsa θεωρώντας το μη αναγνωσμένο πεδίο τιμών, ορισμένο ως εξής: $\mathcal{D}(\mathcal{M},l) = \{\mathbf{p}_i \in \mathcal{D} : \mathcal{M}(\mathbf{p}_i) \geq l\}$.

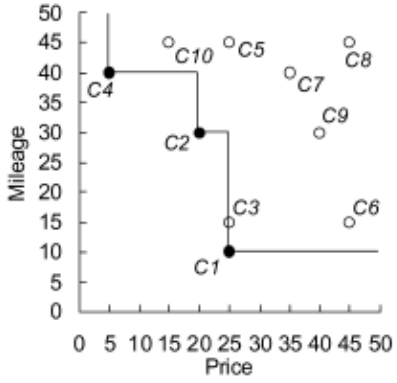
Προφανώς ισχύει ότι $u(M,l) \subseteq D(M,l)$. Σημειώνουμε ότι το D(M,l) σε αντίθεση με το u(M,l) δεν εξαρτάται από μια συγκεκριμένη σχέση εισόδου r. Οπότε ο SaLsa μπορεί να σταματήσει με ασφάλεια όταν και μόνο όταν ισχύει το ακόλουθο: $\forall \mathbf{p}_i \in \mathcal{D}(\mathcal{M},l) : \mathbf{p}_{stop} \succ \mathbf{p}_i$.

Συμβολίζουμε ως $l_{stop}(M)$ το επίπεδο στο οποίο ο SaLsa σταματά κάποια στιγμή, το οποίο ονομάζεται και stop level της M. Ομοίως, $u_{stop}(M)$ και $D_{stop}(M)$ είναι οι τιμές των $u(M,l)$ και $D(M,l)$,αντίστοιχα όπου ο αλγόριθμος σταματά.

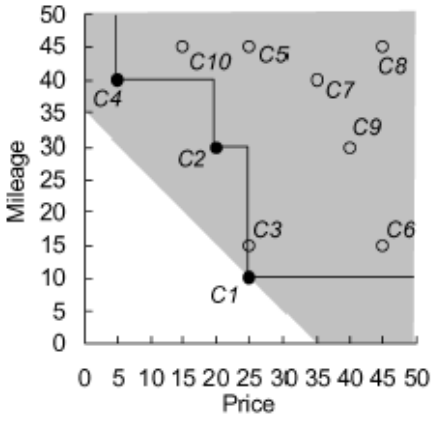
Ας θεωρήσουμε τη σχέση UsedCars(CarID,Price,Mileage,..), ένα στιγμιότυπο της οποίας φαίνεται στην παρακάτω εικόνα.

CarID	Price ($\times 10^3$)	Mileage ($\times 10^3$)
C1	25	10
C2	20	30
C3	25	15
C4	5	40
C5	25	45
C6	45	15
C7	35	40
C8	45	45
C9	40	30
C10	15	45

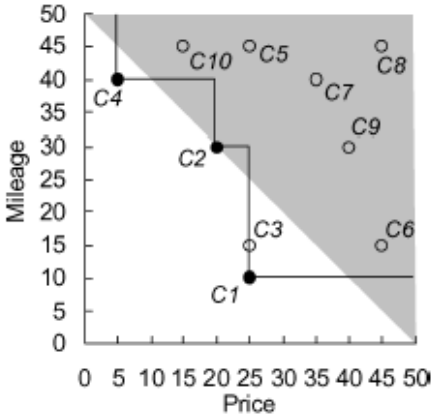
Ένα ερώτημα skyline στα γνωρίσματα Τιμή (Price) και Χιλιόμετρα (Mileage) θα επιστρέψει τα αυτοκίνητα C1,C2 και C4, όπως φαίνεται παρακάτω.



Ας υποθέσουμε τώρα ότι τα σημεία ταξινομούνται με βάση το άθροισμα των τιμών του Price και Mileage, M(p)=p.Price + p.Mileage. Το πρώτο σημείο που θα διαβαστεί είναι το αυτοκίνητο C1, που βρίσκεται στο επίπεδο l=25+10=35. Σ ’αυτό το στάδιο η εκτέλεση του πεδίου που δεν έχει διαβαστεί ακόμα αντιστοιχεί στο σκιασμένο τμήμα του παρακάτω σχήματος.



Αφού διαβαστεί το τελευταίο σημείο του skyline, C2, του οποίου το επίπεδο (βαθμίδα) είναι l=20+30=50, το πεδίο τιμών που δε θα έχει διαβαστεί θα είναι το παρακάτω σκιασμένο κομμάτι.



Παρόλο που ο SaLsa μπορεί να εφαρμοστεί με οποιαδήποτε μονότονη συνάρτηση, μόνο συναρτήσεις που ταξινομούν και περιορίζουν μας ενδιαφέρουν. Λέμε ότι η συνάρτηση M περιορίζει μια σχέση r όταν $u_{stop}(M) \neq \emptyset$ και ότι η M είναι περιοριστική όταν υπάρχει τουλάχιστον μια σχέση r για την οποία το $u_{stop}(M)$ δεν είναι κενό, το οποίο ισοδυναμεί με το να λέγαμε ότι το $D_{stop}(M)$ δεν είναι πάντα κενό.

Μια συνάρτηση M σε d μεταβλητές ονομάζεται συμμετρική αν και μόνο αν $M(x_1, x_2, ..., x_d) = M(x_{\pi(1)}, ..., x_{\pi(d)})$ για κάθε υπόθεση π των $\{1, ..., d\}$, που σημαίνει ότι η τιμή της M δεν μεταβάλλεται μετά από οποιαδήποτε επανατοποθέτηση των μεταβλητών της.

Μια συμμετρική συνάρτηση M δεν πριμοδοτεί καμία ιδιότητα έναντι των άλλων. Αυτό φαίνεται να είναι μια πολύ φυσική προϋπόθεση αν η M χρησιμοποιείται για τον υπολογισμό του skyline, μιας και όλες οι ιδιότητες του skyline είναι εξίσου σημαντικές.

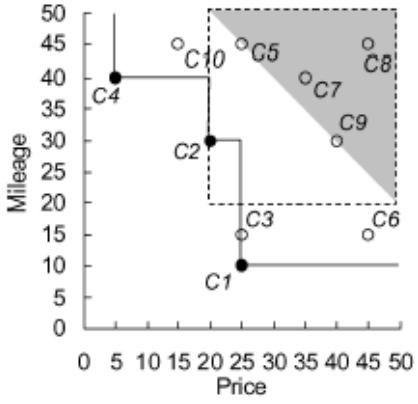
Υποθέτουμε ότι όλες οι τιμές των ιδιοτήτων έχουν κανονικοποιηθεί και άρα για όλα τα j ισχύει: $dom(A_j) = [0,1]$.

Έστω ότι η M είναι μια συμμετρική μονότονη συνάρτηση και ότι l μια οποιαδήποτε τιμή της M. Ορίζουμε ως Low την ελάχιστη τιμή για την οποία ισχύει ότι $M(Low,1,1,...,1) \geq l$.

Αν δεν υπάρχει τέτοιο Low θέτουμε το Low=0. Τότε δεν υπάρχει τέτοιο σημείο p για το οποίο να ισχύουν ταυτόχρονα και τα δύο: i) $M(p) \geq l$, και ii) υπάρχει j τέτοιο ώστε $p[j] < Low$.

Άρα για όλες τις συμμετρικές μονότονες συναρτήσεις $D(M,l)$ περιλαμβάνεται στον υπέρ-κύβο του οποίου οι απέναντι κορυφές είναι τα σημεία $(Low,....,Low)$ και $1 = (1,...,1)$.

Στο παραπάνω παράδειγμα της σχέσης UsedCars, ας υποθέσουμε ότι τα γνωρίσματα Price και Mileage έχουν τιμές στο πεδίο [0,50] και ότι η συμμετρική συνάρτηση που χρησιμοποιείται για να ταξινομηθεί η UsedCars σχέση είναι πάλι το άθροισμα των τιμών του Priceκαι Mileage, M(p)=p.Price + p.Mileage. Αφού διαβαστεί το σημείο C5, το επίπεδο της συνάρτησης αυτής θα είναι l=25+45=70. Η ελάχιστη τιμή του Low ώστε $M(Low,50) \geq 70$ είναι Low=20. Οπότε, όλα τα σημεία που θα διαβάσει ο SaLsa μετά το C5 βρίσκονται στον υπέρ-κύβο, του οποίου οι απέναντι κορυφές είναι τα σημεία (20,20) και (50,50).

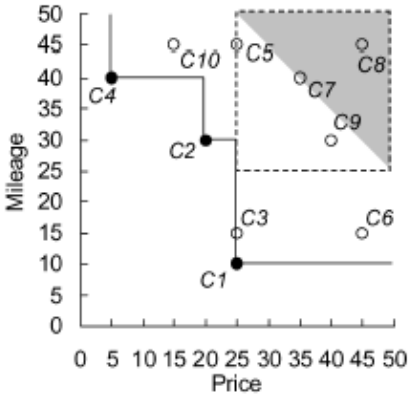


Έστω M μια συμμετρική μονότονη συνάρτηση και ας υποθέσουμε ότι η M φτάνει στο επίπεδο l σε κάποιο σημείο της εκτέλεσης. Τότε, ο SaLsa μπορεί να σταματήσει αν και μόνο αν η συντεταγμένη με τη μέγιστη τιμή του stoppoint p_{stop} ικανοποιεί τη σχέση $p^+_{stop} \leq Low$. Μόνο αν $p_{stop[j]} = p^+_{stop}$ ισχύει για όλα τα j, τότε ο SaLsa πρέπει να διαβάσει όλα (αν υπάρχουν) τα διπλότυπα του p_{stop}.

Δεδομένης μιας συνάρτησης M, ο κανόνας για την επιλογή του stop point είναι ο βέλτιστος αν και μόνο αν για κάθε σχέση r, δεν υπάρχει άλλος κανόνας που να επιτρέπει στον SaLsa να σταματήσει νωρίτερα όταν τα σημεία έχουν ταξινομηθεί χρησιμοποιώντας την M.

Ο ακόλουθος MinMax κανόνας για την επιλογή του stop point είναι βέλτιστος για κάθε συμμετρική μονότονη συνάρτηση M: $p_{stop} = \arg\min_{p_i \in S}\{p_i^+\}$.

Επιστρέφοντας στο παράδειγμα της UsedCars σχέσης, ο κανόνας MinMax μας οδηγεί στο να επιλέξουμε το σημείο C_1, για το οποίο ισχύει ότι C_1^+=25, για stop point. Επιλέγοντας για την ταξινόμηση την ίδια πάλι συνάρτηση, ο SaLsa μπορεί να τερματίσει μόλις διαβάσει το σημείο C7, του οποίου το επίπεδο είναι 75.



Η ελάχιστη τιμή του Low που ικανοποιεί την ισότητα M(Low,50)≥75 είναι η 25≥ C_1^+,που είναι και η συνθήκη τερματισμού με βάση όλα τα παραπάνω.

Αν p_{stop} είναι το τρέχον stop point , όταν ένα νέο σημείο p προστίθεται στο skyline τότε το stop point είτε παραμένει το ίδιο είτε γίνεται ίσο με το P.Αυτό εξαρτάται από το ποιο από τα p^+ και p^+_{stop} είναι μικρότερο.

5.1.2 Επιλογή της συνάρτησης ταξινόμησης

Παρόλο που όπως αναφέρθηκε παραπάνω, μπορούμε να επιλέξουμε το stop point ανεξάρτητα από τη συνάρτηση ταξινόμησης, αυτό δε σημαίνει ότι όλες οι συναρτήσεις συμπεριφέρονται εξίσου καλά. Παρακάτω, παρουσιάζονται, λοιπόν, οι ιδιότητες τριών διαφορετικών μονότονων συναρτήσεων ταξινόμησης, που μπορούν να χρησιμοποιηθούν στον αλγόριθμο SALSA.

- **ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΒΑΣΗ ΤΟΝ ΟΓΚΟ**

Η 1^{η} συνάρτηση που μελετάμε για την ταξινόμηση βασίζεται στον όγκο των σημείων, vol[0](p)= 1-$\prod_{j=1}^{d}(1-p[j])$.

Ο λόγος που χρησιμοποιούμε τον συμβολισμό vol[0] είναι ότι $\prod_{j=1}^{d}(1-p[j])$ είναι ο όγκος της περιοχής κυριαρχίας του p, δηλαδή το σύνολο των σημείων στο $[0,1]^d$ κυριαρχείται από το p. Αν τα σημεία είναι ομοιόμορφα κατανεμημένα στο D, τότε αυτό αναφέρεται επίσης και στο αναμενόμενο κομμάτι των σημείων στην r τα οποία κυριαρχεί το p. Τότε διαβάζοντας πρώτα τα σημεία με μεγαλύτερο όγκο αυξάνεται η πιθανότητα να απομακρυνθούν από νωρίς πολλά σημεία και άρα να μειωθεί ο συνολικός αριθμός των συγκρίσεων. Δυστυχώς παρατηρούμε ότι η vol[0] δεν είναι καλή επιλογή για να περιορίσουμε τον αριθμό των σημείων που θα διαβαστούν. Είναι πιθανό όμως να τροποποιήσουμε τη συνάρτηση όγκου αποφεύγοντας παράγοντες του γινομένου που εξαφανίζονται όταν p[j]=1. Γι' αυτό και θεωρούμε την συνάρτηση $vol[1](p)=2^d-\prod_{i=1}^{d}(2-p[j])$, που έχει πεδίο τιμών το $[0,2^d-1]$.

Θεωρώντας τη συνάρτηση : $vol[m](p)=(m+1)^d-\prod_{j=1}^{d}(m+1-p[j])$,(m>0), της οποίας το πεδίο τιμών είναι$[o,(m+1)^d-m^d]$.Τότε $l_{stop}(vol[m])>(m+1)^d-(m+1)m^{d-1}$για κάθε σχέση r. Από γεωμετρική άποψη κάθε κορυφή v του υπέρ-κύβου $[0,1]^d$ με συντεταγμένες$(1,...,1,0,1,...,1)$ για το οποίο είναι $vol[m](v)=(m+1)^d-(m+1)m^{d-1}$ πρέπει να αποκλειστεί από το $D(vol[m],l)$.

- **ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΒΑΣΗ ΤΟ ΑΘΡΟΙΣΜΑ ΤΩΝ ΣΥΝΤΕΤΑΓΜΕΝΩΝ**

Μια εναλλακτική της vol είναι το άθροισμα των τιμών των γνωρισμάτων του κάθε σημείου: sum(p)= $\sum_{j=1}^{d}p[j]$.

Η sum συνάρτηση ταξινόμησης είναι αυστηρά πιο περιοριστική από την vol[m], για κάθε ορισμένο m, με $l_{stop}(sum)>d-1$.

- **ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΒΑΣΗ ΤΗΝ ΜΙΚΡΟΤΕΡΗ ΣΥΝΤΕΤΑΓΜΕΝΗ**

Η minC ορίζεται ως εξής: $\min C(p)=(\min_j\{p[j]\},sum(p))$.

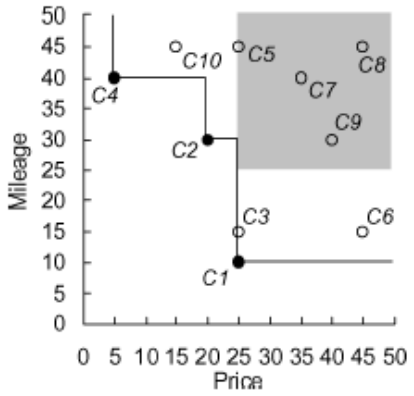
Η minC πρώτα ταξινομεί τα σημεία λαμβάνοντας υπόψη τη συντεταγμένη με τη μικρότερη τιμή. Στη συνέχεια, χρησιμοποιείται ένα άθροισμα των ιδιοτήτων του skyline για να εξασφαλιστεί μονοτονία σε περίπτωση που δεν μπορεί να βγει συμπέρασμα από τη μικρότερη συντεταγμένη.

Άρα, η minC είναι περιοριστική αφού ο SaLsa μπορεί να σταματήσει μόλις διαβαστεί ένα σημείο p για το οποίο είναι $\min_j\{p[j]\} \geq p^+_{stop}$ ι .

Διαισθητικά η βέλτιστη συμπεριφορά της minC απορρέει από την παρατήρηση ότι αν θεωρήσουμε το σημείο p_{diag}, το οποίο βρίσκεται στην κύρια διαγώνιο του χώρου δεδομένων και του οποίου οι συντεταγμένες είναι ίσες με p^+_{stop}, δηλαδή $p_{diag} = (p^+_{stop}, ..., p^+_{stop})$, άρα ισχύει ότι $p_{diag} \succ p$,όταν $\min C(p) > \min C(p_{diag})$ το οποίο και δεν ισχύει για καμία άλλη συμμετρική μονότονη συνάρτηση.

Ο αλγόριθμος 2, παρακάτω, περιγράφει τον SaLsa όταν χρησιμοποιείται η minC συνάρτηση ταξινόμησης. Λαμβάνοντας υπόψη το γενικό πρότυπο του SaLsa (αλγόριθμος 1), εδώ εξετάζουμε αμέσως αν ισχύει η συνθήκη τερματισμού πριν συγκρίνουμε το καινούριο σημείο p που θα διαβαστεί με τα ηδη υπάρχοντα στο skyline S. Σημειώνουμε ότι το βήμα 4 λαμβάνει σωστά υπόψη του την ειδική περίπτωση που το p_{stop} βρίσκεται πάνω στην κύρια διαγώνιο του χώρου των δεδομένων, αλλά και την περίπτωση διπλότυπων.

Επιστρέφοντας στο προηγούμενο παράδειγμα της UsedCars σχέσης, για την οποία το stop point είναι το C1, βλέπουμε ότι όταν η minC εφαρμοστεί στα δεδομένα, ο SALSA μπορεί να τερματίσει αφού διαβάσει το σημείο C5, του οποίου το επίπεδο είναι l=25≥ C_1^+. Το σκιασμένο κομμάτι της παρακάρω εικόνας είναι το κομμάτι του χώρου που δε θα διαβαστεί.



Algorithm 2. SaLsa[minC]

Input: input stream r sorted using the minC function

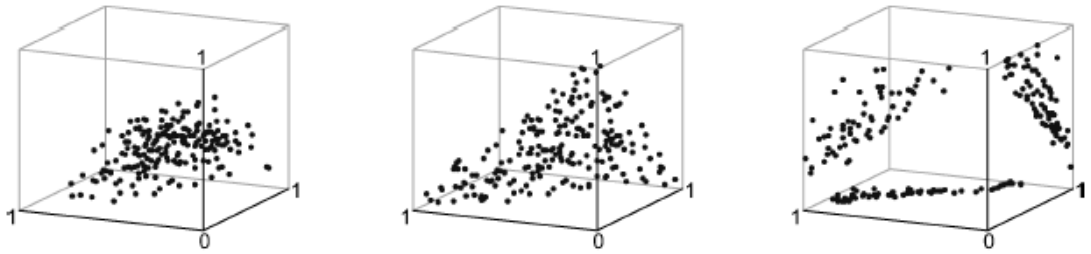
Output: the skyline S(r) of r

```
1.   S ← ∅, stop ← false, p_stop ← undefined , u ← r
2.   while not stop ∧ u ≠ ∅ do
3.       p ← get next point from u, u ← u\{p}
4.       if p+_stop ≤ minC(p) and p_stop ≠ p then stop ← true
5.       else if S ⊁ p then
6.       S ← S ∪ {p},
7.       if p+ < p+_stop then p_stop ← p
8.       return S
```

Ο αλγόριθμος SaLsa μπορεί να επεκταθεί με τη χρήση ασύμμετρων συναρτήσεων, κάτι το οποίο βελτιώνει αρκετά την απόδοση του.

5.2 Ανάλυση του SALSA

Γενικά, είναι γνωστό σύμφωνα με προηγούμενη εργασία του Godfrey [G04], οτι ακόμα και για την απλούστερη περίπτωση όπου τα γνωρίσματα των δεδομένων είναι ανεξάρτητα (independent) δεν υπάρχει γνωστό αναλυτικό αποτέλεσμα για την κατανομή του skyline σε δεδομένα με παραπάνω από δύο διαστάσεις. Παρόλο που μπορεί να θεωρήσουμε ότι η συνάρτηση ταξινόμησης minC είναι καλυτερη από τις υπόλοιπες στο να περιορίζει την πολυπλοκότητα της διαδικασίας του φιλτραρίσματος κατά την εκτέλεση του αλγόριθμου, κάτι τέτοιο δεν ισχύει. Διαισθητικά, συναρτήσεις όπως οι vol[m] και sum είναι πιθανό να έχουν την ιδιότητα που έχει και η vol[0], δηλαδή να επεξεργάζεται πρώτα σημεία που έχουν μεγάλη πιθανότητα να διαγράψουν μεγάλο αριθμό σημείων. Παρολαυτά δεν ισχύει το ίδιο με την minC, καθώς είναι πιθανό η συνάρτηση αυτή να φέρει πρώτα στη μνήμη σημεία που έχουν κακή συμπεριφορά σε όλες τις διαστάσεις εκτός από μία. Κάτι τέτοιο φαίνεται και στην παρακάτω είκονα, που μας δείχνει την κατανομή των πρώτων 200 skyline σημείων που επιστρέφουν οι vol[1], sum και minC αντίστοιχα, σε ένα τρισδιάστατο σύνολο δεδομένων.



(a) vol[1] (b) sum (c) minC

Φαίνεται, λοιπόν οτι η minC πρώτα βρίσκει εκείνα τα σημεία που είναι κοντά στα σύνορα του χώρου και είναι λιγότερο πιθανά να διαγράψουν/αποκλείσουν άλλα σημεία.

Μια άλλη σημαντική παρατήρηση είναι οτι ο αριθμός των ελέγχων κυριαρχίας εξαρτάται και από τη σειρά που τα σημεία στο skyline S συγκρίνονται με το καινούριο σημείο που διαβάζεται κάθε φορά. Για το πρόβλημα αυτό, που εξετάστηκε για πρώτη φορά από τους Borzsoynyi κ.α. [BKS01] προτείνεται η παρακάτω λύση:

Θεώρημα: Ας υποθέσουμε σημεία που έχουν ταξινομηθεί με την minC συνάρτηση. Αν τα γνωρίσματα του κάθε σημείου έχουν κατανεμηθεί ανεξάρτητα και με πανομοιότυπο τρόπο, τότε ο αναμενόμενος αριθμός ελέγχων κυριαρχίας που εκτελούνται διατρέχοντας το τρέχον skyline κατά την ανάποδη σειρά είναι μικρότερος από αυτόν που χρειάζεται όταν τα σημεία διατρέχονται κατά την κανονική σειρά.

6 *Αξιολόγηση*

Σ’ αυτό το κεφάλαιο θα μελετηθεί η απόδοση πέντε διαφορετικών αλγορίθμων για την εύρεση της κορυφογραμμής σε πολυδιάστατα σύνολα δεδομένων. Όλοι οι αλγόριθμοι του παρακάτω πίνακα έχουν υλοποιηθεί σε C++ και όλα τα πειράματα έγιναν σε Intel Core i5-2500 3.60 Hz cpu και 8 GB RAM, σε περιβάλλον Microsoft Windows 7.

Αλγόριθμος	Περιγραφή
BNL	Ο βασικός Block-Nested Loops αλγόριθμος, που λαμβάνει υπόψη του την περίπτωση περιορισμένης μνήμης.
LESS	Ο εξωτερικός (external) Linear Elimination Sort for Skyline αλγόριθμος με μέγεθος EF παραθύρου 1 και χρήση της εξωτερικής συνάρτησης ταξινόμησης Entropy.
SALSA	Ο εξωτερικός Sort and Limit Skyline αλγόριθμος με χρήση της συνάρτησης ταξινόμησης minC.
OSP	Ο Object-Space Partitioning αλγόριθμος για περίπτωση περιορισμένης μνήμης.
SFS	Ο εξωτερικός Sort First Skyline αλγόριθμος με χρήση της συνάρτησης ταξινόμησης Entropy.

6.1 Παράμετροι αξιολόγησης

Επειδή οι βάσεις δεδομένων που μελετούνται είναι συνήθως πολύ μεγάλες για να αποθηκευτούν στην κύρια μνήμη, οι αλγόριθμοι που αναπτύχθηκαν είναι εξωτερικοί (external) υπό την έννοια ότι τα δεδομένα βρίσκονται στην εξωτερική μνήμη (δηλαδή στο δίσκο), ενώ η επεξεργασία τους πραγματοποιείται στην κύρια μνήμη. Γι' αυτόν το λόγο η πρόσβαση στο δίσκο άλλοτε γίνεται σειριακά και άλλοτε τυχαία. Η ακολουθιακή (sequential) πρόσβαση στο δίσκο είναι προτιμότερη από την τυχαία πρόσβαση γιατί είναι σημαντικά ταχύτερη από την τυχαία, καθώς η τελευταία περιλαμβάνει μια σειρά δράσεων αναζήτησης. Επίσης, η διαδοχική πρόσβαση, έχει το πλεονέκτημα ότι με τη χρήση σύγχρονων αρχιτεκτονικών αποθήκευσης, κάνει τον αλγόριθμο ανεξάρτητο από το μέγεθος του μπλοκ.

Οι παράμετροι, οι οποίες αξιολογήθηκαν είναι (1) ο συνολικός αριθμός των συγκρίσεων-ελέγχων κυριαρχίας μεταξύ των σημείων (comparisons), (2) ο συνολικός χρόνος εκτέλεσης του αλγορίθμου (Time) και (3) ο συνολικός αριθμός πρόσβασης στο δίσκο- τα ακολουθιακά (sequential) και τυχαία (random) Inputs-Outputs (I/O).

Με τις πειραματικές μετρήσεις θέλουμε να ερευνήσουμε ποιοι από τους αλγόριθμους κάνουν καλύτερη χρήση της γεωμετρικής ερμηνείας του skyline, είτε αποφεύγοντας άσκοπες συγκρίσεις μεταξύ σημείων, είτε περιορίζοντας τις συγκρίσεις σε μικρά υποσύνολα των σημείων.

6.2 Οργάνωση πειραμάτων

Οι βάσεις δεδομένων που χρησιμοποιήθηκαν στα πειράματα περιλαμβάνουν 1.000, 10.000, 100.000 ή 1.000.000 πλειάδες. Κάθε πλειάδα έχει d γνωρίσματα, των οποίων οι τιμές παράγονται τυχαία μέσα στο

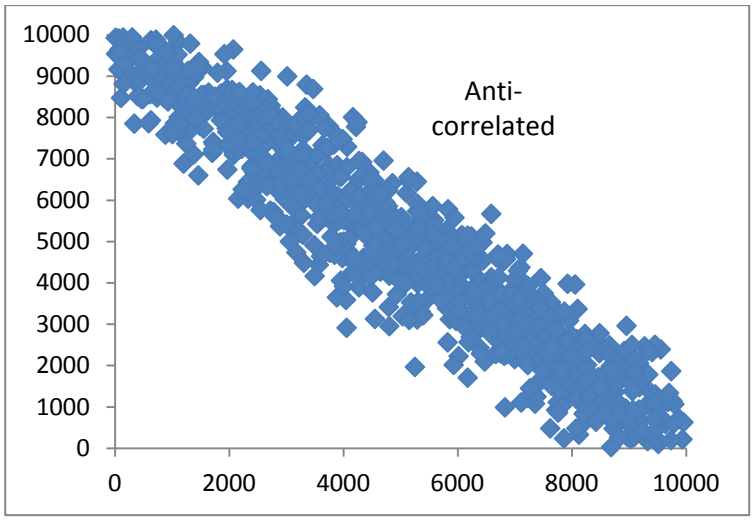
εύρος [0,10000). Όλες οι τιμές των γνωρισμάτων είναι τύπου double (8 bytes). Μελετήθηκαν τρία διαφορετικά συνθετικά είδη βάσεων δεδομένων, που διαφέρουν στον τρόπο με τον οποίο παράγονται οι τιμές των πλειάδων, και παρουσιάζονται παρακάτω.

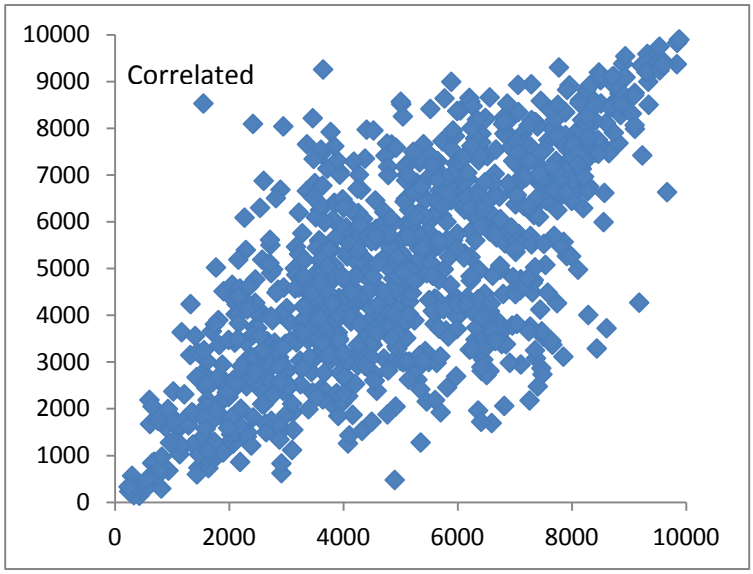
Independent: Γι' αυτό το είδος της βάσης δεδομένων όλες οι τιμές των γνωρισμάτων παράγονται ανεξάρτητα χρησιμοποιώντας μια ομοιόμορφη κατανομή.

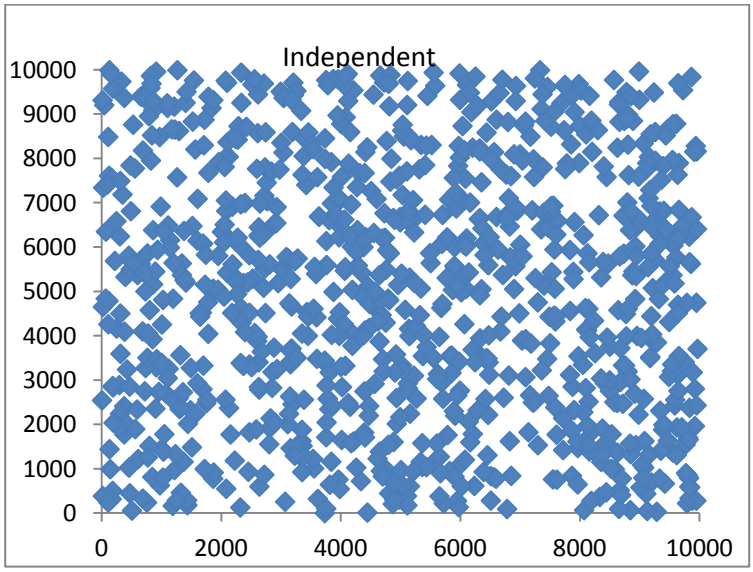
Correlated: Μια τέτοια βάση δεδομένων αναπαριστά ένα περιβάλλον στο οποίο τα σημεία που είναι καλά σε μια διάσταση είναι καλά και σε όλες τις άλλες διαστάσεις. Τα σημεία δημιουργούνται ως εξής: Πρώτα επιλέγουμε ένα επίπεδο κάθετο στη γραμμή από το (0,…,0) έως το (10.000,…,10.000) χρησιμοποιώντας μια κανονική κατανομή. Το καινούριο σημείο θα είναι σ' αυτό το επίπεδο. Χρησιμοποιούμε μια κανονική κατανομή για να επιλέξουμε το επίπεδο, ώστε περισσότερα σημεία να είναι στη μέση απ' ότι στις άκρες. Μέσα στο επίπεδο, οι διαφορετικές τιμές των γνωρισμάτων δημιουργούνται ξανά χρησιμοποιώντας μια κανονική κατανομή, έτσι ώστε τα περισσότερα σημεία να βρίσκονται κοντά στη γραμμή από (0,..,0) έως (10.000,…,10.000).

Anti-correlated: Μια τέτοιου είδους βάση δεδομένων αναπαριστά ένα περιβάλλον στο οποίο τα σημεία που είναι καλά σε μια διάσταση, δεν είναι καλά ,σε μία ή περισσότερες από τις άλλες διαστάσεις. Τα σημεία παράγονται ως εξής: Επιλέγουμε ένα επίπεδο κάθετο στη γραμμή από (0,..,0) έως (10.000,…,10.000) χρησιμοποιώντας μια κανονική κατανομή. Χρησιμοποιούμε μια κανονική κατανομή με πολύ μικρή απόκλιση ώστε όλα τα σημεία να τοποθετούνται σε επίπεδα που είναι κοντά στο επίπεδο μέσω του σημείου (5.000,…,5.000). Μέσα στο επίπεδο, οι διαφορετικές τιμές των γνωρισμάτων δημιουργούνται χρησιμοποιώντας μια ομοιόμορφη κατανομή.

Παρακάτω παρουσιάζονται ενδεικτικά οι γραφικές απεικονίσεις για τα παραπάνω σύνολα δεδομένων, για 1000 σημεία στο δισδιάστατο χώρο.


Anti-
correlated
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000


Correlated
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000


Independent
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000

Για τα πειράματα μελετήθηκαν τρεις διαφορετικές περιπτώσεις. Στην πρώτη περίπτωση, διατηρώντας σταθερό τον αριθμό των διαστάσεων, dimensionality=7 και το μέγεθος της μνήμης (memory size) στα 500 σημεία, μεταβάλλουμε τον αριθμό των σημείων/δεδομένων (cardinality) δίνοντας τις τιμές : 1.000, 10.000,100.000 και 1.000.000. Στη δεύτερη περίπτωση, διατηρώντας σταθερό τον αριθμό των διαστάσεων dimensionality=7 και των αριθμό των σημείων cardinality =10.000, μεταβάλλουμε το μέγεθος της μνήμης (memory size) δίνοντας τις τιμές: 100, 500, 1.000, 5.000 και 10.000 σημεία.Στην τρίτη περίπτωση, διατηρώντας σταθερό τον αριθμό των σημείων, cardinality=10.000, και το μέγεθος της κύριας μνήμης, memory size=500 σημεία, μεταβάλλουμε τον αριθμό των διαστάσεων (dimensionality) d= 2, 3, 5,7,10 και 15. Τα παραπάνω παρουσιάζονται αναλυτικά στον πίνακα:

Παράμετροι	**Τιμές**
Cardinality	1.000, **10.000**, 100.000, 1.000.000
Dimensionality	2,3,5,**7**,10,15
Memory size	100,**500**,1.000,5.000,10.000

Το πλήθος του skyline για τα τρία σύνολα δεδομένων, α) anti-correlated (AC), β) independent (I) και γ) correlated (C) παρουσιάζονται στους παρακάτω πίνακες.

Dimensionality	**AC**	**UI**	**C**
2	38	12	1
3	392	48	1
5	3495	446	14
7	6843	1564	27
10	9179	4460	250
15	9919	9197	1060

Cardinality	**AC**	**I**	**C**
1.000	916	390	20
100.000	41951	4738	30
1.000.000	200700	14039	123

6.3 Αποτελέσματα

Παρακάτω παρουσιάζονται οι γραφικές παραστάσεις των πειραματικών μετρήσεων που έγιναν για κάθε σύνολο δεδομένων ξεχωριστά.

6.3.1 Μεταβάλλοντας το πλήθος των σημείων

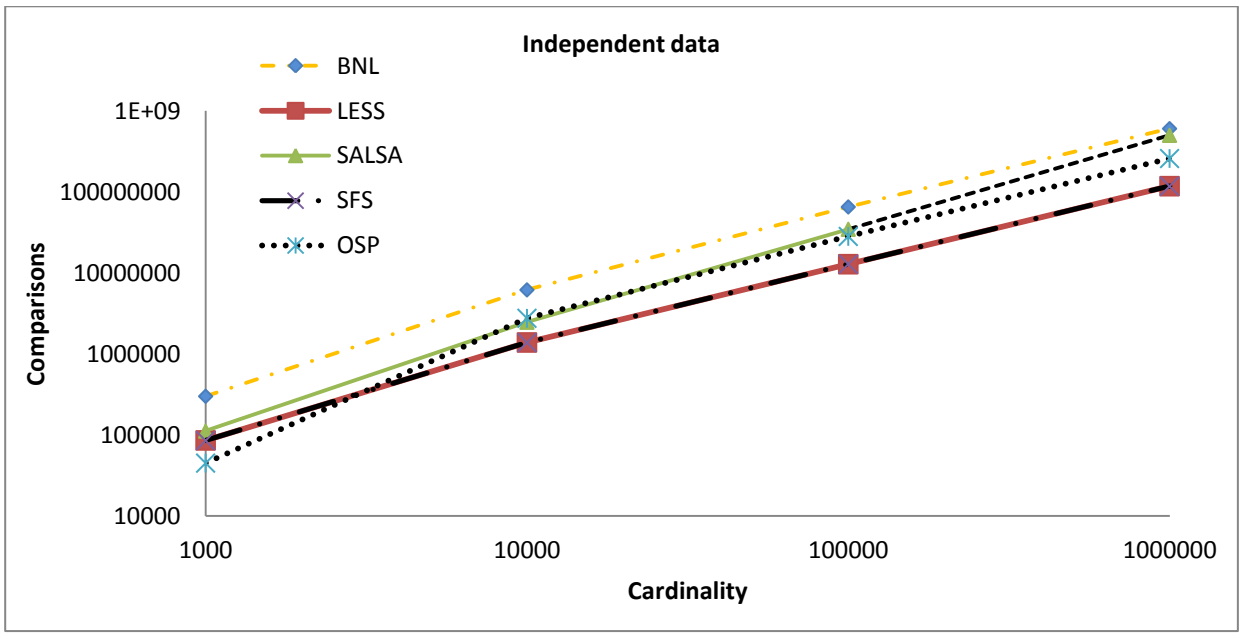
- **Independent δεδομένα**

Σ'αυτήν την περίπτωση, παρατηρούμε ότι καθώς αυξάνεται το πλήθος των σημείων, ο αριθμός των συγκρίσεων που πραγματοποιεί κάθε αλγόριθμος, αυξάνεται και αυτός σημαντικά.

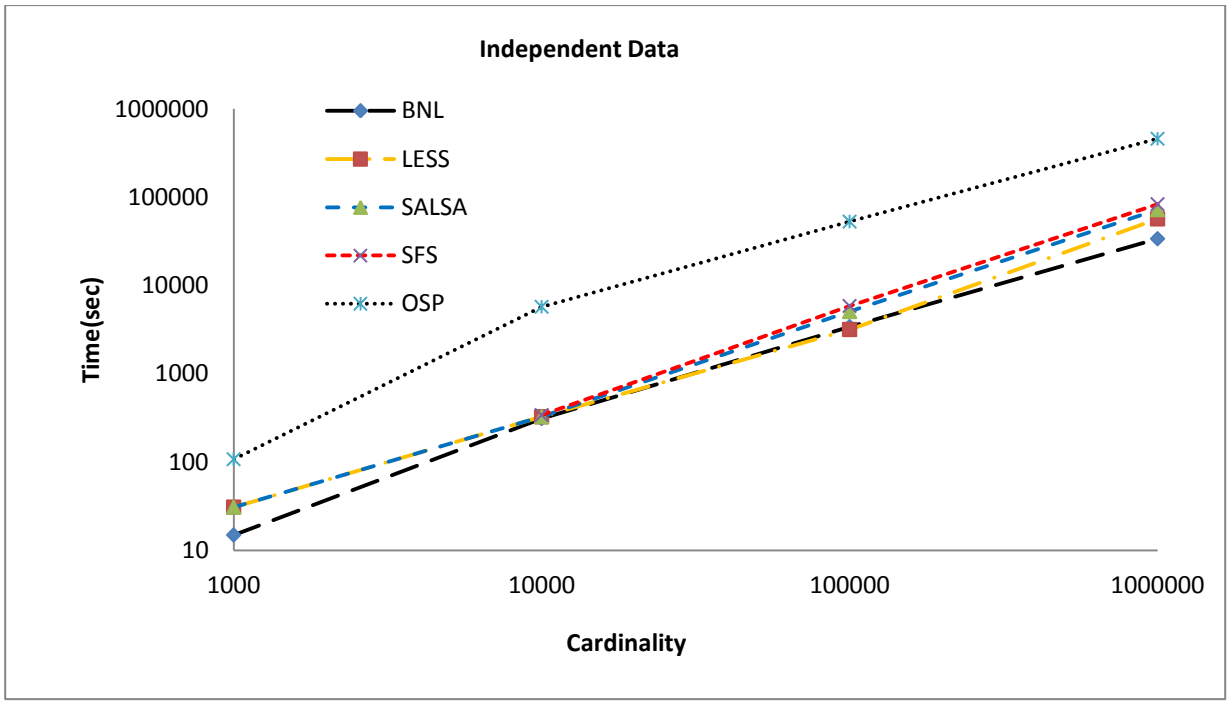
Για τις διαφορετικές τιμές του cardinality, οι αλγόριθμοι που πραγματοποιούν το μικρότερο αριθμό συγκρίσεων είναι ο ο LESS, ο SFS και ο OSP, ενώ ο BNL έχει τη χειρότερη απόδοση καθώς απαιτεί το μεγαλύτερο αριθμό συγκρίσεων ακόμα και για πολύ μικρό πλήθος σημείων.

Ο αλγόριθμος με το μικρότερο χρόνο εκτέλεσης είναι ο BNL, ενώ για cardinality<1.000.000 και ο LESS παρουσιάζει αρκετά χαμηλούς χρόνους εκτέλεσης, παρεμφερείς αυτών του BNL. Αντίθετα ο αλγόριθμος με το μεγαλύτερο κόστος σε χρόνο είναι ο OSP.

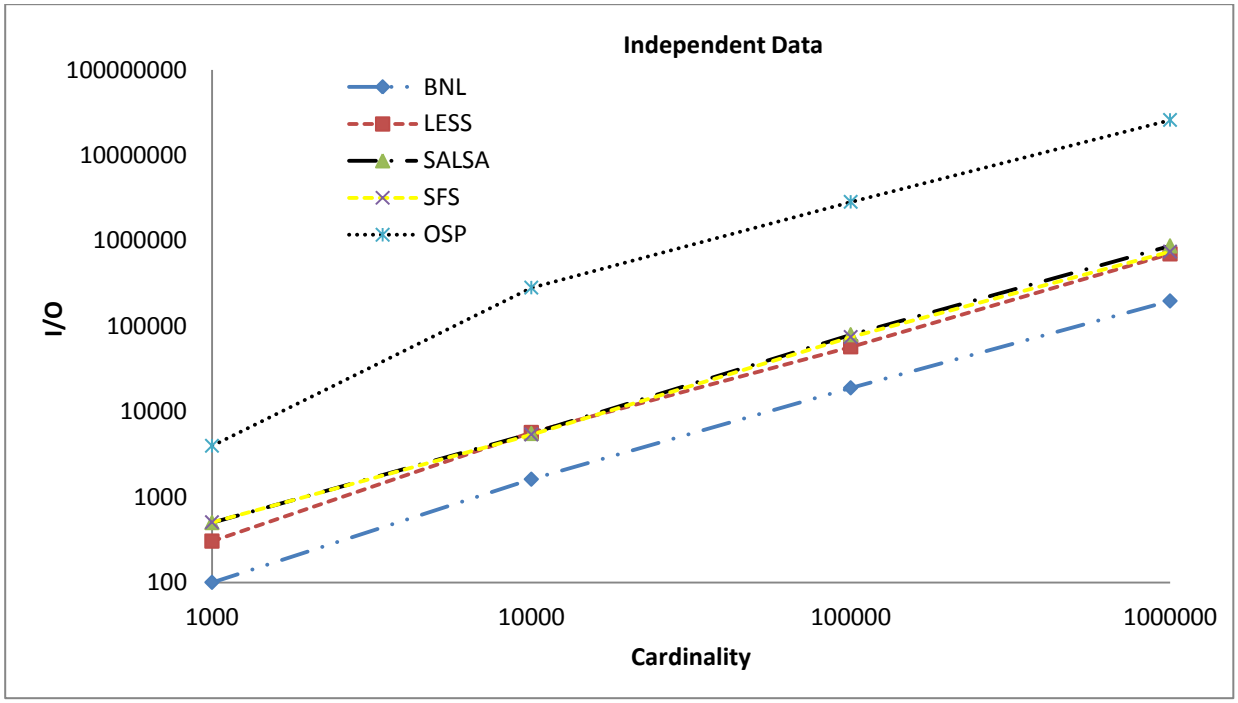
Όσον αφορά τα I/O παρατηρούμε οτι ο BNL χρειάζεται τις λιγοτέρες προσβάσεις I/O, ενώ ο OSP τις περισσότερες. Επίσης είναι αξιοσημείωτο οτι ο LESS, SFS και SALSA δείχνουν να αποδίδουν το ίδιο, καθώς δεν παρουσιάζουν σημαντικές διακυμάνσεις μεταξύ τους.



Εικόνα 1:Μέτρηση αριθμού συγκρίσεων μεταβάλλοντας το πλήθος των σημείων (Independent)



Εικόνα 2:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το πλήθος των σημείων (Independent)



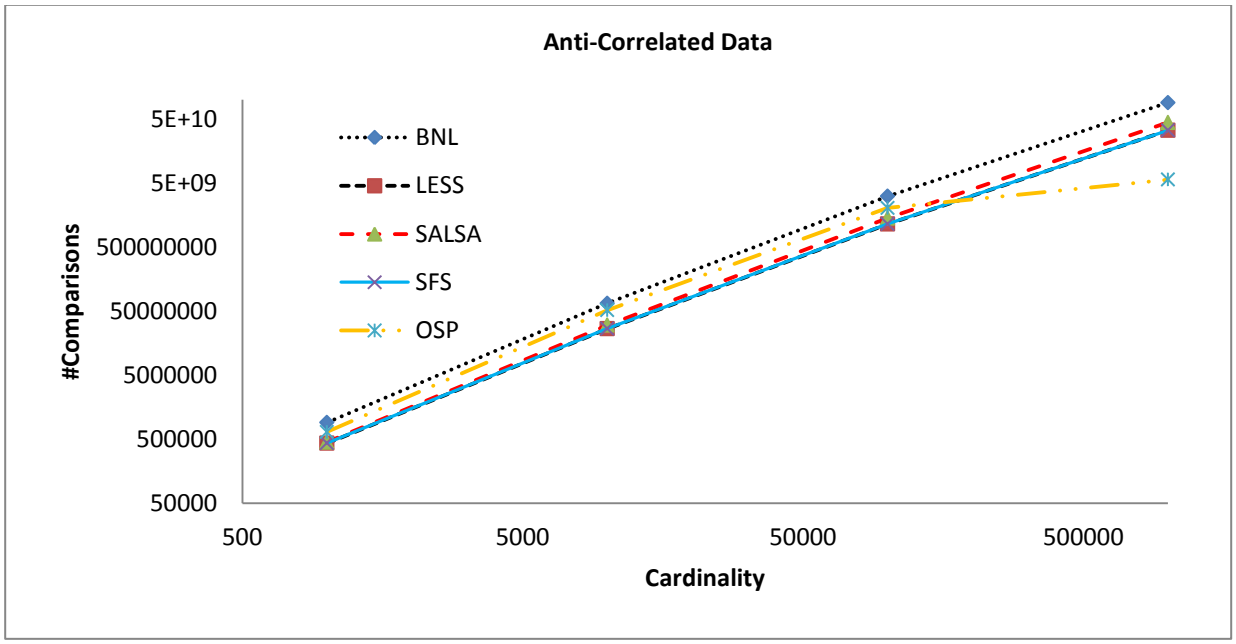
Εικόνα 3:Μέτρηση του αριθμού I/O μεταβάλλοντας το πλήθος των σημείων (Independent)

- **Anti-correlated δεδομένα**

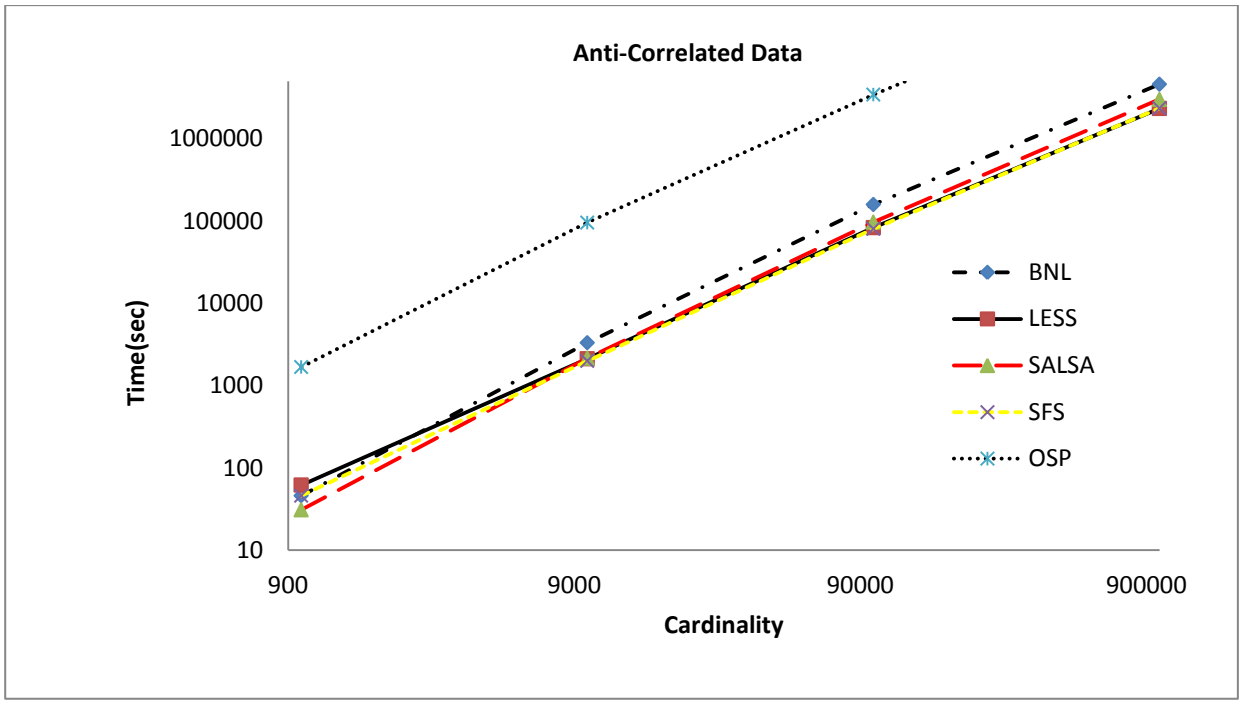
Με την αύξηση του cardinality, είναι λογικό να έχουμε και αύξηση του αριθμού των συγκρίσεων. Στην περίπτωση αυτή, παρατηρούμε ότι για αριθμό σημείων μικρότερο του 100.000 ο αριθμός των συγκρίσεων είναι περίπου ο ίδιος για όλους τους αλγόριθμους, με τον LESS να απαιτεί τις λιγότερες συγκρίσεις και τον BNL και OSP τις περισσότερες. Για μέγαλο πλήθος σημείων τώρα, παρατηρούμε ότι ο αλγόριθμος που απαιτεί τις περισσότερες συγκρίσεις είναι ο BNL, καθώς χρειάζεται να διατρέξει ολόκληρο το σύνολο των δεδομένων αρκετές φορές.

Επίσης, με την αύξηση του αριθμού των σημείων παρατηρούμε ότι ο αλγόριθμος που εκτελείται πιο γρήγορα είναι ο LESS. Αντίθετα για πολύ μικρό cardinality, παρατηρούμε ότι ο χρόνος εκτέλεσης δε διαφέρει σημαντικά μεταξύ των διαφορετικών αλγορίθμων. Και εδώ ο OSP έχει τη χειρότερη απόδοση καθώς με την αύξηση του cardinality, όλο και λιγότερα σημεία μπορούν να επεξεργαστούν πάνω στην κύρια μνήμη.

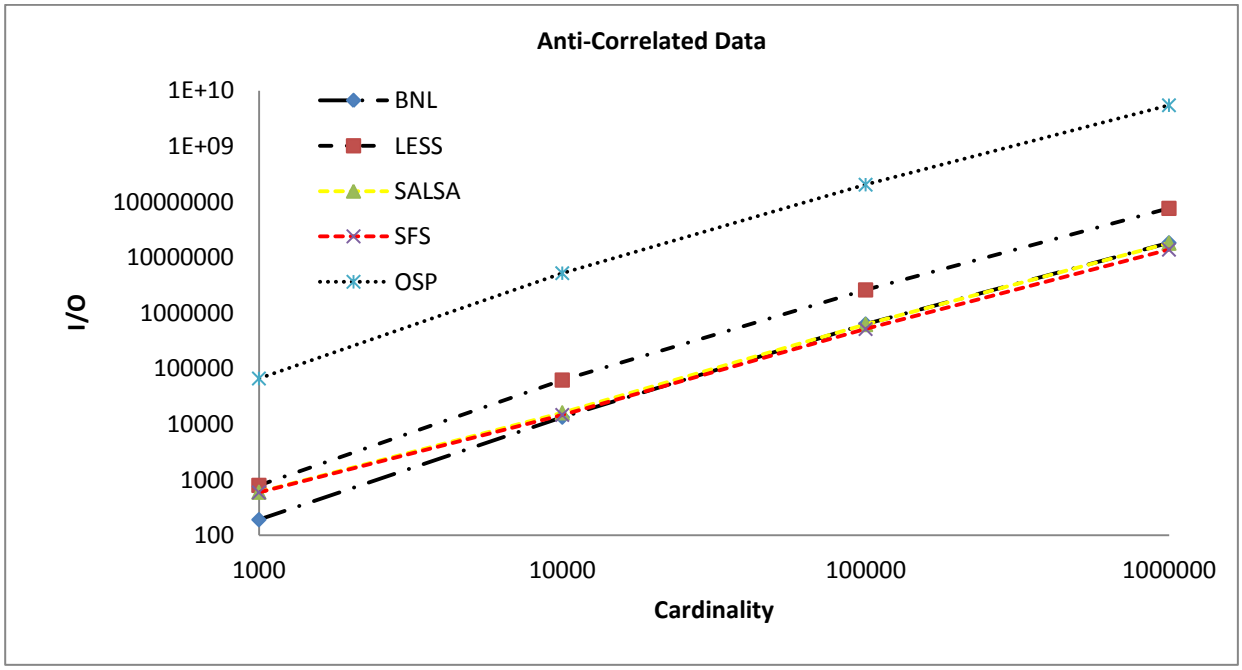
Όσον αφορά τις προσβάσεις στη μνήμη, ο αλγόριθμος με το μικρότερο αριθμό I/O είναι ο BNL και ακολουθεί ο SALSA και ο SFS, ενώ εκείνος που χρειάζεται το μεγαλύτερο αριθμό I/O είναι ο OSP. Ο LESS φαινέται να αποδίδει ελαφρώς χειρότερα από τους SFS και SALSA.



Εικόνα 4:Μέτρηση αριθμού συγκρίσεων μεταβάλλοντας το πλήθος των σημείων (Anti-Correlated)



Εικόνα 5:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το πλήθος των σημείων (Anti-Correlated)



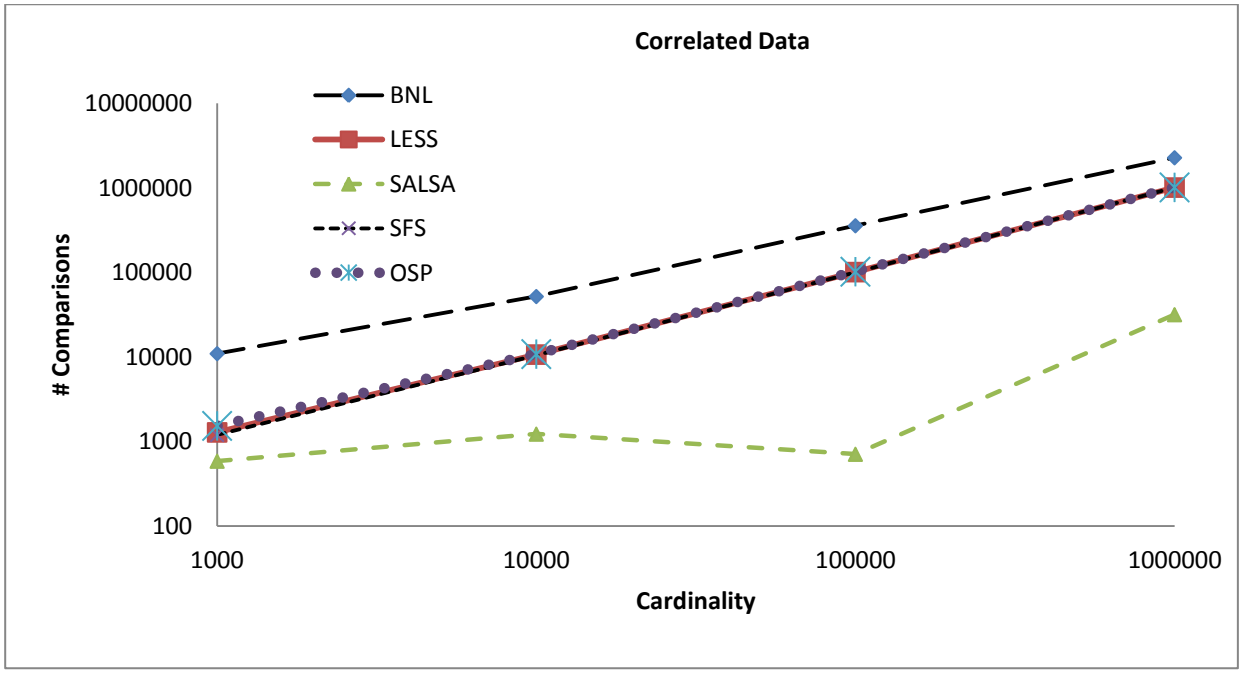
Εικόνα 6:Μέτρηση του αριθμού I/O μεταβάλλοντας το πλήθος των σημείων (Anti-Correlated)

- **Correlated δεδομένα**

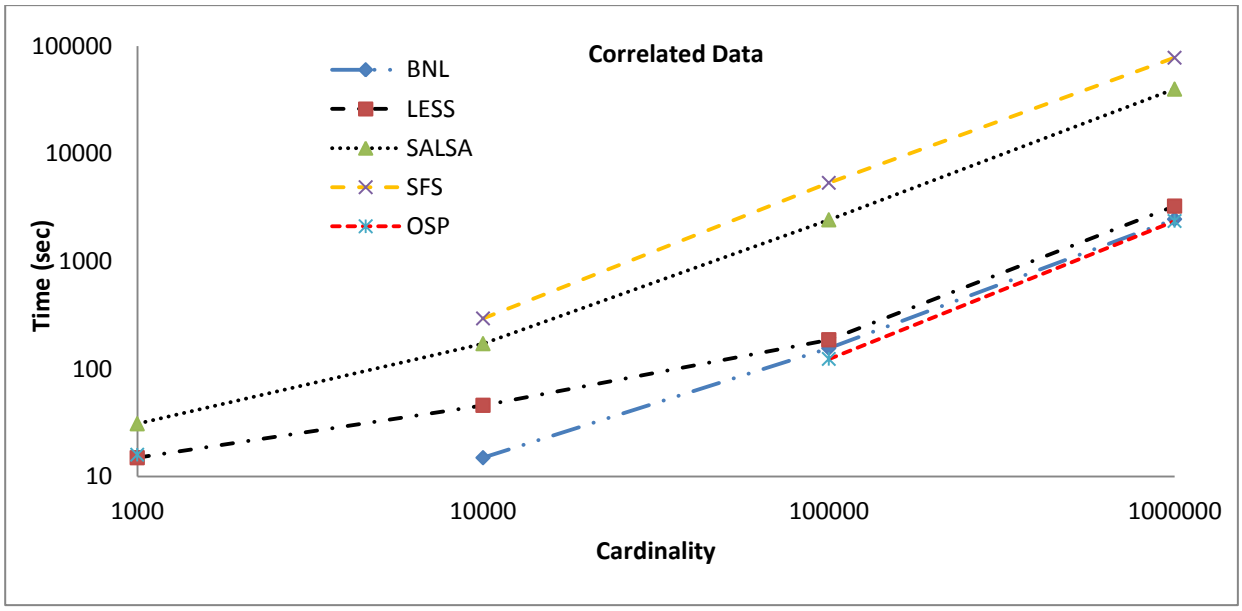
Για τις διαφορετικές τιμές της πληθικότητας (cardinality) βλέπουμε, ότι ο αλγόριθμος που απαιτεί το μικρότερο αριθμό συγκρίσεων με μεγάλη διαφορά από τους άλλους και είναι άρα είναι και ο πιο αποδοτικός, είναι ο SALSA. Ο αριθμός των συγκρίσεων που απαιτούν οι LESS, SFS και OSP είναι περίπου ο ίδιος, ενώ ο BNL απαιτεί το μεγαλύτερο αριθμό συγκρίσεων για όλες τις τιμές του cardinality, οπότε και θεωρείται ο λιγότερο αποδοτικός. Επίσης, αξίζει να σημειώσουμε οτι ενώ οι LESS, SFS και OSP χρειάζονται αριθμό συγκρίσεων περίπου ίσο με το cardinality, ο BNL χρειάζεται αριθμό συγκρίσεων πολλαπλάσιο του cardinality.

Για τις διάφορες τιμές του cardinality οι BNL, LESS, OSP παρουσιάζουν τους μικρότερους χρόνους εκτέλεσης από τους υπόλοιπους. Οι χρόνοι εκτέλεσης μεταξύ των τριών αυτών αλγόριθμων είναι περίπου ίδιοι. Αντίθετα ο SFS και μετά ο SALSA παρουσιάζει τους μεγαλύτερους χρόνους εκτέλεσης από όλους.

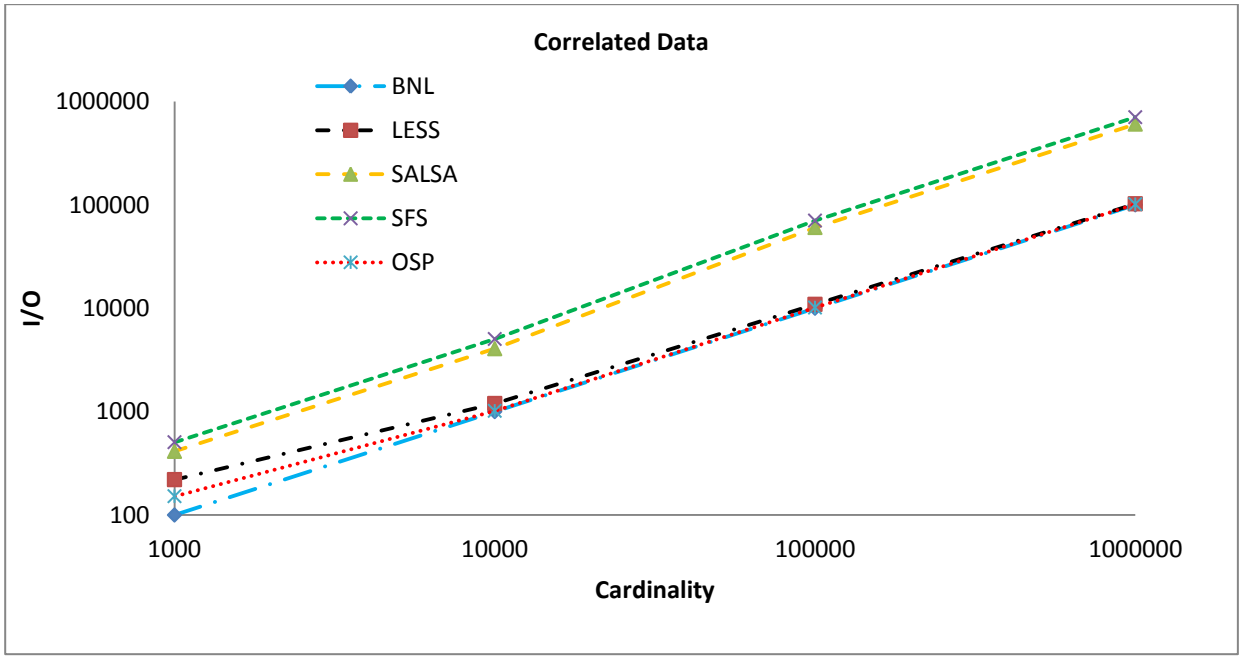
Απ΄την άλλη, οι αλγόριθμοι που χρειάζονται το μικρότερο αριθμό I/O είναι ο BNL, και ο OSP, ενώ οι τιμές των I/O για τον SFS και τον SALSA είναι αρκετά μεγάλοι.



Εικόνα 7:Μέτρηση αριθμού συγκρίσεων μεταβάλλοντας το πλήθος των σημείων (Correlated)



Εικόνα 8:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το πλήθος των σημείων (Correlated)



Εικόνα 9:Μέτρηση του αριθμού I/O μεταβάλλοντας το πλήθος των σημείων (Correlated)

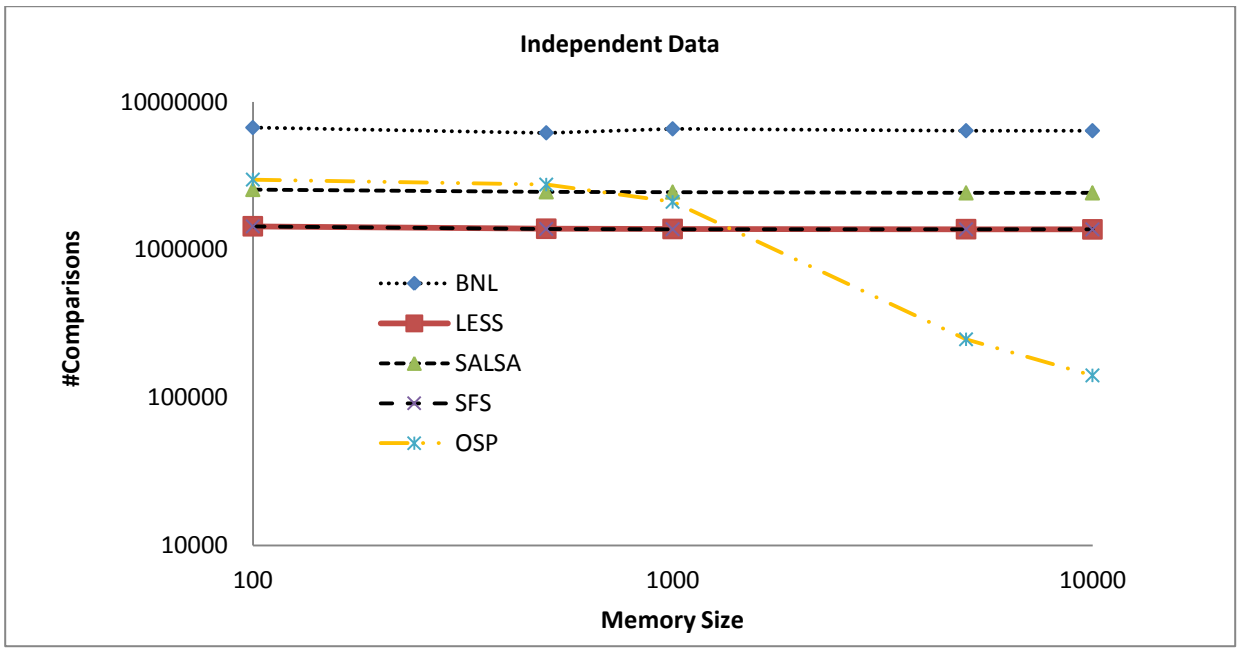
6.3.2 Μεταβάλλοντας το μέγεθος της μνήμης

- **Independent δεδομένα**

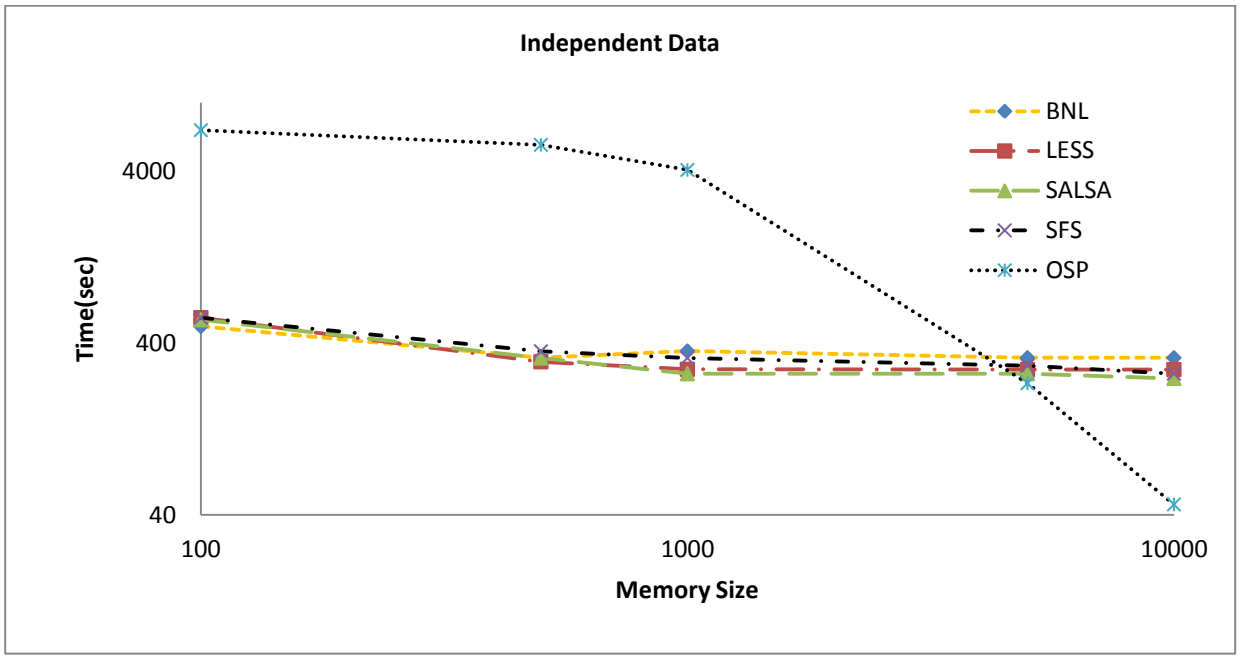
Για τις διάφορες τιμές του memory size, οι αλγόριθμοι που πραγματοποιούν το μικρότερο αριθμό συγκρίσεων σε σχέση με τους άλλους είναι ο LESS και ο SFS, γεγονός που οφείλεται στο στάδιο της ταξινόμησης που εφαρμόζουν πριν την περαιτέρω επεξεργασία των δεδομένων, και το οποίο μειώνει αρκετά τους ελέγχους κυριαρχίας που χρειάζεται να ακολουθήσουν. Επίσης παρατηρούμε οτι με την αύξηση του μεγέθους της μνήμης ο αριθμός των ελέγχων κυριαρχίας που εκτελεί ο OSP μειώνεται σημαντικά.

Όλοι οι αλγόριθμοι εκτός του OSP πραγματοποιούν παρεμφερείς χρόνους εκτέλεσης, οι οποίοι δεν επηρεάζονται από το μέγεθος του memory size. Απ΄την άλλη ο OSP για μικρό μέγεθος μνήμης παρουσιάζει σημαντικά μεγαλύτερους χρόνους εκτέλεσης σε σχέση με τους υπόλοιπους αλγόριθμους, ενώ όταν το μέγεθος της μνήμης γίνεται ίσο με το πλήθος των σημείων, ο OSP είναι αρκετές τάξεις πιο γρήγορος από όλους τους άλλους αλγόριθμους.

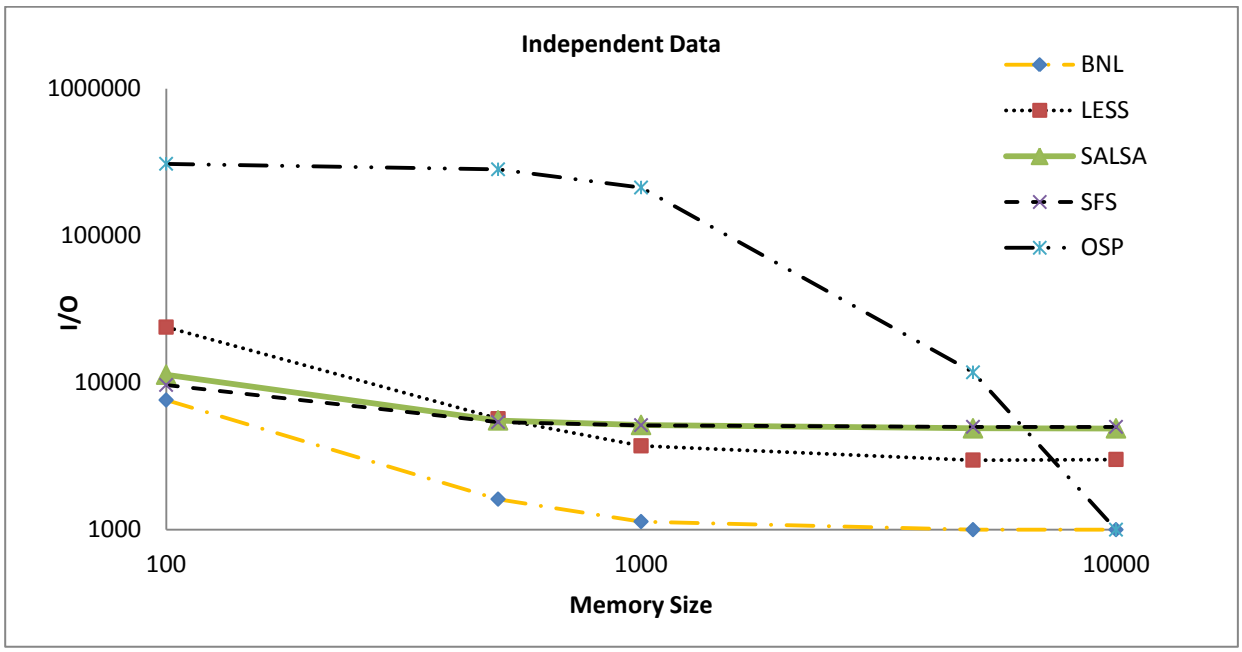
Για τις διαφορετικές τιμές του memory size, ο BNL παρουσιάζει το μικρότερο αριθμό I/O, ενώ ο OSP χρειάζεται σημαντικά μεγαλύτερο αριθμό I/O συγκριτικά με τους άλλους αλγόριθμους. Όταν όμως το μέγεθος της μνήμης γίνει ίσο με το πλήθος των σημείων ο αριθμός των I/O του OSP μειώνεται σημαντικά στο ελάχιστο. Οι LESS, SFS και SALSA χρειάζονται περίπου τον ίδιο αριθμό I/O και παρόλο που ο αριθμός των I/O δε φαίνεται ανα επηρεάζεται από το memory size στους SFS και SALSA, κάτι τέτοιο δεν ισχύει και για τον LESS.



Εικόνα 10:Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας το μέγεθος της μνήμης (Independent)



Εικόνα 11:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το μέγεθος της μνήμης (Independent)

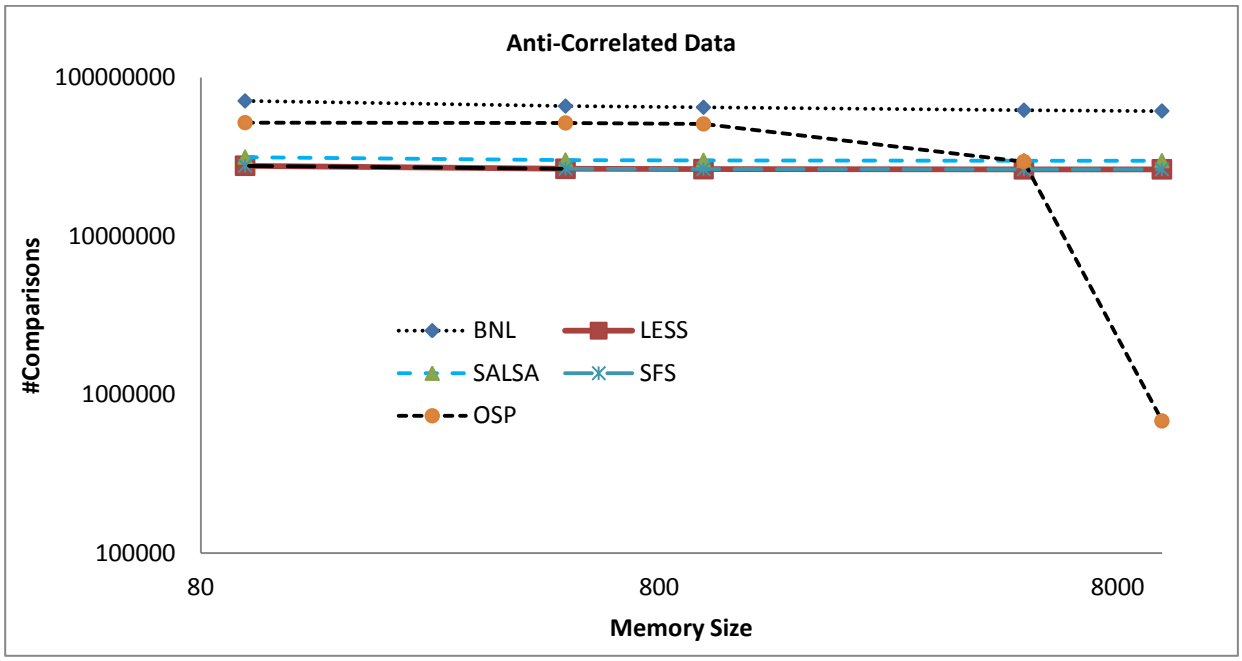


Εικόνα 12:Μέτρηση του αριθμού I/O μεταβάλλοντας το μέγεθος της μνήμης (Independent)

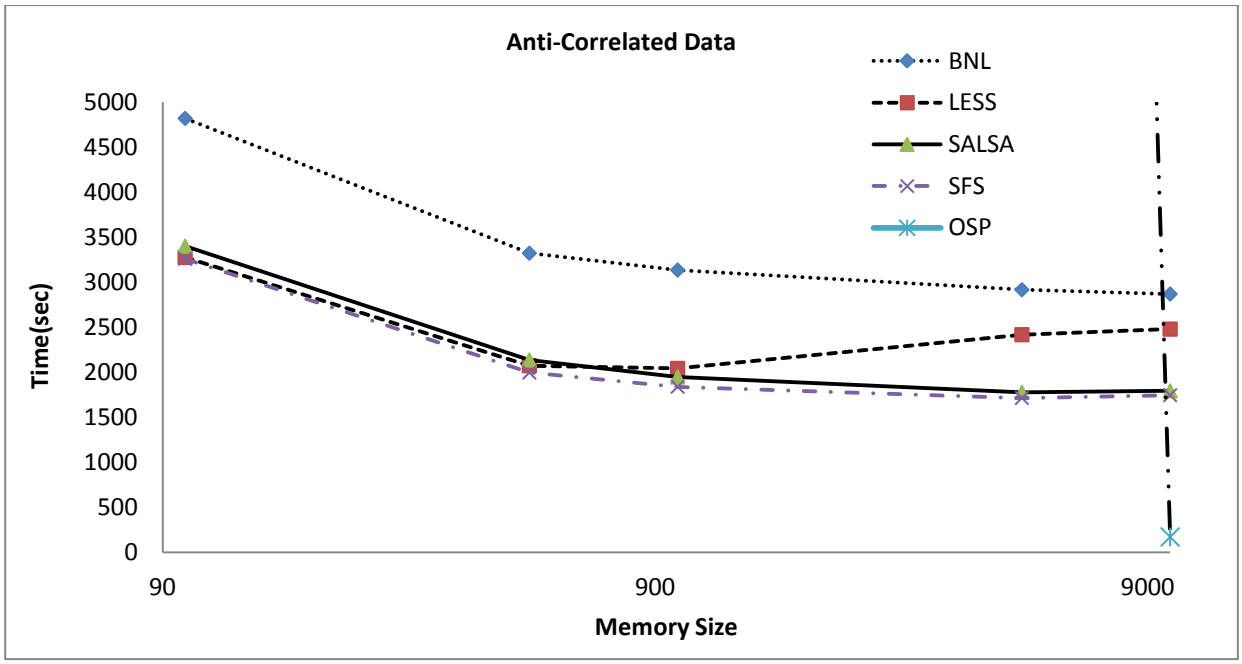
- **Anti-correlated δεδομένα**

Στην περίπτωση αυτή παρατηρούμε οτι η αύξηση του μεγέθους της μνήμης δεν επηρεάζει σχεδόν καθόλου τον αριθμό των συγκρίσεων τπου πρέπει να εκτελέσει κάθε αλγόριθμος, με τον LESS και τον SFS να απαιτούν τις λιγότερες συγκρίσεις. Ο μόνος αλγόριθμος που διαφοροποιείται όπως και παραπάνω είναι ο OSP που όταν το μέγεθος της μνήμης γίνει ίσο με το πλήθος των σημείων, χρειάζεται σημαντικά μικρότερο αριθμό συγκρίσεων από όλους τους άλλους αλγόριθμους. Στο πείραμα αυτό βλέπουμε γενικά οτι ο BNL είναι ο αλγόριθμος που εκτελεί το μεγαλύτερο αριθμό συγκρίσεων.

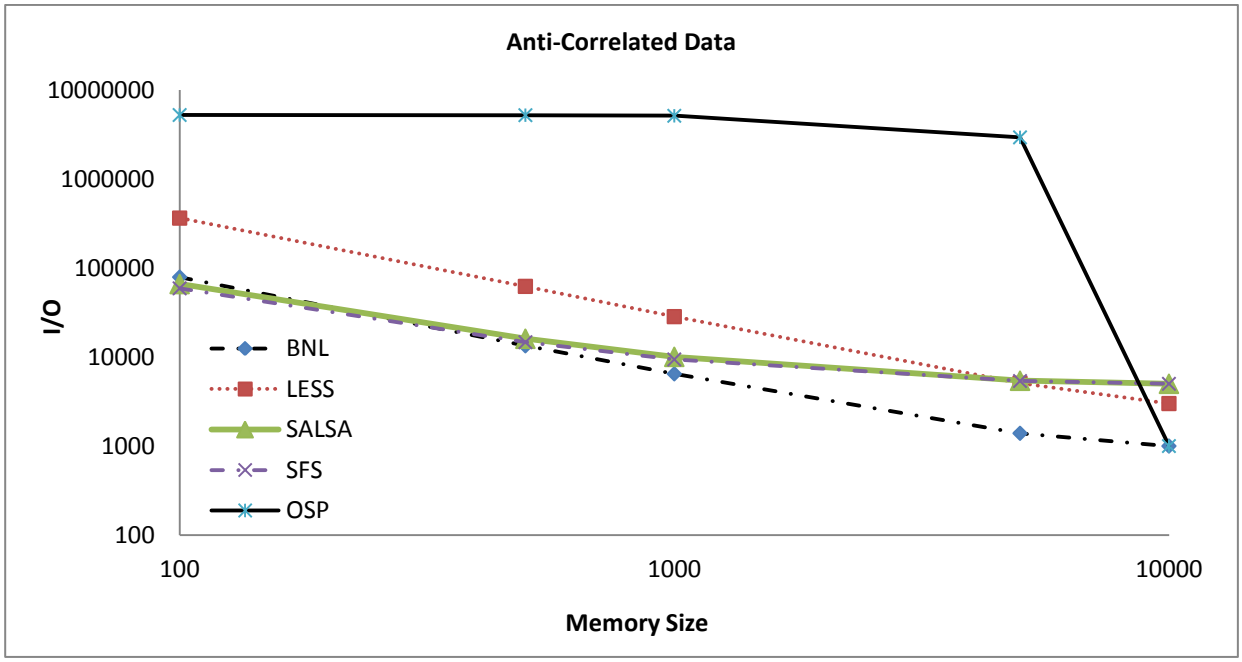
Από άποψη χρόνου ο BNL είναι επίσης αρκετά αργός, όμως το μεγαλύτερο κόστος χρόνου το παρουσιάζει ο OSP στην περίπτωση που η διαθέσιμη μνήμη είναι πολύ μικρή. Αντίθετα ο BNL είναι εκείνος που απαιτεί τα λιγότερα I/O, τα οποία μειώνονται με την αύξηση της μνήμης. Και σ'αυτήν την περίπτωση ο OSP αντιδρά όπως και προηγουμένως.



Εικόνα 13:Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας το μέγεθος της μνήμης (Anti-Correlated)



Εικόνα 14:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το μέγεθος της μνήμης (Anti-Correlated)



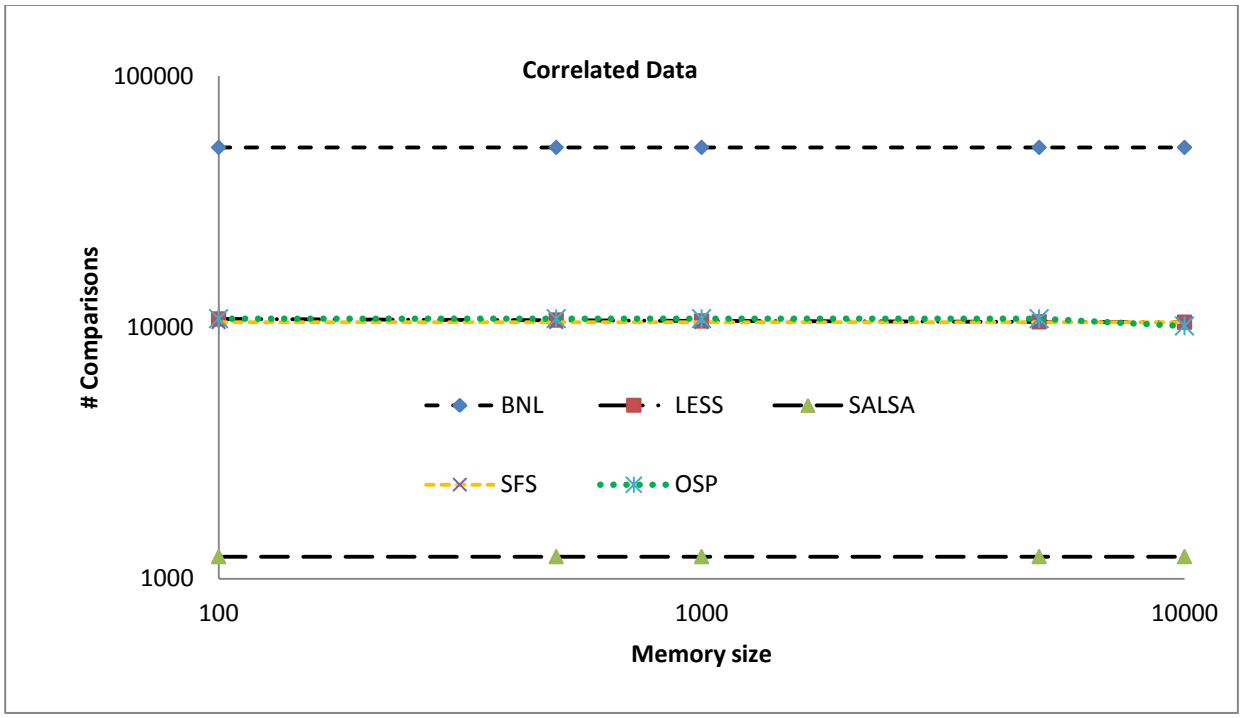
Εικόνα 15:Μέτρηση του αριθμού I/O μεταβάλλοντας το μέγεθος της μνήμης (Anti-Correlated)

- **Correlated δεδομένα**

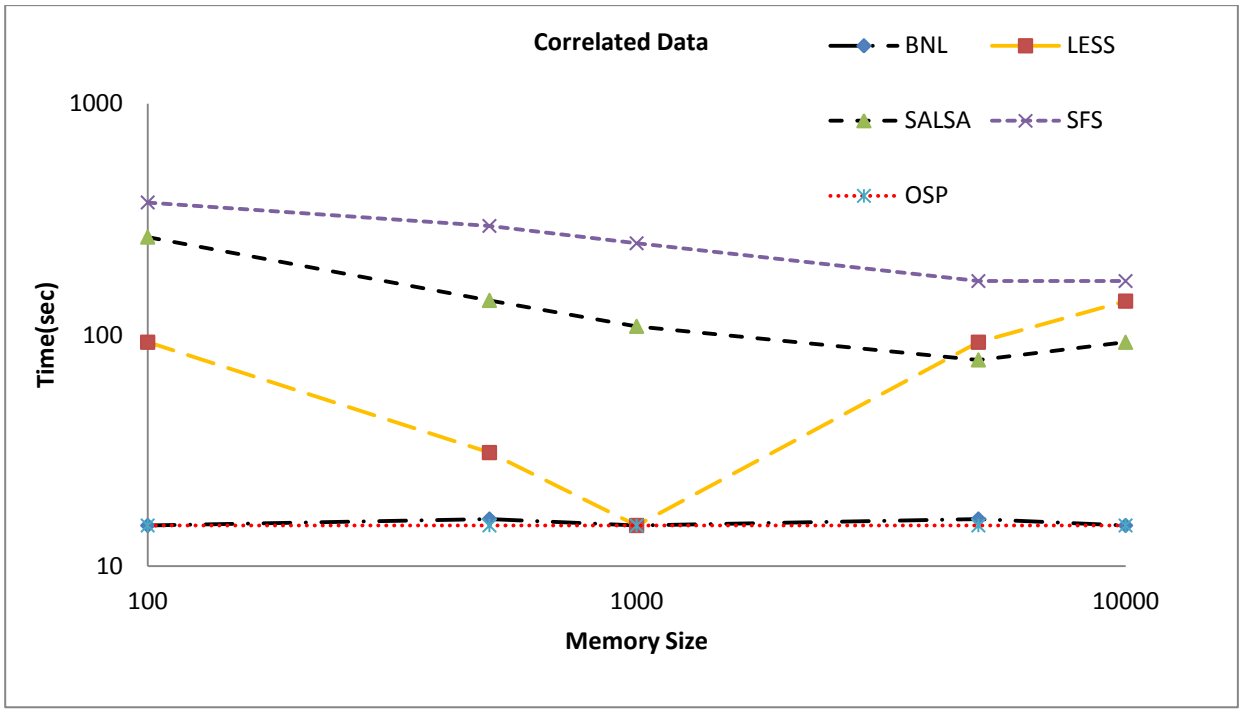
Στην περίπτωση αυτή, ο αλγόριθμος SALSA απαιτεί το λιγότερο αριθμό συγκρίσεων, ενώ ο BNL το μεγαλύτερο με μεγάλη διαφορά στη μεταξύ τους απόδοση. Απ΄την άλλη οι SFS, LESS και OSP έχουν την ίδια σχεδόν απόδοση μεταξύ τους.

Για τις διάφορες τιμές του memory size οι BNL και OSP παρουσιάζουν τους ίδιους χρόνους εκτέλεσης, οι οποίοι παραμένουν ανεπηρέαστοι από τις αυξομειώσεις του memory size. Οι αλγόριθμοι αυτοί είναι και οι ταχύτεροι στη συγκεκριμένη περίπτωση, ενώ ο SFS παρουσιάζει το μεγαλύτερο κόστος σε χρόνο από όλους τους άλλους.

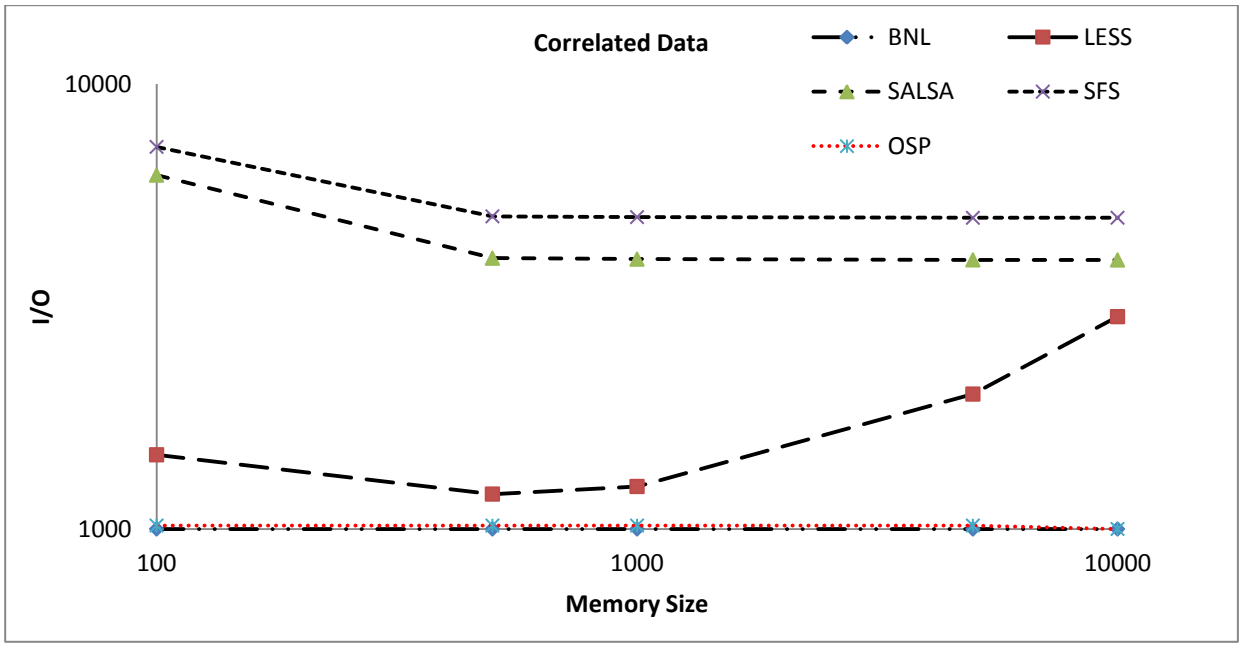
Οι αλγόριθμοι που απαιτούν τα λιγότερα I/O είναι ο BNL και ο OSP και μάλιστα ο αριθμός των I/O παραμένει σταθερός και ανεξάρτητος από το memory size. Απ΄την άλλη, ο SFS και ο SALSA παρουσιάζουν το μεγαλύτερο αριθμό I/O, χωρίς μεγάλες διακυμάνσεις όμως, λόγω μεταβολής του memory size.



Εικόνα 16:Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας το μέγεθος της μνήμης (Correlated)



Εικόνα 17:Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας το μέγεθος της μνήμης (Correlated)



Εικόνα 18:Μέτρηση του αριθμού I/O μεταβάλλοντας το μέγεθος της μνήμης (Correlated)

6.3.3 Μεταβάλλοντας τον αριθμό των διαστάσεων

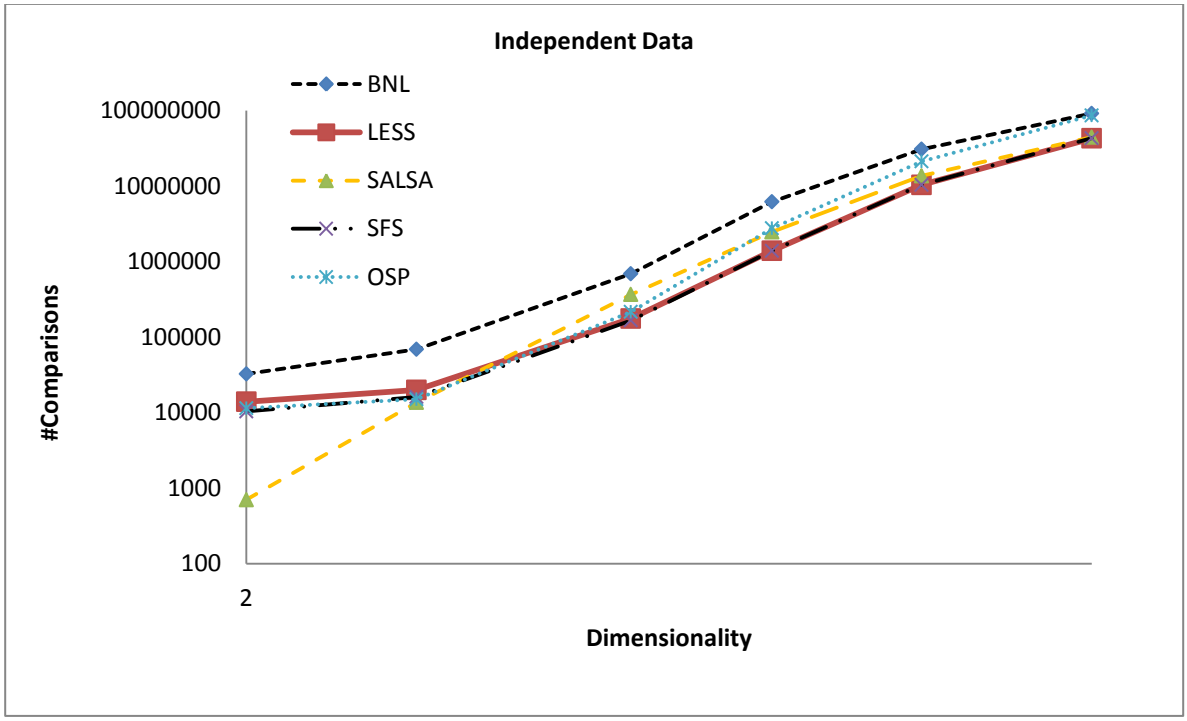
- **Independent δεδομένα**

Σ'αυτήν την περίπτωση, παρατηρούμε ότι καθώς αυξάνεται το dimensionality ο αριθμός των συγκρίσεων που πραγματοποιεί κάθε αλγόριθμος, αυξάνεται και αυτός σημαντικά.

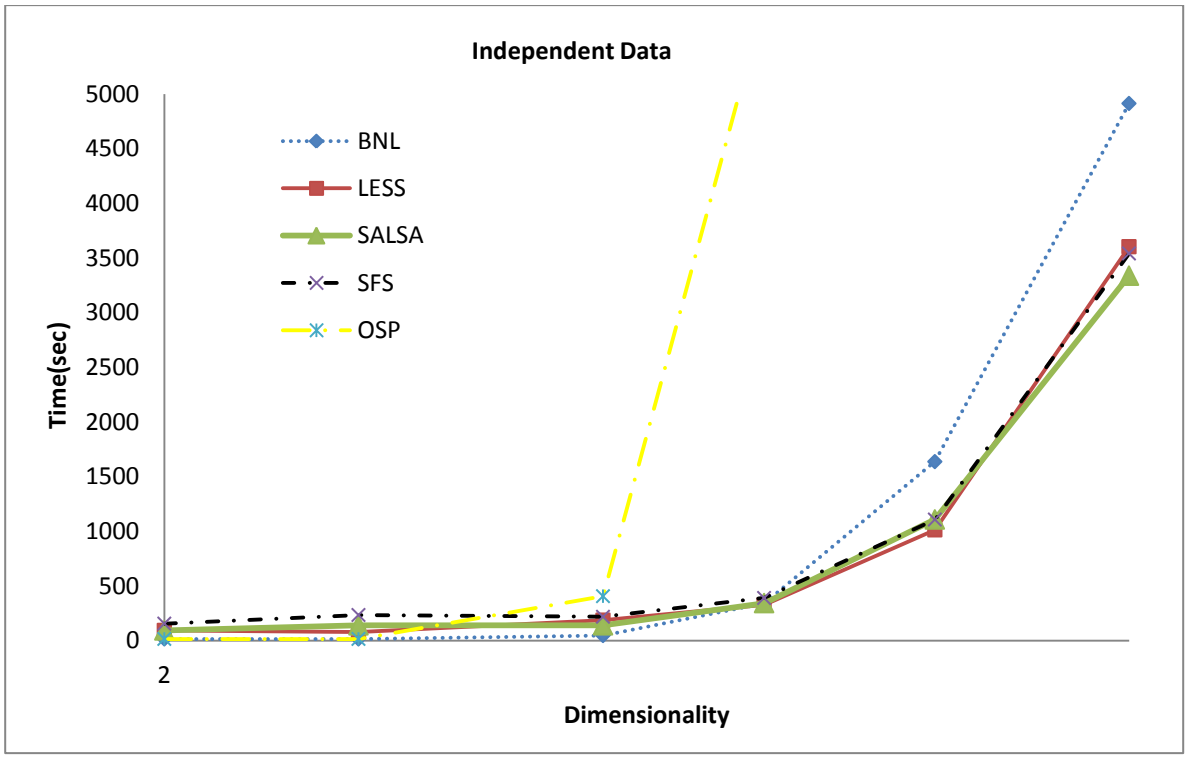
Συγκεκριμένα, ο αλγόριθμος που απαιτεί το μικρότερο αριθμό συγκρίσεων για τις διάφορες τιμές του dimensionality είναι ο LESS, ενώ αυτός που πραγματοποιεί τις περισσότερες συγκρίσεις είναι ο BNL. Για dimensionality<7 ο LESS, ο SFS και ο OSP πραγματοποιούν παρόμοιο αριθμό συγκρίσεων.

Οι αλγόριθμοι που έχουν τους μικρότερους χρόνους εκτέλεσης είναι οι SFS, LESS και SALSA και οι χρόνοι και των τριών αυτών αλγόριθμων είναι σχεδόν ίδιοι. Απ΄την άλλη, οι BNL και OSP ενώ είναι παρά πολύ γρήγοροι για χαμηλό dimensionality, ο χρόνος εκτέλεσης τους αυξάνει σημαντικά για dimensionality>7.

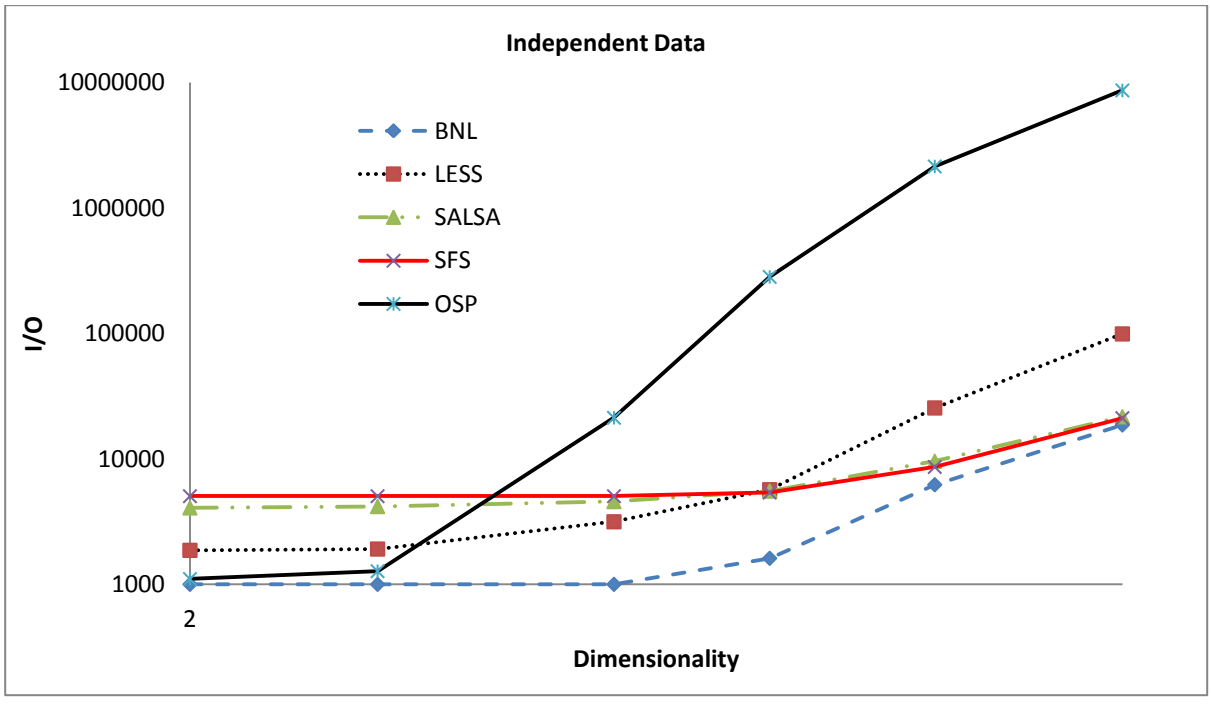
Ο αλγόριθμος που πραγματοποιεί τα λιγότερα I/O είναι ο BNL. Αντίθετα, ο OSP έχει τη χειρότερη απόδοση από όλους, καθώς απαιτεί αρκετά μεγάλο αριθμό I/O. Επίσης, αξίζει να σημειωθεί ότι ο SFS και ο SALSA παρουσιάζουν παρόμοια συμπεριφορά.



Εικόνα 19: Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας τη διάσταση(Independent)



Εικόνα 20: Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας τη διάσταση(Independent)



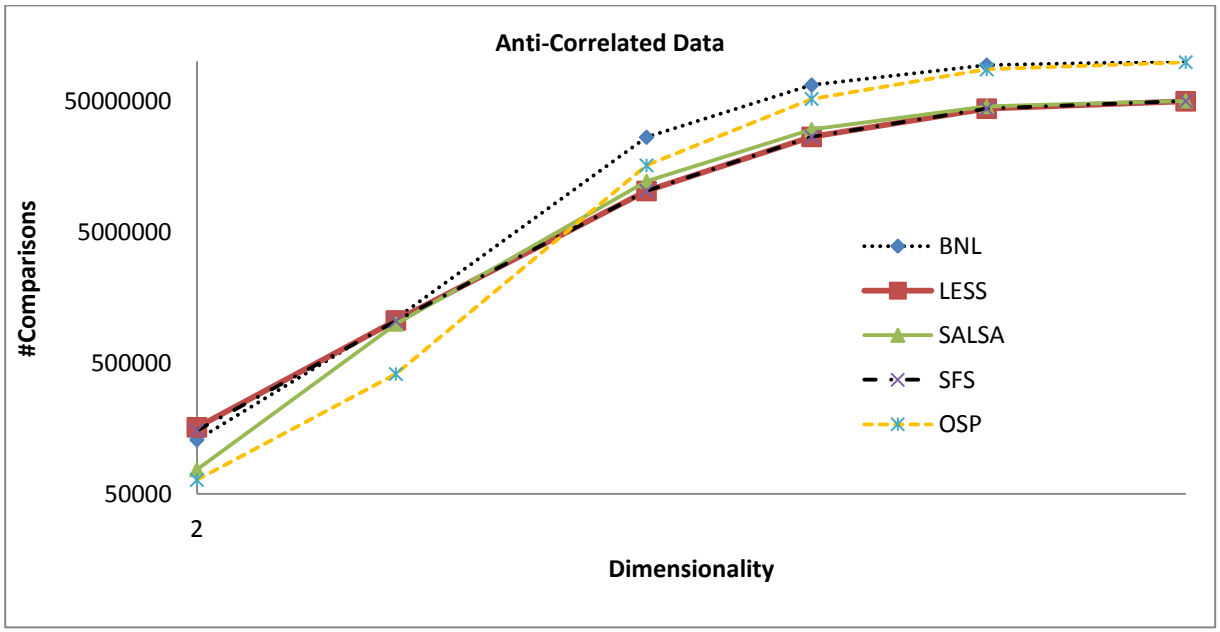
Εικόνα 21: Μέτρηση του αριθμού I/O μεταβάλλοντας τη διάσταση(Independent)

- **Anti-correlated δεδομένα**

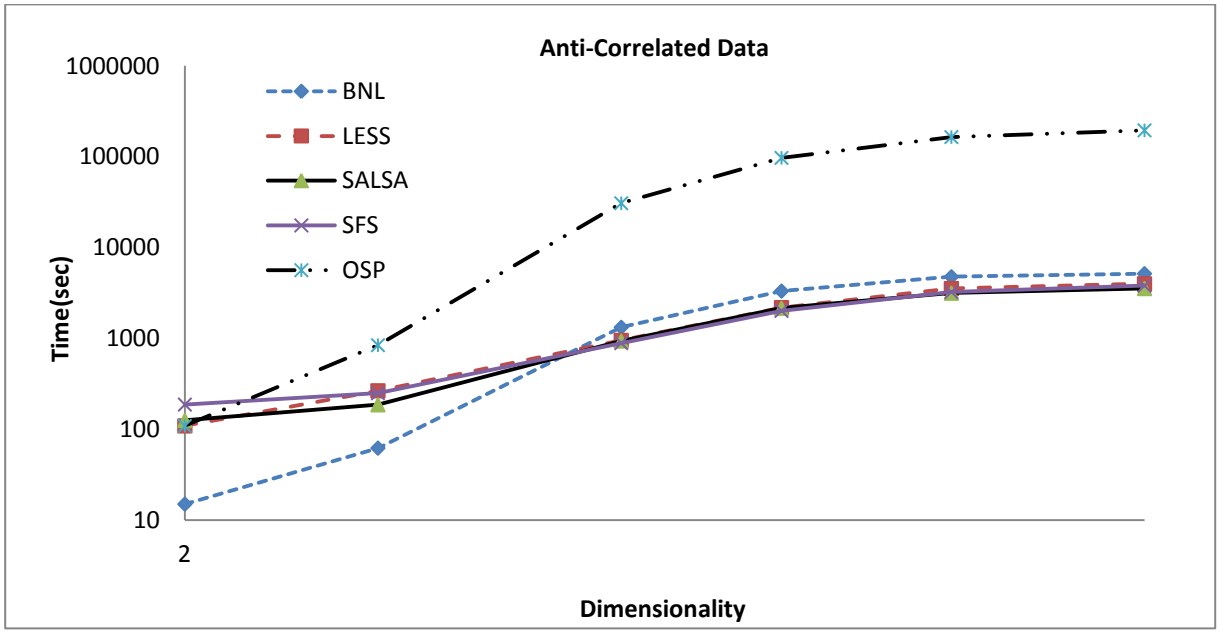
Με την αύξηση του dimensionality παρατηρούμε ότι ο αλγόριθμος που απαιτεί τις περισσότερες συγκρίσεις είναι ο BNL, ενώ ακολουθεί ο OSP. Και οι δύο αυτοί αλγόριθμοι δείχνουν να επηρεάζονται αρκετά από τον αριθμό των διαστάσεων, καθώς ο αριθμός των ελέγχων κυριαρχίας που χρειάζεται να πραγματοποιήσουν αυξάνει σημαντικά καθώς αυξάνει το dimensionality. Αντίθετα, οι SALSA, SFS, και LESS χρειάζονται λιγότερες συγκρίσεις από τον BNL και OSP, με τον LESS και τον SFS να είναι οι πιο αποδοτικοί από όλους.

Ειδικότερα, παρατηρούμε ότι ενώ για μικρό dimensionality ο BNL έχει τους μικρότερους χρόνους εκτέλεσης από όλους τους υπόλοιπους αλγόριθμους, όσο αυξάνεται το dimensionality, οι αλγόριθμοι που αποδίδουν καλύτερα είναι οι LESS, SALSA και SFS, με τον SALSA να είναι ο γρηγορότερος από όλους. Αντίθετα,ο OSP με την αύξηση του dimensionality γίνεται σημαντικά πιο αργός από όλους τους άλλους. Και πάλι εδώ φαίνεται οτι οι BNL και OSP είναι εκείνοι που ο αριθμός των διαστάσεων τους επηρεάζει αρκετά.

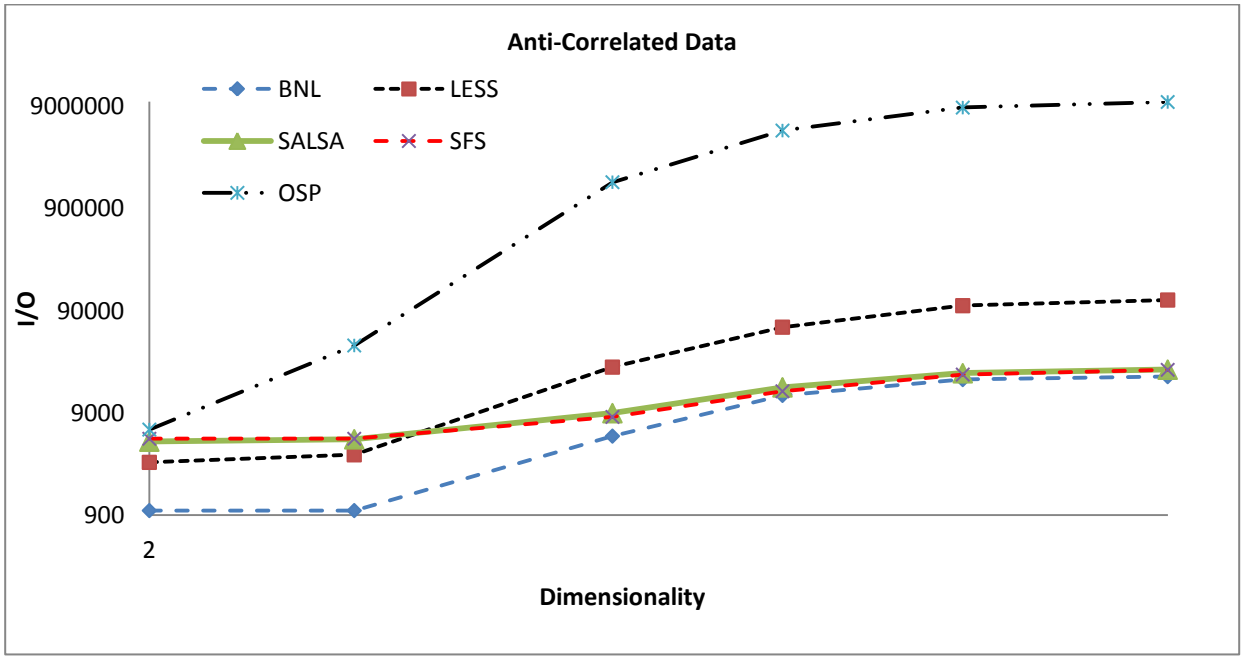
Ο αλγόριθμος που χρειάζεται το μικρότερο αριθμό I/O είναι ο BNL και ακολουθεί ο SALSA και ο SFS, ενώ εκείνος που χρειάζεται το μεγαλύτερο αριθμό I/O είναι ο OSP.



Εικόνα 22: Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας τη διάσταση(Anti-Correlated)



Εικόνα 23: Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας τη διάσταση(Anti-Correlated)



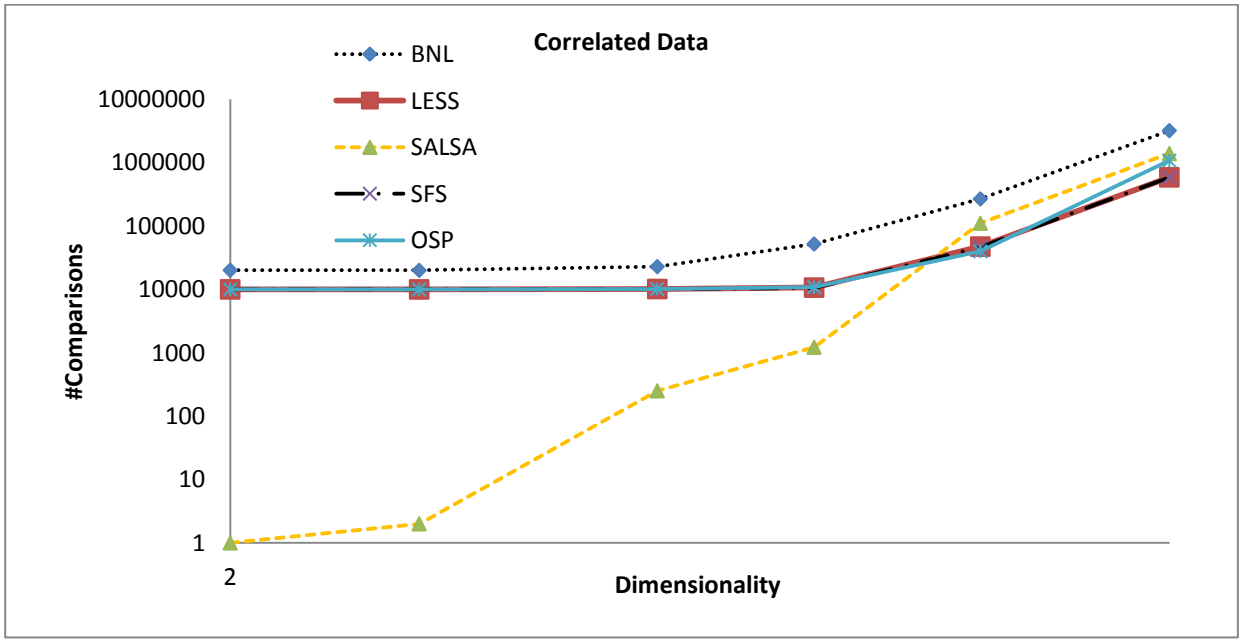
Εικόνα 24: Μέτρηση του αριθμού I/O μεταβάλλοντας τη διάσταση(Anti-Correlated)

- **Correlated δεδομένα**

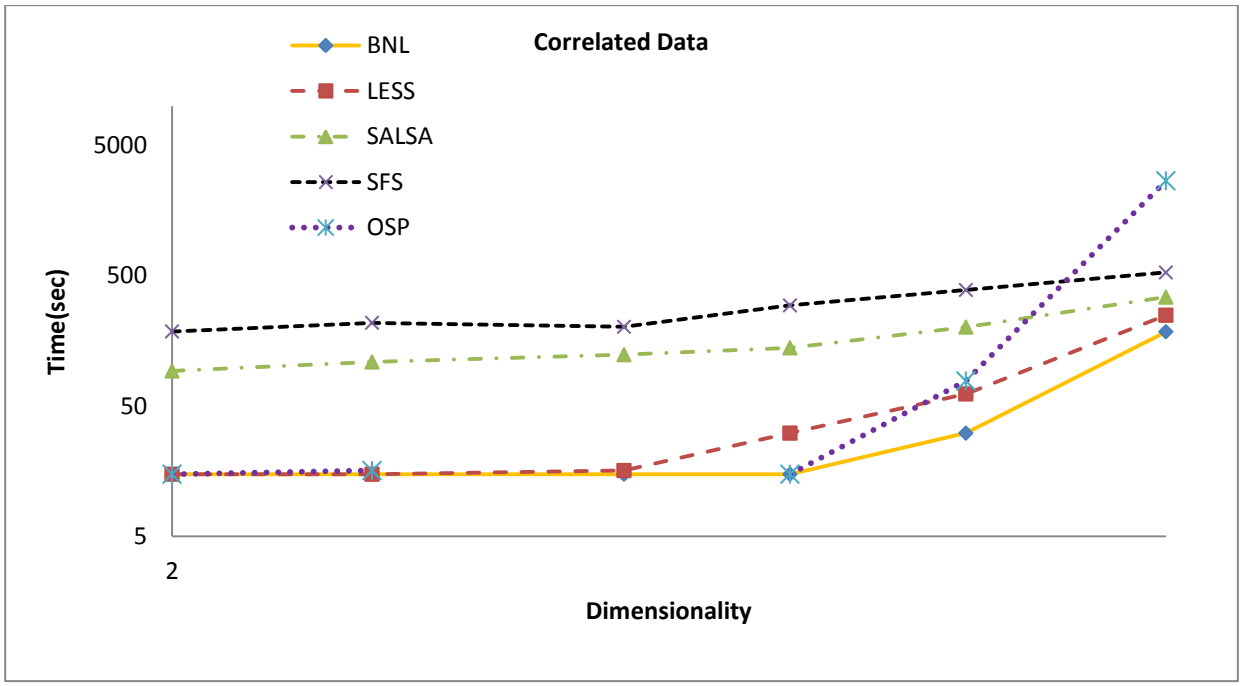
Για τις διάφορες τιμές του dimensionality, ο SFS, ο LESS και ο OSP απαιτούν περίπου τον ίδιο αριθμό συγκρίσεων, ενώ ο BNL απαιτεί το μεγαλύτερο αριθμό συγκρίσεων από όλους. Απ΄την άλλη ο SALSA φαίνεται να επηρεάζεται σημαντικά από την αύξηση της διάστασης των σημείων καθώς απαιτεί σημαντικά μικρότερο αριθμό συγκρίσεων από όλους τους υπόλοιπους αλγόριθμους για μικρό αριθμό διαστάσεων, ενώ για dimensionality>7, ο αριθμός των συγκρίσεων που εκτελεί πλησιάζει αυτόν των άλλων.

Από άποψη χρόνου, ο SFS είναι γενικά ο πιο αργός από όλους, ενώ ο BNL ο πιο γρήγορος, με το χρόνο του OSP να αυξάνεται σημαντικά όταν το dimensionality γίνει >7.

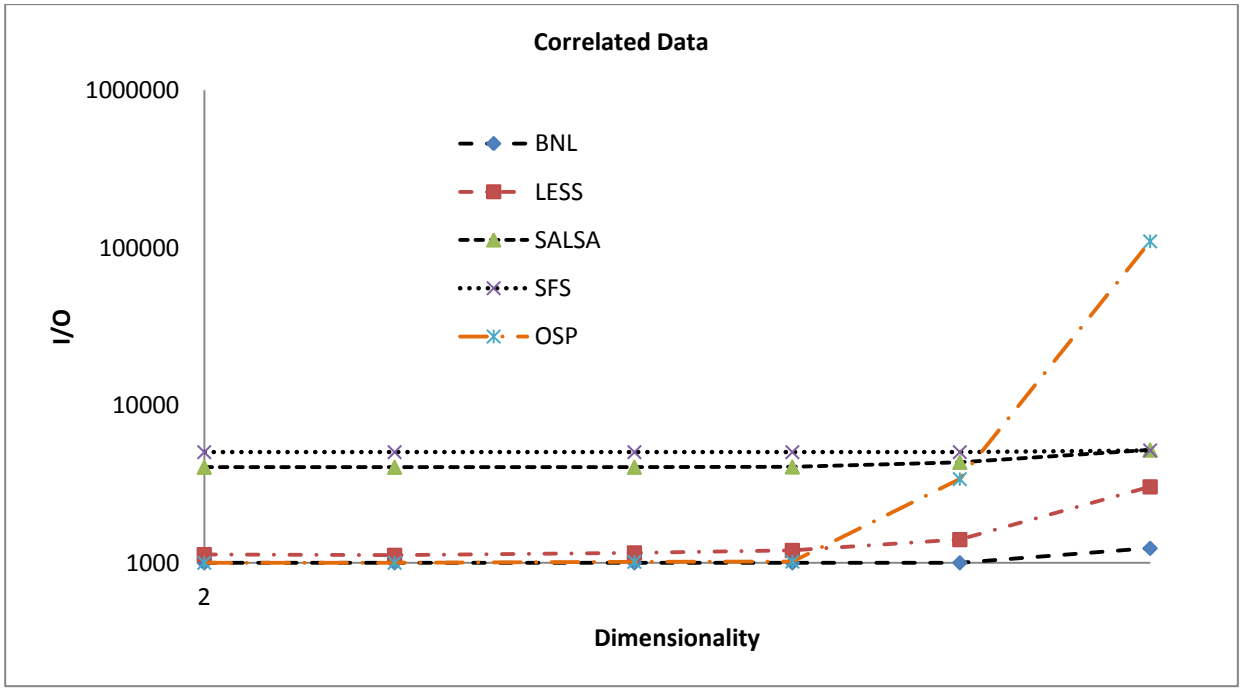
Το μεγαλύτερο αριθμό I/O χρειάζονται οι SFS και BNL, ενώ το λιγότερο ο SALSA και ο LESS. Επίσης αξίζει να σημειωθεί οτι ο OSP για μικρό αριθμό διαστάσεων χρειάζεται αρκετά μικρό αριθμό I/O, παρόμοιο με αυτόν των SALSA και LESS, όταν όμως dimensionality γίνει >7, παρουσιάζει μια μεγάλη αύξηση των I/O.



Εικόνα 25: Μέτρηση του αριθμού συγκρίσεων μεταβάλλοντας τη διάσταση(Correlated)



Εικόνα 26: Μέτρηση του χρόνου εκτέλεσης μεταβάλλοντας τη διάσταση(Correlated)



Εικόνα 27: Μέτρηση του αριθμού I/O μεταβάλλοντας τη διάσταση(Correlated)

6.4 Σύνοψη συμπερασμάτων αξιολόγησης

Συνοπτικά από τις παραπάνω παρατηρήσεις βγαίνουν τα εξής συμπεράσματα:

1. Στο σύνολο των anti-correlated δεδομένων λόγω του μεγάλου μεγέθους του skyline, οι αλγόριθμοι με την καλύτερη απόδοση είναι οι LESS, SFS και SALSA, δηλαδή οι αλγόριθμοι που βασίζονται στην εξωτερική ταξινόμηση των σημείων. Επίσης, ο OSP αλγόριθμος, στην περίπτωση μη περιορισμένης μνήμης γίνεται ο πιο αποδοτικός από όλους κατά μεγάλο βαθμό.
2. Στο σύνολο των independent δεδομένων, οι αλγόριθμοι που χρειάζονται το μικρότερο αριθμό συγκρίσεων είναι ο SFS και ο LESS, ενώ ο BNL είναι ο ταχύτερος από όλους και πραγματοποιεί το μικρότερο αριθμό I/O. Επίσης, στην περίπτωση που δεν υπάρχει περιορισμός στην κύρια μνήμη, ο αλγόριθμος OSP γίνεται ο πιο αποδοτικός από όλους κατά μεγάλο βαθμό.
3. Στο σύνολο των correlated δεδομένων, ο αλγόριθμος SALSA πραγματοποιεί το μικρότερο αριθμό συγκρίσεων από όλους, ενώ οι αλγόριθμοι που είναι πιο γρήγοροι και πραγματοποιούν λιγότερα I/O είναι ο BNL και ο OSP. Αξιοσημείωτο είναι και το ότι το μέγεθος της κύριας μνήμης αφήνει ανεπηρέαστα τον αριθμό συγκρίσεων, το χρόνο εκτέλεσης και τα I/O. Τα παραπάνω οφείλονται στο σημαντικά μικρό μέγεθος του skyline σ'αυτό το σύνολο δεδομένων, καθώς και στο είδος της κατανομής.
4. Με την αύξηση του μεγέθους του συνόλου δεδομένων (cardinality), ο BNL απαιτεί το μεγαλύτερο αριθμό συγκρίσεων ανεξάρτητα από την κατανομή των δεδομένων. Αντίθετα οι υπόλοιποι αλγόριθμοι φαίνεται να επηρεάζονται από την κατανομή, με τον SALSA να αποδίδει καλύτερα στα correlated δεδομένα, τον LESS και SFS για independent δεδομένα και τον OSP για anti-correlated. Από άποψη χρόνου, ο SFS εμφανίζεται να είναι ο πιο αργός για correlated δεδομένα, ενώ για τις άλλες δύο κατανομές ο OSP έχει μεγάλο κόστος χρόνου, καθώς και μεγάλο αριθμό σε I/O. Στα correlated δεδομένα οι OSP, LESS και BNL γίνονται οι πιο γρήγοροι και απαιτούν μικρότερο αριθμό I/O από τους άλλους.
5. Αυξάνοντας το μέγεθος της μνήμης, οι SFS και LESS απαιτούν τις λιγότερες συγκρίσεις για την independent και anti-correlated κατανομή, ενω στην correlated κατανομή ο SALSA γίνεται πιο αποδοτικός. Αντίθετα σε όλα τα σύνολα δεδομένων ο BNL είναι εκείνος που εκτελεί τους περισσότερους ελέγχους κυριαρχίας. Από άποψη χρόνου, ο OSP και BNL είναι οι πιο γρήγοροι για την correlated κατανομή, ενώ ο SFS και SALSA για τις άλλες δύο. Στα correlated δεδομένα ο SFS απαιτεί τα περισσότερα I/O, ενώ οι OSP και BNL τα λιγότερα. Αντίθετα στα independent δεδομένα ο OSP απαιτεί τα περισσότερα I/O, ενώ ο BNL τα λιγότερα. Στην anti-correlated κατανομή, οι SALSA, SFS και OSP απαιτούν παρεμφερή αριθμό I/O, ενώ ο BNL το μεγαλύτερο.

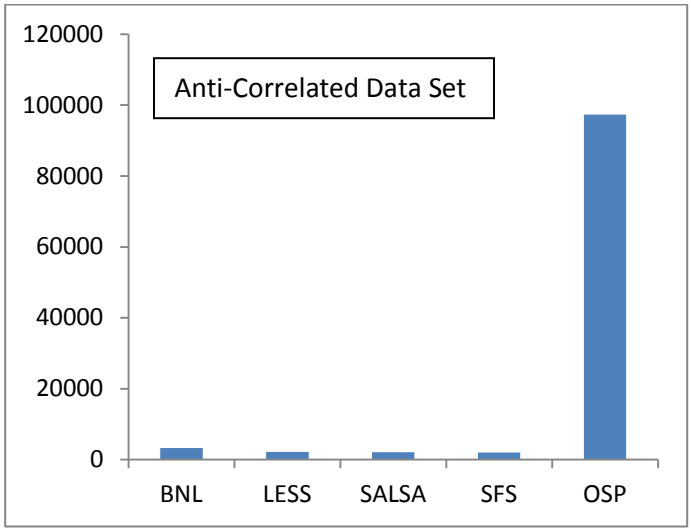
6. Εξετάζοντας την επίδραση που έχει η dimensionality στους αλγόριθμους παρατηρούμε οτι αυξάνοντας τις διαστάσεις των σημείων του συνόλου δεδομένων μας, ο πιο αποδοτικός από όλους τους αλγόριθμους σε όλες τις περιπτώσεις είναι ο SFS. Ο BNL είναι εκείνος που χρειάζεται το μεγαλύτερο αριθμό συγκρίσεων, ενώ ο OSP είναι ο πιο αργός και παρουσιάζει το μεγαλύτερο αριθμό I/O.

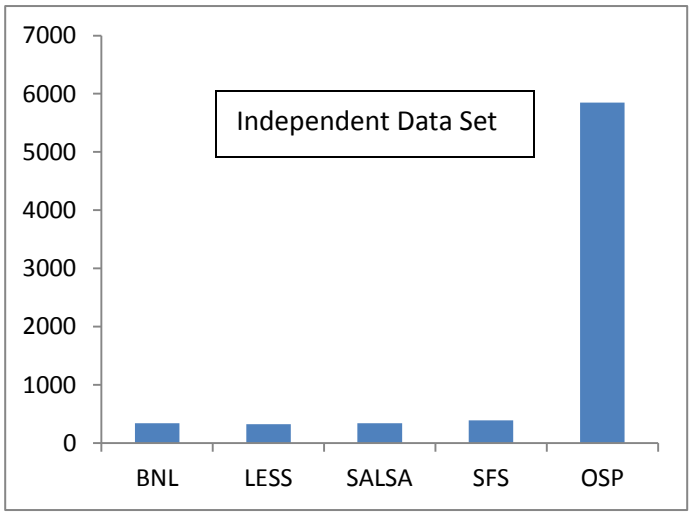
Για τις μέσες τιμές των παραμέτρων των μετρήσεων μας, δηλαδή για dimensionality=7, cardinality=10.000, memory size=500, παρουσιάζονται παρακάτω διαγραμματικά οι αποδόσεις των διαφορετικών αλγορίθμων που μελετήθηκαν, για όλες τις διαφορετικές κατανομές των δεδομένων μας.

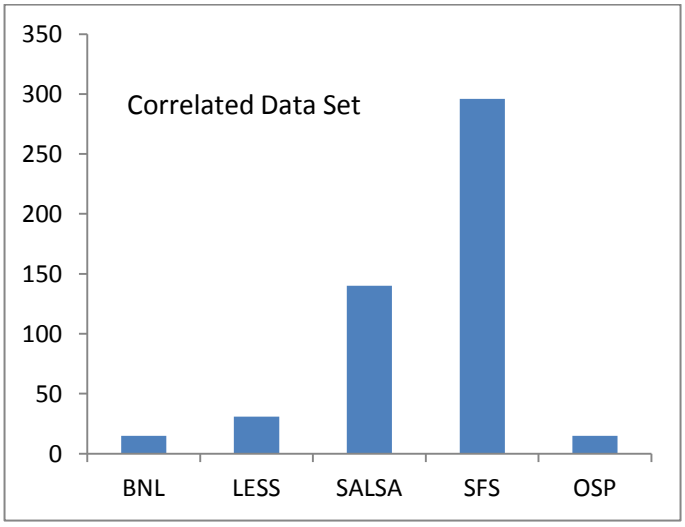
Στα παρακάτω διαγράμματα φαίνεται οτι η κατανομή των δεδομένων επηρεάζει σημαντικά το χρόνο εκτέλεσης των SALSA, SFS και OSP, τον αριθμό των ελέγχων κυριαρχίας στον SALSA και τον OSP, καθώς και τον αριθμό I/O των SALSA, SFS και OSP.

Σε γενικές γραμμές ο LESS φαίνεται να είναι ο πιο αποδοτικός αλγόριθμος από όλους. Αξιοσημείωτο είναι οτι ο BNL έχει πολύ μικρό κόστος σε χρόνο και I/O, αλλά απαιτεί μεγάλο αριθμό συγκρίσεων ανεξάρτητα από την κατανομή των δεδομένων, ενώ ο OSP γίνεται εξαιρετικά αποδοτικός για την correlated κατανομή δεδομένων.

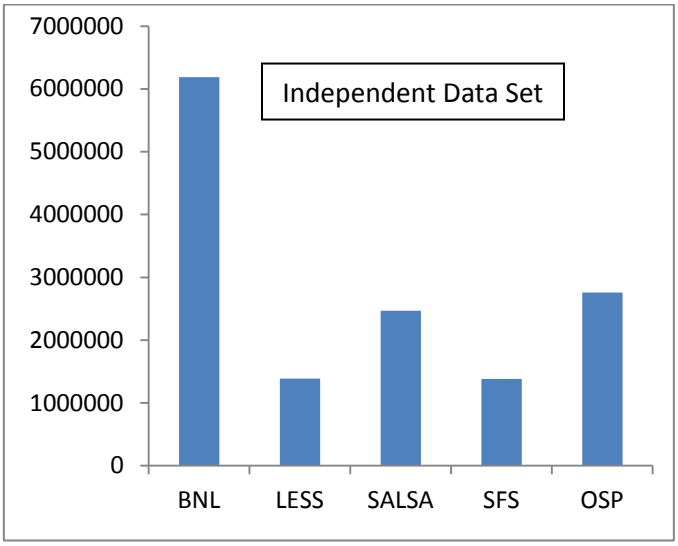
- **Μέτρηση Χρόνου Εκτέλεσης**

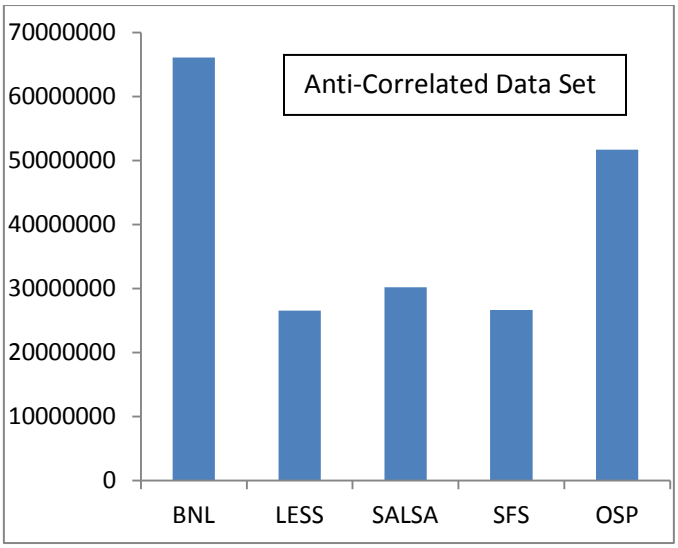


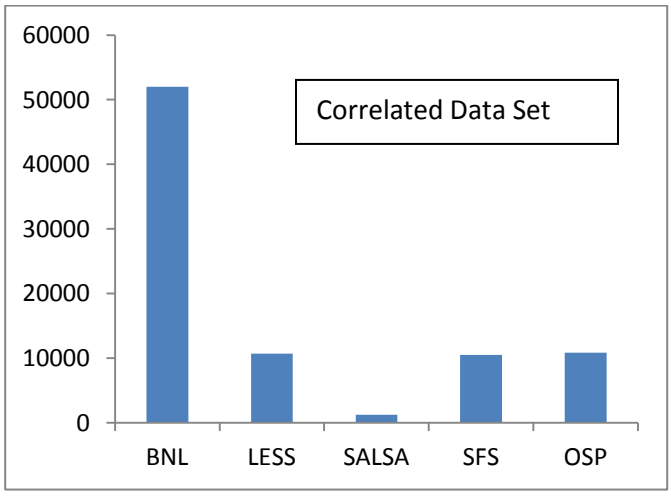




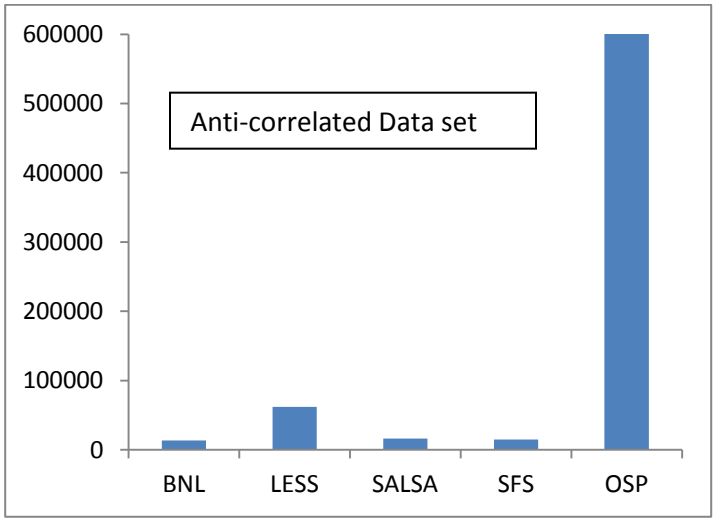
- **Μέτρηση Αριθμού συγκρίσεων**

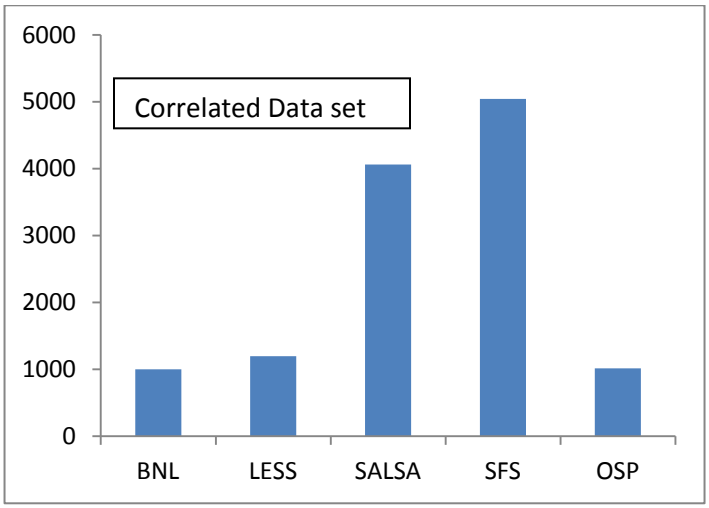


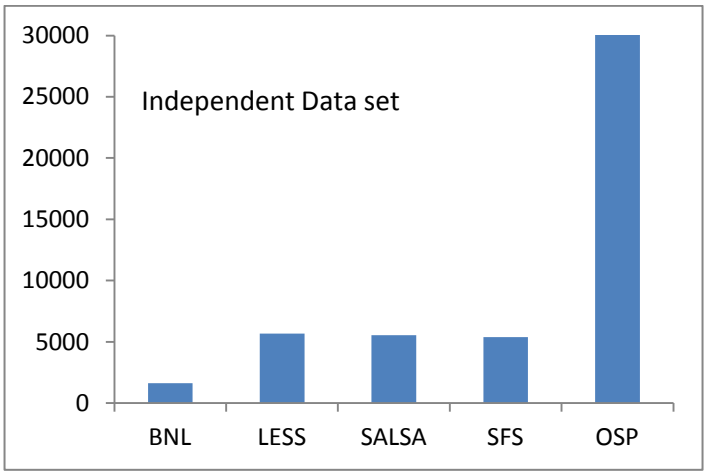




- **Μέτρηση Αριθμού I/O**







7 *Τεχνικές Λεπτομέρειες*

Στο κεφάλαιο αυτό θα παρουσιαστούν οι λεπτομέρειες σχετικά με την υλοποίηση των αλγορίθμων OSP και SALSA σε περίπτωση περιορισμένης μνήμης.

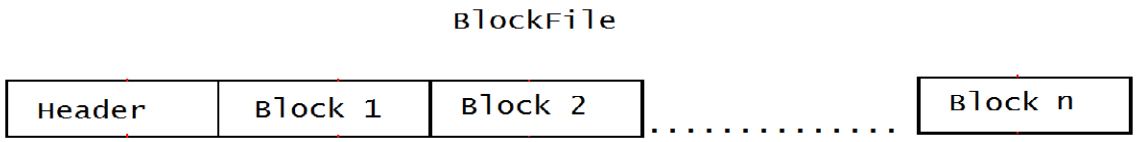
7.1 Λεπτομέρειες Υλοποίησης

Επειδή στους αλγόριθμους που υλοποιήθηκαν, λήφθηκε υπόψη και το σενάριο που τα σύνολα των δεδομένων δε χωράνε ολόκληρα στην εσωτερική μνήμη του υπολογιστή και πρέπει άρα να γίνει χρήση εξωτερικής μνήμης, φαινόμενο αρκετά συχνό για μεγάλες βάσεις πολυδιάστατων δεδομένων, τα σημεία αποθηκεύονται σε αρχεία μπλοκ (blockfiles).

Στα μπλοκ αρχεία τα δεδομένα είναι αποθηκευμένα σε μπλοκ δεδομένων. Κάθε μπλοκ έχει μέγεθος 1024 bytes και περιέχει 10 σημεία. Κάθε αρχείο έχει και μια «επικεφαλίδα» (header), η οποία έχει μέγεθος 16 bytes και περιέχει πληροφορίες σχετικά με το blockfile. Συγκεκριμένα, περιέχει το μέγεθος του μπλοκ (blocksize), τον αριθμό των μπλοκ του αρχείου (number), τη διαστασιμότητα-dimensionality των σημείων που γράφονται στο μπλοκ αρχείο (dim), και την πληθικότητα των σημείων (cardinality).

Ο υπολογιστής περιλαμβάνει μια εσωτερική μνήμη που μπορεί να διατηρήσει μέχρι memsize σημεία και μια απεριόριστη εξωτερική μνήμη. Σε κάθε input/output διεργασία (I/O) το σύστημα μπορεί να μεταφέρει ένα συνεχόμενο μπλοκ του αρχείου αποτελούμενο από 10 σημεία (ή λιγότερα, αν πρόκειται για το τελευταίο μπλοκ του αρχείου).

Η διάταξη των μπλοκ αρχείων φαίνεται στο παρακάτω σχήμα.



Για να υλοποιηθούν λοιπόν, οι αλγόριθμοι, φέρνουμε στη μνήμη ένα μπλοκ δεδομένων κάθε φορά με τη βοήθεια της μεθόδου **readPointsFromBlkFile**, η οποία δηλώνεται ως εξής:

void **readPointsFromBlkFile**(vector<**float***>&points,**int** dimension,**int** numOfBlkToRead,**int** blkSize,**int** pointsPerBlock,myBlockFile *file,**int** &readedBlockCount);

Η readPointsFromBlkFile δέχεται σαν ορίσματα τον buffer που θα αποθηκευτούν τα δεδομένα (vector<**float***>&points), τη διαστάσιμότητα των σημείων (**int** dimension), το συνολικό αριθμό των μπλοκ που περιέχει το αρχείο (**int** numOfBlkToRead), το μέγεθος του μπλοκ (**int** blkSize), τον αριθμό των σημείων που χωράνε σε κάθε μπλοκ (**int** pointsPerBlock), το μπλοκ αρχείο που διαβάζουμε (myBlockFile *file), και το μπλοκ που θα διαβαστεί (**int** &readedBlockCount).

Έχοντας υπόψη μας το συνολικό αριθμό των μπλοκ, numOfBlkToRead, που είναι γραμμένα στο αρχείο και χρειάζεται να διαβαστούν, τα σημεία φορτώνονται ανά μπλοκ στη μνήμη με τη βοήθεια της εντολής file->read_curBlock (tmpBlock). Στην περίπτωση που πρέπει να διαβαστεί το τελευταίο μπλοκ του αρχείου, πρέπει να λάβουμε υπόψη μας και την περίπτωση που το μπλοκ αυτό δεν είναι γεμάτο, έτσι ώστε να μη διαβαστούν οι κενές του θέσεις. Αυτό γίνεται με τη βοήθεια της εντολής file->emptyPoints(), που επιστρέφει τον αριθμό των κενών σημείωντου μπλοκ (**int** emptyBlkPoints). Αφού βρεθεί, λοιπόν ο αριθμός αυτός, διαγράφουμε τις θέσεις αυτές από το μπλοκ για να μην περαστούν έτσι κατά λάθος στον buffer points.

Κάθε φορά που διαβάζεται ένα μπλοκ δεδομένων η μεταβλητή readedBlockCount αυξάνεται κατά 1, μετρώντας έτσι πόσα μπλοκ έχουν διαβαστεί μέχρι στιγμής. Η μεταβλητή αυτή εξυπηρετεί λοιπόν, στο να κάνουμε καλύτερο έλεγχο των input που γίνονται στο δίσκο.

Παράλληλα, με την κλήση της blockToList(points, tmpBlock, blkSize, dimension), τα σημεία του μπλοκ γράφονται στον buffer points. Η blockToList δηλώνεται ως εξής:

void **blockToList**(vector<**float** *>&points, **char** *block, **int** blocklength, **int** dim);

Τα ορίσματα της περιλαμβάνουν τον vector points, όπου θα αποθηκευτούν τα σημεία, το εκάστοτε μπλοκ, tmpBlock, το μέγεθος του κάθε μπλοκ, blkSize, και τέλος τη διάσταση των σημείων, dimension.

Αντίθετα, όταν χρειάζεται να γράψουμε σημεία σε ένα προσωρινό μπλοκ αρχείο για να μπορέσουμε να τα επεξεργαστούμε αργότερα, όταν τα σημεία που φορτώνουμε στην κύρια μνήμη γίνουν ίσα με τον αριθμό των σημείων που χωράνε σε ένα μπλοκ, καλούμε τη συνάρτηση **writePointsToBlkFile**, η οποία δηλώνεται ως εξής:

void writePointsToBlkFile(vector<**float***> points, **int** dimension, **int** blkSize, **int** pointsPerBlock, myBlockFile *file);

Η συνάρτηση αυτή παίρνει ως ορίσματα τον buffer, όπου είναι αποθηκευμένα τα σημεία που θέλουμε να γράψουμε στο μπλοκ αρχείο (vector<**float***>points), τη διαστασιμότητα των σημείων που γράφονται στο αρχείο(**int** dimension), το μέγεθος του μπλοκ (**int** blkSize), τον αριθμό των σημείων που χωράνε σε κάθε μπλοκ (**int** pointsPerBlock), και το όνομα του μπλοκ αρχείου στο οποίο θα αποθηκευτούν τα σημεία (myBlockFile *file).

Τα σημεία διαβάζονται ένα-ένα και αντιγράφονται στο μπλοκ του myBlockFile, με τη βοήθεια της εντολής file->append_block(tmpFileBlock). Ταυτόχρονα, αφού γραφτεί το μπλοκ στο αρχείο,τα περιεχόμενα του buffer points διαγράφονται, ώστε να μπορούν να αποθηκευτούν εκεί τα επόμενα χωρίς να δημιουργηθεί πρόβλημα με τη μνήμη που έχουμε στη διάθεση μας. Εξασφαλίζουμε δηλαδή έτσι οτι ποτέ δε θα βρίσκονται φορτωμένα στη μνήμη πάνω από memory size σημεία.

Κατά την εγγραφή των σημείων στο προσωρινό μπλοκ αρχείο, λαμβάνουμε υπόψη μας και την περίπτωση που θα πρέπει να γραφτεί το τελευταίο μπλοκ του αρχείου και τα σημεία είναι λιγότερα από το pointsPerBlock, δηλαδή από τον αριθμό των σημείων που έχει καθοριστεί εξαρχής ότι πρέπει να γράφονται σε κάθε μπλοκ (δηλαδή 10). Μετά την εγγραφή του μπλοκ η εντολή:

file->fwrite_Cardinality(file->cardinality + pointToWrite) ανανεώνει το πλήθος των σημείων που περιέχει το αρχείο (file->cardinality), έτσι ώστε να υπάρχει διαθέσιμο και αυτό στο header του αρχείου.

Για την εύρεση του skyline, οι αλγόριθμοι λαμβάνουν υπόψη την περίπτωση του dynamic skyline και την περίπτωση του orthant skyline.

Οι αλγόριθμοι SALSA και OSP αρχικοποιούνται ως:

SaLSa *salsa = **new** SaLSa(inputFile, queries[q], dominanceType, memSize, readBufferBlkNum, writeBufferBlkNum, sortFunc);

και

OSP *osp = **new** OSP(inputFile, queries[q], dominanceType, memSize, readBufferBlkNum, writeBufferBlkNum);

αντίστοιχα.

Ως inputFile ορίζεται το μπλοκ αρχείο που περιέχει το αρχικό σύνολο δεδομένων μας και ως queries[q], ο πίνακας που περιέχει τα διαφορετικά σημεία που θα αποτελέσουν την αρχή των αξόνων μας με βάση την οποία θα υπολογιστούν τα orthant και dynamic skyline. Η μεταβλητή dominanceType μας δίνει τη δυνατότητα να επιλέξουμε αν το skyline που θα υπολογιστεί θα είναι orthant ή dynamic και με τη μεταβλητή memSize δηλώνουμε το μέγεθος της μνήμης μας σε αριθμό σημείων. Οι μεταβλητές readBufferBlkNum και writeBufferBlkNum μας δίνουν τη δυνατότητα να επιλέξουμε πόσα μπλοκ θέλουμε να διαβαστούν ή να γραφτούν στα μπλοκ αρχεία αντίστοιχα. Τέλος, στον αλγόριθμο SALSA μπορούμε να προσδιορίσουμε με χρήση της μεταβλητής sortFunc, με ποια συνάρτηση θέλουμε να ταξινομηθούν τα δεδομένα μας.

7.1.1 Αλγόριθμος SALSA

Για να υλοποιηθεί ο αλγόριθμος SALSA, επειδή απαιτεί στην είσοδο του ένα ταξινομημένο σύνολο δεδομένων, χρησιμοποιήθηκε η τεχνική του external sorting, η οποία περιλαμβάνει δύο στάδια.

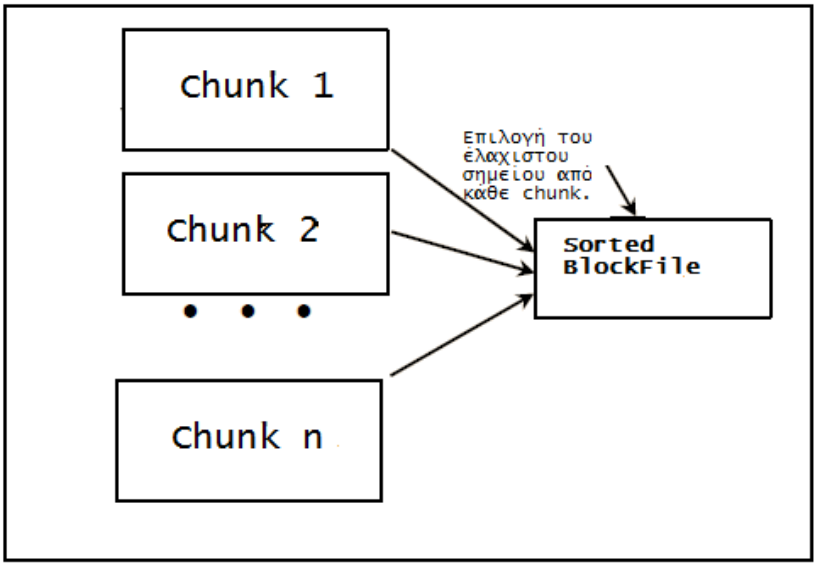
Ο όρος external sorting χρησιμοποιείται, καθώς μελετάμε και την περίπτωση που τα δεδομένα που θα ταξινομηθούν δε χωρούν στην κύρια μνήμη και έτσι πρέπει να βρίσκονται αποθηκευμένα σε κάποια εξωτερική μνήμη, που στην περίπτωση μας είναι τα μπλοκ αρχεία στο δίσκο.

Στο πρώτο στάδιο του external sorting, που είναι και το στάδιο της ταξινόμησης, (σωροί) chunks δεδομένων αρκετά μικροί ώστε να χωρούν στη μνήμη διαβάζονται, ταξινομούνται και γράφονται σε ένα προσωρινό αρχείο. Για το σκοπό αυτό χρησιμοποιείται η μέθοδος **sortChunks()**. Τα σημεία ταξινομούνται με βάση τη μικρότερη τιμή κάθε διάστασης τους, δηλαδή με τη βοήθεια της minC συνάρτησης που παρουσιάστηκε στο Κεφάλαιο 5.

Στο δεύτερο στάδιο του external sorting, που είναι και το στάδιο της συγχώνευσης, τα ταξινομημένα υπό-κομμάτια του αρχείου συνδυάζονται σε ένα μεγαλύτερο ταξινομημένο μπλοκ αρχείο. Για το σκοπό αυτό χρησιμοποιείται η μέθοδος **merge()**.

Ο αριθμός των μπλοκ που θα αποθηκευτούν ανά chunk (numOfblocksPerChunk) είναι ίσος με το (μέγεθος της μνήμης)/(τον αριθμό των σημείων που διαβάζουμε κάθε φορά από το αρχικό μπλοκ αρχείο,δηλ.10). Ο αριθμός των σημείων που γράφονται σε κάθε chunk είναι ίσος με: (numOfblocksPerChunk)*(τον αριθμό των σημείων που χωράνε σε κάθε μπλοκ, δηλ.10).

Η παραπάνω διαδικασία περιγράφεται και στο παρακάτω σχήμα.



Για να μετρήσουμε τον αριθμό των προσβάσεων στο δίσκο στην είσοδο/έξοδο (I/O) χρησιμοποιούμε τις μεθόδους countIn και countOut, οι οποίες δηλώνονται ως εξής:

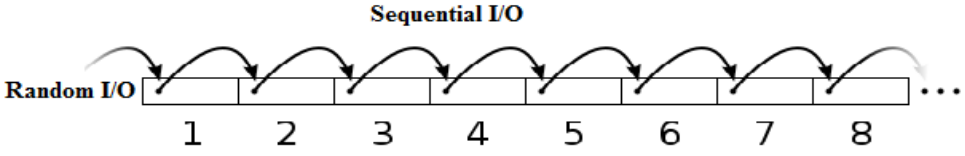
void SaLSa::countIn(**int** readedBlocks, **int** phase);

και

void SaLSa::countOut(**int** writtenBlocks, **int** phase);

αντίστοιχα. Η countIn, παίρνει σαν ορίσματα τον αριθμό των μπλοκ που έχουν διαβαστεί, readedBlocks, και τη φάση, phase, στην οποία βρίσκεται ο αλγόριθμος. Ο αλγόριθμος χωρίζεται σε δύο φάσεις, την εξωτερική που περιλαμβάνει το external sorting και το merge στάδιο, και την εσωτερική φάση που περιλαμβάνει το κύριο μέρος του αλγόριθμου SALSA. Απ΄την άλλη η countOut παίρνει σαν ορίσματα τον αριθμό των μπλοκ που έχουν γραφτεί, writtenBlocks, και τη φάση, phase, στην οποία βρίσκεται ο αλγόριθμος.

Κατά τις προσβάσεις στο δίσκο, το πρώτο μπλοκ που διαβάζεται ή γράφεται επιλέγεται πάντα τυχαία (InRand, OutRand), ενώ όλα τα υπόλοιπα από εκεί και πέρα σειριακά (InSeq, OutSeq). Η σειριακή πρόσβαση είναι προτιμότερη καθώς είναι ταχύτερη και κάνει πιο αποδοτική χρήση του δίσκου.



Μετά το στάδιο αυτό ο αλγόριθμος υλοποιείται όπως ο Αλγόριθμος 2, που περιγράφηκε στο Κεφάλαιο 5.

Αρχικά εξετάζουμε αν αληθεύει η συνθήκη τερματισμού (stop).

Αν όχι, τα σημεία διαβάζονται από το ταξινομημένο αρχείο ανά μπλοκ με την εντολή readPointsFromBlkFile. Αν πρόκειται για την πρώτη εκτέλεση του αλγόριθμου σαν stop point επιλέγεται το πρώτο σημείο που διαβάζουμε.

Η συνθήκη τερματισμού (stop condition), ανανεώνεται ως εξής:

pntmax(stopoint.data,query,dimension)<=pntMinDim(inBuffer[i].data,query,dimension) && (samePoints(stopoint.data,inBuffer[i].data,dimension) = =false).

Η συνάρτηση **pntmax** λαμβάνοντας υπόψη τις τιμές του σημείου σε όλες του τις διαστάσεις επιστρέφει τη μέγιστη από αυτές. Ενώ, η **pntMinDim** επιστρέφει τη μικρότερη από αυτές. Η **samePoints**, παίρνοντας στην είσοδο της, 2 σημεία, και ελέγχει αν συμπίπτουν.

Αν η παραπάνω συνθήκη ισχύει ο αλγόριθμος τερματίζεται και έχουμε στη διάθεση μας το τελικό skyline, αλλιώς εξετάζουμε αν το σημείο κυριαρχείται ή όχι από τα σημεία που έχουν ήδη τοποθετηθεί στη skyline λίστα (cSL). Αν το σημείο δεν κυριαρχείται από κανένα σημείο της cSL και η cSL χωράει στη μνήμη, μπαίνει και αυτό στη cSL και το stop point ανανέωνεται.

Η ανανέωση του stop point γίνεται με την ακόλουθη εντολή:

```
if(pntmax(inBuffer[i].data,query,dimension)<pntmax(stopoint.data,query,dimens
ion)){

  stopoint=inBuffer[i];

}
```

, η οποία ελέγχει αν το καινούριο σημείο που θα μπει στο skyline έχει pntmax μικρότερο από το pntmax του σημείου που θεωρείται μέχρι στιγμής το stop point.

Αν διαπιστώσουμε όμως οτι η cSL δε χωράει στη μνήμη, το σημείο γράφεται στο προσωρινό αρχείο, το οποίο θα αποτελέσει την είσοδο στο επόμενο πέρασμα του αλγόριθμου μας. Οι έλεγχοι κυριαρχίας γίνονται με τη βοήθεια της εντολής: isSkylineSorted(inBuffer[i].data, cSL, query, dominanceType, dimension, compCnt). Η isSkylineSorted δηλώνεται ως εξής:

bool isSkylineSorted(**float** *pnt, vector<cSLPoint> cSL, **float** *query, **int** dominanceType, **int** dimension, **long long** &comparCnt);

Η isSkylineSorted ελέγχει αν το τρέχον σημείο inBuffer[i].data, κυριαρχείται ή όχι από κάποιο από τα σημεία της skyline λίστας cSL. Αν κυριαρχείται, τότε διαγράφεται. Για να γίνουν οι έλεγχοι κυριαρχίας η isSkylineSorted χρησιμοποιεί στην περίπτωση που υπολογίζουμε δυναμικό skyline, τη συνάρτηση PdyndomP, αλλιώς, αν υπολογίζουμε orthant skyline, χρησιμοποιεί την PorthdomP.

Η PdyndomP δηλώνεται ως εξής:

bool PdyndomP (**float** *pt1, **float** *pt2, **float** *query, **int** dimension);

και παίρνει σαν ορίσματα δύο σημεία (**float** *pt1, **float** *pt2), το σημείο query με βάση το οποίο θα υπολογιστεί το skyline (**float** *query) και τη διάσταση των σημείων (**int** dimension).

Η PorthdomP δηλώνεται ως εξής:
bool PorthdomP(**float** *pt1, **float** *pt2, **float** *query, **int** dimension);
και παίρνει σαν ορίσματα δύο σημεία (**float** *pt1, **float** *pt2), το σημείο query με βάση το οποίο θα υπολογιστεί το skyline (**float** *query) και τη διάσταση των σημείων (**int** dimension).

7.1.2 Αλγόριθμος OSP

Ο αλγόριθμος αυτός υλοποιήθηκε με τη βοήθεια του Αλγορίθμου 5, που παρουσιάστηκε στο Κεφάλαιο 4, και λαμβάνει υπόψη του την περίπτωση που το σύνολο των δεδομένων μας είναι αρκετά μεγάλο για να χωρέσει στην κύρια μνήμη.

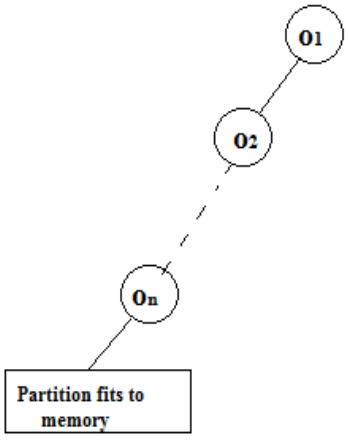
Αρχικά, λαμβάνοντας στην είσοδο το μπλοκ αρχείο, όπου είναι αποθηκεύμενα τα σημεία μας ο αλγόριθμος ελέγχει αν η πληθικότητα των σημείων (cardinality) υπερβαίνει το μέγεθος της μνήμης (memSize). Στην περίπτωση αυτή, ο αλγόριθμος κάνοντας ελέγχους κυριαρχίας (dominance checks) βρίσκει ένα skyline σημείο και με βάση αυτό απορρίπτει/διαγράφει όσα σημεία κυριαρχούνται από αυτό. Όσα σημεία δεν κυριαρχούνται, θεωρούνται ως ένα subpartition του συνόλου δεδομένων και σώζονται σε ένα μπλοκ αρχείο με τη βοήθεια της μεθόδου που περιγράφηκε παραπάνω. Το προσωρινό αυτό μπλοκ αρχείο θα αποτελέσει την είσοδο για το επόμενο πέρασμα του αλγόριθμου. Οι έλεγχοι κυριαρχίας γίνονται με τη βοήθεια της συνάρτησης PAddress, η δήλωση της οποίας είναι η εξής:

unsigned long PAddress(cSLPoint* base, cSLPoint* Obj,**float*** query,**int** dimension, **int** dominanceType);

Η PAddress παίρνει σαν ορίσματα το σημείο με βάση το όποιο θα συγκρίνονται τα υπόλοιπα (cSLPoint* base), το σημείο του οποίου θέλουμε να υπολογίσουμε την partition address (cSLPoint* Obj), το σημείο query με βάση το οποίο γίνονται οι έλεγχοι κυριαρχίας (**float*** query=NULL), τη διαστασιμότητα των σημείων (**int** dimension), και το είδος του skyline που υπολογίζεται, δηλ. δυναμικό ή ορθοκανονικό (**int** dominanceType).

Τα σημεία των οποίων το partition address σε σχέση με το skyline σημείο, base, είναι 0- 2^d -2, είναι πιθανόν να ανήκουν στο skyline, ενώ τα σημεία των οποίων το partition address είναι ίσο με τη μέγιστη τιμή που μπορεί να πάρει η partition διεύθυνση (MAXDMNT), δηλαδή 2^d , διαγράφονται.

Θα μπορούσαμε να παρομοιάσουμε το LCRS δέντρο που σχηματίζεται κατά την διάρκεια εκτέλεσης του αλγόριθμου μέχρι τη στιγμή που τα δεδομένα θα χωρέσουν στη μνήμη με αυτό του παρακάτω σχήματος.



Όταν ο αριθμός των σημείων, είτε του προσωρινού μας μπλοκ αρχείου, είτε του αρχικού blockFile, μπορεί να χωρέσει στην κύρια μνήμη ο αλγόριθμος ακολουθεί τα όσα περιγράφηκαν στον Αλγόριθμο 5 του Κεφαλαίου 4.

Η κλήση του αλγόριθμου αυτού γίνεται με την εντολή: PPSSP (SRC,query,dimension,dominanceType)

Η συνάρτηση PPSSP δηλώνεται ως εξής:

int PPSSP(PSUBSTRUCT *&pnode,**float*** query,**int** dimension,**int** dominanceType);

Η PPSSP λαμβάνει ως είσοδο τα εξής ορισματα: το τρέχον partition που θα επεξεργαστούμε (PSUBSTRUCT *&pnode), το query, το οποίο θα αποτελέσει την αρχή των αξόνων για τον υπολογισμό

του skyline (**float*** query), τη διαστασιμότητα των σημείων (**int** dimension), και τον τύπο του skyline που θα υπολογίσουμε, dynamic ή orthant (**int** dominanceType).

Το partition tree σχηματίζεται με την βοήθεια του structure PSUBSTRUCT και του αντίστοιχου constructor του, που δηλώνεται ως:

PSUBSTRUCT(**unsigned long** padd, cSLPoint** pObj, PSUBSTRUCT *prt=NULL, PSUBSTRUCT *sb=NULL, PSUBSTRUCT *cld=NULL, **unsignedlong** s_len=1, **bool** inSky=**false**);

και παίρνει σαν ορίσματα την διεύθυνση του εκάστοτε partition (**unsigned long** padd), η οποία μπορεί να πάρει τιμές: από 0 έως 2^d -1, και την οποία αρχικά την ορίζουμε ως 0, έναν πίνακα με τα σημεία που ανήκουν στο συγκεκριμένο partition (cSLPoint** pObj), το partition γονέα, το γειτονικό partition και το partition παιδί (PSUBSTRUCT *prt=NULL, PSUBSTRUCT *sb=NULL και PSUBSTRUCT *cld=NULL αντίστοιχα), τον αριθμό των σημείων που περιέχει το συγκεκριμένο partition (**unsignedlong** s_len=1) και μια μεταβλητή που δείχνει αν το συγκεκριμένο partition είναι κομμάτι του skyline ή όχι.

Ο κάθε κόμβος του skyline partition tree σχηματίζεται με τη βοήθεια του structure SNODE και του αντίστοιχου constructor του, που δηλώνεται ως:

SNODE(cSLPoint * pobj, **unsigned long** padd, SNODE * sb=NULL, SNODE * chld=NULL, **bool** inSky=**true**);

και παίρνει σαν ορίσματα το σημείο από το οποίο θα αποτελείται ο κόμβος (cSLPoint *pobj), τη διεύθυνση του partition που αντιπροσωπεύεται από αυτό το σημείο (**unsigned long** padd), ένα δείκτη στο δεξί γειτονικό partition και ένα δείκτη στο πιο αριστερό υπό-partition (SNODE * sb=NULL και SNODE * chld=NULL αντίστοιχα), καθώς και μια μεταβλητή που δείχνει αν το σημείο του κόμβου είναι skyline σημείο ή όχι (**bool** inSky=**true**).

Για τους ελέγχους κυριαρχίας χρησιμοποιείται η μέθοδος Dominate του SNODE, που αποτελεί υλοποίηση του Αλγόριθμου 1 που περιγράφηκε στο Κεφάλαιο 5.

Η δήλωση της είναι η εξής:

bool Dominate(cSLPoint * Obj, **unsignedlong** padd, **bool** isLocated=**false**,**float*** query=NULL,**int** dimension=0,**int** dominanceType=0);

Η μέθοδος αυτή παίρνει ως ορίσματα ένα σημείο (cSLPoint * pobj), τη διεύθυνση του partition που βρίσκεται το σημείο (**unsigned long** padd), μια μεταβλητή που μας δείχνει αν το SNODE βρίσκεται στο

partitionτου σημείου pobj (**bool** isLocated=**false**), το σημείο query με βάση το οποίο γίνονται οι έλεγχοι κυριαρχίας (**float*** query=NULL), τη διαστασιμότητα των σημείων (**int** dimension=0), και το είδος του skyline που υπολογίζεται, δηλ. δυναμικό ή ορθοκανονικό (**int** dominanceType=0).

Για το φιλτράρισμα των γειτονικών partition χρησιμοποιείται η συνάρτηση filter, που αποτελεί υλοποίηση του Αλγόριθμου 4 που περιγράφεται στο Κεφάλαιο 5. Η δήλωση της είναι:

void filter(PSUBSTRUCT *&node,SNODE *SL,**float** *query,**int** dimension,**int** dominanceType);

Η filter παίρνει σαν ορίσματα ένα partition (PSUBSTRUCT *&node), ένα skyline δέντρο που είναι γειτονικό του node (SNODE *SL), το σημείο query που λαμβάνεται ως αρχή των αξόνων για να υπολογιστεί το skyline (**float*** query=NULL), τη διαστασιμότητα των σημείων (**int** dimension),και το είδος του skyline που υπολογίζεται, δηλ. δυναμικό ή ορθοκανονικό (**int** dominanceType).

Ο ρόλος της filter είναι να φιλτράρει το partition node με βάση το skyline δέντρο SL, απομακρύνοντας όσα σημεία του node, αλλά και των γειτονικών του partition κυριαρχούνται από το SL.

Στην υλοποίηση της συνάρτησης PPSSP, χρησιμοποιείται η συνάρτηση OSPSOnSortingFirst, που αποτελεί υλοποίηση του αλγόριθμου 2 που παρουσιάστηκε στο κεφάλαιο 5. Η συνάρτηση που επιλέγουμε για την ταξινόμηση των στοιχείων είναι η sum, δηλαδή τα σημεία ταξινομούνται με βάση το άθροισμα όλων των συντεταγμένων του κάθε σημείου. Επίσης, ως πρώτο skyline σημείο επιλέγεται ένα τυχαίο. Θα μπορούσε ως παραλλαγή του αλγόριθμου να χρησιμοποιήσουμε ως αρχικό skyline σημείο και το πρώτο σημείο του partition.

Η δήλωση της συνάρτησης αυτής είναι η:

SNODE ***OSPSOnSortingFirst**(PSUBSTRUCT *&pnode,**float*** query,**int** dimension,**int** dominanceType);

Η OSPSOnSortingFirst παίρνει ως ορίσματα ένα partition (PSUBSTRUCT *&node), το σημείο query που λαμβάνεται ως αρχή των αξόνων για να υπολογιστεί το skyline (**float*** query=NULL), τη διαστασιμότητα των σημείων (**int** dimension), και το είδος του skylineπου υπολογίζεται, δηλ. δυναμικό ή ορθοκανονικό (**int** dominanceType). Με βάση το πρώτο skyline σημείο, firstRef, η συνάρτηση διατρέχει το partition pnode και μετά από έλεγχους κυριαρχίας βγάζει στην έξοδο το τελικό skyline δέντρο.

7.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Το υποσύστημα έχει αναπτυχθεί με τη γλώσσα προγραμματισμού ANSI C++ και τρέχει σε πλατφόρμα Microsoft Windows, αλλά και Linux. Η επιλογή τόσο της γλώσσας προγραμματισμού, της γενικότερης πλατφόρμας αλλά και του λειτουργικού συστήματος έγινε καταρχάς με κριτήριο την ταχύτερη απόδοση του συστήματος αλλά και τη στο μέτρο του δυνατού απλούστερη υλοποίηση, ενώ σημαντικό ρόλο έπαιξε και το γεγονός ότι όλα τα εργαλεία που χρησιμοποιήθηκαν ανήκουν στην κατηγορία του ελεύθερου λογισμικού.

Για την ανάπτυξη χρησιμοποιήθηκε ο μεταγλωτιστής Cygwin gcc, ενώ ως περιβάλλον ανάπτυξης χρησιμοποιήθηκε το Eclipse CDT. Σε όλο τον κώδικα έγινε χρήση της στάνταρντ βιβλιοθήκης προτύπων της C++ (STL) και των βιβλιοθηκών Boost.

8 *Επίλογος*

Παρακάτω ακολουθεί μια σύνοψη της παρούσας διπλωματικής εργασίας.

8.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική υλοποιήθηκαν δύο αλγόριθμοι για την εύρεση της κορυφογραμμής (skyline), σε πολυδιάστασταστα σύνολα δεδομένων (σημείων). Το skyline εξυπηρετεί στην εύρεση εκείνων των σημείων που είναι καλύτερα σε σχέση με τα υπόλοιπα, σε τουλάχιστον μια διάσταση, συνεισφέροντας έτσι στην επίλυση του προβλήματος για λήψη απόφασης με πολλαπλά κριτήρια.

Βασιζόμενοι σε προηγούμενες εργασίες, υλοποιήσαμε του αλγόριθμους: Sort and Limit Skyline Algorithm (SALSA) και Object Space Partitioning algorithm (OSP), λαμβάνοντας υπόψη το πρόβλημα που προκύπτει στην περίπτωση που το σύνολο δεδομένων μας είναι αρκετά μεγάλο για να χωρέσει στην

κύρια μνήμη του συστήματος. Σκοπός μας ήταν οι αλγόριθμοι να είναι όσο αποδοτικοί και όσο ταχύτεροι γίνεται, καθώς το πλήθος των δεδομένων και η διάσταση τους αυξάνεται.

Οι δύο αυτοί αλγόριθμοι, μετά από πειραματικές μετρήσεις συγκρίθηκαν με τρεις ακόμα που παρουσιάζονται στη βιβλιογραφία, τον Block Nested Loops Alogrithm (BNL), τον Sort First Skyline Algorithm (SFS), και τον Linear Elimination Sort for Skyline algorithm (LESS). Οι πειραματικές μετρήσεις που λάβαμε ήταν (i) ο αριθμός των συγκρίσεων που πραγματοποιεί ο κάθε αλγόριθμος, (ii) ο χρόνος εκτέλεσης των αλγόριθμων και (iii) το σύνολο των προσβάσεων εισόδου/εξόδου. Το συμπέρασμα των μετρήσεων αυτών ήταν οτι ο SALSA είναι αρκετά αποδοτικός, σε όλα τα σύνολα δεδομένων, αλλά παρολαυτά δεν έχει σημαντικό πλεονέκτημα έναντι των SFS και LESS, που χρησιμοποιούν και αυτοί την τεχνική της τοπολογικής ταξινόμησης των σημείων. Στα περισσότερα πειράματα η συμπεριφορά του SALSA είναι παρεμφερής με αυτή του SFS. Από την άλλη ο OSP, στην περίπτωση που το σύνολο των δεδομένων μας χωράει ακέραιο στην κύρια μνήμη είναι αρκετές τάξεις καλύτερος από όλους τους υπόλοιπους, καθώς ελαχιστοποιεί τον αριθμό των απαιτούμενων συγκρίσεων μεταξύ των δεδομένων, χωρίζοντας το χώρο σε κομμάτια, τα οποία οργανώνει με τη βοήθεια ένος δέντρου. Στην περίπτωση όμως, που η μνήμη του συστήματος μας είναι περιορισμένη η υλοποίηση του αλγόριθμου ακολουθεί μια διαφορετική διαδικασία που δεν του επιτρέπει να αξιοποιήσει το βασικό του χαραχτηριστικό στο μέγιστο, με αποτέλεσμα να γίνεται αρκετά αργός και καθόλου αποδοτικός στις περιπτώσεις που εξετάζουμε το πρόβλημα της cardinality (συνολικού αριθμού σημείων). Είδαμε επίσης, οτι ο OSP επηρεάζεται αρκετά από την κατανομή των δεδομένων, αλλά και από τον αριθμό των διαστάσεων (dimensionality).

8.2 Μελλοντικές επεκτάσεις

Παρακάτω παρουσιάζουμε τυχόν επεκτάσεις που θα μπορούσαν να γίνουν στα πλαίσια της παρούσας διπλωματικής.

Αρκετό ενδιαφέρον θα παρουσιάζε, η λήψη πειραματικών μετρήσεων σε σύνολα πραγματικών δεδομένων και η συγκριτική μελέτη των αποτελεσμάτων με εκείνα που λάβαμε στα σύνολα συνθετικών δεδομένων. Θα μπορούσαμε επίσης, εκτός από τα σύνολα δεδομένων που χρησιμοποιήθηκαν (δηλαδή anti-correlated, correlated και independent), να χρησιμοποιήσουμε ανάμεικτα σύνολα δεδομένων, όπου τα σημεία αλλού κατανέμονται στο χώρο ως anti-correlated και αλλού ως independent.

Θα μπορούσαμε ακόμα να συμπεριλάβουμε στα πειράματα μας και index-based αλγόριθμους, όπως για παράδειγμα ο ZSearch [LZL+07], και να μελετήσουμε έτσι τη συμπεριφορά τους σε σχέση με τον OSP και SALSA.

9 *Βιβλιογραφία*

[BKS01] S. Borzsonyi, D. Kossmann, K. Stocker. The Skyline Operator, 2001.

[BCP08] I.Bartolini, P. Ciaccia, M. Patella. Efficient Sort-Based Skyline Evaluation, 2008.

[C03] J. Chomicki. Preference Formulas in Skyline Queries, 2003.

[CGG+02] J. Chomicki, P. Godfrey, J. Gryz, D. Liang. Skyline with Presorting, 2002.

[G04] P. Godfrey. Skyline Cardinality for Relational Processing, 2004.

[GSG06] P.Godfrey,R.Shirpley,J.Gryz. Algorithms and Analysis for Maximal Vector Computation. VLDB Journal 2006.

[KB11] M. Kokkidis, P. Bhattacharya. Implementation of Skyline Query Algorithms, 2011.

[KLP75] H.T Kung, F.Luccio, F.P. Preparata. On Finding the Maxima of a Set of Vectors,1975.

[KRR02] D.Kossman, F.Ramsak, S.Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, 2002.

[LZL+07] K.C.K. Lee, B.Zheng, H. Li, W.Lee. Approaching the Skyline in Z Order.

[MPJ07] M. Morse, J.M. Patel, H.V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains.

[PTF+03] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD), pages 467–478, 2003.

[PTF+05] D. Papadias, Y. Tao, G.Fu, and B. Seeger. Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS), 30(1):41–82, 2005.

[SBS08] D. Sacharidis, P. Bouros, T. Sellis.Caching Dynamic Skyline Queries, 2008.

[TEO01] K. Tan, P. Eng, B. C. Ooi. Efficient progressive skyline computation. In Proceedings of 27th International Conference on Very Large Data Bases (VLDB), pages 301–310, 2001.

[ZMC09] S. Zhang, N. Mamoulis, D.W. Cheung. Scalable Skyline Computation Using Object-Based Space Partitioning, 2009.