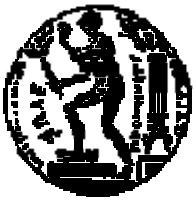


ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αποδοτικά ευρετήρια για ερωτήματα ομοιότητας σε τυχαίους υποχώρους πολυδιάστατων δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΕΤΡΟΥ Ι. ΒΕΝΕΤΗ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2007

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

Αποδοτικά ευρετήρια για ερωτήματα ομοιότητας σε τυχαίους υποχώρους πολυδιάστατων δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΕΤΡΟΥ Ι. ΒΕΝΕΤΗ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Σεπτεμβρίου 2007.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2007

.....

ΠΕΤΡΟΣ Ι. ΒΕΝΕΤΗΣ

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

Περίληψη

Ο σκοπός της διπλωματικής είναι η ανάπτυξη μίας δομής ευρετηρίων, που θα αναφέρεται σε πολυδιάστατα δεδομένα, και θα επιτρέπει την κατά το δυνατόν πιο γρήγορη απάντηση ερωτημάτων ομοιότητας σε τυχαίους υποχώρους.

Για να βρούμε αυτή τη δομή, αποφασίσαμε να κατατμήσουμε τα δεδομένα μας κάθετα και να αναθέσουμε τις διάφορες διαστάσεις τους σε μία γνωστή δομή ευρετηρίου (συγκεκριμένα είτε σε ένα R^*-δέντρο είτε σε ένα B^+-δέντρο). Μετά, αφού έχουμε τα δεδομένα μας κατατμημένα σε διάφορες υπάρχουσες δομές ευρετηρίων και χρησιμοποιώντας τον Threshold Algorithm, έναν αλγόριθμο που απαντά σε ερωτήματα που επιστρέφουν τα κορυφαία δεδομένα ως προς κάποιο κριτήριο, μπορούμε να απαντήσουμε σε ερωτήματα ομοιότητας, πραγματοποιώντας ένα συγκεκριμένο πλήθος από ανακλήσεις δεδομένων, τον οποίο και καταμετρούμε.

Σύμφωνα με αυτή τη δομή που περιγράψαμε, προσπαθήσαμε να βρούμε πως πρέπει να τοποθετηθούν τα R^*-δέντρα και τα B^+-δέντρα στις διαστάσεις των δεδομένων μας, ώστε να έχουμε τις λιγότερες δυνατές ανακλήσεις δεδομένων από το δίσκο, καθώς αυτές καθορίζουν το κόστος της απάντησης ενός ερωτήματος ομοιότητας. Έτσι, αφού πρώτα πραγματοποιήσαμε αρκετές μετρήσεις με διάφορους συνδυασμούς από R^*-δέντρα και B^+-δέντρα, εξήγαμε διάφορα συμπεράσματα και σχεδιάσαμε έναν αλγόριθμο κατανομής των διαστάσεων της βάσης μας στις προαναφερθείσες δομές ευρετηρίων, ο οποίος δίνει καλές λύσεις.

Λέξεις Κλειδιά: ερωτήματα ομοιότητας, πλησιέστεροι γείτονες, ευρετήριο, πολυδιάστατα δεδομένα, αλγόριθμος κατωφλίου.

Abstract

The scope of this thesis is the development of an index structure, for multidimensional data, which will allow proximity queries in subspaces to be answered in as little time as possible.

In order to develop this structure, we decided to fragment our data vertically and use well-known index structures (either R^*-trees or B^+-trees). Afterwards, using Threshold Algorithm, an algorithm that returns the top objects according to some criterion, we were able to answer proximity queries and count the number of data recalls that had to take place in order to answer the query.

Using the structure we just described, we tried to find how we should organize the R^*-trees and the B^+-trees so that we have as less recalls from the disk as possible, because these recalls determine the time required in order an answer to a proximity query to be given. Later, we took several measurements in many different combinations of R^*-trees and B^+-trees, we were able to make many conclusions and then to design an algorithm that will distribute the dimensions of our database in one of the index structures we just mentioned, which gives quite good solutions.

Keywords: proximity queries, nearest neighbors, index, multidimensional data, threshold algorithm.

Πίνακας περιεχομένων

Πίνακας εικόνων

1

Εισαγωγή

1.1 Εξελίξεις στο χώρο των βάσεων δεδομένων – Πολυδιάστατα δεδομένα

Τα σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (database management systems, DBMS), που έχουν τις ρίζες τους στα πρώτα συστήματα Ingres και System R, στα πρώτα στάδια της εξέλιξής τους είχαν τη δυνατότητα να εξυπηρετήσουν διαφόρων ειδών απλά ερωτήματα (queries) όπως η προβολή (projection), η επιλογή (selection) και οι σύνδεσμοι (joins) για βάσεις δεδομένων (databases, DB) που δεν είχαν πάρα πολλές διαστάσεις (dimensions), ενώ και η χρήση των ευρετηρίων (indexes) δεν ήταν εκτεταμένη. Με την πάροδο του χρόνου όμως, εμφανίστηκε η ανάγκη τα DBMS να πρέπει να διαχειρίζονται ερωτήματα αρκετά πιο περίπλοκα από τα προηγούμενα. Ένα από αυτά τα ερωτήματα είναι και το ερώτημα του πλησιέστερου γείτονα (nearest neighbor query), ενώ άλλα πολύπλοκα ερωτήματα είναι ερωτήματα αντίστροφου πλησιέστερου γείτονα (reverse nearest neighbor), διαρκή ερωτήματα σε ρεύματα δεδομένων και πολλά άλλα. Πέρα όμως από το γεγονός ότι τα ερωτήματα που άρχισαν να τίθενται στα DBMS είναι αρκετά πιο περίπλοκα σε σχέση με αυτά του παρελθόντος, και οι ίδιες οι βάσεις δεδομένων διαθέτουν σχέσεις με πάρα πολλές διαστάσεις (ή όπως αλλιώς λέγονται ιδιότητες), με αποτέλεσμα να υπάρχουν σήμερα σχέσεις με κάποιες χιλιάδες διαστάσεις (για παράδειγμα βάσεις δεδομένων που αφορούν σε αστρονομικά στοιχεία). Το γεγονός αυτό του αυξημένου αριθμού των διαστάσεων των σχέσεων, οδηγεί σε πολύ μεγάλο αριθμό προβλημάτων στην πράξη, που αντικατοπτρίζονται κάτω από τη φράση curse of dimensionality (η κατάρα των πολυδιάστατων δεδομένων δηλαδή σε ελεύθερη απόδοση).

Σήμερα πραγματοποιείται ιδιαίτερα έντονη έρευνα ώστε να δημιουργηθούν ευρετήρια που «αντέχουν» στις αυξημένες διαστάσεις των δεδομένων, καθώς κλασικά ευρετήρια που έχουν εξαιρετική απόδοση σε χαμηλές διαστάσεις φαίνεται πως εκφυλίζονται όταν πρέπει να χειριστούν πολυδιάστατα δεδομένα. Για παράδειγμα, η οικογένεια των R-trees που έχουν εξαιρετική συμπεριφορά σε διαστάσεις 5 ή 10 ή και 15 ακόμα διαστάσεων, για περισσότερες διαστάσεις από αυτές καταντούν να είναι μία γραμμική δομή ευρετηρίου, χωρίς ιδιαίτερο νόημα. Εκτός από τη δημιουργία νέων ευρετηρίων για πολυδιάστατα δεδομένα γίνονται και άλλες προσπάθειες για τη διαχείριση τέτοιων δεδομένων, όπως θα δούμε στην πορεία της εργασίας, όπως είναι ο μετασχηματισμός των δεδομένων σε νέους χώρους (πχ από τον $\mathbb{R}^{100}$ στον $\mathbb{R}^{30}$ που μπορούμε να χειριστούμε σχετικά πιο εύκολα).

1.2 Αντικείμενο διπλωματικής

Πραγματοποιείται ιδιαίτερη έρευνα αυτό το διάστημα για την απάντηση ερωτημάτων πλησιέστερου γείτονα σε πολυδιάστατα δεδομένα. Στην παρούσα φάση δεν υπάρχει κάποιο ευρετήριο ή κάποια τεχνική μέθοδος που να κρίνεται ικανοποιητική για την απάντηση τέτοιων ερωτημάτων ως προς το χρόνο απόκρισης. Έτσι, έγινε μία προσπάθεια να κατασκευαστεί ένα ευρετήριο, που αφορά σε πολυδιάστατα δεδομένα και επιτρέπει την απάντηση τέτοιων ερωτημάτων με τις λιγότερες δυνατές αναγνώσεις και εγγραφές στο δίσκο, δηλαδή στο μικρότερο δυνατό χρόνο.

Ας γίνουμε όμως λίγο πιο συγκεκριμένοι. Τα ερωτήματα που θα μας απασχολήσουν είναι ερωτήματα πλησιέστερου γείτονα. Τι σημαίνει όμως αυτό με απλά λόγια; Αν υποθέσουμε ότι έχουμε μία βάση δεδομένων με m διαστάσεις και ένα σημείο Q που επιλέγει αυτός που θέτει την ερώτηση (θα το ονομάζουμε σημείο ερώτησης ή query point από εδώ και πέρα), τότε αναζητούμε τα k πλησιέστερα προς το Q στοιχεία που περιλαμβάνει η βάση δεδομένων. Θα θεωρήσουμε ότι η απόσταση του Q από ένα στοιχείο της βάσης δεδομένων είναι η ευκλείδεια απόστασή τους. Σύμφωνα με τα παραπάνω θα πρέπει να είναι σαφές ως τώρα, ότι αν το $Q = (q_1, q_2, \dots, q_m)$ και το $D = (d_1, d_2, \dots, d_m)$, τότε η μεταξύ τους απόσταση είναι $dist(Q, D) = \sqrt{(d_1 - q_1)^2 + (d_2 - q_2)^2 + \cdots + (d_m - q_m)^2}$. Αυτά τα ερωτήματα ονομάζονται kNN (k-nearest neighbor ή k-πλησιέστεροι γείτονες) καθώς επιστρέφονται τα k κοντινότερα στοιχεία της βάσης δεδομένων μας προς το σημείο ερώτησης Q. Εμείς θα μελετήσουμε μία παραλλαγή αυτών των ερωτημάτων, ή καλύτερα μία γενίκευση αυτών των ερωτημάτων. Θα επιτρέψουμε στα ερωτήματα k-NN να τίθενται όχι μόνο προς όλες τις διαστάσεις, αλλά να μπορούμε να αναζητήσουμε τους πλησιέστερους γείτονες του σημείου Q ως προς τις διαστάσεις d_2 και d_5 μόνο επί παραδείγματι, δηλαδή να βρούμε τα στοιχεία της βάσης δεδομένων εκείνα, που έχουν από το Q τις k μικρότερες αποστάσεις μόνο ως προς τις διαστάσεις d_2 και d_5, ήτοι θέλουμε τα k στοιχεία της βάσης δεδομένων που έχουν τα μικρότερα $dist_{2,5}(Q, D) = \sqrt{(d_2 - q_2)^2 + (d_5 - q_5)^2}$. Άρα, με απλά λόγια επιτρέπουμε να γίνονται ερωτήματα πλησιέστερου γείτονα σε οποιεσδήποτε διαστάσεις επιθυμεί ο χρήστης και όχι μόνο ως προς όλες τις διαστάσεις που έχει η σχέση μας.

Ας επανέλθουμε τώρα στον απώτερο σκοπό της εργασίας μας. Αυτός είναι η δημιουργία ενός ευρετηρίου που θα μας επιτρέψει να απαντάμε σε τέτοιου είδους ερωτήματα στο

μικρότερο δυνατό χρόνο. Θα υποθέσουμε για το σκοπό αυτό ότι έχουμε μία βάση δεδομένων πολλών διαστάσεων, στην οποία τίθενται διαφόρων ειδών ερωτήματα k-NN με συγκεκριμένες συχνότητες το καθένα τους, ενώ επίσης το κάθε ερώτημα k-NN αφορά και σε διαφορετικές διαστάσεις. Έτσι, μπορεί να έχουμε ένα ερώτημα k-NN που αφορά στις διαστάσεις 15, 67, 102 μιας σχέσης 10,000 ιδιοτήτων και έρχεται με συχνότητα 40%, ένα άλλο ερώτημα που αφορά στις διαστάσεις 1, 4, 10, 41, 67, 78, 89, 1032 της ίδιας σχέσης και έρχεται με συχνότητα 20% στη βάση δεδομένων μας κ.ο.κ.. Ως προς το σημείο ερώτησης δεν τίθεται κανένας περιορισμός. Μπορεί ο χρήστης σε κάθε ερώτημα που θέτει να το αλλάζει αυθαίρετα, αρκεί τα ερωτήματα που θέτει να έρχονται με τις συχνότητες βάσει των οποίων έχει σχεδιαστεί το ευρετήριό μας.

Για να βρεθεί το βέλτιστο ευρετήριο μελετήθηκε ο αλγόριθμος Threshold Algorithm (TA) και οι διάφορες παραλλαγές του και εκτελέστηκαν διάφορα πειράματα, στα οποία μετρήθηκαν οι αναγνώσεις και εγγραφές δίσκων με τη χρήση του TA πάνω σε B^+-trees και R^*-trees. Ο TA θεωρείται ο καλύτερος δυνατός αλγόριθμος για την απάντηση ερωτημάτων top-k όπως είναι γνωστά, δηλαδή ερωτημάτων που ζητούν τα k καλύτερα στοιχεία ως προς ένα κριτήριο. Στην περίπτωσή μας το κριτήριο είναι η ελάχιστη απόσταση από το σημείο ερώτησης στις διάφορες διαστάσεις που θέλει ο χρήστης. Έτσι, μελετήσαμε την απόδοση του TA πάνω σε B^+-trees και R^*-trees και δημιουργήσαμε ένα ευρετήριο, ή καλύτερα μία ομάδα από ευρετήρια που αποτελούνται από B^+-trees και R^*-trees, τα οποία μας επιτρέπουν να απαντήσουμε σε k-NN ερωτήματα που έρχονται με δεδομένες συχνότητες πολύ αποδοτικά.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήθηκε και υλοποιήθηκε μία παραλλαγή του αλγορίθμου TA πάνω σε B-δέντρα και R-δέντρα για δεδομένα τεσσάρων διαστάσεων με όλες τις δυνατές παραλλαγές (δηλαδή B-δέντρα σε κάθε διάσταση, ένα R-δέντρο στις δύο πρώτες διαστάσεις και δύο B-δέντρα για τις άλλες δύο διαστάσεις, ένα R-δέντρο στις τρεις πρώτες διαστάσεις και ένα B-δέντρο για την άλλη μία, ένα R-δέντρο 4 διαστάσεων και τέλος δύο R-δέντρα δύο διαστάσεων για τις πρώτες δύο και για τις δύο τελευταίες) με δύο διαφορετικούς τρόπους.
2. Αξιολογήσαμε τα αποτελέσματα που πήραμε από την πειραματική μελέτη των υλοποιήσεων που μόλις αναφέραμε.
3. Βασισμένοι στην αξιολόγηση των αποτελεσμάτων που λάβαμε από τα πειράματα, αναπτύχτηκε ο αλγόριθμος σχεδιασμού του τελικού ευρετηρίου που βοηθά στην απάντηση ερωτήσεων k-NN.

1.3 Οργάνωση κειμένου

Εργασίες σχετικές με το αντικείμενο της διπλωματικής παρουσιάζονται στο κεφάλαιο 2. Στο κεφάλαιο 3 παρουσιάζεται ο αλγόριθμος TA και διάφορες παραλλαγές του, οι δομές ευρετηρίων R-trees και B^+-trees καθώς και η λογική βάση της οποία θα εκτελέσουμε τα πειράματά μας. Στο κεφάλαιο 4 παραθέτουμε ορισμένα πειραματικά αποτελέσματα καθώς

και συμπεράσματα που εξάγονται από αυτά. Στο κεφάλαιο 5 τώρα περιγράφουμε τον αλγόριθμο κατανομής των διαστάσεων σε ευρετήρια. Καταληκτικά, στο κεφάλαιο 6 αναφερόμαστε στα γενικά συμπεράσματα που εξάγονται από την εργασία μας και στο 7 βλέπουμε τη βιβλιογραφία που χρειάστηκε για την εκπόνηση της εργασίας αυτής.

2

Παρεμφερή προβλήματα και οι λύσεις τους

Στη βιβλιογραφία βρέθηκαν αρκετά παρεμφερή προβλήματα και διάφορες λύσεις που έχουν δοθεί σε αυτά. Το πρώτο από αυτά είναι τα ερωτήματα ομοιότητας. Στα ερωτήματα αυτά δίνουμε ένα αντικείμενο στόχο και επιστρέφονται τα πιο όμοια αντικείμενα ως προς αυτό που περιλαμβάνονται στη βάση δεδομένων. Πρακτικά, η έννοιες όμοια αντικείμενα και κοντινά αντικείμενα είναι αυτούσιες, οπότε τα ερωτήματα ομοιότητας και τα ερωτήματα k-NN είναι πρακτικώς η ίδια έννοια.

Ένα άλλο παρεμφερές πρόβλημα είναι το πρόβλημα του reverse nearest neighbor. Μία αρκετά κατατοπιστική εργασία πάνω σε αυτό το ερώτημα είναι η [15]. Στο πρόβλημα αυτό, δίνεται ένα αντικείμενο στόχος και επιστρέφονται τα αντικείμενα της βάσης δεδομένων που έχουν αυτό το αντικείμενο στόχο πλησιέστερο αντικείμενο τους σε σχέση με όλα τα υπόλοιπα της βάσης δεδομένων.

Τέλος, αξίζει να σημειωθεί ότι έχουν καταβληθεί πολλές προσπάθειες επανασχεδιασμού των συναρτήσεων απόστασης καθώς και των εφαρμογών που ασχολούνται με πολυδιάστατα δεδομένα. Η έρευνα προς αυτήν την κατεύθυνση οφείλεται στο γεγονός ότι παραδοσιακές μετρικές (όπως η ευκλείδεια) δεν είναι ιδιαίτερα αντιπροσωπευτικές για πολυδιάστατα δεδομένα όπως θα δούμε στην αντίστοιχη ενότητα.

Αξίζει να σημειωθεί ότι η μόνη εργασία που είναι σε γνώση μας για το θέμα με το οποίο ασχολούμαστε σε αυτήν την εργασία είναι η [9], αλλά με χρήση μόνο R-trees με k-NN ερωτήματα που αναφέρονται σε όλες τις διαστάσεις της σχέσης, ενώ εμείς δίνουμε τη δυνατότητα για την πραγματοποίηση ερωτημάτων σε οποιοδήποτε σύνολο διαστάσεων επιλέξει ο χρήστης και με την αρωγή B^+-trees.

2.1 Ερωτήματα ομοιότητας και μετασχηματισμοί σχέσεων

Πολύ περίπλοκοι τύποι δεδομένων, όπως εικόνες, κείμενα και ακολουθίες DNA, γίνονται ολοένα και πιο σημαντικοί σε μοντέρνες εφαρμογές που σχετίζονται με τις βάσεις δεδομένων. Ένα ερώτημα που τίθεται συχνά σε τέτοιες εφαρμογές είναι η εύρεση αντικειμένων τα οποία είναι όμοια (δηλαδή έχουν μικρή απόσταση) με κάποιο δοσμένο αντικείμενο. Μία μέθοδος που ακολουθείται για να δοθεί απάντηση σε τέτοιου είδους ερωτήματα είναι να μετασχηματίσεις τα δεδομένα της DB από ένα διανυσματικό χώρο σε έναν άλλο, στον οποίο οι αποστάσεις προσεγγίζουν τις αρχικές, πραγματικές αποστάσεις. Έτσι, τα ερωτήματα εκτελούνται στα μετασχηματισμένα δεδομένα και κατόπιν, με κατάλληλη επεξεργασία, παίρνουμε τα πραγματικά δεδομένα που επιθυμούμε. Από τις κυριότερες εργασίες σε αυτό το χώρο είναι και η [11].

Η αν-ομοιότητα μεταξύ δύο διαφορετικών αντικειμένων δίνεται από μία συνάρτηση απόστασης d[1]. Το πρόβλημα είναι ότι πολλές φορές, ο υπολογισμός της απόστασης d μεταξύ δύο αντικειμένων είναι από μόνη της πάρα πολύ ακριβή. Για παράδειγμα, ο υπολογισμός της «απόστασης» δύο πρωτεϊνών χρειάζεται κάποιες εκατοντάδες δευτερολέπτων για τον υπολογισμό του. Για αυτό το λόγο και μετασχηματίζουμε τα δεδομένα μας με διπλό σκοπό:

- Οι αποστάσεις στο μετασχηματισμένο χώρο προσεγγίζουν κατά το δυνατό τις πραγματικές αποστάσεις των δεδομένων στον αρχικό χώρο.
- Η αναζήτηση στο μετασχηματισμένο χώρο είναι λιγότερο ακριβή από τον αρχικό χώρο.

Υπάρχουν μέθοδοι οι οποίες μετασχηματίζουν το διανυσματικό χώρο βασισμένες μόνο στη μετρική d. Το θετικό με αυτές τις μεθόδους είναι ότι μπορούν να εφαρμοστούν χωρίς να έχουμε κάποια γνώση σχετικά με τη φύση των δεδομένων μας, αφού η μετρική d αντιμετωπίζεται σαν μαύρο κουτί. Παρ' όλα αυτά, υπάρχουν πάμπολλες μέθοδοι μετασχηματισμού των δεδομένων μας σύμφωνα με το είδος των δεδομένων που έχουμε να επεξεργαστούμε (όπως εικόνες και χρονοσειρές), οι οποίες έχουν προκύψει από την εμπειρία που έχουμε αποκτήσει από την επεξεργασία τέτοιων δεδομένων.

2.1.1 Γενικά περί συναρτήσεων απόστασης

Η δυάδα (S, d) λέγεται πεπερασμένος μετρικός χώρος, αν το S είναι πεπερασμένο (πληθικότητας N) και η d είναι μία συνάρτηση απόστασης, $d: S \times S \to \mathbb{R}^+$. Έχει γίνει πολύ δουλειά στην πάροδο των χρόνων σχετική με τις συναρτήσεις απόστασης. Οι πιο συχνά

[1] Μία συνάρτηση απόστασης έχει τις ακόλουθες ιδιότητες:
- $d(x, y) \geq 0$
- $d(x, y) = 0 \Leftrightarrow x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(z, y)$

χρησιμοποιούμενες είναι οι μετρικές Minkowski, που ορίζονται ως $\|x\|^p = (\sum|x_i|^p)^{1/p}$ για $p \geq 1$. Για $p = 2$ έχουμε τη γνωστή μας ευκλείδεια απόσταση, για $p = 1$ έχουμε την απόσταση Manhattan και για $p = \infty$ έχουμε την απόσταση Chessboard.

Ένας μετασχηματισμός από έναν πεπερασμένο μετρικό χώρο (S, d) σε έναν $(\mathbb{R}^k, \delta)$ είναι μία αντιστοίχηση $F: S \rightarrow \mathbb{R}^k$, όπου k είναι το πλήθος των διαστάσεων του μετασχηματισμένου χώρου και $\delta: \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}^+$ είναι η συνάρτηση απόστασης του μετασχηματισμένου χώρου. Ιδανικά, θα θέλαμε η απόσταση $\delta(F(o_1, o_2))$ στο μετασχηματισμένο χώρο να είναι όσο το δυνατόν πιο κοντά στην πραγματική απόσταση $d(o_1, o_2)$ του αρχικού χώρου. Θα θέλαμε δηλαδή στην πραγματικότητα να είναι $\delta(F(o_1, o_2)) = d(o_1, o_2)$ για όλα τα $o_1, o_2 \in S$, δηλαδή οι δύο πεπερασμένοι μετρικοί χώροι να είναι ισομετρικοί.

2.1.2 Γενικά περί ερωτημάτων ομοιότητας και ιδιοτήτων των συναρτήσεων απόστασης

Όταν εκτελούμε ένα ερώτημα ομοιότητας σε ένα μετασχηματισμένο χώρο δεδομένων τότε (εκτός από την περίπτωση που $\delta(F(o_1, o_2)) = d(o_1, o_2)$ για όλα τα $o_1, o_2 \in S$) το ερώτημα θα μας επιστρέψει ένα σύνολο από αντικείμενα R_e, το οποίο θα είναι εν γένει διαφορετικό από το R_o που μας επιστρέφεται από το ερώτημα στον αρχικό χώρο, πράγμα που σημαίνει ότι το R_e, μπορεί να έχει κάποια αντικείμενα που δεν βρίσκονται στο R_o καθώς και το αντίστροφο.

Υπάρχουν δύο βασικά κριτήρια βάσει των οποίων αξιολογούμε την ποιότητα του συνόλου R_e που μας επιστρέφεται από το μετασχηματισμένο χώρο. Αυτά είναι τα ακόλουθα:

- *Ακρίβεια:* η έννοια της ακρίβειας έχει να κάνει με το πόσα από τα αντικείμενα που βρίσκονται στο R_e είναι σωστά, δηλαδή ανήκουν και στο R_o. Συγκεκριμένα, η ακρίβεια απάντησης του ερωτήματος στο μετασχηματισμένο χώρο ορίζεται ως $\frac{|R_e \cap R_o|}{|R_e|}$, δηλαδή ορίζεται ως ο αριθμός των αντικειμένων του R_e που σωστά επιστράφηκαν από το ερώτημα προς τον πληθάριθμο του συνόλου που περιλαμβάνει όλα τα αντικείμενα που επιστράφηκαν από το ερώτημα, όταν αυτό πραγματοποιήθηκε στο μετασχηματισμένο χώρο.
- *Ανάκληση:* η έννοια της ανάκλησης έχει να κάνει με το πόσα από τα αντικείμενα που θα έπρεπε να ανακληθούν από το ερώτημα, ανακλήθηκαν τελικά και στο μετασχηματισμένο χώρο. Συγκεκριμένα, ο ορισμός της ανάκλησης είναι $\frac{|R_e \cap R_o|}{|R_o|}$, δηλαδή ως ο αριθμός των αντικειμένων του R_e που ανακλήθηκαν στο μετασχηματισμένο χώρο, προς τον πληθάριθμο του συνόλου που περιλαμβάνει τα αντικείμενα που θα έπρεπε να ανακληθούν συνολικά.

Από τη στιγμή που θα εκτελέσουμε το ερώτημα στο μετασχηματισμένο χώρο, θα πάρουμε ως αποτέλεσμα το σύνολο αντικειμένων της βάσης δεδομένων R_e. Από εκεί και πέρα έχουμε δύο επιλογές:

- Να αναφέρουμε τα αντικείμενα που περιλαμβάνει το R_e όπως έχει, ως απάντηση στο ερώτημα. Παρ' ότι το κόστος με αυτόν τον τρόπο είναι ιδιαίτερα μικρό, πολλές φορές δεν γνωρίζουμε εκ των προτέρων ούτε την ακρίβεια ούτε την ανάκληση και άρα η ποιότητα της απάντησης στο ερώτημα μπορεί να μην είναι ικανοποιητική.
- Να χρησιμοποιήσουμε τη μέθοδο filter and refine. Δηλαδή, να θεωρήσουμε ότι η εύρεση του R_e είναι το στάδιο "filter" και η χρήση της πραγματικής συνάρτησης d χρησιμοποιείται για να κάνουμε "refine" το αποτέλεσμά μας, οπότε παίρνουμε ως αποτέλεσμα το R_f. Η μέθοδος filter and refine χρησιμοποιείται για να αφαιρέσουμε από το R_e όλα τα ανόμοια αντικείμενα σε σχέση με το αντικείμενο στόχο και άρα η ακρίβεια οδηγείται στο 100%. Παρ' όλα αυτά, αν στο R_o υπάρχουν αντικείμενα που δεν υπάρχουν στο R_e τότε αυτά προφανώς θα λείπουν και από το R_f. Για αυτό το λόγο και θέλουμε στη μέθοδο filter and refine η ανάκληση να είναι 100%, ώστε τελικά να πάρουμε όλα τα αντικείμενα που μας ενδιαφέρουν και μόνο αυτά.

Ένα πολύ σημαντικό στοιχείο είναι ότι μπορούμε πάντοτε να έχουμε 100% ανάκληση όταν ο μετασχηματισμός που πραγματοποιούμε είναι contractive.

Ορισμός. Ένας μετασχηματισμός είναι contractive όταν ισχύει $\delta\big(F(o_1, o_2)\big) \leq d(o_1, o_2)$ για κάθε $o_1, o_2 \in S$.

Όπως προαναφέραμε, όταν ένας μετασχηματισμός είναι contractive τότε μπορούμε να έχουμε 100% ανάκληση. Αυτό ισχύει διότι αν $\delta\big(F(o_1, o_2)\big) > r$ τότε $d(o_1, o_2) > r$. Έτσι, μπορούμε να κάνουμε prune όλα τα αντικείμενα o που έχουν $d(o_1, o_2) > r$, χωρίς να διώξουμε κάποιο αντικείμενο που τελικά θα ήταν σωστό να κρατήσουμε. Συνεπώς είναι εύκολο σε αυτήν την περίπτωση να έχουμε 100% ανάκληση.

Τώρα, οι μετασχηματισμοί που συνήθως πραγματοποιούνται είναι οι SparseMap, FastMap και MetricMap, στους οποίους και θα αναφερθούμε εν συντομία και χωρίς ιδιαίτερες λεπτομέρειες στη συνέχεια.

2.1.3 Μετασχηματισμοί

2.1.3.1 SparseMap

Το SparseMap βασίζεται στους μετασχηματισμούς Lipschitz. Επειδή το αρχικό SparseMap είχε κάποια σημαντικά μειονεκτήματα[2] έγινε μία προσπάθεια με κάποιες ευριστικές μεθόδους να βελτιωθεί. Οι ευριστικές αυτές μέθοδοι μείωσαν τις διαστάσεις του μετασχηματισμένου χώρου, κάνοντας το χειρισμό του πιο απλό, ενώ παράλληλα δεν απαιτούσαν τετραγωνικό κόστος για τον υπολογισμό όλων των αποστάσεων. Επίσης, το SparseMap δεν ήταν contractive και άρα δεν έδινε με βεβαιότητα 100% ανάκληση όπως αναφέραμε στην ενότητα 2.1.2. Και αυτό το πρόβλημα αντιμετωπίστηκε επιτυχώς και πλέον το SparseMap έχει πολλές και καλές ιδιότητες.

[2] Κατ' αρχάς, σε πολλές περιπτώσεις απαιτούσε τον υπολογισμό των αποστάσεων ενός αντικειμένου σε σχέση με όλα τα υπόλοιπα αντικείμενα, γεγονός που είχε τετραγωνικό κόστος ως προς τον αριθμό των δεδομένων. Επίσης, ο μετασχηματισμένος χώρος είχε πολλές διαστάσεις κάτι που δεν ήταν επιθυμητό για τη διατήρηση της απλότητας.

2.1.3.2 FastMap

Το FastMap είναι εμπνευσμένο από τη μείωση διαστάσεων σε ευκλείδειους χώρους που πραγματοποιεί ο μετασχηματισμός Karhunen-Loeve transform (KLT). Ο KLT είναι ένας αλγόριθμος για τον προσδιορισμό αξόνων συντεταγμένων. Το αρνητικό σημείο του FastMap είναι ότι είναι contractive μόνον όταν ο αρχικός χώρος που θα μετασχηματίσουμε είναι ισομετρικός (βλέπετε ενότητα 2.1.1) ενός ευκλείδειου χώρου. Κατά τα άλλα, ο μετασχηματισμός αυτός δίνει ιδιαίτερα καλά αποτελέσματα.

2.1.3.3 MetricMap

Το MetricMap είναι ένας ιδιαίτερα πολύπλοκος μετασχηματισμός, ο οποίος βασίζεται στο FastMap και το SVD. Αξίζει να σημειωθεί ότι όπως το FastMap έτσι και αυτός ο μετασχηματισμός δεν εξασφαλίζει πάντοτε ότι θα είναι contractive.

2.1.4 Ομοιότητες και διαφορές σε σχέση με την εργασία μας

Τα ερωτήματα ομοιότητας που τίθενται στις εργασίες που παρουσιάσαμε στην ενότητα 2.1 αφορούν σε όλες τις διαστάσεις μίας σχέσης (ή ενός αντικειμένου) σε αντίθεση με τη δική μας εργασία που δίνει τη δυνατότητα στο χρήστη να επιλέξει αυτός τις διαστάσεις στις οποίες θα αναζητήσει τον πλησιέστερο γείτονα. Έτσι, η μέθοδος των μετασχηματισμών δεν μπορεί να εφαρμοστεί. Οι μετασχηματισμοί μπορούν να μας αναφέρουν τους γείτονες ενός αντικειμένου στόχου όταν το ερώτημα γίνεται σε όλες τις διαστάσεις τις σχέσεις, δεν μπορεί όμως να χρησιμοποιηθεί ένας τέτοιος μετασχηματισμός για να κάνουμε ερωτήματα k-NN σε ορισμένες διαστάσεις μόνο της σχέσης της βάσης δεδομένων μας.

2.2 Ερωτήματα αντίστροφου πλησιέστερου γείτονα

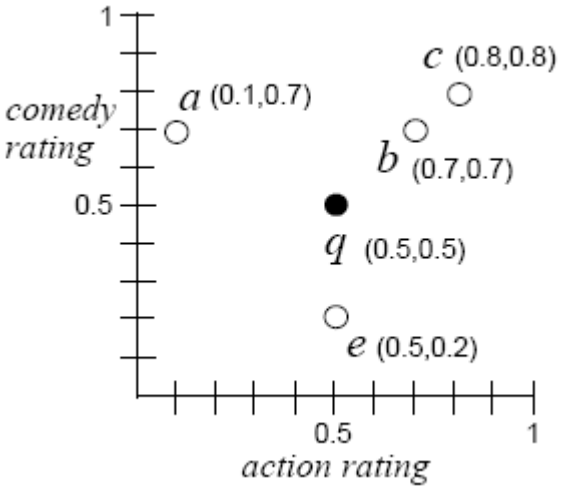
Τα ερωτήματα αντίστροφου πλησιέστερου γείτονα (reverse nearest neighbor ή RNN) είναι ερωτήματα που επιστρέφουν από τη βάση δεδομένων όλα τα δεδομένα που έχουν ένα αντικείμενο στόχο πιο κοντά τους από κάθε άλλο δεδομένο της βάσης δεδομένων.

Ας δούμε ένα παράδειγμα για να καταλάβουμε καλύτερα την έννοια του αντίστροφου πλησιέστερου γείτονα. Ας υποθέσουμε ότι έχουμε μία βάση δεδομένων με τέσσερα στοιχεία και δύο μόνον ιδιότητες:

- Την αξιολόγηση της ταινίας από την κωμική της πλευρά (το 1 είναι άριστη κωμωδία και το 0 κάκιστη)
- Την αξιολόγηση της ταινίας από την πλευρά της δράσης που αυτή περιέχει (και πάλι το 1 είναι άριστη περιπέτεια και το 0 κάκιστη).

Στην εικόνα Εικόνα 1 φαίνονται τα αντικείμενα της εν λόγω βάσης δεδομένων. Αν το αντικείμενο στόχος είναι το q, τότε τα στοιχεία της βάσης δεδομένων που έχουν το q πιο κοντά (με την ευκλείδεια απόσταση τουλάχιστον) από κάθε στοιχείο της βάσης δεδομένων είναι τα a και e. Το b δεν έχει το q πιο κοντά από κάθε στοιχείο της βάσης δεδομένων, καθώς η απόσταση των b, c είναι πιο μικρή από την απόσταση των q, b. Αξίζει να

σημειώσουμε ότι το γεγονός ότι το b είναι ο πλησιέστερος γείτονας του q δεν σημαίνει (όπως εύκολα μπορούμε να επαληθεύσουμε εκτελώντας μερικές πράξεις) ότι θα είναι και αντίστροφος πλησιέστερος γείτονας του q.



Εικόνα 1. Βάση δεδομένων με αξιολογήσεις ταινιών.

Μία παραλλαγή του RNN είναι το RkNN. Στο RkNN ερώτημα επιστρέφονται τα στοιχεία εκείνα τα οποία έχουν το q στα k πλησιέστερα αντικείμενα τους.

Και σε αυτά τα ερωτήματα η συνήθης πρακτική είναι η filter and refine. Στο στάδιο filter επιστρέφεται ένα υπερσύνολο του συνόλου των αποδεκτών/σωστών λύσεων και στο στάδιο refine απορρίπτονται τα δεδομένα εκείνα που τελικά δεν ήταν αποδεκτά (καθώς το στάδιο filter γενικά επιστρέφει περισσότερα από τα απόλυτα αναγκαία δεδομένα, έχει δηλαδή ακρίβεια μικρότερη από 100%, αλλά ανάκληση 100%).

Έτσι, για το στάδιο filter έχουν προταθεί διάφοροι αλγόριθμοι (αναφορικά μόνο αξίζει να σημειωθούν: TPL filter, greedy filter).

Είναι προφανές ότι το πρόβλημα RNN σχετίζεται ιδιαίτερα με το NN. Όμως και πάλι η λύση που έχει προταθεί ως τώρα δεν μπορεί να μας εξασφαλίσει καλή λειτουργία για το πρόβλημα που μελετούμε εμείς, το οποίο προϋποθέτει ότι αναζητούμε πλησιέστερους γείτονες σε ad-hoc υποχώρους που επιλέγει ο χρήστης του συστήματος.

2.3 Επανασχεδίαση συναρτήσεων απόστασης

Τα τελευταία χρόνια έχει παρατηρηθεί έντονα ότι διάφορα προβλήματα, όπως η ομαδοποίηση στοιχείων (clustering), η αναζήτηση του πλησιέστερου γείτονα (nearest neighbor search) και η κατασκευή ευρετηρίων (indexing) δεν μπορούν να λυθούν με καλά αποτελέσματα σε πολυδιάστατα δεδομένα. Αυτό οφείλεται στο γεγονός ότι οι πολυδιάστατοι χώροι είναι αραιοί και έτσι παραδοσιακά ευρετήρια και αλγοριθμικές τεχνικές αποτυγχάνουν να δώσουν τα αναμενόμενα αποτελέσματα. Για παράδειγμα, αξίζει να αναφερθεί ότι σε ερωτήματα πλησιέστερου γείτονα απαιτείται η προσπέλαση όλων των

δεδομένων της βάσης σχεδόν για να δοθεί απάντηση, ενώ διάφοροι αλγόριθμοι για clustering έχουν ασταθή συμπεριφορά και πολλές φορές δεν καταφέρνουν να λειτουργήσουν αποδοτικά.

Πολύ συχνά, η κατευθείαν χρήση μεθόδων που βασίζεται σε απόσταση (που αρχικά είχαν σχεδιαστεί με την σιωπηλή προϋπόθεση των λίγων διαστάσεων) έχουν ως αποτέλεσμα η απόδοση να μην είναι η αναμενόμενη και το κόστος να είναι ιδιαίτερα μεγάλο. Αυτό οφείλεται σε μεγάλο βαθμό στις χρησιμοποιούμενες συναρτήσεις απόστασης. Οι παραδοσιακές συναρτήσεις απόστασης δεν έχουν ιδιαίτερο νόημα για πολυδιάστατα δεδομένα. Για παράδειγμα, αν χρησιμοποιούμε την ευκλείδεια απόσταση σε ένα ερώτημα πλησιέστερου γείτονα σε μία σχέση 10,000 διαστάσεων, τότε όποιο και να είναι το σημείο ερώτησης με όλα σχεδόν τα δεδομένα της βάσης δεδομένων θα έχει αρκετές διαστάσεις με μεγάλες διαφορές από το σημείο της ερώτησης. Αυτό προκαλεί πολλά προβλήματα, καθώς είναι εξαιρετικά δύσκολος ο εντοπισμός του πλησιέστερου γείτονα από τη μία, ενώ και αυτό που θα επιστραφεί βάσει της ευκλείδειας απόστασης μπορεί να μην είναι το στοιχείο της βάσης δεδομένων που θα επιθυμούσαμε.

Προς επίρρωση των προηγουμένων θα παραθέσουμε ένα θεώρημα που καταδεικνύει τη σημασία που έχει η μεγάλη διάσταση των δεδομένων.

Θεώρημα του Beyer. Αν d είναι η διάσταση των δεδομένων, N είναι το πλήθος των δεδομένων της βάσης δεδομένων, X_d είναι ένα σημείο του d-διάστατου χώρου δεδομένων, $dist_d(X_d)$ είναι η απόσταση του X_d από την αρχή $(0,0,\dots,0)$, $Dmin_d$, $Dmax_d$ είναι η ελάχιστη και μέγιστη απόσταση των N σημείων από την αρχή, τότε:

Αν $\lim_{d\to\infty}\left(\frac{dist_d(X_d)}{E[dist_d(X_d)]}\right)=0$

Τότε $\frac{Dmax_d-Dmin_d}{Dmin_d}$ συγκλίνει όσο το d τείνει στο άπειρο κατά πιθανότητα στο 0.

Το προηγούμενο θεώρημα μας λέει πρακτικά ότι κάτω από ορισμένες συνθήκες, που ισχύουν σχετικά εύκολα, η διαφορά μεταξύ της μέγιστης και της ελάχιστης απόστασης από ένα δοσμένο σημείο είναι μικρή όταν συγκρίνεται με την απόσταση του πλησιέστερου σημείου σε ένα πολυδιάστατο χώρο. Το γεγονός αυτό καθιστά τα ερωτήματα που έχουν ως στοιχείο τους την απόσταση ασταθή. Για να γίνει αυτό κατανοητό, ας σκεφτούμε για λίγο πως μία μικρή αλλαγή του σημείου ερώτησης σε ένα ερώτημα πλησιέστερου γείτονα μπορεί να κάνει τον μέχρι πρότινος πλησιέστερο γείτονα να είναι το πιο απομακρυσμένο στοιχείο πλέον.

Το θεώρημα όμως, δεν μας λέει μόνο πόσο ασταθές είναι ένα ερώτημα πλησιέστερου γείτονα. Μας ενημερώνει ταυτόχρονα πως και το κόστος απάντησης ενός τέτοιου ερωτήματος θα είναι ιδιαίτερα υψηλό. Ας αναλογιστούμε ότι τα ερωτήματα πλησιέστερου γείτονα χρησιμοποιούν την εξής ιδέα: αν είναι γνωστό ότι κάποιος γείτονας X είναι αρκετά κοντά στο στόχο, τότε μπορούμε να κάνουμε pruning όλων των στοιχείων που έχουν μεγαλύτερη απόσταση από το X. Με αυτή τη λογική έχουν σχεδιαστεί και κάποια ευρετήρια. Όμως, αν όλα τα δεδομένα της βάσης μας έχουν την ίδια απόσταση από το

σημείο στόχο, τότε δεν μπορούμε να κάνουμε και πολλά. Το pruning περιορίζεται ραγδαία. Αυτός είναι και ένας λόγος που τα ευρετήρια που υπάρχουν οδηγούν πρακτικά σε ανάκληση όλων των δεδομένων από το δίσκο.

Όλα τα προηγούμενα συνηγορούν στο ότι θα πρέπει να δημιουργήσουμε συναρτήσεις απόστασης οι οποίες θα έχουν νόημα και για πολυδιάστατα δεδομένα, ενώ θα εξυπηρετούν ταυτόχρονα και τις ανάγκες σε απόδοση που έχουν οι εφαρμογές μας. Έτσι, κάποιες ιδιότητες που θα θέλαμε να έχουν οι συναρτήσεις απόστασης που αφορούν σε πολυδιάστατα δεδομένα είναι οι ακόλουθες:

- Να μπορούν να διακρίνουν καθαρά τα δεδομένα που απέχουν πολύ από ένα σημείο από αυτά που απέχουν λίγο. Όπως είδαμε, σε πολύ μεγάλες διαστάσεις σχεδόν πάντα κάποιες διαστάσεις θα απέχουν πολύ από το σημείο στόχο με αποτέλεσμα η απόσταση ενός δεδομένου από το σημείο στόχο να είναι σχεδόν σταθερή για όλα τα δεδομένα της βάσης μας
- Να έχουν στατιστική ευαισθησία. Τα δεδομένα που έχουν οι διάφορες βάσεις δεδομένων δεν είναι ομοιόμορφα κατανεμημένα σε ένα διάστημα συνήθως, γεγονός που οι συναρτήσεις απόστασης θα πρέπει να το λαμβάνουν υπ' όψιν τους.
- Να μπορούν να λάβουν υπ' όψιν τους ότι δύο ή περισσότερες διαστάσεις είναι συσχετισμένες μεταξύ τους. Αν αναζητήσουμε την απόσταση δύο δεδομένων της βάσης μας και οι τιμές τους στις ιδιότητες 1 είναι παρόμοιες, αλλά και στη διάσταση 2 και στην 3 κ.ο.κ. θα πρέπει να το λάβουμε υπ' όψιν μας ώστε να πούμε ότι αυτά τα δεδομένα είναι παρόμοια μεταξύ τους.
- Να είναι εύκολα υπολογίσιμη. Είναι αδιανόητο να σχεδιάσουμε μία συνάρτηση απόστασης η οποία για να υπολογιστεί απαιτεί κάποια δευτερόλεπτα. Θέλουμε να ο υπολογισμός μίας συνάρτησης απόστασης να γίνεται κατά το δυνατόν ταχύτερα.

Μία προσπάθεια σχεδιασμού μία μετρικής είναι αυτή που περιγράφεται στο [5] και μπορεί να εφαρμοστεί ανεξάρτητα από τα δεδομένα που πραγματευόμαστε.

Ας δούμε όμως τώρα γιατί η παραπάνω μέθοδος δεν αποτελεί ιδιαίτερη βοήθεια για την εργασία μας. Εμείς έχουμε πολυδιάστατα δεδομένα, αλλά τα ερωτήματα που θα θέτουμε θα αναφέρονται σε κάποιους υποχώρους. Οι υποχώροι αυτοί θα είναι σαφώς μικρότερης διάστασης από τη συνολική διάσταση των δεδομένων μας. Σε υποχώρους τέτοιους, μικρής διάστασης, η ευκλείδεια νόρμα που χρησιμοποιήσαμε είναι ιδιαίτερα αποτελεσματική. Δεν υπάρχει ανάγκη αλλαγής της νόρμας για να μπορέσουμε να έχουμε αποτελεσματικά απαντήσεις σε ερωτήματα NN. Έτσι, παρ' ότι για παράδειγμα η συνάρτηση απόστασης που προτείνεται στο [5], θα μπορούσε να χρησιμοποιηθεί δεν χρησιμοποιήθηκε για δύο λόγους:

- Δεν θα προσέφερε κάτι ουσιαστικό, καθώς στους υποχώρους που εργαζόμαστε οι διαστάσεις δεν είναι τόσο μεγάλες όσο είναι το συνολικό πλήθος των διαστάσεων της βάσης δεδομένων μας. Αυτό σημαίνει ότι η ευκλείδεια νόρμα θα λειτουργήσει πολύ αποτελεσματικά.
- Το κόστος υπολογισμού της θα ήταν αρκετά μεγαλύτερο από το κόστος της ευκλείδειας νόρμας, της οποίας ο υπολογισμός είναι ιδιαίτερα απλός.

3

Δομές ευρετηρίων και βασικοί αλγόριθμοι

Ο σκοπός της διπλωματικής εργασίας είναι να δημιουργήσει ένα βέλτιστο κατά το δυνατό ευρετήριο για ερωτήματα πλησιέστερου γείτονα, όταν έχουμε πολυδιάστατα δεδομένα. Αυτό θα το κάνουμε με τη χρήση ήδη υπαρχόντων ευρετηρίων σε μεσαίας διάστασης αντικείμενα. Δηλαδή, θα χρησιμοποιήσουμε R^*-trees και B^+-trees (είτε σε λίγες διαστάσεις είτε σε μία αντίστοιχα) για να μπορούν να τίθενται και να απαντώνται ερωτήματα πλησιέστερου γείτονα με τις λιγότερες δυνατές αναγνώσεις και εγγραφές στο δίσκο. Τα R^*-trees και B^+-trees χρησιμοποιούνται κατάλληλα από έναν αλγόριθμο που ονομάζεται TA και θα εξετάσουμε ακολούθως, ώστε να δοθεί απάντηση σε ερωτήματα πλησιέστερου γείτονα.

Θα δούμε στην ακόλουθη ενότητα τον αλγόριθμο TA και στη μεθεπόμενη θα μελετήσουμε τις δομές ευρετηρίων R^*-trees και B^+-trees που θα χρησιμοποιήσουμε για την εργασία μας.

3.1 Παρουσίαση του TA και διάφορων παραλλαγών του

Ας υποθέσουμε ότι έχουμε μία βάση δεδομένων με m διαστάσεις. Για παράδειγμα μπορεί να έχουμε μία βάση δεδομένων που περιγράφει τις πωλήσεις των καταστημάτων μίας μεγάλης εταιρείας. Για κάθε δεδομένο (για κάθε row δηλαδή στην πράξη) έχουμε τέσσερις στήλες. Στην πρώτη αναφέρονται οι πωλήσεις ενός προϊόντος, στη δεύτερη ενός άλλου κ.ο.κ.. Προφανώς, οι συνολικές πωλήσεις ενός καταστήματος της εταιρείας είναι το άθροισμα των πωλήσεων των διαφόρων προϊόντων.

Κάθε μία από αυτές τις διαστάσεις αντιπροσωπεύει ένα βαθμό, ενώ σε κάθε κατάστημα μπορεί να ανατεθεί ένας γενικός βαθμός ο οποίος προκύπτει από τους μερικούς βαθμούς που επιδεικνύουν τις πωλήσεις σε κάθε προϊόν με τη βοήθεια μίας συνάρτησης (όπως η συνάρτηση minimum ή average). Για να δούμε ποια είναι τα k κορυφαία (top-k) αντικείμενα (ή δεδομένα), δηλαδή στο παράδειγμά μας για να εντοπίσουμε τα k καταστήματα με τις μεγαλύτερες πωλήσεις σε άθροισμα, θα πρέπει σε μία πρώτη σκέψη να βρούμε τις τιμές που έχει ένα αντικείμενο σε κάθε διάσταση και να εκτελέσουμε την

κατάλληλη συνάρτηση σε αυτές τις τιμές. Αφού ανακαλέσουμε όλα τα δεδομένα από τη βάση δεδομένων μας και βρούμε τους γενικούς βαθμούς που αντιστοιχούν στο καθένα είναι εύκολο να βρούμε και τα k κορυφαία.

Υπάρχουν όμως πολύ αποδοτικότεροι αλγόριθμοι που επιλύουν το πρόβλημα αυτό. Ο Fagin έχει δώσει έναν αλγόριθμο που είναι πολύ πιο αποδοτικός από τον απλό αλγόριθμο που μόλις περιγράψαμε. Ο αλγόριθμος αυτός ονομάζεται, προς τιμή του Fagin, Fagin's Algorithm (FA). Ο FA είναι βέλτιστος με μεγάλη πιθανότητα στη χειρότερη περίπτωση για κάποιες μονότονες aggregation συναρτήσεις. Όμως, υπάρχει ένας αλγόριθμος που είναι καλύτερος και από αυτόν του Fagin. Ο αλγόριθμος αυτός ονομάζεται Threshold Algorithm (TA) και είναι βέλτιστος υπό μία έννοια πολύ ισχυρότερη από αυτή του FA. Ο TA είναι βέλτιστος όχι μόνο για κάποιες μονότονες aggregation συναρτήσεις, αλλά για όλες τις μονότονες aggregation συναρτήσεις και όχι με υψηλή πιθανότητα στη χειρότερη περίπτωση, αλλά είναι βέλτιστος για όλες τις πιθανές βάσεις δεδομένων. Επίσης, σε αντίθεση με τον FA που απαιτεί μεγάλους buffers (μάλιστα το μέγεθος των buffers αυξάνεται όσο αυξάνεται και το μέγεθος της βάσης δεδομένων χωρίς φραγμό), ο TA απαιτεί έναν buffer σταθερού μεγέθους. Μία πολύ καλή ανάλυση του TA καθώς και παραλλαγών του δίνεται στο [10].

Ας δούμε σε αυτή τη φάση ορισμένες έννοιες που θα χρησιμοποιήσουμε στη συνέχεια της περιγραφής του TA.

3.1.1 Χρήσιμες έννοιες για την περιγραφή του TA

Ορισμός. Εάν $x_1, x_2, \dots, x_m$ είναι οι τιμές ενός αντικειμένου R στις m διαστάσεις που αυτό έχει, τότε $t(x_1, x_2, \dots, x_m)$ είναι ο γενικός βαθμός του αντικειμένου R. Πολύ συχνά χρησιμοποιείται και ο συμβολισμός $t(R)$, αντί του $t(x_1, x_2, \dots, x_m)$. Η t ονομάζεται τότε aggregation συνάρτηση.

Είναι φανερό τώρα τι είναι μία aggregation συνάρτηση. Τρεις συναρτήσεις που χρησιμοποιούνται πολύ συχνά στη θέση της t είναι η minimum, η average και η sum.

Ορισμός. Μία aggregation συνάρτηση είναι μονότονη αν $t(x_1, x_2, \dots, x_m) \leq t(x_1', x_2', \dots, x_m')$ όταν $x_i \leq x_i'$ για κάθε $i \in \{1,2, \dots, m\}$.

Η μονοτονία είναι μία ιδιότητα που αναμένουμε από μία aggregation συνάρτηση. Αν για κάθε αντικείμενο R' η τιμή όλων των ιδιοτήτων είναι τουλάχιστον όσο μεγάλη αυτής του αντικειμένου R, τότε αναμένουμε και ο γενικός βαθμός της R' να είναι ίσος ή μεγαλύτερος από αυτόν του αντικειμένου R.

Συνήθως, στις παραδοσιακές βάσεις δεδομένων, η απάντηση ενός ερωτήματος είναι ένα σύνολο το οποίο δεν έχει κάποια διάταξη, δεν είναι δηλαδή ταξινομημένο. Αντίθετα, όταν αναζητούμε τα top-k αντικείμενα, η απάντηση στο ερώτημα είναι βαθμολογημένη. Έτσι, η απάντηση είναι ένα βαθμολογημένο σύνολο από δυάδες (x, g) όπου x είναι το δεδομένο μας και g είναι ο βαθμός του, όπως αυτός προκύπτει από την aggregation συνάρτηση που χρησιμοποιούμε. Συνήθως, τα βαθμολογημένα σύνολα ταξινομούνται βάσει του γενικού βαθμού των αντικειμένων.

Ας δούμε τώρα το μοντέλο κάτω από το οποίο θα τρέχει ο TA.

Μοντέλο. Υποθέτουμε ότι η βάση δεδομένων μας αποτελείται από ένα πεπερασμένο σύνολο από αντικείμενα. Θα συμβολίσουμε το πλήθος των αντικειμένων με N. Συσχετισμένα με κάθε αντικείμενο R είναι m πεδία ιδιοτήτων $x_1, x_2, \dots, x_m$, με $x_i \in [0,1]$. Θα αναφερόμαστε στο x_i ως το i-οστό πεδίο ή ιδιότητα ή διάσταση του αντικειμένου R. Τη βάση δεδομένων μας μπορούμε να τη φανταστούμε όπως έχουμε συνηθίσει, δηλαδή ως μία σχέση με m διαστάσεις μαζί με μία στήλη που δηλώνει το id. Θα μπορούσαμε όμως – και αυτό θα κάνουμε σε αυτή την εργασία – να σκεφτούμε τη βάση μας σαν ένα σύνολο από m ταξινομημένες λίστες $L_1, L_2, \dots, L_m$, η κάθε μία μήκους N (υπάρχει μία εγγραφή για το κάθε αντικείμενο της βάσης δεδομένων μας στην κάθε λίστα). Στη συνέχεια μπορεί να αναφερόμαστε στη κάθε λίστα L_i ως η i-οστή λίστα. Κάθε εγγραφή σε κάθε λίστα L_i θεωρείται ότι είναι μία δυάδα (R, x_i), όπου x_i είναι το i-οστό πεδίο του R. Κάθε λίστα είναι ταξινομημένη σε φθίνουσα μορφή ως προς τη μεταβλητή x_i. Θα θεωρούμε ότι η βάση δεδομένων μας έχει αυτή τη μορφή, καθώς αναφερόμαστε στους ακόλουθους αλγορίθμους. Επίσης, θα λαμβάνουμε υπ' όψιν μας μόνο το κόστος της ανάκλησης πληροφορίας και όχι το υπολογιστικό κόστος.

Το κόστος της ανάκλησης της πληροφορίας έρχεται σε δύο διαφορετικές μορφές.

- Η πρώτη είναι η σειριακή (sequential ή sorted). Στις sorted ανακλήσεις πληροφορίας ο αλγόριθμος ανακαλεί από μία από τις ταξινομημένες λίστες που προαναφέραμε την τιμή σε ένα πεδίο ενός αντικειμένου. Αυτές οι ανακλήσεις γίνονται από τις λίστες από την κορυφή τους και μετά κατεβαίνουν μία - μία τις θέσεις μέχρι και το τέλος της κάθε λίστας.
- Η δεύτερη είναι η και η τυχαία (random). Εδώ, ο αλγόριθμος ζητά την τιμή του αντικειμένου R στη i-οστή λίστα και την παίρνει με μία τυχαία προσπέλαση.

Αν σε έναν αλγόριθμο πραγματοποιηθούν s σειριακές προσπελάσεις δεδομένων και r τυχαίες, και το κόστος της μία σειριακής προσπέλασης είναι c_s και μίας τυχαίας c_r, τότε το συνολικό κόστος του αλγορίθμου (από την πλευρά των προσπελάσεων που το εξετάζουμε εμείς) είναι $sc_s + rc_r$.

3.1.2 Fagin's Algorithm

Ας δούμε σε αυτή την ενότητα τον αλγόριθμο του Fagin:

Fagin's Algorithm (FA)

1. Κάνε παράλληλες[3] προσπελάσεις στα δεδομένα της κάθε μίας από τις m ταξινομημένες λίστες. Περίμενε μέχρι να υπάρξουν k αντικείμενα τα οποία έχεις δει σε όλες τις ταξινομημένες λίστες.
2. Για κάθε αντικείμενο *R* που έχει ιδωθεί σε κάποια λίστα, κάνε τυχαίες προσπελάσεις δεδομένων για να βρεις τα πεδία του αντικειμένου που δεν έχουν ακόμα ιδωθεί.

[3] Με τη λέξη παράλληλα εννοούμε: προσπέλασε πρώτα όλα τα κορυφαία δεδομένα της κάθε ταξινομημένης λίστας, μετά όλα τα δεύτερα κ.ο.κ..

3. Υπολόγισε το γενικό βαθμό του κάθε αντικειμένου που έχει ιδωθεί: $t(R) = t(x_1, x_2, \dots, x_m)$. Ας είναι *Y* το σύνολο των αντικειμένων που έχουν ιδωθεί με τους k καλύτερους βαθμούς. Η έξοδος του αλγορίθμου είναι το σύνολο $\{(R, t(R)): R \in Y\}$.

Η απόδειξη της ορθότητας του αλγορίθμου είναι αρκετά εύκολη. Αξίζει να αναφερθεί όμως ότι αν υπάρχουν N αντικείμενα στη βάση δεδομένων τότε το κόστος του αλγορίθμου είναι $O\left(N^{m-1/m} k^{1/m}\right)$ με μεγάλη πιθανότητα.

Απόδειξη ορθότητας του FA. Ο αλγόριθμος του Fagin δίνει όντως το σωστό αποτέλεσμα. Έστω ότι η τελευταία τιμή που είδαμε σε κάθε λίστα L_i είναι η $\underline{x_i}$ με την εκτέλεση του αλγορίθμου του Fagin. Ήδη από το βήμα 1 του αλγορίθμου έχουμε εξασφαλίσει ότι τα k αντικείμενα που έχουμε δει πλήρως έχουν βαθμό μεγαλύτερο ή το πολύ ίσο με την τιμή $t\left(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}\right)$. Επίσης, όλα τα αντικείμενα που δεν έχουμε δει, θα έχουν γενικό βαθμό μικρότερο από το $t\left(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}\right)$, καθώς οι τιμές τους σε όλα τα πεδία είναι μικρότερες από τα x_i και η $t(.)$ είναι μονότονη, σύμφωνα με τις προϋποθέσεις που έχουμε θέσει για τον αλγόριθμο του Fagin. Αυτό σημαίνει ότι όλα τα αντικείμενα που δεν έχουμε δει, δεν μπορεί να είναι υποψήφιοι για το σύνολο που περιέχει τα top-k αντικείμενα. Για αυτό και δεν εξετάζουμε αυτά τα στοιχεία. Τώρα, στα βήματα 2 και 3 του αλγορίθμου εξετάζουμε τους γενικούς βαθμούς όλων των στοιχείων που έχουμε δει σε κάποιο πεδίο. Έτσι, βρίσκουμε ορθά τα στοιχεία που αναζητούμε.

Ορισμός. Μία aggregation συνάρτηση είναι αυστηρή όταν $t(x_1, x_2, \dots, x_m) = 1$ τότε και μόνον τότε όταν $x_i = 1$ για κάθε i.

Ο Fagin απέδειξε ότι ο αλγόριθμός του είναι βέλτιστος με μεγάλη πιθανότητα όταν η aggregation συνάρτηση είναι αυστηρή. Αξίζει να αναφερθεί επίσης για τον αλγόριθμο του Fagin ότι το κόστος του είναι ανεξάρτητο από την aggregation συνάρτηση που χρησιμοποιούμε. Ακόμα και μία σταθερή aggregation συνάρτηση να είχαμε, οι ίδιες ακριβώς προσπελάσεις δεδομένων θα συνέβαιναν.

Ο περιορισμός ότι ο αλγόριθμος του Fagin είναι βέλτιστος με μεγάλη πιθανότητα όταν η aggregation συνάρτηση είναι αυστηρή δεν μας καλύπτει σε αρκετές περιπτώσεις δυστυχώς. Για παράδειγμα, όταν η aggregation συνάρτηση είναι η max, εύκολα μπορούμε να δούμε ότι το μόνο που χρειάζεται είναι mk σειριακές προσπελάσεις δεδομένων (κατά μέγιστο) και καμία τυχαία προσπέλαση για να βρούμε τα top-k στοιχεία. Αντίθετα, ο TA που θα εξετάσουμε αμέσως μετά, είναι βέλτιστος για κάθε μονότονη aggregation συνάρτηση, με πολύ χαλαρές συνθήκες.

3.1.3 Threshold algorithm

Ο Threshold Algorithm (ή TA) αναπτύχτηκε από τρεις ομάδες επιστημόνων ανεξάρτητα. Οι Nepal και Ramakrishna ήταν οι πρώτοι που τον δημοσίευσαν, αλλά παράλληλα τον ανέπτυξαν και ο Guntzer και άλλοι, καθώς και οι Fagin, Lotem και Naor.

Ας παρουσιάσουμε τώρα τον Threshold Algorithm, ή όπως θα τον λέμε στη συνέχεια τον TA.

Threshold Algorithm (TA)

1. Κάνε παράλληλες προσπελάσεις στα δεδομένα της κάθε μίας από τις m ταξινομημένες λίστες. Για κάθε αντικείμενο *R* που βλέπεις σε μία λίστα, κάνε τυχαίες προσπελάσεις στις υπόλοιπες λίστες ώστε να βρεις τις τιμές του *R* σε όλα τα πεδία του. Υπολόγισε το $t(R) = t(x_1, x_2, \dots, x_m)$. Αν το $t(R)$ είναι ένα από τα k υψηλότερα που έχεις δει ως τώρα, να θυμάσαι το *R* και το γενικό βαθμό του $t(R)$.
2. Για κάθε ταξινομημένη λίστα L_i ας είναι $\underline{x_i}$ το τελευταίο στοιχείο που έχει ιδωθεί με σειριακές προσπελάσεις. Ορίζουμε ως τιμή κατωφλίου την τιμή $\tau = t\left(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}\right)$. Όταν έχεις δει τουλάχιστον k αντικείμενα με τιμή μεγαλύτερη ή ίση με το τ, τότε σταμάτα.
3. Ας είναι *Y* το σύνολο των αντικειμένων που έχουν ιδωθεί με τους k καλύτερους βαθμούς. Η έξοδος του αλγορίθμου είναι το σύνολο $\{(R, t(R)) : R \in Y\}$.

Ο TA είναι σωστός για κάθε μονότονη aggregation συνάρτηση t.

Απόδειξη ορθότητας του TA. Ας είναι Y το σύνολο του βήματος 3 του αλγορίθμου. Για να αποδείξουμε την ορθότητα του TA αρκεί να δείξουμε ότι κάθε αντικείμενο που ανήκει στο Y έχει γενικό βαθμό μεγαλύτερο ή ίσο από κάθε στοιχείο z που δεν ανήκει στο Y. Από τον ίδιο τον ορισμό του Y, η προηγούμενη πρόταση ισχύει για κάθε στοιχείο z που έχουμε δει τρέχοντας τον TA. Ας υποθέσουμε τώρα ότι έχουμε ένα z που δεν έχουμε δει σε καμία διάσταση τρέχοντας τον TA. Έστω ότι τα πεδία του είναι τα $x_1, x_2, \dots, x_m$. Καθώς δεν έχουμε δει το z πουθενά (δηλαδή σε κανένα πεδίο) τα τρέχοντα $\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}$ θα είναι σίγουρα μεγαλύτερα από τα αντίστοιχα $x_1, x_2, \dots, x_m$ του αντικειμένου z. Δηλαδή θα ισχύει $x_i \leq \underline{x_i}$. Συνεπώς, θα είναι $t(z) = t(x_1, x_2, \dots, x_m) \leq t\left(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}\right) = \tau$, λόγω της μονοτονίας της συνάρτησης t. Όμως, από τον ίδιο τον ορισμό του συνόλου Y, για κάθε αντικείμενο $y \in Y$ έχουμε ότι $t(y) \geq \tau$. Άρα, για κάθε στοιχείο του συνόλου *Y* έχουμε την ποθητή $t(y) \geq \tau \geq t(z)$.

Αξίζει επίσης να αναφερθεί ότι ο TA σταματά τουλάχιστον όσο γρήγορα σταματά και ο FA, δηλαδή, ο TA εκτελεί το πολύ τόσες σειριακές προσπελάσεις όσες εκτελεί και ο FA. Στον FA, αν *R* είναι ένα αντικείμενο το οποίο έχουμε δει με σειριακές προσπελάσεις σε κάθε λίστα, τότε λόγω της μονοτονίας της t ο βαθμός του *R* θα είναι τουλάχιστον ίσος με την τιμή κατωφλίου. Για αυτό το λόγο, όταν υπάρχουν τουλάχιστον k αντικείμενα που έχουν ιδωθεί με τον FA σε κάθε λίστα με σειριακές προσπελάσεις (ο κανόνας τερματισμού του FA), τότε υπάρχουν τουλάχιστον k αντικείμενα που έχουν βαθμό τουλάχιστον ίσο με την τιμή κατωφλίου (που είναι η συνθήκη τερματισμού του TA).

Αυτό σημαίνει ότι οι σειριακές προσπελάσεις του TA είναι το πολύ όσες είναι οι σειριακές προσπελάσεις του FA, αλλά δεν σημαίνει ότι το συνολικό κόστος του TA είναι κατ' ανάγκη μικρότερο από το κόστος του FA (καθώς στο συνολικό κόστος ενός αλγορίθμου, εκτός από τις σειριακές προσπελάσεις δεδομένων, πρέπει να λάβουμε υπ' όψιν μας και τις τυχαίες.

Δεν μας εξασφαλίζει κανείς ότι ο TA δεν θα κάνει περισσότερες τυχαίες προσπελάσεις δεδομένων σε σχέση με τον FA.

Ας δούμε λίγο τη φιλοσοφία του αλγορίθμου TA. Πού βασίζεται δηλαδή, σε ποια ιδέα. Για απλότητα, ας υποθέσουμε ότι k=1, δηλαδή ο χρήστης αναζητά την καλύτερη δυνατή απάντηση και μόνο. Ας υποθέσουμε ότι μέχρι στιγμής δεν έχουμε δει κανένα αντικείμενο το οποίο να έχει βαθμό μεγαλύτερο ή ίσο από το τ. Συνεπώς, δεν μπορούμε μέχρι τώρα να πούμε ότι ξέρουμε ποιο είναι το καλύτερο αντικείμενο, καθώς το επόμενο αντικείμενο που μπορεί να δούμε σε κάποια λίστα μπορεί να έχει γενικό βαθμό ίσο με τ, που να είναι δηλαδή μεγαλύτερος από όλους τους βαθμούς που έχουμε δει. Από την άλλη, όταν δούμε ένα αντικείμενο με βαθμό τουλάχιστον τ, τότε είναι ασφαλές να σταματήσουμε τον αλγόριθμο, σύμφωνα με την απόδειξη ορθότητας που προηγήθηκε. Παρόμοια, για k διαφορετικό του 1 η διαίσθηση είναι κοίταζε σειριακά τα αντικείμενα, ως που να έχεις δει τα k καλύτερα.

Θεώρημα. Ο TA απαιτεί buffers σταθερού μεγέθους, που είναι ανεξάρτητοι από το μέγεθος της βάσης δεδομένων.

Απόδειξη. Η απόδειξη είναι αρκετά απλή. Το μόνο που χρειάζεται για να τρέξει ο TA είναι τα top-k αντικείμενα μέχρι εκείνη τη στιγμή και τους βαθμούς τους, καθώς και δείκτες στα τελευταία αντικείμενα που έχουν ιδωθεί με σειριακές προσπελάσεις των λιστών.

Σε αντίθεση, ο FA απαιτεί buffers που μπορεί να μεγαλώνουν χωρίς άνω φράγμα όσο το μέγεθος της βάσης δεδομένων αυξάνει, καθώς ο FA κάθε αντικείμενο που βλέπει με σειριακές προσπελάσεις σε όλες τις λίστες για να τα ελέγξει κατόπιν.

Το αρνητικό στοιχεί του TA όμως είναι ότι μπορεί να κάνει κάθε φορά που βλέπει ένα αντικείμενο μέχρι και m-1 τυχαίες προσπελάσεις στοιχείων, για να βρει τις τιμές όλων των πεδίων του αντικειμένου που είδε σε μία λίστα. Αυτό ισχύει ακόμα και αν σε προηγούμενη φάση το στοιχείο το είχε δει σε προηγούμενες φάσεις του τρεξίματος του αλγορίθμου, καθώς δεν κρατάει δεδομένα που θα του επέτρεπαν να μην χρειαστεί να ξανακάνει τυχαίες προσπελάσεις.

Τώρα θα παραθέσουμε, χωρίς απόδειξη, ένα σπουδαίο θεώρημα για τη σημασία του TA.

Ορισμός. Έστω **A** ένα σύνολο από αλγόριθμους και **D** μία κλάση από νόμιμες εισόδους σε αυτούς τους αλγόριθμους. Έστω και ένα μέτρο κόστους $cost(\mathcal{A}, \mathcal{D})$ που αντιπροσωπεύει το κόστος του αλγορίθμου $\mathcal{A}$ όταν τρέχει με είσοδο $\mathcal{D}$, ή στην περίπτωσή μας το κόστος του αλγορίθμου $\mathcal{A}$ όταν τρέχει πάνω στη βάση δεδομένων $\mathcal{D}$. Λέμε ότι ένας αλγόριθμος $\mathcal{B}$ είναι instance optimal επί του **A** και **D** όταν

$$cost(\mathcal{B}, \mathcal{D}) = O\big(cost(\mathcal{A}, \mathcal{D})\big)$$

για κάθε $\mathcal{A} \in \boldsymbol{A}$ και $\mathcal{D} \in \boldsymbol{D}$. Προφανώς $\mathcal{B} \in \boldsymbol{A}$.

Θεώρημα. Ο TA είναι instance optimal επί όλων των αλγορίθμων που βρίσκουν σωστά τα top-k στοιχεία και επί όλων των δυνατών βάσεων δεδομένων.

Ας δώσουμε τώρα και ένα παράδειγμα τρεξίματος του TA για να γίνει σαφέστερος ο τρόπος με τον οποίο λειτουργεί:

L_1
$(1,1)$
$(2,1)$
$(3,1)$
$\dots$
$(n+1,1)$
$(n+2,0)$
$(n+3,0)$
$\dots$
$(2n+1,0)$

L_2
$(2n+1,1)$
$(2n,1)$
$(2n-1,1)$
$\dots$
$(n+1,1)$
$(n,0)$
$(n-1,0)$
$\dots$
$(1,0)$

Εικόνα 2. Ένα παράδειγμα τρεξίματος του TA.

Υποθέτουμε ότι έχουμε μία βάση δεδομένων με 2n+1 στοιχεία, όπως αυτή που φαίνεται στην εικόνα Εικόνα 2. Αν υποθέσουμε ότι η aggregation συνάρτηση είναι η minimum τότε το κορυφαίο αντικείμενο όπως φαίνεται είναι το n+1. Υποθέτουμε ότι k=1. Τα βήματα του αλγορίθμου θα είναι τα επόμενα: πρώτα θα δει τα στοιχεία 1 στη λίστα 1 και 2n+1 στη λίστα 2. Θα καθορίσει την τιμή κατωφλίου κατ' αυτόν τον τρόπο στη μονάδα. Επειδή ούτε το 1 αντικείμενο, ούτε το 2n+1 έχουν γενικούς βαθμούς μεγαλύτερους ή ίσους από το 1 (έχουν και τα δύο 0) ο αλγόριθμος θα τρέξει ένα ακόμα βήμα. Θα δει τα αντικείμενα 2 και 2n στις λίστες 1 και 2 αντίστοιχα. Και πάλι η τιμή κατωφλίου θα είναι 1, αλλά οι γενικοί βαθμοί των αντικειμένων 2 και 2n είναι 0. Οπότε θα εκτελεστεί ένα ακόμα βήμα. Αυτή η διαδικασία θα συνεχίσει μέχρι να φτάσουμε στο αντικείμενο n+1. Τότε, η τιμή κατωφλίου θα είναι 1, αλλά το αντικείμενο n+1 θα έχει γενικό βαθμό 1 (καθώς min(1,1)=1) και άρα βρήκαμε το top-1 αντικείμενο και ο αλγόριθμος τερματίζει.

Ο αλγόριθμος είδε όλα τα στοιχεία πρακτικά (είτε με τυχαίες είτε με σειριακές προσπελάσεις). Το γεγονός αυτό παρατηρείται σε anticorrelated στοιχεία (αντίθετα συσχετισμένα στοιχεία δηλαδή) όπως είναι αυτά που έχουμε. Το στοιχείο αυτό του TA θα μελετηθεί αρκετά σε επόμενα στάδια της εργασίας, όπου θα μας ενδιαφέρει αν δύο διαστάσεις μία βάσης δεδομένων είναι correlated, anticorrelated ή τελείως ανεξάρτητες μεταξύ τους, για να εκτιμήσουμε τα βήματα που θα χρειαστεί ο TA για να βρει τα top-k αντικείμενα που αναζητούμε.

3.1.4 Όταν δεν μπορούμε να κάνουμε σειριακές προσπελάσεις σε κάποια δεδομένα

Σε πολλές περιστάσεις είναι αδύνατο να εκτελέσεις τυχαίες προσπελάσεις σε δεδομένα. Ένα πολύ εύστοχο παράδειγμα είναι αυτό που έδωσαν ο Bruno και άλλοι. Έστω ότι ένας χρήστης θέλει να πάρει πληροφορίες για εστιατόρια. Ο χρήστης αξιολογεί το κάθε εστιατόριο με τρία κριτήρια (ή πεδία): πόσο καλό είναι, πόσο φτηνό είναι και πόσο κοντά του είναι. Υπάρχουν τώρα τρεις διαφορετικές ιστοσελίδες οι οποίες αξιολογούν το κάθε εστιατόριο στα τρία κριτήρια που προαναφέρθηκαν. Όμως μόνο η ιστοσελίδα που αξιολογεί το πόσο καλό είναι ένα εστιατόριο επιτρέπει σειριακές προσπελάσεις στα δεδομένα.

Αυτό είναι ένα απλό παράδειγμα που δείχνει πως μπορεί να είναι απαραίτητος ένας αλγόριθμος ο οποίος επιτρέπει σειριακές προσπελάσεις σε κάποιες λίστες και τυχαίες προσπελάσεις σε κάποια άλλα πεδία των δεδομένων.

Ακολούθως θα εξετάσουμε έναν αλγόριθμο ο οποίος θεωρεί ότι κάποια πεδία των δεδομένων μπορούμε να τα προσπελάσουμε σειριακά, ενώ κάποια άλλα μόνο τυχαία. Ο αλγόριθμος αυτός είναι ο TA_Z.

3.1.4.1 Ο αλγόριθμος TA_Z

Στον αλγόριθμο TA_Z γίνονται τυχαίες προσπελάσεις δεδομένων μόνον σε ορισμένες από όλες τις διαστάσεις των δεδομένων.

Έστω ότι Z είναι το σύνολο των δεικτών των λιστών που μπορούμε να προσπελάσουμε σειριακά. Υποθέτουμε ότι το Z δεν είναι κενό και πως ο πληθάριθμός του είναι ίσος με m'. Τότε ο TA_Z θα είναι η ακόλουθη παραλλαγή του TA:

Threshold Algorithm Z (TA_Z)

---

1. Κάνε παράλληλες προσπελάσεις στα δεδομένα της κάθε μίας από τις m' ταξινομημένες λίστες L_i $i \in Z$. Για κάθε αντικείμενο *R* που βλέπεις σε μία λίστα, κάνε τυχαίες προσπελάσεις στις υπόλοιπες λίστες ώστε να βρεις τις τιμές του *R* σε όλα τα πεδία του. Υπολόγισε το $t(R) = t(x_1, x_2, \dots, x_m)$. Αν το $t(R)$ είναι ένα από τα k υψηλότερα που έχεις δει ως τώρα, να θυμάσαι το *R* και το γενικό βαθμό του $t(R)$.
2. Για κάθε ταξινομημένη λίστα L_i $i \in Z$ ας είναι $\underline{x_i}$ το τελευταίο στοιχείο που έχει ιδωθεί με σειριακές προσπελάσεις. Επίσης, για κάθε λίστα L_i $i \notin Z$ θεωρούμε ότι $\underline{x_i} = 1$. Ορίζουμε ως τιμή κατωφλίου την τιμή $\tau = t\left(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}\right)$. Όταν έχεις δει τουλάχιστον k αντικείμενα με τιμή μεγαλύτερη ή ίση με το τ, τότε σταμάτα.
3. Ας είναι *Y* το σύνολο των αντικειμένων που έχουν ιδωθεί με τους k καλύτερους βαθμούς. Η έξοδος του αλγορίθμου είναι το σύνολο $\{(R, t(R)): R \in Y\}$.

Ένα αξιόλογο θεώρημα για τον TA_Z είναι το ακόλουθο.

Θεώρημα. Αν **A** είναι η κλάση όλων των αλγορίθμων που βρίσκουν σωστά τα top-k αντικείμενα της κλάσης **D** όλων των πιθανών βάσεων δεδομένων κάνοντας σειριακές προσπελάσεις αντικειμένων μόνον στις λίστες L_i $i \in Z$, τότε ο TA_Z είναι instance optimal επί του **A** και **D**.

Το παραπάνω θεώρημα παρατίθεται χωρίς απόδειξη, απλά για να γίνει κατανοητό πόσο ισχυρή είναι και αυτή η παραλλαγή του TA. Ο TA_Z είναι (με απλά λόγια) ο καλύτερος δυνατός αλγόριθμος που μπορούμε να έχουμε όταν μπορούμε να κάνουμε σειριακές προσπελάσεις μόνον στις λίστες με L_i $i \in Z$.

L_1
$(R, 1)$
$\ldots$
$(\cdot, 0.7)$

L_2
$\ldots$
$(R, 0.6)$
$\ldots$

L_3
$(R, 1)$
$\ldots$
$\ldots$

Εικόνα 3. Παράδειγμα τρεξίματος του TA_Z.

Ας δούμε τώρα ένα παράδειγμα τρεξίματος του TA_Z όπου μόνο τα στοιχεία της πρώτης διάστασης μπορούμε να τα προσπελάσουμε σειριακά. Δηλαδή, $Z = \{1\}$. Ας είναι η $t(x, y, z) = mix(x, y)$ $\alpha\nu$ $z = 1$ και $t(x, y, z) = \frac{mix(x,y,z)}{2}$ $\alpha\nu$ $z \neq 1$. Φαίνεται πολύ εύκολα ότι η t είναι μονότονη και αυστηρή (βλέπετε αντίστοιχα για τους ορισμούς στις ενότητες 3.1.1 και 3.1.2). Στη βάση αυτή υποθέτουμε επίσης ότι δεν είναι δυνατόν δύο διαφορετικά δεδομένα να έχουν την ίδια τιμή στο ίδιο πεδίο.

Είναι φανερό από την εικόνα Εικόνα 3 ότι το αντικείμενο *R* έχει γενικό βαθμό 0.6. Για κάθε αντικείμενο *R'* με διαφορετικό από το *R* ο βαθμός στη λίστα 3 είναι σίγουρα διάφορος του 1 (λόγω της υπόθεσης που κάναμε σχετικά με τη διαφορετικότητα των τιμών για τη βάση μας) και άρα $t(R') \leq 0.5$. Για αυτό και το *R* είναι το top-1 αντικείμενο.

Ο TA_Z θα έκανε σειριακές προσπελάσεις σε όλα τα δεδομένα της λίστας 1, μέχρι να φτάσει στο μικρότερο, που έχει βαθμό 0.7. Τότε θα ανακάλυπτε ότι ούτε καν αφού είδε όλα τα στοιχεία στη λίστα 1 δεν κατάφερε να δώσει τη σωστή απάντηση, παρ' όλο που την «εγνώριζε» από την αρχή. Αυτό συμβαίνει στο συγκεκριμένο παράδειγμα διότι η εκτίμηση που δίνει η τιμή κατωφλίου είναι πολύ συντηρητική. Γενικά όμως ο αλγόριθμος κάνει το καλύτερο που θα μπορούσε να κάνει ένας αλγόριθμος με τις ελευθερίες που του έχουμε δώσει (δηλαδή σειριακές προσπελάσεις μόνο σε μία λίστα και μόνον τυχαίες στις υπόλοιπες).

3.1.5 Ελαχιστοποιώντας τις τυχαίες προσπελάσεις

Ως εδώ δεν έχουμε μελετήσει καθόλου την επίδραση των τυχαίων προσπελάσεων στο συνολικό κόστος του τρεξίματος ενός αλγορίθμου. Υπενθυμίζουμε ότι το συνολικό κόστος ενός αλγορίθμου, όπως το μετράμε σε αυτήν εδώ την εργασία τουλάχιστον, είναι το $rc_r + sc_s$, δηλαδή το πλήθος των τυχαίων προσπελάσεων που λαμβάνουν χώρα επί το κόστος της μίας τυχαίας προσπέλασης συν το πλήθος των σειριακών προσπελάσεων επί το κόστος της μία σειριακής προσπέλασης.

Υπάρχουν πολλές περιπτώσεις που υποχρεωνόμαστε είτε να μην έχουμε καθόλου τυχαίες προσπελάσεις είτε μας συμφέρει να τις μειώσουμε σε πολύ μεγάλο βαθμό. Κατ' αρχάς, στα αποτελέσματα που μας επιστρέφει μία μηχανή αναζήτησης επί παραδείγματι δεν έχουμε τη δυνατότητα να εκτελέσουμε καμία τυχαία προσπέλαση. Πάντοτε κάνουμε σειριακές προσπελάσεις σε τέτοιου είδους αποτελέσματα. Σε αυτήν την περίπτωση το κόστος είναι $c_r = \infty$. Από την άλλη, όταν οι προσπελάσεις σε δεδομένα ταυτίζονται με αναγνώσεις ή εγγραφές στους δίσκους, τότε το σειριακό κόστος είναι κατά πολύ μικρότερο από το κόστος των τυχαίων προσπελάσεων.

Για αυτές τις περιπτώσεις θέλουμε αλγορίθμους που είτε δεν κάνουν καθόλου τυχαίες προσπελάσεις στα δεδομένα είτε κάνουν πολύ προσεκτικά τέτοιου είδους προσπελάσεις. Ο No Random Accesses (NRA) είναι ο αλγόριθμος που δεν κάνει καθόλου τυχαίες προσπελάσεις σε δεδομένα, ενώ ο Combined Algorithm (CA) είναι ο αλγόριθμος που λαμβάνει υπ' όψιν του τα κόστη των τυχαίων και των σειριακών προσπελάσεων και τρέχει κατάλληλα.

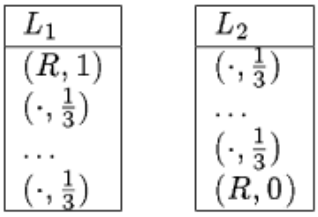
Και οι δύο αλγόριθμοι σταματούν στο σημείο εκείνο που ξέρουν ότι καμία βελτίωση δεν μπορεί να πραγματοποιηθεί. Για κάθε αντικείμενο *R* κρατάμε τα πεδία στα οποία έχει γίνει είτε σειριακή είτε τυχαία προσπέλαση και έτσι έχουμε το σύνολο $S(R) = \{i_1, i_2, \dots, i_l\} \subseteq \{1,2, \dots, m\}$, που δηλώνει ότι οι τιμές $x_{i_1}, x_{i_2}, \dots, x_{i_l}$ είναι γνωστές. Έτσι, μπορούμε για κάθε αντικείμενο να διατηρούμε ένα άνω και ένα κάτω όριο για το $t(R)$. Οι αλγόριθμοι σταματούν όταν δεν υπάρχει κάποιο στοιχείο που να έχει μεγαλύτερο άνω φράγμα του γενικού του βαθμού από το κάτω φράγμα του k-οστού αντικειμένου των top-k.

Τώρα για κάθε αντικείμενο *R* για το οποίο γνωρίζουμε το προαναφερθέν σύνολο $S(R) = \{i_1, i_2, \dots, i_l\}$ που προαναφέραμε, είναι εύκολο να βρούμε ένα άνω φράγμα του γενικού του βαθμού και ένα κάτω φράγμα του γενικού του βαθμού. Το κάτω φράγμα είναι το $W_S(R) = t\big(x_{i_1}, x_{i_2}, \dots, x_{i_l}, 0,0, \dots, 0\big)$. Παρόμοια, το άνω φράγμα του γενικού του βαθμού θα είναι το $B_S(R) = t\left(x_{i_1}, x_{i_2}, \dots, x_{i_l}, \underline{x_{i_{l+1}}}, \underline{x_{i_{l+2}}}, , \dots, \underline{x_m}\right)$. Το γεγονός ότι $t(R) \geq W_S(R)$ και $t(R) \geq B_S(R)$ προκύπτει πολύ εύκολα από την προηγηθείσα ανάλυση.

3.1.5.1 No Random Accesses (NRA)

Σε αυτή την ενότητα θα μελετήσουμε τον αλγόριθμο NRA. Θα κάνουμε την υπόθεση ότι πλέον δεν θέλουμε τα top-k αντικείμενα ταξινομημένα σύμφωνα με τους γενικούς τους βαθμούς, αλλά απλά αναφέρονται με μία τυχαία σειρά. Αυτό το κάνουμε διότι δεν γνωρίζουμε ακριβώς το γενικό βαθμό του κάθε αντικειμένου, παρά μόνο ένα άνω φράγμα και ένα κάτω φράγμα. Βάσει αυτών των φραγμάτων προσπαθούμε να σταματήσουμε την εκτέλεση του αλγορίθμου κατά το δυνατόν πιο σύντομα, ώστε να έχουμε τις λιγότερες δυνατές ανακλήσεις δεδομένων.

Ας δούμε ένα παράδειγμα μίας βάσης δεδομένων για να καταλάβουμε πως δεν είναι απαραίτητο να γνωρίζουμε ακριβώς τις τιμές του κάθε αντικειμένου σε όλες τις διαστάσεις. Μας ενδιαφέρουν μόνο τα φράγματα του πραγματικού γενικού βαθμού του αντικειμένου. Η βάση δεδομένων μας θα είναι αυτή που απεικονίζεται στην εικόνα Εικόνα 4.



Εικόνα 4. Ένα παράδειγμα που βοηθά στην κατανόηση του NRA.

Θα υποθέσουμε ότι αναζητούμε το top-1 αντικείμενο της βάσης δεδομένων. Η βάση δεδομένων υποθέτουμε ότι έχει σε όλα τα αντικείμενα τιμές, και στα δύο πεδία, $\frac{1}{3}$, εκτός από το αντικείμενο *R* που έχει στο πρώτο πεδίο τιμή 1 και στο δεύτερο 0. Επίσης, κάνουμε την υπόθεση ότι η aggregation συνάρτηση που θα χρησιμοποιήσουμε είναι η average. Τότε, όλα τα αντικείμενα της βάσης δεδομένων έχουν γενικό βαθμό $\frac{1}{3}$, εκτός από το αντικείμενο *R* που έχει γενικό βαθμό $\frac{1}{2}$ και αυτό είναι το top-1 αντικείμενο. Εάν θέλουμε να βρούμε το γενικό βαθμό του αντικειμένου *R* τότε θα πρέπει να κάνουμε σειριακές ανακλήσεις δεδομένων σε ολόκληρη τη λίστα 2. Αυτό προφανώς δεν είναι αποδοτικό. Αν θέλουμε να έχουμε με ασφάλεια τη σωστή λύση, αρκεί να κάνουμε 2 σειριακές ανακλήσεις δεδομένων στην πρώτη λίστα και μία στη δεύτερη. Τότε θα ξέρουμε ότι το μέγιστο που θα μπορούν να έχουν τα αντικείμενα που δεν έχουμε δει είναι $\frac{1}{3}$ ενώ η ελάχιστη τιμή του γενικού βαθμού του αντικειμένου *R* είναι $\frac{1}{2}$.

Ας δούμε τώρα τον αλγόριθμο NRA.

No Random Accesses (NRA)

---

1. Κάνε παράλληλες προσπελάσεις στα δεδομένα της κάθε μίας από τις m ταξινομημένες λίστες L_i. Σε βάθος d (δηλαδή όταν έχουν ιδωθεί d αντικείμενα από κάθε λίστα με σειριακές προσπελάσεις) κάνε τα εξής:
    a. Διατήρησε τις τιμές $\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}$, που είναι οι τελευταίες που έχουν ιδωθεί στην κάθε λίστα.
    b. Για κάθε αντικείμενο *R* που έχουν ιδωθεί οι διαστάσεις του $S(R) = \{i_1, i_2, \dots, i_l\}$, υπολόγισε τις τιμές $W(R)$ και $B(R)$. (Για αντικείμενα που δεν έχουν ακόμα ιδωθεί θεωρήστε ότι $W(R) = t(0,0,\dots,0)$ και $B(R) = t(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m})$.)
    c. Ας είναι T_k η λίστα με τα k αντικείμενα με τα μεγαλύτερα $W(R)$ που έχουμε ως τώρα. Αν έχουμε δύο αντικείμενα με την ίδια τιμή $W(R)$ τότε προηγείται στη λίστα αυτό που έχει μεγαλύτερο $B(R)$. Ας είναι M_k η k-οστή χειρότερη τιμή του $W(R)$ που παρουσιάζεται στη λίστα T_k.
2. Ένα αντικείμενο *R* θεωρείται viable αν $B(R) > M_k$. Σταμάτα όταν έχουν ιδωθεί k διαφορετικά αντικείμενα (δηλαδή η λίστα T_k περιλαμβάνει k διαφορετικά μεταξύ τους αντικείμενα) και δεν υπάρχουν viable αντικείμενα εκτός της λίστας (δηλαδή για κάθε αντικείμενο που δεν ανήκει στη λίστα T_k ισχύει $B(R) \leq M_k$). Επίστρεψε τα αντικείμενα που βρίσκονται στη λίστα T_k.

Ας αποδείξουμε τώρα την ορθότητα του NRA.

Απόδειξη ορθότητας του NRA. Αν η συνάρτηση t είναι μονότονη τότε ο NRA βρίσκει τα top-k αντικείμενα.

Έστω ότι ο NRA σταματά μετά από d βήματα. Τότε η λίστα T_k θα έχει τη μορφή $T_k = \{R_1, R_2, \dots, R_k\}$. Τότε βέβαια η έξοδος του αλγορίθμου είναι $R_1, R_2, \dots, R_k$. Θα δείξουμε ότι αν το αντικείμενο R δεν είναι ένα από τα $R_1, R_2, \dots, R_k$, τότε θα πρέπει να ισχύει $t(R) \leq t(R_i)$ για κάθε i. Αυτό και θα δείξουμε ακολούθως.

Εφόσον ο αλγόριθμος σταματά μετά από d βήματα, δηλαδή σε βάθος d, ξέρουμε ότι το *R* δεν είναι viable. Αυτό σημαίνει ότι $B(R) \leq M_k$. Γνωρίζουμε όμως επίσης ότι ισχύει $t(R) \leq B(R)$. Επίσης για καθένα από τα k αντικείμενα $R_1, R_2, \dots, R_k$ έχουμε $M_k \leq W(R_i) \leq t(R_i)$.

Από τα ανωτέρω είναι φανερό ότι $t(R) \leq t(R_i)$, για κάθε i, όπως και επιθυμούσαμε.

Συνεπώς ο NRA είναι σωστός αλγόριθμος. Όμως δεν είναι αυτό που χαρακτηρίζει την ποιότητά του. Το σημαντικό του χαρακτηριστικό είναι το γεγονός ότι ο NRA είναι instance optimal επί όλων των αλγορίθμων που δεν χρησιμοποιούν τυχαίες προσπελάσεις επί όλων των δυνατών βάσεων δεδομένων.

Σε αυτή τη διπλωματική εργασία θα χρησιμοποιηθεί ο NRA. Αυτό πραγματοποιείται διότι ο NRA μας επιτρέπει να μην κάνουμε τυχαίες προσπελάσεις δεδομένων, κάτι που κοστίζει πολύ όταν «τραβάμε» δεδομένα από το δίσκο, όπως στην περίπτωσή μας.

3.1.5.2 Combined Algorithm

Ο Combined Algorithm (CA) κάνει τυχαίες προσπελάσεις δεδομένων, αλλά λαμβάνει υπ' όψιν του το κόστος των τυχαίων και τον σειριακών προσπελάσεων. Για τους σκοπούς αυτής της ενότητας θα υποθέσουμε ότι $c_R > c_S$ και ότι $h = \left\lfloor \frac{c_R}{c_S} \right\rfloor \geq 1$. Η ιδέα του CA είναι να τρέχουμε τον NRA και κάτι h βήματα να κάποιες τυχαίες προσπελάσεις στοιχείων.

Ο αλγόριθμος CA είναι ο ακόλουθος:

Combined Algorithm (CA)

1. Κάνε παράλληλες προσπελάσεις στα δεδομένα της κάθε μίας από τις m ταξινομημένες λίστες L_i. Σε βάθος d (δηλαδή όταν έχουν ιδωθεί d αντικείμενα από κάθε λίστα με σειριακές προσπελάσεις) κάνε τα εξής:
    a. Διατήρησε τις τιμές $\underline{x_1}, \underline{x_2}, \dots, \underline{x_m}$, που είναι οι τελευταίες που έχουν ιδωθεί στην κάθε λίστα.
    b. Για κάθε αντικείμενο *R* που έχουν ιδωθεί οι διαστάσεις του $S(R) = \{i_1, i_2, \dots, i_l\}$, υπολόγισε τις τιμές $W(R)$ και $B(R)$. (Για αντικείμενα που δεν έχουν ακόμα ιδωθεί θεωρήστε ότι $W(R) = t(0,0,\dots,0)$ και $B(R) = t(\underline{x_1}, \underline{x_2}, \dots, \underline{x_m})$.)

c. Ας είναι T_k η λίστα με τα k αντικείμενα με τα μεγαλύτερα $W(R)$ που έχουμε ως τώρα. Αν έχουμε δύο αντικείμενα με την ίδια τιμή $W(R)$ τότε προηγείται στη λίστα αυτό που έχει μεγαλύτερο $B(R)$. Ας είναι M_k η k-οστή χειρότερη τιμή του $W(R)$ που παρουσιάζεται στη λίστα T_k.

2. Ένα αντικείμενο R θεωρείται viable αν $B(R) > M_k$. Κάθε $h = \left\lfloor \frac{c_R}{c_S} \right\rfloor$ κάνε το ακόλουθο: πάρε ένα αντικείμενο που δεν έχεις δει όλα τα πεδία του και κάνε τυχαίες προσπελάσεις στις λίστες. Κατά μέγιστο θα γίνουν m-1 τυχαίες προσπελάσεις. Αν δεν υπάρχει τέτοιο αντικείμενο μην κάνεις κάποια τυχαία προσπέλαση σε αυτό το βήμα.
3. Σταμάτα όταν έχουν ιδωθεί k διαφορετικά αντικείμενα (δηλαδή η λίστα T_k περιλαμβάνει k διαφορετικά μεταξύ τους αντικείμενα) και δεν υπάρχουν viable αντικείμενα εκτός της λίστας (δηλαδή για κάθε αντικείμενο που δεν ανήκει στη λίστα T_k ισχύει $B(R) \leq M_k$). Επίστρεψε τα αντικείμενα που βρίσκονται στη λίστα T_k.

Είναι φανερό ότι όταν το h είναι πολύ μεγάλο τότε ο CA πρακτικά γίνεται NRA, καθώς δεν θα γίνουν ποτέ περισσότερα από h βήματα στον CA.

Το γεγονός ότι ο CA επιστρέφει τα σωστά αποτελέσματα είναι φανερό.

3.1.6 Σημασία του TA για την εργασία μας

Εμείς θα χρησιμοποιήσουμε τον TA (συγκεκριμένα θα χρησιμοποιήσουμε την παραλλαγή του NRA που μελετήσαμε). Αυτό που θα κάνουμε είναι να έχουμε ένα B^+-tree, να ξεκινάμε από το σημείο ερώτησης και κατόπιν πηγαίνοντας δεξιά και αριστερά να επεκτεινόμαστε. Το αν θα πάμε δεξιά ή αριστερά εξαρτάται από το ποιο από τα σημεία είναι πιο κοντά στα σημεία ερώτησης. Για παράδειγμα, αν έχουμε στα φύλλα ενός B-δέντρου τις τιμές 2, 3, 4, 5, 6, 7, 10, 12, 14, 17 και το σημείο του ερωτήματος είναι το 10, τότε θα ξεκινήσουμε από το σημείο 10, μετά θα πάμε στο 12 (καθώς το 12 είναι πιο κοντά από το 7 στο 10, που είναι το σημείο ερώτησης), μετά στο 7 (καθώς το 7 είναι πιο κοντά από το 15 στο 10) κ.ο.κ.. Με αυτόν τον τρόπο η απόσταση του σημείου πάνω στο B-δέντρο από το σημείο ερώτησης αυξάνεται σταδιακά σε κάθε βήμα. Έτσι έχουμε ένα «βαθμό» που εμείς (σε αντίθεση με τον TA που περιγράψαμε) τον θέλουμε μικρό, γιατί ψάχνουμε τον πλησιέστερο γείτονα. Αυτό δεν αλλάζει κάτι ιδιαίτερο στους αλγορίθμους που περιγράψαμε. Το μόνο που αλλάζει είναι η φορά των ανισόσεων. Έχουμε δηλαδή, μία δομή που λειτουργεί σαν τις λίστες που είχαμε στον TA. Αυτή η δομή είναι τα B-δέντρα.

Θα χρησιμοποιήσουμε επίσης και τα R-trees στον NRA που θα κατασκευάσουμε. Στα R-trees έχουμε τη δυνατότητα να κάνουμε αυξητικά ερωτήματα πλησιέστερου γείτονα με έναν αλγόριθμο που θα περιγράψουμε στο ακόλουθο κεφάλαιο. Δηλαδή, μπορούμε να έχουμε ένα R-tree και να ζητήσουμε τον πιο κοντινό γείτονα προς ένα δεδομένο, μετά το επόμενο πιο κοντινό κ.ο.κ.. Με αυτόν τον τρόπο έχουμε μία ακόμα δομή που μας επιστρέφει αντικείμενα βάσει του βαθμού τους και μάλιστα ξεκινά από τα αντικείμενα με

μικρούς βαθμούς και φτάνει στα αντικείμενα με μεγαλύτερους βαθμούς (ο βαθμός εδώ ταυτίζεται με την έννοια της απόστασης).

Στο επόμενο κεφάλαιο θα εξετάσουμε συνοπτικά τον τρόπο με τον οποίο βρίσκεται (αυξητικά) ο επόμενος πλησιέστερος γείτονας κάθε φορά. Για παράδειγμα, θα εξηγήσουμε ότι όταν τρέχει ο εν λόγω αλγόριθμος στο πρώτο βήμα επιστρέφει τον πλησιέστερο γείτονα, στο δεύτερο γείτονα επιστρέφει τον αμέσως επόμενο πλησιέστερο γείτονα κ.ο.κ.. Αυτά θα τα δούμε στο επόμενο κεφάλαιο.

3.2 Δομές ευρετηρίων που θα χρησιμοποιηθούν

3.2.1 Εισαγωγή στα B⁺-trees

Σε αυτήν την ενότητα θα μελετήσουμε κάποια στοιχεία από τα B$^+$-trees που πρέπει να είναι γνωστά για αυτή τη διπλωματική εργασία.

Η δομή των B$^+$-trees είναι η πιο ευρέως χρησιμοποιούμενη σε δομές ευρετηρίου που διατηρούν την αποτελεσματικότητά τους για δυναμικά δεδομένα, δηλαδή για δεδομένα στα οποία επιτρέπονται διαγραφές παρόντων δεδομένων και εισαγωγές νέων δεδομένων. Ένα B$^+$-tree έχει τη μορφή ενός ισορροπημένου δέντρου, στο οποίο η διαδρομή από τη ρίζα προς οποιοδήποτε φύλλο είναι ίδιου μήκους. Κάθε κόμβος του δέντρου αυτού (εκτός και αν είναι φύλλο) έχει από $\lceil n/2 \rceil$ ως και n παιδιά, όπου το n είναι σταθερό για ένα δέντρο.

Ας μελετήσουμε τώρα τη δομή ενός B$^+$-tree.

Ένας τυπικός κόμβος ενός B$^+$-tree είναι σαν αυτόν που φαίνεται στην εικόνα Εικόνα 5. Όπως φαίνεται περιέχει $n-1$ τιμές αναζήτησης $K_1, K_2, \dots, K_{n-1}$ και n δείκτες $P_1, P_2, \dots, P_n$. Οι τιμές του κάθε κόμβου διατηρούνται ταξινομημένες έτσι ώστε αν $i < j$ να είναι και $K_i < K_j$.

P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

Εικόνα 5. Ένας κόμβος ενός κλασικού B$^+$-tree.

Ας δούμε τώρα τη δομή των φύλλων του B$^+$-tree. Για $i = 1,2, \dots, n-1$ ο δείκτης P_i δείχνει είτε σε μία εγγραφή του αρχείου με τιμή κλειδιού K_i ή σε μια θέση (bucket) με δείκτες καθένας από τους οποίους δείχνει σε μία εγγραφή του αρχείου με τιμή κλειδιού K_i (προφανώς τα buckets θα χρησιμοποιούνται όταν το κλειδί αναζήτησης δεν σχηματίζει πρωτεύον κλειδί). Δεν συζητήσαμε προς το παρόν τη χρησιμότητα του δείκτη P_n, την οποία θα κατανοήσουμε όμως στη συνέχεια. Θα πρέπει να αναφέρουμε επίσης ότι κατά μέγιστο

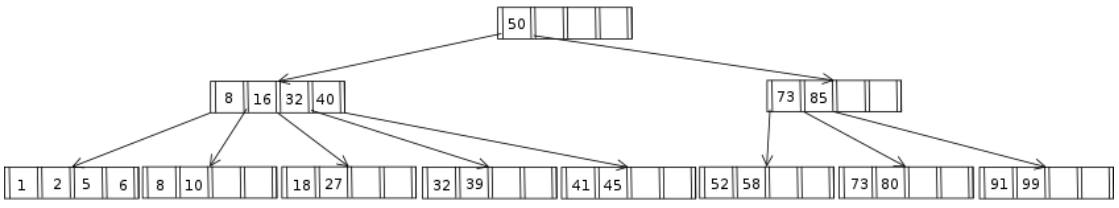
ένα B⁺-tree έχει $n-1$ τιμές κλειδιών και κατ' ελάχιστον έχει $\left\lceil {(n-1)}/{2} \right\rceil$. Αξίζει επίσης να αναφερθεί ότι το εύρος των τιμών στο οποίο κινείται το κάθε φύλλο είναι ξένο προς το εύρος των τιμών στο οποίο κινείται το οποιοδήποτε άλλο φύλλο. Μάλιστα, όχι απλά είναι ξένα τα δύο σύνολα διαστήματα τιμών. Αν υποθέσουμε ότι το L_i είναι φύλλο, όπως και το L_j με $i < j$, τότε οποιαδήποτε τιμή του φύλλου i (δηλαδή του αριστερού φύλλο) είναι μικρότερη από οποιαδήποτε τιμή του φύλλου j. Τώρα θα καταστήσουμε σαφές και το λόγο ύπαρξης του δείκτη P_n. Ο δείκτης αυτός δείχνει προς το αμέσως πιο δεξιό φύλλο από το τρέχον, δηλαδή το φύλλο που έχει «μία σκάλα» πιο αυξημένες τιμές κλειδιών από το φύλλο στο οποίο αναφερόμαστε[4]. Ο δείκτης P_n μας επιτρέπει να επεξεργαζόμαστε σειριακά το αρχείο πάρα πολύ αποδοτικά.

Τώρα ας εξετάσουμε λεπτομερώς τους εσωτερικούς κόμβους του B⁺-tree. Η δομή και αυτών των κόμβων είναι η ίδια με τη δομή των κόμβων φύλλων, αλλά οι δείκτες των μη φύλλων κόμβων δείχνουν σε άλλους κόμβους. Ένας εσωτερικός κόμβος μπορεί να περιέχει από $\lceil {n}/{2} \rceil$ ως n δείκτες. Όμως πού δείχνουν αυτοί οι δείκτες; Ας υποθέσουμε ότι διαθέτουμε έναν κόμβο με m δείκτες. Η λογική είναι η εξής: για $i = 2,3,\dots,m-1$ ο δείκτης P_i δείχνει στο υποδέντρο που περιέχει τιμές κλειδιών μικρότερες από K_i και μεγαλύτερες ή ίσες με το K_{i-1}. Ο δείκτης P_m δείχνει στο υποδέντρο που έχει τιμές κλειδιών μεγαλύτερες από K_m, ενώ ο δείκτης P_1 δείχνει στο υποδέντρο με τιμές κλειδιών μικρότερες από K_1.

Τώρα θα εξετάσουμε τη ρίζα που είναι κατά κάποιο τρόπο λίγο ιδιόρρυθμη. Η ρίζα μπορεί να περιέχει λιγότερους από $\lceil {n}/{2} \rceil$ δείκτες, αλλά πρέπει να περιέχει περισσότερους από 2 (εκτός και αν το δέντρο αποτελείται μόνο από τη ρίζα, οπότε μπορεί να αποτελείται και μόνο από ένα δείκτη).

Έχει αποδειχτεί ότι είναι πάντα δυνατό να κατασκευάσουμε ένα δέντρο που να ικανοποιεί όλες τις προαναφερθείσες ιδιότητες.

Στην εικόνα φαίνεται ένα δέντρο, όπου τα κλειδιά είναι ακέραιοι από το 1 ως το 7. Είναι χαρακτηριστικά παραδείγματα που βρέθηκαν στη Wikipedia.



Εικόνα 6. Ένα απλό B⁺-tree με $n = 5$.

[4] Στα δικά μας πειράματα θα έχουμε και ένα δείκτη P_{-1}, ο οποίος θα δείχνει προς το αμέσως πιο αριστερά φύλλο. Αυτό θα μας εξυπηρετεί σημαντικά όταν θα πρέπει να βρίσκουμε αντικείμενα με τιμές σε μία τους διάσταση πιο μικρές από το τρέχον αντικείμενο που εξετάζουμε.

3.2.2 Εισαγωγή στα R-trees

Μιας και η εργασία μας θα βασιστεί σε μεγάλο μέρος στα R-trees ως δομή ευρετηρίου, θα αναφέρουμε εδώ κάποια βασικά πράγματα πάνω στα R-trees.

Γενικά, οι περισσότερες δομές ευρετηρίων που είχαν μελετηθεί μέχρι και τη δημοσίευση του [4] δεν επέτρεπαν ούτε αποδοτική αναζήτηση σε πολυδιάστατα δεδομένα. Είχαν προταθεί αρκετές δομές, οι οποίες όμως είτε είχαν τη δυνατότητα να χειρίζονται μόνο συγκεκριμένους τύπους δεδομένων (για παράδειγμα τα KDB-trees ήταν χρήσιμα μόνο για σημειακά δεδομένα) είτε δεν μπορούσαν να χρησιμοποιηθούν σε δυναμικά δεδομένα (όπως οι μέθοδοι κελιών που δεν μπορούσαν να αποδώσουν καλά για δυναμικά δεδομένα καθώς τα όρια των κελιών πρέπει να αποφασιστούν εκ των προτέρων).

Το R-tree είναι ένα ισοζυγισμένο δέντρο παρόμοιο με το B-tree, στο οποίο οι εγγραφές των φύλλων περιέχουν δείκτες σε δεδομένα. Ο κάθε κόμβος αντιστοιχεί σε μία σελίδα του δίσκου. Η δομή επιτρέπει διαγραφές και εισαγωγές νέων στοιχείων και είναι τελείως δυναμική.

Η δική μας βάση δεδομένων (αυτή που θα χρειαστούμε για να εκτελέσουμε τα πειράματά μας) αποτελείται από ένα αναγνωριστικό. Έτσι, τα φύλλα του R-tree περιλαμβάνουν εγγραφές της μορφής

$$(I, tuple - id)$$

όπου $tuple - id$ αναφέρεται στο tuple της βάσης δεδομένων και το I είναι το n-διάστατο ορθογώνιο κουτί που περιλαμβάνει το δεδομένο μας (το σημείο στην περίπτωσή μας). Δηλαδή,

$$I = (I_0, I_1, \dots, I_{n-1})$$

Προφανώς το n είναι το πλήθος των διαστάσεων που έχουμε και I_i είναι ένα κλειστό διάστημα της μορφής $[a, b]$ που περιγράφει πόσο εκτείνεται το αντικείμενο μας (στην περίπτωση το σημείο μας, οπότε έχουμε διαστήματα της μορφής $[a, a]$) στη διάσταση i. Τώρα, οι κόμβοι του δέντρου που δεν είναι φύλλα έχουν εγγραφές της μορφής

$$(I, child - pointer)$$

όπου child-pointer είναι ένας δείκτης σε ένα σε έναν κόμβο κατώτερου επιπέδου και το I περιλαμβάνει όλα τα ορθογώνια των εγγραφών των κατώτερων κόμβων.

Ας είναι M ο μέγιστος αριθμός εγγραφών που χωρούν σε ένα κόμβο και $m \leq {}^{M}/_{2}$ μία παράμετρος που καθορίζει τον ελάχιστο αριθμό εγγραφών σε ένα κόμβο. Τότε, ένα R-tree ικανοποιεί τις ακόλουθες ιδιότητες:

- Κάθε φύλλο περιέχει από m μέχρι M εγγραφές, εκτός και αν είναι η ρίζα
- Για κάθε εγγραφή, κάθε κόμβου του δέντρου $(I, tuple - id)$ σε ένα φύλλο το I είναι το μικρότερο δυνατό ορθογώνιο που περιλαμβάνει το n-διάστατο δεδομένο που αντιπροσωπεύει το tuple στο οποίο αναφερόμαστε

- Κάθε κόμβος του δέντρου που δεν είναι φύλλο έχει μεταξύ m και M εγγραφών, εκτός και αν είναι η ρίζα
- Για κάθε εγγραφή, κάθε κόμβου του δέντρου $(I, child - pointer)$ σε έναν κόμβο που δεν είναι φύλλο, το I είναι το μικρότερο δυνατό ορθογώνιο που περιλαμβάνει όλα τα ορθογώνια των παιδιών του κόμβου
- Η ρίζα έχει τουλάχιστον δύο φύλλα, εκτός και αν είναι φύλλο
- Όλα τα φύλλα εμφανίζονται στο ίδιο επίπεδο

Στην εικόνα Εικόνα 7 που ακολουθεί, φαίνεται ένα χαρακτηριστικό παράδειγμα ενός R-tree στο οποίο φαίνονται και οι επικαλύψεις μεταξύ ορθογωνίων καθώς και οι σχέσεις περίληψης ορθογωνίων.

Το ύψος ενός R-tree που περιλαμβάνει N δεδομένα είναι κατά μέγιστο $log_m N - 1$, καθώς κάθε κόμβος έχει κατ΄ ελάχιστον m παιδιά. Επίσης, το ελάχιστο του utilization είναι (για όλους τους κόμβους εκτός από τη ρίζα) $\frac{m}{M}$.

Τα R-trees άλλαξαν αρκετά τα πράγματα σε δεδομένα πολλών διαστάσεων και για μία μεγάλη περίοδο πραγματοποιήθηκε μεγάλη έρευνα στον τομέα. Διάφορες παραλλαγές των R-trees εμφανίστηκαν, όπως το R^+-tree και το R^*-tree. Πλέον θεωρείται ότι το R^*-tree αποτελεί τη λύση για δεδομένα μεσαίας κλίμακας διαστάσεων. Για πολυδιάστατα όμως δεδομένα δεν έχει βρεθεί κάποια κοινά αποδεκτή δομή ευρετηρίου. Εμείς θα χρησιμοποιήσουμε τα R^*-trees για την εργασία μας, αφού αυτά θεωρούνται και ότι καλύτερο για μεσαίας διάστασης δεδομένα. Μία ανάλυση των R^*-trees γίνεται στο [8].

Η διαφορά των R^*-trees με τα R-trees έγκειται κυρίως στον αλγόριθμο εισαγωγής στοιχείων. Ο Beckman και η ομάδας του είδαν βελτιώσεις ως και 50% σε σχέση με το παραδοσιακό R-tree. Η δική τους υλοποίηση δείχνει ότι μπορεί να βελτιωθεί εξαιρετικά και η χρησιμοποίηση των σελίδων που αποθηκεύουν το δέντρο.

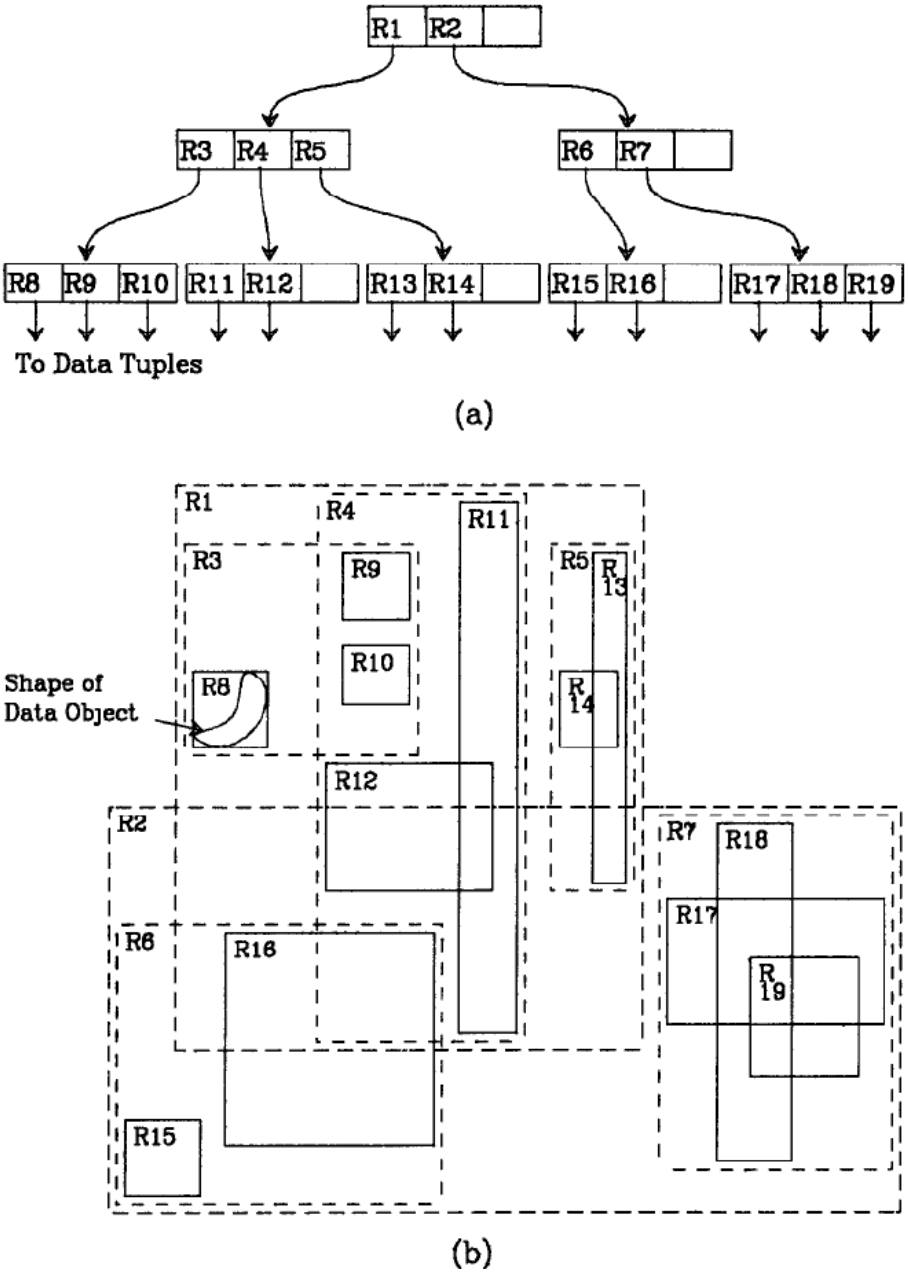
Στο R^*-tree εγκαινιάζεται μία καινούρια μέθοδος εισόδου στοιχείων, που ονομάζεται forced reinsert (εξαναγκασμένη επανεισαγωγή). Σε αυτή τη μέθοδο, αυτό που γίνεται είναι ότι αν ένας κόμβος είναι γεμάτος και του έρθουν νέα δεδομένα για να μπουν, δεν θα «σπάσει» (split) αμέσως, αλλά p εγγραφές θα αφαιρεθούν από τον κόμβο και θα ξαναμπούν στο δέντρο. Η παράμετρος p μπορεί να πάρει διάφορες τιμές, αλλά ο Beckman έδειξε ότι θα πρέπει να είναι περίπου το 30% του μέγιστου αριθμού εγγραφών που χωρούν ανά σελίδα.

Επιπλέον, στο απλό R-tree το μόνο κριτήριο για το splitting των κόμβων είναι το μέγεθος του ελάχιστου περιβάλλοντος ορθογωνίου (minimum bounding rectangle (MBR)) να είναι το ελάχιστο. Όμως, στα R^*-trees ελήφθησαν υπ' όψιν και τα ακόλουθα:

- Τα MBRs στο ίδιο επίπεδο ενός R^*-tree να είναι το ελάχιστο δυνατό. Με αυτόν τον τρόπο μικραίνει η πιθανότητα να χρειαστεί να ακολουθήσουμε διάφορα μονοπάτια για να απαντήσουμε σε ένα query για παράδειγμα.
- Τα MBRs να έχουν την ελάχιστη δυνατή περίμετρο. Με αυτή τη λογική το τετράγωνο είναι το ιδανικό σχήμα ενός MBR στις δύο διαστάσεις, ο κύβος στις τρεις και οι επεκτάσεις τους σε περισσότερες διαστάσεις.

- Η χρησιμοποίηση των σελίδων του δίσκου θέλουμε να είναι κατά το δυνατόν μέγιστη.

Ας δούμε όμως τώρα τον αλγόριθμο που χρησιμοποιείται για την εύρεση του πλησιέστερου γείτονα όταν χρησιμοποιούμε R-trees.



Εικόνα 7. Ένα κλασικό παράδειγμα (από το [4]) ενός R-tree.

3.2.2.1 Αυξητικός αλγόριθμος πλησιέστερου γείτονα

Ας δούμε τώρα τον καλύτερο κατά γενική ομολογία αλγόριθμο που μας επιστρέφει κατά βήματα τον πλησιέστερο γείτονα, μετά τον επόμενο πλησιέστερο γείτονα, μετά τον επόμενο κ.ο.κ..

Ο αλγόριθμος αυτός παίρνει ως είσοδο το σημείο ερώτησης q. Τότε ο αλγόριθμος είναι ο εξής:

Αυξητικός αλγόριθμος εύρεσης πλησιέστερων γειτόνων (incremental nearest neighbor algorithm (INN))

```
Static Ουρά προτεραιότητας queue;
queue.insert(root, 0);
όσο η queue δεν είναι άδεια κάνε τα ακόλουθα
        Element first=queue.top();
        Αν το first είναι ενδιάμεσος κόμβος τότε
                Για κάθε παιδί του first κάνε queue.insert(child, mindist(child, q))
        Αλλιώς αν το first είναι φύλλο τότε
                Για κάθε αντικείμενο του first κάνε queue.insert(object, mindist(object, q))
        Αλλιώς αν το first είναι αντικείμενο
                Ανάφερε το object
                Return // βρέθηκε ένας ακόμα nn. Όταν ξανακληθεί η συνάρτηση θα
//βρεθεί ο επόμενος κ.ο.κ..
```

Ο προηγούμενος αλγόριθμος επιστρέφει σταδιακά τους πλησιέστερους γείτονες. Όταν αποφασίσουμε ότι δεν θέλουμε άλλους γείτονες τότε σταματάμε να καλούμε τη συνάρτηση.

Η ορθότητα της συνάρτησης που περιγράψαμε βασίζεται στο γεγονός ότι η ουρά προτεραιότητα διατηρεί ταξινομημένα πάντα τα objects και nodes βάσει της ελάχιστης απόστασης από το σημείο ερώτησης. Έτσι, όταν φτάνουμε στο σημείο να αναφέρουμε ένα αντικείμενο είμαστε σίγουροι ότι όλα τα υπόλοιπα αντικείμενα έχουν απόσταση μεγαλύτερη από αυτό που αναφέρουμε, καθώς σε διαφορετική περίπτωση κάποιος κόμβος που θα τα περιείχε θα ήταν πιο κοντά στην κορυφή της ουράς προτεραιότητας από το δεδομένο που αναφέρουμε. Μεταξύ δύο αντικειμένων πάλι προφανώς η ταξινόμησή τους γίνεται σωστά, αφού βασίζεται στην απόστασή τους από το σημείο ερώτησης.

Άρα η προηγούμενη συνάρτηση μας λέει με βεβαιότητα ότι το πρώτο δεδομένο που θα επιστραφεί θα είναι ο πλησιέστερος γείτονας, το δεύτερο δεδομένο που θα επιστραφεί θα είναι ο αμέσως επόμενος πλησιέστερος γείτονας κ.ο.κ.. Αυτό θέλει και ο ΤΑ για να τρέξει. Να μπορεί να ανακαλεί με κάποιο τρόπο τα δεδομένα έτσι ώστε να απομακρύνονται από το σημείο ερώτησης όλο και περισσότερο από τις διάφορες διαστάσεις.

3.3 Ανάπτυξη προγραμμάτων

3.3.1 Τι προγράμματα θέλουμε να αναπτύξουμε

Ας υποθέσουμε ότι τίθεται ένα ερώτημα σε πέντε διαστάσεις. Για τους λόγους που προαναφέραμε, θέλουμε να μελετήσουμε πότε είναι καλύτερο αυτά τα δεδομένα να βρίσκονται και με τις πέντε τους διαστάσεις σε ένα R^*-tree, πότε είναι καλύτερο να βρίσκονται οι τρεις διαστάσεις σε ένα R^*-tree και οι υπόλοιπες δύο σε ένα άλλο R^*-tree, ή πότε είναι καλύτερο να έχουμε πέντε B^+-trees, ή όλες τις υπόλοιπες ενδιάμεσες περιπτώσεις (συνδυασμοί δηλαδή R^*-trees και B^+-trees).

Με αυτό το σκεπτικό κατασκευάσαμε τα προγράμματά μας. Τι κάναμε δηλαδή; Κατ' αρχάς κατασκευάσαμε μία γεννήτρια για να παράγουμε τα σύνολα δεδομένων που θα χρησιμοποιήσουμε για τα πειράματά μας. Αυτό έγινε με τη βοήθεια της βιβλιοθήκης tools, που μπορεί να βρεθεί στο site: http://research.att.com/~marioh/tools/index.html. Η βιβλιοθήκη tools, έχει γεννήτριες για κάθε λογής αριθμούς (double, float, integers, long integers και άλλους). Εμείς για τα πειράματά μας χρησιμοποιήσαμε double δεδομένα. Τα datasets τα παρήγαμε με τέτοιο τρόπο ώστε είτε δύο διαστάσεις να είναι ανεξάρτητες μεταξύ τους, είτε δύο διαστάσεις να είναι συσχετισμένα μεταξύ τους (correlated) ή είναι αντίθετα συσχετισμένα μεταξύ τους (anti-correlated), καθώς όπως είδαμε στο κεφάλαιο 3 της αναφοράς, έχει σημασία για τον TA (αλλά και τις παραλλαγές του) αν τα δεδομένα που διαχειρίζεται είναι correlated ή anticorrelated. Έτσι, παρήγαμε όλα τα δυνατά δεδομένα και κατόπιν προχωρήσαμε στην κατασκευή των υπόλοιπων προγραμμάτων. Τα δυνατά δεδομένα που παρήγαμε είναι:

- Όλες οι διαστάσεις ανεξάρτητες
- Δύο διαστάσεις correlated
- Δύο διαστάσεις anticorrelated
- Τρεις διαστάσεις correlated
- Τρεις διαστάσεις anticorrelated
- Δυο διαστάσεις correlated και μία anticorrelated προς τις δύο πρώτες
- Τέσσερις διαστάσεις correlated
- Τρεις διαστάσεις correlated και μία anticorrelated προς τις υπόλοιπες
- Δύο διαστάσεις correlated μεταξύ τους και άλλες δύο correlated μεταξύ τους
- Δύο διαστάσεις correlated μεταξύ τους και άλλες δύο anticorrelated μεταξύ τους

Κατόπιν, κατασκευάσαμε ένα πρόγραμμα που μας δηλώνει τα πραγματικά αποτελέσματα ενός ερωτήματος. Δηλαδή, παίρνει ως εισόδους τα datasets και το σημείο ερώτησης και βγάζει ως έξοδο τα δεδομένα με τις αποστάσεις τους από το σημείο ερώτησης ταξινομημένα ως προς την απόστασή τους από το σημείο ερώτησης. Έτσι, αν δώσουμε ένα ερώτημα στα προγράμματα που θα κατασκευάσουμε κατόπιν, θα μπορούμε να εκτιμήσουμε αν λειτουργούν όπως θα έπρεπε.

Τέλος, υλοποιήσαμε τον αλγόριθμο NRA (υπενθυμίζουμε ότι πρόκειται για παραλλαγή του TA κατά την οποία δεν πραγματοποιούνται τυχαίες προσπελάσεις σε δεδομένα, βλέπετε

ενότητα 3.1.5.1) για όλες τις δυνατές περιπτώσεις. Δουλέψαμε σε τέσσερις διαστάσεις με τη λογική ότι τα συμπεράσματα που θα βγάλουμε θα είναι επεκτάσιμα και για λίγο μεγαλύτερες. Έτσι, φτιάξαμε ένα πρόγραμμα που να έχει ένα R*-tree και στις τέσσερις διαστάσεις και στο οποίο μπορούμε να θέτουμε ερωτήματα πλησιέστερου γείτονα είτε για 4, είτε για 3, είτε για 2 είτε για 1 διάσταση, μετά φτιάξαμε ένα πρόγραμμα που να έχει δύο R*-trees δύο διαστάσεων το καθένα και τα οποία μπορούν να δέχονται ερωτήματα και πάλι για όλες τις πιθανές διαστάσεις, μετά ένα πρόγραμμα με ένα R*-tree σε δύο διαστάσεις και δύο B+-trees στις υπόλοιπες διαστάσεις (πάλι δέχονται ερωτήματα σε οποιεσδήποτε διαστάσεις), ένα R*-tree στις 3 διαστάσεις και ένα B+-tree στην άλλη διάσταση καθώς και ένα πρόγραμμα με τέσσερα B+-trees (B+-trees σε όλες τις διαστάσεις δηλαδή).

Με αυτόν τον τρόπο, έχουμε τη δυνατότητα να μελετήσουμε όλες τις περιπτώσεις για δεδομένα τεσσάρων διαστάσεων και να βγάλουμε πολύτιμα συμπεράσματα. Μπορούμε να τρέξουμε επί παραδείγματι ένα ερώτημα πλησιέστερου γείτονα όταν έχουμε B+-trees σε δύο διαστάσεις και μετά να τρέξουμε το ίδιο ερώτημα όταν έχουμε ένα R*-tree στις τρεις διαστάσεις και ένα B+-tree στην άλλη διάσταση και να μπορούμε να συγκρίνουμε πότε γίνονται περισσότερες αναγνώσεις και εγγραφές δεδομένων στο δίσκο (μετράμε μάλιστα όχι απλά αναγνώσεις δεδομένων, αλλά αριθμό εγγραφών και αναγνώσεων στο δίσκο, δηλαδή πόσες μεταφορές σελίδων γίνονται από και προς το δίσκο).

3.3.2 Ο αλγόριθμος των προγραμμάτων σε ψευτοκώδικα

Στην ενότητα θα δώσουμε τον αλγόριθμο των προγραμμάτων που κατασκευάσαμε σε ψευτοκώδικα. Είναι πρακτικά κατάλληλα τροποποιημένος ο NRA που περιγράψαμε στην ενότητα 3.1.5.1. Ας δώσουμε τον αλγόριθμο όταν έχουμε και στις τέσσερις διαστάσεις B+-trees:

Αλγόριθμος με B+-trees

---

1. Κατάλληλες αρχικοποιήσεις: $(M_k = 4, topk = \{dummy1, \dots, dummyk\}, candidates = \emptyset)$
2. Ταξινόμησε τις τιμές της κάθε διάστασης των δεδομένων σε διπλά συνδεδεμένες λίστες, οι οποίες έχουν τα δεδομένα ταξινομημένα ως προς την τιμή του πεδίου. Δηλαδή, οι λίστες αυτές (που προσομοιώνουν τα B+-trees) έχουν το id του κάθε δεδομένου και την τιμή του στην εν λόγω διάσταση, ώστε οι τιμές στις διαστάσεις να ξεκινούν από το 0 και να φτάνουν ως το 1 σε αύξουσα σειρά.
3. Σε κάθε μία από τις λίστες που προαναφέραμε βρες παράλληλα το δεδομένο με την κοντινότερη τιμή στο σημείο ερώτησης σε αυτή τη διάσταση που δεν έχεις δει ακόμα. Σημείωσε ότι έχεις δει το τρέχον αντικείμενο που βλέπεις στην κάθε λίστα στην κατάλληλη διάσταση που το βλέπεις. Επίσης, για κάθε στοιχείο που βλέπεις βρες την ελάχιστη και τη μέγιστη απόσταση που μπορεί να έχει από το σημείο ερώτησης. Αν δεν ανήκει στους candidates ούτε στο topk βάλε το στους candidates.
4. Εάν η μεγαλύτερη απόσταση που έχει το τρέχον πρόσφατα ιδωθέν δεδομένο είναι μικρότερη από M_k τότε
    a. Αν το πρόσφατα ιδωθέν δεδομένο δεν ανήκει στα topk βάλε το στα topk
    b. Ανανέωσε το M_k

5. Αν το πρόσφατα ιδωθέν δεδομένο έχει ελάχιστη απόσταση από το σημείο ερώτησης μεγαλύτερη ή ίση από το M_k τότε αφαίρεσέ το από τους candidates
6. Threshold=η ελάχιστη καλύτερη απόσταση από το σημείο ερώτησης που έχει κάποιος από τους candidates
7. Πήγαινε στο 3 αν threshold<=M_k, αλλιώς ανάφερε τα topk.

Σε μία πρώτη ανάγνωση, ο αλγόριθμος δεν μοιάζει ίδιος με αυτόν που περιγράψαμε στην ενότητα 3.1.5.1. Όμως είναι το ίδιο ακριβώς πράγμα.

Αξίζει να αναφερθεί κατ' αρχάς ότι στο βήμα 3 του αλγορίθμου ο υπολογισμός της μέγιστης και της ελάχιστης απόστασης από το σημείο ερώτησης θα γίνεται βάσει των ακόλουθων τύπων:

- Η ελάχιστη απόσταση βρίσκεται από το ακόλουθο άθροισμα: $minDist = \sqrt{\sum_{\sigma\varepsilon\ \text{όποιες διαστάσεις έχουμε δει το δεδομένο}} \left(data_{dim} - query_{point_{dim}}\right)^2 + \sum_{\sigma\varepsilon\ \text{όποιες διαστάσεις δεν έχουμε δει το δεδομένο}} \left(last_{seen_{in_{queues}}{}_{dim}} - query_{point_{dim}}\right)^2}$
- Η μέγιστη απόσταση βρίσκεται από το άθροισμα: $maxDist = \sqrt{\sum_{\sigma\varepsilon\ \text{όποιες διαστάσεις έχουμε δει το δεδομένο}} \left(data_{dim} - query_{point_{dim}}\right)^2 + \sum_{\sigma\varepsilon\ \text{όποιες διαστάσεις δεν έχουμε δει το δεδομένο}} \left(max(1 - query\_point_{dim}, query\_point_{dim})\right)^2}$

Η μόνη διαφορά που έχουμε όταν χρησιμοποιούμε R^*-trees αντί για B^+-trees μόνο είναι στο βήμα 3 του αλγορίθμου, όπου δεν βρίσκουμε παράλληλα σε λίστες (δηλαδή τα B^+-trees) τα δεδομένα που είναι πιο κοντά στο σημείο ερώτησης, αλλά βρίσκουμε παράλληλα σε όποιες δομές ευρετηρίων τα δεδομένα που είναι πιο κοντά στο σημείο ερώτησης. Αν έχουμε ένα R^*-trees σε δύο διαστάσεις και δύο B^+-trees στις άλλες δύο επί παραδείγματι, τότε θα βρούμε από το R^*-tree το πιο κοντινό σημείο στις διαστάσεις στις οποίες αναφέρεται και από τα δύο B^+-trees θα βρούμε το πιο κοντινό σημείο του καθενός στη διάσταση στην οποία αναφέρεται. Έτσι θα λειτουργεί και ο NRA. Θα παίρνει σημεία με φθίνουσα απόσταση κάθε φορά, θα υπολογίζει κατάλληλα τις αποστάσεις (μέγιστες και ελάχιστες δυνατές) και θα σταματά όσο το δυνατόν πιο γρήγορα.

Έχοντας υλοποιήσει αυτά τα προγράμματα θα τρέξουμε τα πειράματά μας και θα μελετήσουμε το πλήθος των εγγραφών και αναγνώσεων από το δίσκο. Έτσι θα καταλήξουμε σε αρκετά χρήσιμα συμπεράσματα και θα φτιάξουμε την καλύτερη δυνατή δομή ευρετηρίου για πολυδιάστατα δεδομένα στα οποία υποβάλλονται ερωτήματα πλησιέστερου γείτονα.

4

Πειράματα και σχολιασμός των αποτελεσμάτων

Εδώ θα δούμε τα πειράματα που εκτελέσαμε, καθώς και τα συμπεράσματα που προέκυψαν από τα πειράματα αυτά. Υπενθυμίζουμε ότι τα πειράματα εκτελέστηκαν σε δεδομένα τεσσάρων διαστάσεων. Επίσης, τα προγράμματα που έχουμε αναπτύξει έχουν όλες τις δυνατές περιπτώσεις συνδυασμών B^+-trees και R^*-trees (δηλαδή ένα πρόγραμμα με ένα R^*-tree τεσσάρων διαστάσεων, ένα με δύο R^*-trees από δύο διαστάσεις το καθένα, ένα με ένα R^*-tree δύο διαστάσεων και στις υπόλοιπες δύο διαστάσεις είχαμε B^+-trees, ένα με ένα R^*-tree τριών διαστάσεων και στην άλλη διάσταση είχαμε ένα B^+-tree, και τέλος ένα πρόγραμμα στο οποίο και στις τέσσερις διαστάσεις είχαμε B^+-trees.

Τώρα, θα αναφέρουμε για άλλη μία φορά ότι τα πιθανά σύνολα δεδομένων με τα οποία και πειραματιστήκαμε είναι τα ακόλουθα:

- Όλες οι διαστάσεις ανεξάρτητες
- Δύο διαστάσεις correlated
- Δύο διαστάσεις anticorrelated
- Τρεις διαστάσεις correlated
- Τρεις διαστάσεις anticorrelated
- Δυο διαστάσεις correlated και μία anticorrelated προς τις δύο πρώτες
- Τέσσερις διαστάσεις correlated
- Τρεις διαστάσεις correlated και μία anticorrelated προς τις υπόλοιπες
- Δύο διαστάσεις correlated μεταξύ τους και άλλες δύο correlated μεταξύ τους
- Δύο διαστάσεις correlated μεταξύ τους και άλλες δύο anticorrelated μεταξύ τους

Αυτό που κάναμε είναι για κάθε είδος συνόλου δεδομένων (για παράδειγμα όλες οι διαστάσεις να είναι ανεξάρτητες) βάλαμε τα δεδομένα μας είτε σε ένα R^*-tree τεσσάρων διαστάσεων, είτε σε δύο R^*-trees δύο διαστάσεων το καθένα, είτε όλες τις υπόλοιπες

περιπτώσεις. Έτσι, πήραμε διάφορες μετρήσεις σχετικά με τις ανακλήσεις δεδομένων που απαίτησε το κάθε πρόγραμμα και καταλήξαμε σε διάφορα συμπεράσματα σχετικά με το ποια μέθοδος πρέπει να ακολουθηθεί για την διάταξη των διαφόρων διαστάσεων σε ευρετήρια τύπου R^*-tree ή B^+-tree, όπως έχουμε προαναφέρει.

Επειδή όμως οι δυνατές περιπτώσεις είναι πάρα πολλές (έχουμε δέκα διαφορετικά σύνολα δεδομένων, τα οποία θέλουμε να εξετάσουμε για σημεία ερώτησης 1, 2, 3 και 4 διαστάσεων, σε πέντε διαφορετικές δομές ευρετηρίων (συνδυασμοί R^*-trees και B^+-trees) και μάλιστα όταν έχουμε για παράδειγμα σύνολα δεδομένων με δύο διαστάσεις correlated και τις υπόλοιπες ανεξάρτητες, τότε έχουμε να κάνουμε ακόμα περισσότερους συνδυασμούς. Το τελευταίο είναι αληθές, διότι όταν θέλουμε να τρέξουμε τα πειράματα της τελευταίας περίπτωσης σε ένα R^*-tree δύο διαστάσεων και δύο B^+-trees, τότε θα βάλουμε τα correlated στο R^*-tree, αλλά για να εξετάσουμε όλες τις περιπτώσεις και τη μία διάσταση στο R^*-tree και την άλλη correlated σε ένα B^+-tree, καθώς και τις δύο στα B^+-trees. Αυτό σημαίνει ότι μπορούμε να εκτελέσουμε περί τα 1000 διαφορετικά πειράματα). Προφανώς δεν θα παραθέσουμε όλα τα πειραματικά αποτελέσματα σε αυτό το κείμενο. Θα παραθέσουμε κάποια βασικά πειραματικά αποτελέσματα που μας επέτρεψαν να βγάλουμε και τα κύρια συμπεράσματά μας.

Αυτό θα κάνουμε στις ακόλουθες ενότητες. Σημειώνουμε ότι για όλα τα πειράματα που εκτελέστηκαν θεωρήσαμε $k = 10$ για τα ερωτήματα kNN.

4.1 Σύνολο δεδομένων τεσσάρων διαστάσεων

4.1.1 Δεδομένα με ανεξάρτητες διαστάσεις

Εδώ θα παραθέσουμε τα αποτελέσματα που λάβαμε όταν εκτελέσαμε πειράματα σε δεδομένα με τέσσερις διαστάσεις ανεξάρτητες μεταξύ τους.

Τα αποτελέσματα φαίνονται στους τέσσερις ακόλουθους πίνακες:

Πίνακας 1. Μετρήσεις για 4d σημείο ερώτησης σε δεδομένα 4 διαστάσεων ανεξάρτητων μεταξύ τους.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
4	4 B^+-trees	4×308	-	-	1232
	1 R^*-tree 2d+ 2 B^+-trees	2×306	33	501	1146
	1 R^*-tree 3d +1 B^+-tree	244	60	675	979
	1 R^*-tree 4d	-	9	10	19
	2 R^*-trees 2d	-	$7 + 6$	$69 + 57$	139

Πίνακας 2. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 4 διαστάσεων ανεξάρτητων μεταξύ τους.

Διαστάσεις του σημείου	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees	IOs στα R^*-trees (data	Συνολικά IOs

ερώτησης			(index nodes)	nodes)	
3	4 B^+-trees	3×133	-	-	399
	1 R^*-tree 2d+ 2 B^+-trees (το ερώτημα αφορά σε δύο διαστάσεις του R^*-tree και μία ενός B^+-tree)	128	17	216	361
	1 R^*-tree 2d +2 B^+-trees (το ερώτημα αφορά σε μία διάσταση του R^*-tree και δύο των B^+-trees)	2×134	27	249	544
	1 R^*-tree 3d+ 1 B^+-tree (το ερώτημα αφορά σε τρεις διαστάσεις του R^*-tree)	-	4	4	8
	1 R^*-tree 3d+ 1 B^+-tree (το ερώτημα αφορά σε δύο διαστάσεις του R^*-tree και μία του B^+-tree)	128	61	425	614
	1 R^*-tree 4d	-	22	41	63
	2 R^*-trees 2d	-	$17 + 21$	$216 + 230$	484

Πίνακας 3 . Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 4 διαστάσεων ανεξάρτητων μεταξύ τους.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
2	4 B^+-trees	2×34	-	-	68
	1 R^*-tree 2d+ 2 B^+-trees (το ερώτημα αφορά σε δύο διαστάσεις του R^*-tree)	-	3	4	7
	1 R^*-tree 2d +2 B^+-trees (το ερώτημα αφορά σε μία διάσταση του R^*-	28	15	101	144

	tree και μία ενός B^+-tree)				
	1 R^*-tree 3d+ 1 B^+-tree (το ερώτημα αφορά σε δύο διαστάσεις του R^*-tree)	-	9	18	27
	1 R^*-tree 3d+ 1 B^+-tree (το ερώτημα αφορά σε μία διαστάσεις του R^*-tree και μία του B^+-tree)	28	47	352	427
	1 R^*-tree 4d	-	37	98	135
	2 R^*-trees 2d	-	17 + 18	120 + 104	259

Πίνακας 4. Μετρήσεις για 1d σημείο ερώτησης σε δεδομένα 4 διαστάσεων ανεξάρτητων μεταξύ τους.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
1	4 B^+-trees	4	-	-	4
	1 R^*-tree 2d+ 2 B^+-trees (το ερώτημα αφορά σε μία διάσταση του R^*-tree)	-	13	50	63
	1 R^*-tree 3d+ 1 B^+-tree (το ερώτημα αφορά σε μία διάσταση του R^*-tree)	-	48	294	342
	1 R^*-tree 4d	-	108	715	823

4.1.1.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Από τις παραπάνω μετρήσεις μπορούμε να καταλήξουμε σε ορισμένα σημαντικά σε πρώτη φάση συμπεράσματα. Ας δούμε τα κυριότερα από αυτά συνοπτικά.

Κατ' αρχάς, από τις παραπάνω μετρήσεις μπορούμε εύκολα να διαπιστώσουμε ότι όταν το σημείο ερώτησης αναφέρεται σε x διαστάσεις, τότε η καλύτερη λύση είναι να έχουμε ένα R^*-tree και στις x διαστάσεις αυτές διαστάσεις. Βλέπουμε για παράδειγμα από τον πίνακα Πίνακας 1, ότι όταν το σημείο ερώτησης είναι τεσσάρων διαστάσεων τότε ένα R^*-tree 4d (4 dimensional) δίνει τα λιγότερα IOs. Το ίδιο συμπέρασμα προκύπτει από τον

πίνακα Πίνακας 2 όταν το σημείο ερώτησης είναι τριών διαστάσεων και από τον πίνακα Πίνακας 3 όταν το σημείο ερώτησης είναι δύο διαστάσεων. Τέλος, από τον πίνακα Πίνακας 4 βλέπουμε ότι όταν το σημείο ερώτησης είναι μίας μόνο διάστασης τότε ένα B^+-tree σε αυτή τη διάσταση δίνει τα λιγότερα δυνατά IOs. Αυτό είναι λογικό, καθώς κατεβαίνουμε από τη ρίζα το B^+-tree, βρίσκουμε στα φύλλα το κοντινότερο δεδομένο προς το σημείο ερώτησης σε αυτή τη διάσταση που μας ενδιαφέρει και μετά κινούμαστε δεξιά και αριστερά για να βρούμε τα υπόλοιπα $k-1$ δεδομένα που αποτελούν τους υπόλοιπους πλησιέστερους γείτονες.

Τώρα, ας δούμε ένα ακόμα συμπέρασμα που μπορεί να προκύψει από τις προηγηθείσες μετρήσεις. Ας υποθέσουμε ότι το ερώτημα πλησιέστερου γείτονα αναφέρεται σε x διαστάσεις. Τότε, όπως προαναφέραμε, μία δομή ευρετηρίου R^*-tree που αναφέρεται σε αυτές τις x διαστάσεις έχουμε τα καλύτερα αποτελέσματα. Αξίζει όμως να αναφερθεί ότι αν χρησιμοποιούμε μία δομή R^*-tree y διαστάσεων, με $x \neq y$, και σε όλες τις υπόλοιπες διαστάσεις που μας ενδιαφέρουν για το ερώτημα kNN έχουμε B^+-trees, τότε όσο αυξάνεται το $|x-y|$ τόσο χειρότερο είναι το αποτέλεσμα που θα πάρουμε, δηλαδή τόσο περισσότερα είναι τα IOs που θα πραγματοποιηθούν. Για παράδειγμα, αν έχουμε ένα ερώτημα που αφορά σε τέσσερις διαστάσεις, τότε καλύτερα είναι να έχουμε αυτές τις διαστάσεις σε ένα R^*-tree 4d. Αν όμως έχουμε ένα R^*-tree 3d και ένα B^+-tree τότε τα αποτελέσματα θα είναι χειρότερα, ενώ αν έχουμε ένα R^*-tree 2d και δύο B^+-trees τα αποτελέσματα θα είναι ακόμα χειρότερα. Μπορείτε να δείτε τον πίνακα Πίνακας 1 για να το εξακριβώσετε.

Τέλος, αξίζει να αναφέρουμε ένα ακόμα συμπέρασμα που μπορούμε να δούμε σε όλες τις μετρήσεις που έχουμε λάβει. Ας υποθέσουμε ότι έχουμε μία δομή με ένα R^*-tree και κάποια B^+-trees. Τότε, όσες περισσότερες από τις διαστάσεις του ερωτήματος πλησιέστερου γείτονα που τίθεται βρίσκονται στο R^*-tree, τόσο καλύτερα αποτελέσματα θα πάρουμε. Αν για παράδειγμα υποθέσουμε ότι θέλουμε να κάνουμε ένα ερώτημα πλησιέστερου γείτονα που αφορά σε τρεις διαστάσεις, και η δομή μας αποτελείται από ένα R^*-tree 2d και δύο B^+-trees, τότε τα καλύτερα αποτελέσματα τα παίρνουμε όταν στο R^*-tree έχουμε τις δύο από τις διαστάσεις που αφορούν στο ερώτημα. Αντίθετα, όταν στο R^*-tree έχουμε τη μία από τις διαστάσεις που αφορούν στο ερώτημα παίρνουμε αισθητά περισσότερα IOs.

4.1.2 Δεδομένα με τις δύο διαστάσεις τους correlated

Εδώ θα δούμε τις μετρήσεις που λάβαμε όταν οι δύο από τις τέσσερις συνολικά διαστάσεις των δεδομένων μας ήταν correlated. Επειδή ακριβώς ήταν correlated, υπήρξε η ανάγκη να γίνουν ακόμα περισσότερες μετρήσεις. Για παράδειγμα, όταν το σημείο ερώτησης ήταν τεσσάρων διαστάσεων και είχαμε ένα R^*-tree 2d και δύο B^+-trees τότε μπορεί στο R^*-tree να είχαμε τα δύο correlated data, ή να είχαμε το ένα από αυτά στο R^*-tree και το άλλο σε κάποιο B^+-tree ή ακόμα - ακόμα να βρίσκονται και τα δύο σε ένα B^+-tree.

Ας δούμε λοιπόν τα πειραματικά αποτελέσματα που λάβαμε για αυτό το σύνολο δεδομένων.

Πίνακας 5. Μετρήσεις για 4d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
4	4 B^+-trees	4×174	-	-	696
	1 R^*-tree 2d+ 2 B^+-trees (τα cor βρίσκονται στο R^*-tree)	2×174	14	259	621
	1 R^*-tree 2d +2 B^+-trees (ένα από τα cor βρίσκεται στο R^*-tree)	2×134	20	231	519
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται στα B^+-trees)	2×126	14	207	473
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree)	134	28	309	471
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα cor βρίσκεται στο R^*-tree)	109	40	327	476
	1 R^*-tree 4d	-	7	6	13
	2 R^*-trees 2d (τα cor βρίσκονται στο ίδιο R^*-tree)	-	$10 + 14$	$188 + 203$	415
	2 R^*-trees 2d (τα cor βρίσκονται σε διαφορετικό R^*-tree)	-	$6 + 5$	$27 + 25$	63

Πίνακας 6. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
3	4 B^+-trees (το ερώτημα περιλαμβάνει τα cor)	3×26	-	-	78
	1 R^*-tree 2d+ 2 B^+-trees (τα cor βρίσκονται	26	5	34	65

	στο R^*-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο ένα B^+-tree)				
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία διάσταση του R^*-tree και στα δύο B^+-trees)	2×156	12	231	555
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο cor B^+-tree)	15	6	29	50
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο B^+-tree που δεν είναι cor)	109	17	196	322
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην cor διάσταση του R^*-tree και στα B^+-trees)	2×31	16	99	177
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην μη cor διάσταση του R^*-tree και στα B^+-trees)	2×582	49	950	2163

	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται στα B^+-trees-το ερώτημα αφορά στις διαστάσεις του R^*-tree και μία από τις cor)	127	15	207	349
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται στα B^+-trees-το ερώτημα αφορά σε μία διάσταση του R^*-tree και τις cor)	2×30	19	101	180
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα γίνεται στο R^*-tree)	-	4	3	7
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα γίνεται στα cor του R^*-tree και το B^+-tree)	31	22	103	156
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε ένα cor του R^*-tree, τη μη cor διάσταση του R^*-tree και το B^+-tree)	109	19	240	368
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα cor βρίσκεται στο R^*-tree-το ερώτημα γίνεται στα μη cor του R^*-tree και το άλλο cor)	127	35	342	504
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα cor βρίσκεται στο R^*-tree-το ερώτημα γίνεται σε ένα μη	22	26	110	158

	cor του R^*-tree, και στα δύο cor)				
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει και τα δύο cor)	-	15	23	38
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει το ένα cor μόνο)	-	7	5	12
	2 R^*-trees 2d (τα cor βρίσκονται στο ίδιο R^*-tree-το ερώτημα περιλαμβάνει τα cor)	-	$5 + 19$	$40 + 101$	165
	2 R^*-trees 2d (τα cor βρίσκονται στο ίδιο R^*-tree-το ερώτημα δεν περιλαμβάνει και τα δύο cor)	-	$15 + 10$	$207 + 188$	402
	2 R^*-trees 2d (τα cor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα τα περιλαμβάνει)	-	$8 + 17$	$40 + 89$	154
	2 R^*-trees 2d (τα cor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα περιλαμβάνει το ένα από αυτά)	-	$17 + 13$	$196 + 186$	412

Πίνακας 7. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι correlated.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
2	4 B^+-trees (τα cor περιλαμβάνονται)	2×4	-	-	8
	1 R^*-tree 2d+ 2 B^+-trees (τα cor βρίσκονται στο R^*-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree)	-	3	2	5
	1 R^*-tree 2d	24	4	30	58

	+2 B^+-trees (τα cor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία διάσταση του R^*-tree και ένα B^+-tree)				
	1 R^*-tree 2d +2 B^+-trees (τα cor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην cor διάσταση του R^*-tree και στο cor B^+-tree)	4	13	55	72
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα στις cor του R^*-tree)	-	16	44	60
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε μία cor του R^*-tree και μία ανεξάρτητη του R^*-tree)	-	4	3	7
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε ένα cor του R^*-tree και το B^+-tree)	26	22	99	147
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα cor βρίσκεται στο R^*-tree-το ερώτημα γίνεται στα δύο cor)	4	37	252	289
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει και τα δύο cor)	-	54	221	275
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει το	-	16	24	40

	ένα cor μόνο)				
	2 R^*-trees 2d (τα cor βρίσκονται στο ίδιο R^*-tree-το ερώτημα περιλαμβάνει το ένα cor και τη μη cor διάσταση του άλλου δέντρου)	-	$5 + 25$	$31 + 108$	169
	2 R^*-trees 2d (τα cor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα τα περιλαμβάνει)	-	$13 + 24$	$55 + 79$	171

Πίνακας 8. Μετρήσεις για 1d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι correlated.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
1	1 R^*-tree 2d+ 2 B^+-trees (τα cor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία cor διάσταση του R^*-tree)	-	3	2	5
	1 R^*-tree 3d+ 1 B^+-tree (τα cor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία cor του R^*-tree)	-	26	88	114
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει ένα cor)	-	54	221	275

Θα μπορούσαμε να συνεχίσουμε να παραθέτουμε πίνακες με πειραματικά αποτελέσματα, αλλά δεν έχει ιδιαίτερο νόημα. Θα σχολιάσουμε τα αποτελέσματα που έχουμε πάρει και καταγράψει ως εδώ, και έτσι θα καταλήξουμε στα συμπεράσματά μας.

4.1.2.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Ας καταστήσουμε σαφή τώρα ορισμένα συμπεράσματα που μπορούν να προκύψουν από τη μελέτη των προηγηθέντων μετρήσεων.

Κατ' αρχάς, όπως και στην ενότητα 4.1.1.1 έτσι και εδώ μπορούμε να πούμε ότι όταν το σημείο ερώτησης αναφέρεται σε x διαστάσεις, τότε η καλύτερη λύση είναι να έχουμε ένα R^*-tree και στις x διαστάσεις αυτές διαστάσεις. Για παράδειγμα, αν θέσουμε ένα ερώτημα τριών διαστάσεων εκ των οποίων οι δύο είναι μεταξύ τους correlated, τότε τα ελάχιστα IOs τα έχουμε όταν έχουμε ένα R^*-tree με τρεις διαστάσεις: αυτές στις οποίες τίθεται το ερώτημα.

Τώρα θα δούμε κάποιες περιπτώσεις.

- Όταν το ερώτημα τίθεται σε έναν αριθμό διαστάσεων και υποθέτουμε ότι δύο από αυτές είναι μεταξύ τους correlated, τότε η καλύτερη περίπτωση είναι οι δύο διαστάσεις που είναι correlated να είναι η μία σε ένα R^*-tree και η άλλη σε ένα B^+-tree, η αμέσως καλύτερη είναι να είναι στο ίδιο R^*-tree και η χειρότερη είναι να βρίσκονται η κάθε μία σε ένα διαφορετικό R^*-tree.
- Όταν όμως το ερώτημα τίθεται σε έναν αριθμό διαστάσεων ανεξάρτητων μεταξύ τους, όμως οι δομές ευρετηρίων έχουν κάποιες διαστάσεις που είναι μεταξύ τους correlated, τότε έχουμε να πούμε ότι αφενός αν χρησιμοποιούσαμε δομές που έχουν μόνο ανεξάρτητες διαστάσεις θα είχαμε λιγότερα IOs (μπορείτε να το δείτε με μία σύγκριση μεταξύ των πειραματικών αποτελεσμάτων των ενοτήτων 4.1.1.1 και 4.1.2.1), αφετέρου αν είμαστε «υποχρεωμένοι» να χρησιμοποιήσουμε δομές ευρετηρίων που έχουν διαστάσεις που μεταξύ τους είναι correlated τότε ας είναι όσο λιγότερες γίνεται.

4.1.3 Δεδομένα με τις δύο διαστάσεις τους anti-correlated

Εδώ θα δούμε τις μετρήσεις που λάβαμε όταν οι δύο από τις τέσσερις συνολικά διαστάσεις των δεδομένων μας ήταν anti-correlated. Ας δούμε λοιπόν τα πειραματικά αποτελέσματα που λάβαμε για αυτό το σύνολο δεδομένων.

Πίνακας 9. Μετρήσεις για 4d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι anticorrelated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
4	4 B^+-trees	4×174	-	-	696
	1 R^*-tree 2d+ 2 B^+-trees (τα anticor βρίσκονται στο R^*-tree)	2×174	14	254	616
	1 R^*-tree 2d +2 B^+-trees (ένα από τα anticor βρίσκεται στο R^*-tree)	2×134	16	234	418
	1 R^*-tree 2d +2 B^+-trees (τα anticor	2×126	14	207	473

	βρίσκονται στα B^+-trees)				
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree)	134	22	304	460
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα anticor βρίσκεται στο R^*-tree)	109	40	326	475
	1 R^*-tree 4d	-	5	5	10
	2 R^*-trees 2d (τα anticor βρίσκονται στο ίδιο R^*-tree)	-	$10 + 14$	$185 + 207$	420
	2 R^*-trees 2d (τα anticor βρίσκονται σε διαφορετικό R^*-tree)	-	$6 + 6$	$25 + 25$	62

Πίνακας 10. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι anticorrelated.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
3	4 B^+-trees (το ερώτημα περιλαμβάνει τα anticor)	3×26	-	-	78
	1 R^*-tree 2d+ 2 B^+-trees (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο ένα B^+-tree)	26	5	34	65
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία διάσταση του R^*-tree και στα δύο B^+-trees)	2×156	13	229	554

	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο anticor B^+-tree)	15	6	30	51
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree και στο B^+-tree που δεν είναι anticor)	109	14	193	316
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην anticor διάσταση του R^*-tree και στα B^+-trees)	2×31	16	99	177
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην μη anticor διάσταση του R^*-tree και στα B^+-trees)	2×582	49	950	2163
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται στα B^+-trees-το ερώτημα αφορά στις διαστάσεις του R^*-tree και μία από τις	127	15	207	349

	anticor)				
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται στα B^+-trees-το ερώτημα αφορά σε μία διάσταση του R^*-tree και τις anticor)	2×30	19	101	180
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα γίνεται στο R^*-tree)	-	4	3	7
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα γίνεται στα anticor του R^*-tree και το B^+-tree)	31	22	103	156
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε ένα anticor του R^*-tree, τη μη anticor διάσταση του R^*-tree και το B^+-tree)	109	19	240	368
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα anticor βρίσκεται στο R^*-tree-το ερώτημα γίνεται στα μη anticor του R^*-tree και το άλλο anticor)	127	35	342	504
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα anticor βρίσκεται στο R^*-tree-το ερώτημα γίνεται σε ένα μη anticor του R^*-tree, και στα δύο anticor)	22	26	110	158
	1 R^*-tree 4d (το	-	15	23	38

	ερώτημα περιλαμβάνει και τα δύο anticor)				
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει το ένα anticor μόνο)	-	7	5	12
	2 R^*-trees 2d (τα anticor βρίσκονται στο ίδιο R^*-tree-το ερώτημα περιλαμβάνει τα anticor)	-	$5 + 19$	$40 + 101$	165
	2 R^*-trees 2d (τα anticor βρίσκονται στο ίδιο R^*-tree-το ερώτημα δεν περιλαμβάνει και τα δύο anticor)	-	$15 + 10$	$213 + 188$	408
	2 R^*-trees 2d (τα anticor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα τα περιλαμβάνει)	-	$8 + 17$	$39 + 90$	154
	2 R^*-trees 2d (τα anticor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα περιλαμβάνει το ένα από αυτά)	-	$17 + 13$	$192 + 186$	408

Πίνακας 11. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι anticorrelated.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
2	4 B^+-trees (τα anticor περιλαμβάνονται)	2×4	-	-	8
	1 R^*-tree 2d+ 2 B^+-trees (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά στις διαστάσεις του R^*-tree)	-	3	2	5
	1 R^*-tree 2d	24	4	30	58

	+2 B^+-trees (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία διάσταση του R^*-tree και ένα B^+-tree)				
	1 R^*-tree 2d +2 B^+-trees (τα anticor βρίσκονται ένα στο R^*-tree και ένα σε ένα B^+-tree-το ερώτημα αφορά στην anticor διάσταση του R^*-tree και στο anticor B^+-tree)	4	13	55	72
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα στις anticor του R^*-tree)	-	16	44	60
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε μία anticor του R^*-tree και μία ανεξάρτητη του R^*-tree)	-	4	3	7
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα γίνεται σε ένα anticor του R^*-tree και το B^+-tree)	26	22	99	147
	1 R^*-tree 3d+ 1 B^+-tree (ένα από τα anticor βρίσκεται στο R^*-tree-το ερώτημα γίνεται στα δύο anticor)	4	37	252	289
	1 R^*-tree 4d (το	-	54	221	275

	ερώτημα περιλαμβάνει και τα δύο anticor)				
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει το ένα anticor μόνο)	-	16	24	40
	2 R^*-trees 2d (τα anticor βρίσκονται στο ίδιο R^*-tree-το ερώτημα περιλαμβάνει το ένα anticor και τη μη anticor διάσταση του άλλου δέντρου)	-	$5 + 25$	$31 + 108$	169
	2 R^*-trees 2d (τα anticor βρίσκονται σε διαφορετικό R^*-tree-το ερώτημα τα περιλαμβάνει)	-	$13 + 24$	$55 + 79$	171

Πίνακας 12. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 4 διαστάσεων, με τις δύο διαστάσεις να είναι anticorrelated.

Διαστάσεις του σημείου ερώτησης	**Χρησιμοποιούμενη δομή ευρετηρίου**	**IOs στα B^+-trees**	**IOs στα R^*-trees (index nodes)**	**IOs στα R^*-trees (data nodes)**	**Συνολικά IOs**
1	1 R^*-tree 2d+ 2 B^+-trees (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία anticor διάσταση του R^*-tree)	-	3	2	5
	1 R^*-tree 3d+ 1 B^+-tree (τα anticor βρίσκονται στο R^*-tree-το ερώτημα αφορά σε μία anticor του R^*-tree)	-	26	88	114
	1 R^*-tree 4d (το ερώτημα περιλαμβάνει ένα anticor)	-	54	221	275

4.1.3.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Ας καταστήσουμε σαφή τώρα ορισμένα συμπεράσματα που μπορούν να προκύψουν από τη μελέτη των προηγηθέντων μετρήσεων.

Κατ' αρχάς, όπως και στην ενότητα 4.1.1.1 έτσι και εδώ μπορούμε να πούμε ότι όταν το σημείο ερώτησης αναφέρεται σε x διαστάσεις, τότε η καλύτερη λύση είναι να έχουμε ένα R^*-tree και στις x διαστάσεις αυτές διαστάσεις. Για παράδειγμα, αν θέσουμε ένα ερώτημα τριών διαστάσεων εκ των οποίων οι δύο είναι μεταξύ τους anticorrelated, τότε τα ελάχιστα IOs τα έχουμε όταν έχουμε ένα R^*-tree με τρεις διαστάσεις: αυτές στις οποίες τίθεται το ερώτημα.

Τώρα θα δούμε κάποιες περιπτώσεις, όπως είδαμε και στην ενότητα 4.1.2.1. Τα συμπεράσματα που προκύπτουν (όπως άλλωστε και οι αντίστοιχες μετρήσει όπως μπορείτε να παρατηρήσετε) είναι τα ίδια:

- Όταν το ερώτημα τίθεται σε έναν αριθμό διαστάσεων και υποθέτουμε ότι δύο από αυτές είναι μεταξύ τους anticorrelated, τότε η καλύτερη περίπτωση είναι οι δύο διαστάσεις που είναι anticorrelated να είναι η μία σε ένα R^*-tree και η άλλη σε ένα B^+-tree, η αμέσως καλύτερη είναι να είναι στο ίδιο R^*-tree και η χειρότερη είναι να βρίσκονται η κάθε μία σε ένα διαφορετικό R^*-tree.
- Όταν όμως το ερώτημα τίθεται σε έναν αριθμό διαστάσεων ανεξάρτητων μεταξύ τους, όμως οι δομές ευρετηρίων έχουν κάποιες διαστάσεις που είναι μεταξύ τους correlated, τότε έχουμε να πούμε ότι αφενός αν χρησιμοποιούσαμε δομές που έχουν μόνο ανεξάρτητες διαστάσεις θα είχαμε λιγότερα IOs (μπορείτε να το δείτε με μία σύγκριση μεταξύ των πειραματικών αποτελεσμάτων των ενοτήτων 4.1.1.1 και 4.1.2.1), αφετέρου αν είμαστε «υποχρεωμένοι» να χρησιμοποιήσουμε δομές ευρετηρίων που έχουν διαστάσεις που μεταξύ τους είναι correlated τότε ας είναι όσο λιγότερες γίνεται.

4.1.4 Γενικά συμπεράσματα

Εδώ θα παραθέσουμε τα κυριότερα συμπεράσματα που εξαγάγαμε από τα πειράματα που εκτελέσαμε:

- Εάν το σημείο ερώτησης είναι x διαστάσεων, τότε τα λιγότερα δυνατά IOs θα τα έχουμε όταν χρησιμοποιήσουμε ένα R^*-tree επίσης x διαστάσεων.
- Εάν ένα ερώτημα τίθεται σε διαστάσεις, κάποιες από τις οποίες είναι correlated ή anticorrelated μεταξύ τους, τότε είναι καλό να διασπείρουμε αυτές τις διαστάσεις σε διάφορες δομές. Η καλύτερη λύση είναι να έχουμε μία από αυτές τις διαστάσεις σε ένα R^*-tree και τα υπόλοιπα σε B^+-trees για κάθε σύνολο διαστάσεων που είναι μεταξύ τους correlated ή anticorrelated.
- Ένα ερώτημα που τίθεται σε μία δοσμένη δομή ευρετηρίων αποδίδει καλύτερα όταν τα δοθέντα ευρετήρια δεν περιλαμβάνουν κάποιες διαστάσεις που είναι μεταξύ τους correlated ή anticorrelated. Αν έχουμε ένα σύνολο από B^+-trees και R^*-

trees, τότε τα IOs αυξάνονται όσο οι διαστάσεις που είναι μεταξύ τους correlated ή anticorrelated αυξάνονται.

4.2 Δεδομένα δέκα διαστάσεων

Σε αυτά τα δεδομένα χρησιμοποιήσαμε δύο δομές ευρετηρίων. Είτε δέκα B^+-trees, ένα για κάθε διάσταση δηλαδή, είτε ένα R^*-tree δέκα διαστάσεων, δηλαδή ένα R^*-tree για όλες τις διαστάσεις των δεδομένων μας. Έτσι πήραμε τις ακόλουθες μετρήσεις.

4.2.1 Δεδομένα με ανεξάρτητες διαστάσεις

Πίνακας 13. Μετρήσεις για 10d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
10	1 R^*-tree 10d	-	18	165	183
	10 B^+-trees	1220	-	-	1220

Πίνακας 14. Μετρήσεις για 7d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
7	1 R^*-tree 10d	-	12	114	126
	10 B^+-trees	651	-	-	651

Πίνακας 15. Μετρήσεις για 5d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
5	1 R^*-tree 10d	-	5	98	103
	10 B^+-trees	255	-	-	225

Πίνακας 16. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
3	1 R^*-tree 10d	-	21	198	219
	10 B^+-trees	48	-	-	48

Πίνακας 17. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index	IOs στα R^*-trees (data nodes)	Συνολικά IOs

			nodes)		
2	1 R*-tree 10d	-	21	239	260
	10 B+-trees	12	-	-	12

Πίνακας 18. Μετρήσεις για 1d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τις διαστάσεις ανεξάρτητες.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B+-trees	IOs στα R*-trees (index nodes)	IOs στα R*-trees (data nodes)	Συνολικά IOs
1	1 R*-tree 10d	-	27	484	509
	10 B+-trees	2	-	-	2

4.2.1.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Αυτό που προκύπτει από τις μετρήσεις μας είναι ότι με τα B+-trees, όσο αυξάνονται οι διαστάσεις στις οποίες τίθεται το ερώτημα (από το 1 ως το 10) τόσο περισσότερα IOs γίνονται.

Με τα R*-trees παρατηρούμε κάτι διαφορετικό. Ξεκινώντας από τα ερωτήματα με 10 διαστάσεις και μειώνοντας τις διαστάσεις στις οποίες αναφέρεται το ερώτημα, παρατηρούμε ότι μέχρι και τις 5 διαστάσεις τα IOs μειώνονται. Αντίθετα όμως, αν πάμε σε ερωτήματα με κάτω από 5 διαστάσεις τα IOs αυξάνονται και μάλιστα πάρα πολύ. Χαρακτηριστικό είναι το γεγονός ότι όταν το R*-tree είναι δέκα διαστάσεων και το ερώτημα αφορά σε μία μόνο από τις διαστάσεις του, τότε γίνονται 509 IOs, που είναι πολύ περισσότερα από κάθε άλλο ερώτημα, όπως φαίνεται στις μετρήσεις μας (το μεγαλύτερο επόμενο μέγιστο είναι 260 όταν το ερώτημα αφορά σε δύο διαστάσεις).

4.2.2 Δεδομένα με τρεις διαστάσεις correlated

Πίνακας 19. Μετρήσεις για 10d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B+-trees	IOs στα R*-trees (index nodes)	IOs στα R*-trees (data nodes)	Συνολικά IOs
10	1 R*-tree 10d	-	12	48	60
	10 B+-trees	1050	-	-	1050

Πίνακας 20. Μετρήσεις για 7d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B+-trees	IOs στα R*-trees (index nodes)	IOs στα R*-trees (data nodes)	Συνολικά IOs
7	1 R*-tree 10d	-	11	59	70
	10 B+-trees	413	-	-	413

Πίνακας 21. Μετρήσεις για 5d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
5	1 R^*-tree 10d	-	5	35	40
	10 B^+-trees	90	-	-	90

Πίνακας 22. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
3	1 R^*-tree 10d	-	9	153	162
	10 B^+-trees	6	-	-	6

Πίνακας 23. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
2	1 R^*-tree 10d	-	9	153	162
	10 B^+-trees	4	-	-	4

Πίνακας 24. Μετρήσεις για 1d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
1	1 R^*-tree 10d	-	9	153	162
	10 B^+-trees	2	-	-	2

4.2.2.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Να αναφέρουμε κατ' αρχάς ότι όπως και στα δεδομένα με ανεξάρτητες διαστάσεις, έτσι και εδώ, όταν χρησιμοποιούμε B^+-trees και ξεκινήσουμε από ένα ερώτημα 10 διαστάσεων, τότε όσο μειώνονται οι διαστάσεις στις οποίες αναφέρεται το ερώτημα, τόσο πιο λίγα IOs πραγματοποιούνται. Επίσης, αξίζει να αναφερθεί ότι για ερωτήματα ίδιων διαστάσεων όταν η βάση μας έχει διαστάσεις που είναι correlated τότε γίνονται λιγότερα (αισθητά λιγότερα) IOs.

Τώρα για τα R^*-trees, παρατηρούμε ότι όταν οι διαστάσεις είναι 5 έχουμε τα λιγότερα δυνατά IOs. Για ερωτήματα πάνω των 5 διαστάσεων έχουμε μία αύξηση των IOs. Αντίθετα,

όταν έχουμε λιγότερες διαστάσεις από 5 στο ερώτημα, και περιλαμβάνονται στις διαστάσεις και οι correlated έχουμε πολύ μεγάλη αύξηση των IOs.

4.2.3 Δεδομένα με δύο διαστάσεις correlated και δύο άλλες anticorrelated

Πίνακας 25. Μετρήσεις για 10d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
10	1 R^*-tree 10d	-	13	47	60
	10 B^+-trees	1240	-	-	1240

Πίνακας 26. Μετρήσεις για 7d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
7	1 R^*-tree 10d	-	9	32	41
	10 B^+-trees	392	-	-	392

Πίνακας 27. Μετρήσεις για 5d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
5	1 R^*-tree 10d	-	6	54	60
	10 B^+-trees	95	-	-	95

Πίνακας 28. Μετρήσεις για 3d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
3	1 R^*-tree 10d	-	6	60	66
	10 B^+-trees	21	-	-	21

Πίνακας 29. Μετρήσεις για 2d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
2	1 R^*-tree 10d	-	10	254	264
	10 B^+-trees	2	-	-	2

Πίνακας 30. Μετρήσεις για 1d σημείο ερώτησης σε δεδομένα 10 διαστάσεων, με τρεις διαστάσεις τους correlated.

Διαστάσεις του σημείου ερώτησης	Χρησιμοποιούμενη δομή ευρετηρίου	IOs στα B^+-trees	IOs στα R^*-trees (index nodes)	IOs στα R^*-trees (data nodes)	Συνολικά IOs
1	1 R^*-tree 10d	-	10	254	264
	10 B^+-trees	2	-	-	2

4.2.3.1 Συμπεράσματα και σχόλια επί των μετρήσεων

Να αναφέρουμε κατ' αρχάς ότι όπως και στα δεδομένα με ανεξάρτητες διαστάσεις, έτσι και εδώ, όταν χρησιμοποιούμε B^+-trees και ξεκινήσουμε από ένα ερώτημα 10 διαστάσεων, τότε όσο μειώνονται οι διαστάσεις στις οποίες αναφέρεται το ερώτημα, τόσο πιο λίγα IOs πραγματοποιούνται. Επίσης, αξίζει να αναφερθεί ότι για ερωτήματα ίδιων διαστάσεων όταν η βάση μας έχει διαστάσεις που είναι correlated τότε γίνονται λιγότερα (αισθητά λιγότερα) IOs.

Τώρα για τα R^*-trees, παρατηρούμε ότι όταν οι διαστάσεις είναι 5 έχουμε τα λιγότερα δυνατά IOs. Για ερωτήματα πάνω των 5 διαστάσεων έχουμε μία αύξηση των IOs. Αντίθετα, όταν έχουμε λιγότερες διαστάσεις από 5 στο ερώτημα, και περιλαμβάνονται στις διαστάσεις και οι correlated έχουμε πολύ μεγάλη αύξηση των IOs.

4.2.4 Γενικά συμπεράσματα

Τα κύρια συμπεράσματα που μπορούν να εξαχθούν από τις παραπάνω μετρήσεις είναι τα ακόλουθα:

- Όταν χρησιμοποιούμε B^+-trees τότε όσο αυξάνεται το πλήθος των διαστάσεων στις οποίες αναφέρεται το ερώτημα τόσο αυξάνονται και τα IOs που απαιτούνται για την απάντηση του ερωτήματος αυτού.
- Όταν χρησιμοποιούμε B^+-trees τότε για ένα ερώτημα που αναφέρεται σε έστω x διαστάσεις, τότε αν κάποιες από αυτές τις x είναι μεταξύ τους correlated ή anticorrelated τα IOs που θα πραγματοποιηθούν είναι λιγότερα από ότι αν όλες οι x διαστάσεις είναι μεταξύ τους ανεξάρτητες.
- Όταν χρησιμοποιούμε R^*-trees, τότε μέχρι πλήθος 5 διαστάσεων μπορούμε να πούμε ότι είναι αποδοτικά. Για περισσότερες διαστάσεις δεν αποδίδουν καλά. Επίσης, όταν έχουμε R^*-trees που αναφέρονται σε πολλές διαστάσεις και θέτουμε ερωτήματα σε πολύ λίγες από τις διαστάσεις τους, τότε το κόστος εύρεσης των πλησιέστερων γειτόνων είναι πολύ αυξημένο (βλέπουμε μέχρι και το 80% των δεδομένων για να απαντήσουμε στο ερώτημά μας).

4.3 Γενικά συμπεράσματα

Από όλες τις παραπάνω μετρήσεις μπορούμε να εξάγουμε κάποια ασφαλή συμπεράσματα, τα οποία και θα παραθέσουμε εδώ:

- Κατ' αρχάς τα R^*-trees για πάνω από 5 διαστάσεις αρχίζουν και εκφυλίζονται, για αυτό και δεν θα πρέπει να χρησιμοποιούνται.
- Όσο λιγότερες δομές ευρετηρίων χρησιμοποιούμε τόσο καλύτερα αποτελέσματα παίρνουμε. Για παράδειγμα, αν έχουμε ένα ερώτημα που τίθεται σε 15 διαστάσεις (και σύμφωνα με τον προηγούμενο περιορισμό) θα προτιμήσουμε να χρησιμοποιήσουμε 3 R^*-trees από 5 διαστάσεις το καθένα τους.
- Αν χρησιμοποιούμε R^*-trees για να απαντήσουμε ένα ερώτημα πρέπει να προσέχουμε. Αν το R^*-tree έχει περισσότερες διαστάσεις από αυτές που μας ενδιαφέρουν για να απαντήσουμε στο ερώτημα, τότε το κόστος απάντησης του ερωτήματος αυξάνει πολύ. Για παράδειγμα, αν έχουμε να απαντήσουμε σε ένα ερώτημα 5 διαστάσεων και έχουμε 4 B^+-trees στις 4 διαστάσεις και ένα R^*-tree περιλαμβάνει την $5^η$ διάσταση που μας ενδιαφέρει, τότε αυτό το R^*-tree αυξάνει το κόστος. Θα ήταν πολύ καλύτερα αν είχαμε ένα B^+-tree και σε αυτή τη διάσταση.

5

Αλγόριθμος κατανομής των διαστάσεων σε ευρετήρια

Στο κεφάλαιο αυτό θα δώσουμε τον αλγόριθμο που κατασκευάσαμε ο οποίος κατανέμει ένα σύνολο από διαστάσεις (είτε ανεξάρτητες μεταξύ τους είτε correlated ή anticorrelated) σε ευρετήρια B-trees ή R-trees, για ένα δεδομένο σύνολο από ερωτήματα πλησιέστερου γείτονα που έρχονται με δεδομένες συχνότητες στη βάση δεδομένων μας.

Για να σχεδιάσουμε τον αλγόριθμό μας θα θεωρήσουμε ότι γνωρίζουμε δύο πράγματα:

- Ποιες από τις διαστάσεις της βάσης δεδομένων μας είναι μεταξύ τους correlated ή anticorrelated
- Το κόστος που έχει η απάντηση ενός ερωτήματος kNN για κάποιο δεδομένο ευρετήριο

Περισσότερες λεπτομέρειες για τον αλγόριθμο που θα σχεδιάσουμε ακολουθούν στην ενότητα 5.1.

5.1 Εισαγωγή

Υποθέτουμε αρχικά ότι έχουμε μία βάση δεδομένων με n διαστάσεις. Θα αριθμήσουμε αυτές τις διαστάσεις κατά τέτοιο τρόπο ώστε τελικά να έχουμε τις «ονοματισμένες» διαστάσεις $1,2,3,\dots,n$.

Επίσης, θα έχουμε κάποια ερωτήματα kNN που θα τίθενται σε αυτή τη βάση. Αυτά τα ερωτήματα θα τίθενται σε συγκεκριμένες διαστάσεις το καθένα και θα εμφανίζονται με συγκεκριμένες συχνότητες. Προφανώς το σημείο της ερώτησης μπορεί να το επιλέξει ο χρήστης, καθώς όπως είδαμε από τα πειράματα, το σημείο ερώτησης έχει αμελητέα επίδραση στις ανακλήσεις δεδομένων από το δίσκο. Θα αριθμήσουμε τα ερωτήματα kNN που μπορούν να τεθούν στη βάση μας, κατά τέτοιο τρόπο ώστε να έχουμε τα ερωτήματα

$Q_1, Q_2, \dots, Q_\lambda$, δηλαδή συνολικά έχουμε λ ερωτήματα πλησιέστερου γείτονα που μπορούν να τεθούν στη βάση μας. Θεωρούμε ότι το ερώτημα Q_i τίθεται με συχνότητα f_i με $0 < f_i \leq 1$, με $\sum_{i=1}^{\lambda} f_i = 1$. Θεωρούμε επιπλέον ότι το ερώτημα Q_i αφορά στις διαστάσεις $D_i \subseteq \{1,2, \dots, n\}$, δηλαδή σε ένα υποσύνολο από το σύνολο όλων των διαστάσεων που έχει η βάση μας.

Υποθέτουμε επίσης για τις ανάγκες του αλγορίθμου που θα αναπτύξουμε ότι διαθέτουμε μία συνάρτηση $cost(index\ structure\ I, kNN\ query\ Q_i)$, η οποία μας δίνει το αναμενόμενο πλήθος των IOs που εκτελείται όταν έχουμε ένα σύνολο από δομές ευρετηρίων I και τίθεται ένα ερώτημα kNN Q_i. Προφανώς, αυτή τη συνάρτηση μπορούμε να την προετοιμάσουμε με πειράματα αντίστοιχα αυτών που εκτελέσαμε στο κεφάλαιο 4.

Τέλος, υποθέτουμε ότι γνωρίζουμε ποιες είναι οι διαστάσεις που είναι μεταξύ τους correlated ή anticorrelated και αυτό το λαμβάνουμε υπ' όψιν μας στη συνάρτηση $cost(I, Q_i)$.

Αυτό που επιθυμούμε να κάνει ο αλγόριθμός μας είναι να μας επιστρέφει μία δομή ευρετηρίων I από B$^+$-trees και R*-trees έτσι ώστε το συνολικό κόστος $\sum_{i=0}^{\lambda}[f_i \times cost(I, Q_i)]$ να είναι το ελάχιστο δυνατό.

5.2 Ένας απλός αλγόριθμος

Μία πρώτη σκέψη για να σχεδιάσουμε έναν αλγόριθμο είναι η εξής απλή: εξετάζουμε όλες τις πιθανές δομές ευρετηρίων αποτελούμενες από B$^+$-trees και R*-trees στις n διαστάσεις που διαθέτουμε και επιλέγουμε αυτή τη δομή I που δίνει το μικρότερο συνολικό κόστος $\sum_{i=0}^{\lambda}[f_i \times cost(I, Q_i)]$.

Θα αποδείξουμε ότι ο αλγόριθμος αυτός είναι σίγουρα μη αποδοτικός. Ας βρούμε πόσα είναι τα πιθανά I που θα πρέπει να εξεταστούν για να βρούμε το βέλτιστο. Για τις n διαστάσεις που διαθέτουμε μπορούμε να δοκιμάσουμε να βάλουμε ένα R-tree 2 διαστάσεων και σε όλες τις υπόλοιπες διαστάσεις να έχουμε B-trees. Πόσα διαφορετικά τέτοια σενάρια με R-trees δύο διαστάσεων μπορούμε να βρούμε; Προφανώς μπορούμε να έχουμε το R-tree στις διαστάσεις 1 και 2, ή στις 1 και 3, ή στις 1 και 4 ή στις 1 και n, ή στις 2 και 3, ή… Γενικά έχουμε $\frac{n(n-1)}{2}$ διαφορετικές περιπτώσεις, ή καλύτερα (από τους κανόνες της συνδυαστικής) $\binom{n}{2} = \frac{n(n-1)}{2}$ διαφορετικές περιπτώσεις. Ομοίως, για την περίπτωση που εξετάζουμε όλες τις δομές με ένα R-tree σε 3 διαστάσεις και σε όλες τις υπόλοιπες έχουμε B-trees, έχουμε συνολικά $\binom{n}{3}$ διαφορετικά σενάρια. Με αυτήν την έννοια, το συνολικό πλήθος των σεναρίων που έχουμε να εξετάσουμε είναι τουλάχιστον ίσο από $\binom{n}{n} + \binom{n}{n-1} + \binom{n}{n-2} + \binom{n}{n-3} + \cdots + \binom{n}{2}$, αντίστοιχα όταν έχουμε ένα R-tree n διαστάσεων, αν έχουμε ένα R-tree $n-1$ διαστάσεων, αν έχουμε ένα R-tree $n-2$ διαστάσεων κ.ο.κ..

Όπως γνωρίζουμε από στοιχειώδη μαθηματική ανάλυση ισχύει $(a+b)^n = \sum_{\nu=0}^{n} \binom{n}{\nu} a^{n-\nu} b^{\nu}$, άρα αν θεωρήσουμε ότι $a = b = 1$, έχουμε ότι $\sum_{\nu=0}^{n} \binom{n}{\nu} =$

2^n. Άρα, το ελάχιστο πλήθος των σεναρίων που έχουμε να εξετάσουμε είναι $\binom{n}{n} + \binom{n}{n-1} + \binom{n}{n-2} + \binom{n}{n-3} + \cdots + \binom{n}{2} = \sum_{\nu=0}^{n} \binom{n}{\nu} - \binom{n}{1} - \binom{n}{0} = 2^n - n - 1$, δηλαδή $\Omega(2^n)$.

Άρα είναι προφανές ότι ο αλγόριθμος αυτός θα μπορούσε να λειτουργήσει για εξαιρετικά μικρές τιμές του n, όπως για παράδειγμα μέχρι το 5.

Ας δούμε τώρα έναν λίγο καλύτερο αλγόριθμο για την επίλυση του προβλήματος.

5.3 Ένας αποδοτικότερος αλγόριθμος

Τώρα θα δώσουμε τον αλγόριθμο που προτείνουμε για τη κατανομή των διαστάσεων σε ευρετήρια τύπου B$^+$-trees και R*-trees. Ο αλγόριθμος που προτείνουμε είναι greedy. Θα ξεκινήσουμε έχοντας όλες τις διαστάσεις της βάσης δεδομένων μας κάτω από B$^+$-trees και σταδιακά θα συνενώνουμε δύο δομές ευρετηρίων σε κάθε βήμα. Δηλαδή, σε κάθε βήμα τα αρχικά n ευρετήρια (δηλαδή τα αρχικά n B$^+$-trees) στην πρώτη επανάληψη θα συνενωθούν δύο B$^+$-trees και θα δημιουργηθεί ένα R*-tree δύο διαστάσεων και $n-2$ B$^+$-trees, μετά είτε θα συνενωθούν δύο B$^+$-trees και θα καταλήξουμε με δύο R*-trees δύο διαστάσεων και $n-4$ B$^+$-trees είτε θα συνενωθεί το R*-tree με ένα B$^+$-tree οπότε θα καταλήξουμε με ένα R*-tree τριών διαστάσεων και $n-3$ B$^+$-trees, κ.ο.κ.. Δηλαδή, σε κάθε βήμα ο αλγόριθμός μας θα μειώνει το πλήθος των ευρετηρίων που διαθέτουμε κατά ένα.

Ο αλγόριθμος μας δεν εξασφαλίζει ότι θα βρει το απόλυτο ελάχιστο του συνολικού κόστους, αλλά λειτουργεί ευριστικά βάσει των πειραμάτων που εκτελέσαμε.

Κατ' αρχάς, κάνουμε την παραδοχή ότι τα R*-trees θα αναφέρονται μέχρι και σε 5 διαστάσεις και όχι παραπάνω, καθώς μετά από τις 5 διαστάσεις αρχίζουν να εκφυλλίζονται. Βέβαια, ανάλογα και με το σύνολο δεδομένων που έχουμε θα μπορούσαμε να πούμε ότι τα R*-trees μπορούν να λειτουργήσουν καλά ως και τις δέκα διαστάσεις (αυτό είναι κάτι που προσδιορίζεται πειραματικά).

Τώρα ας δούμε τον αλγόριθμο:

Αλγόριθμος Κατανομής Διαστάσεων σε Ευρετήρια.

Βάλε B$^+$-trees σε όλες τις διαστάσεις της βάσης δεδομένων.
Βρες πόσα ερωτήματα περιλαμβάνουν την κάθε διάσταση. Έστω $times_i$ το πλήθος των φορών που μία διάσταση περιλαμβάνεται σε ερωτήματα. Τότε ορίζουμε ως $maxTimes = \max_{i\in\{1,2,\dots,n\}}\{times_i\}$ και $minTimes = \min_{i\in\{1,2,\dots,n\}}\{times_i\}$.
$candidates = \emptyset$
Για $i = maxTimes$ ως $minTimes$ κάνε τα ακόλουθα {
    $candidates = candidates \cup \{j, times_j = i\}$
    Υπολόγισε το συνολικό κόστος για όλα τα ερωτήματα (έστω $cost_1$).
    $cost_2 = 0$
    Όσο $cost_1 > cost_2$ κάνε τα ακόλουθα {

$cost_2 = cost_1$

Βρες τα δύο ευρετήρια τα οποία αν συνενωθούν δίνουν το μικρότερο δυνατό συνολικό κόστος (και αν το προκύπτον ευρετήριο από τη συνένωση είναι R*-tree αναφέρεται σε 5 το πολύ διαστάσεις). Μετά συνένωσέ τα και κάνε το $cost_1$ ίσο με αυτό το μικρότερο δυνατό συνολικό κόστος. Επίσης, αφαίρεσε από το σύνολο $candidates$ τα ευρετήρια που συνενώθηκαν και πρόσθεσε το νέο ευρετήριο

}

}

Επίστρεψε το σύνολο $candidates$.

Το κόστος του αλγορίθμου κατά μέγιστο είναι $O(\lambda n^3)$ στη χειρότερη περίπτωση. Ας το δικαιολογήσουμε αυτό. Προφανώς, το να βρούμε τα δύο ευρετήρια τα οποία δίνουν το μικρότερο δυνατό συνολικό κόστος έχει κόστος n^2. Επειδή σε κάθε βήμα του αλγορίθμου μειώνεται το πλήθος των ευρετηρίων κατά ένα, μπορεί τα βήματα να είναι το πολύ n, το κόστος κατά μέγιστο είναι $O(n^3)$ για να «τακτοποιήσουμε» όλες τις διαστάσεις για δεδομένο i.

Είναι εμφανές ότι ο αλγόριθμος δεν δίνει τη βέλτιστη λύση σε κάθε περίπτωση, αλλά θα δούμε για ποιους λόγους τα βήματα που εκτελεί ο αλγόριθμος είναι λογικό να γίνουν. Γιατί δηλαδή ο αλγόριθμος τείνει να δώσει καλές λύσεις.

Κατ' αρχάς, ο αλγόριθμος είναι φανερό πως θέλει πρώτιστα να «τακτοποιήσει» τις διαστάσεις εκείνες που συμμετέχουν σε πολλά ερωτήματα. Αυτό είναι λογικό, καθώς από τα πειράματα συμπεράναμε πως καλό είναι να έχουμε κατά το δυνατόν λιγότερες δομές (πχ σε ένα ερώτημα 17 διαστάσεων το καλύτερο θα ήταν να έχουμε 4 μεγάλα R-trees), αλλά όταν έχουμε ένα R-tree και ζητάμε πλησιέστερους γείτονες μόνο ως προς λίγες διαστάσεις τους το κόστος αυξάνει. Οπότε αν έχουμε 3 κοινές διαστάσεις σε δύο ερωτήματα έχει πολύ μεγάλη σημασία το πώς θα τις διαχειριστούμε. Αυτό και προσπαθεί να κάνει ο αλγόριθμος.

Μετά ελέγχει όλα τα πιθανά ζεύγη ευρετηρίων που μπορεί να κοιτάξει, βρίσκει αυτά που αν συνενωθούν θα δώσουν το μικρότερο δυνατό κόστος και συνενώνονται.

Το γεγονός ότι ο αλγόριθμος ξεκινά από τις διαστάσεις που συμμετέχουν σε πολλά ερωτήματα είναι ευριστικό, αλλά θα λειτουργεί καλά. Μπορεί για παράδειγμα να έχουμε 4 διαστάσεις που συμμετέχουν σε 10 ερωτήματα η κάθε μία, αλλά καμία δεν συμμετέχει στο ίδιο ερώτημα με κάποια άλλη. Άρα δεν προάγουμε τίποτα σε αυτό το πρώτο βήμα (όλα θα μείνουν σε B+-trees σύμφωνα με τα πειράματα που εκτελέσαμε). Αν όμως 2 συμμετέχουν στο ίδιο ερώτημα , τότε το πιθανότερο είναι να δημιουργηθεί ένα R*-tree σε αυτές τις δύο διαστάσεις.

Παρ' όλη την ευριστικότητα του αλγορίθμου, έχει ισχυρά λογικά ερείσματα και φαίνεται πως δίνει καλά αποτελέσματα. Γιατί όμως; Αυτό μπορούμε να το δούμε βασιζόμενοι στα πειραματικά αποτελέσματα. Είναι φανερό για παράδειγμα ότι αν έχουμε δύο ερωτήματα που αφορούν σε 4 διαστάσεις το καθένα, εκ των οποίων οι 2 είναι κοινές, τότε μεγαλύτερη σημασία έχει το πώς θα χειριστούμε τις 2 κοινές διαστάσεις. Αυτό μπορεί να φανεί από τα

πειράματα που εκτελέσαμε στις 4 διαστάσεις. Αν έχουμε σε όλες τις διαστάσεις B^+-trees τότε το συνολικό αναμενόμενο κόστος είναι 2464 IOs (με την υπόθεση ότι οι διαστάσεις είναι ανεξάρτητες μεταξύ τους). Αν όμως βάζαμε ένα R^*-tree 2 διαστάσεων στις δύο κοινές διαστάσεις των ερωτημάτων, τότε το κόστος θα έπεφτε στα 2292, ενώ αν βάζαμε και στις υπόλοιπες διαστάσεις (τις μη κοινές) σε R^*-trees 2 διαστάσεων (οι δύο διαστάσεις του ενός ερωτήματος που αφορούν μόνο αυτό και οι άλλες δύο που αφορούν μόνο το άλλο), τότε το κόστος πέφτει στα 278 IOs. Πρέπει όμως πρώτα να κατασκευαστούν οι δομές για τις κοινά χρησιμοποιούμενες διαστάσεις και μετά οι υπόλοιπες.

6

Επίλογος

6.1 Σύνοψη και συμπεράσματα

Στο αντικείμενο της διπλωματικής, δηλαδή στα ευρετήρια για πολυδιάστατες βάσεις δεδομένων, πραγματοποιείται πολύ έρευνα στις μέρες μας. Δεν υπάρχει όμως κάποια δομή ευρετηρίου που να δίνει γρήγορα απαντήσεις σε ερωτήματα πλησιέστερου γείτονα (ή kNN). Σε αυτήν την εργασία λοιπόν, προσπαθήσαμε να δημιουργήσουμε μία δομή, η οποία θα απαντά κατά το δυνατόν πιο γρήγορα σε τέτοιου είδους ερωτήματα.

Αρχικά μελετήθηκαν διάφορα παρόμοια προβλήματα τα οποία και περιγράφονται σε αρκετή έκταση στο κεφάλαιο 2.

Έπειτα μελετήσαμε εκτενώς τον αλγόριθμο TA καθώς και διάφορες παραλλαγές του. Ο αλγόριθμος TA είναι ο καλύτερος αλγόριθμος που υπάρχει όταν θέλεις να βρεις τα κορυφαία k αντικείμενα βάσει ενός μέτρου αξιολόγησης. Αυτό κάναμε και όταν χρησιμοποιήσαμε την παραλλαγή του αλγορίθμου TA, τον NRA, και θελήσαμε να βρούμε τα k πιο κοντινά αντικείμενα προς ένα αντικείμενο στόχο.

Κατόπιν, μελετήθηκαν οι δύο δομές ευρετηρίων που θα χρησιμοποιήσουμε για να φτιάξουμε την τελική μας δομή. Οι δομές αυτές είναι τα B^+-trees και τα R^*-trees. Η πορεία της σκέψης μας μάς οδήγησε στο να κάνουμε μία κατακόρυφη κατάτμηση των δεδομένων μας και να βάλουμε τις διάφορες διαστάσεις τους είτε σε ένα B^+-tree και είτε σε ένα R^*-tree. Το πώς θα βάλουμε τα δεδομένα μας στα B^+-trees και τα R^*-trees πλέον ήταν το αντικείμενό μας, μετά από αυτό το στάδιο.

Σε επόμενο στάδιο, εκτελέσαμε αρκετά πειράματα σε τετρα-διάστατα δεδομένα καθώς και σε δεκα-διάστατα δεδομένα, για να βγάλουμε ορισμένα συμπεράσματα ως προς το πώς συμφέρει να κατανείμουμε τις διαστάσεις στις διάφορες δομές ευρετηρίων.

Μετά από την εκτέλεση των πειραμάτων που εκτελέσαμε κατασκευάσαμε τον αλγόριθμο κατανομής των διαστάσεων σε B^+-trees και R^*-trees. Η δομή αυτή κατασκευάζεται με έναν greedy αλγόριθμο, ο οποίος βέβαια δεν εξασφαλίζει ότι δίνει τη βέλτιστη λύση, αλλά λειτουργεί με μία λογική.

6.2 Μελλοντικές επεκτάσεις

Η μελλοντική επέκταση είναι η εκτέλεση περισσότερων πειραμάτων, σε περισσότερες διαστάσεις από αυτές που μελετήσαμε εμείς και με περισσότερους συνδυασμούς από δομές ευρετηρίων. Κατόπιν, αφού παρθούν ασφαλέστερα συμπεράσματα από τα πειράματα αυτά, θα πρέπει να βελτιωθεί κατά το δυνατόν ο αλγόριθμος κατανομής των διαστάσεων στις διάφορες δομές ευρετηρίων, καθώς και να ελεγχθεί σε κάποια πραγματικά δεδομένα.

7

Βιβλιογραφία

1. **Tao, Jufei, et al.** An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *IEEE Transactions on Knowledge and Data Engineering.* October 2004, Vol. 16, 10, pp. 1169-1184.

2. *Answering Top-k Queries Using Views.* **Das, Gautam, et al.** Seoul, Korea : VLDB Endowment, 2006. 32nd international conference on Very large data bases. pp. 451-462.

3. *A model for the prediction of R-tree performance.* **Theodoridis, Yannis and Sellis, Timos.** Montreal, Quebec, Canada : ACM Press, 1996. Symposium on Principles of Database Systems. pp. 161-171.

4. *R-trees: a dynamic index structure for spatial searching.* **Guttman, Antonin.** Boston, Massachusetts : ACM Press, 1984. ACM SIGMOD international conference on Management of data. pp. 47-57.

5. **Aggrawal, Charu.** Re-designing distance functions and distance-based applications for high dimensional data. *ACM SIGMOD Record.* March 2001, Vol. 30, 1, pp. 13-18.

6. *The IGrid Index. Reversing the Dimensionality Curse For Similarity Indexing in High Dimensional Space.* **Aggrawal, Charu and Yu, Philip.** Boston, Massachusetts, United States : ACM Press, 2000. Sixth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 119-129.

7. **Agrawal, Rakesh, et al.** *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. 6003029* Armonk, NY, 1997.

8. *The R*-tree: an efficient and robust access method for points and rectangles.* **Beckman, Norbert, et al.** Atlantic City, New Jersey, United States : ACM Press, 1990. ACM SIGMOD international conference on Management of data. pp. 322-331.

9. **Dellis, Evangellos, Seeger, Bernhard and Vlachou, Akrivi.** Nearest Neighbor Search on Vertically Partitioned High-Dimensional Data. *Data Warehousing and Knowledge Discovery.* s.l. : Springer Berlin / Heidelberg, 2005, pp. 243-253.

10. *Optimal aggregation algorithms for middleware.* **Fagin, Ronald, Lotem, Amnon and Naor, Moni.** Santa Barbara, California, United States : ACM Press, 2001. ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 102-113.

11. **Hjaltason, G.R. and Samet, H.** Properties of embedding methods for similarity searching in metric spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on.* May 2003, Vol. 25, 5, pp. 530-549.

12. *Efficient Query Processing in Arbitrary Subspaces Using Vector Approximations.* **Kriegel, Hans-Peter, et al.** Wien, Austria : s.n., 2006. 18th Conference on Scientific and Statistical Database Management (SSDBM). pp. 184-190.

13. **Seidl, Thomas and Kriegel, Hans-Peter.** Optimal Multi-Step k-Nearest Neighbor Search. *ACM SIGMOD Record.* 1998, Vol. 27, 2, pp. 154-165.

14. **Tanay, Amos, Sharan, Roden and Shamir, Ron.** *Biclustering Algorithms: A Survey.* 2004.

15. **Yiu, Man Lung and Mamoulis, Nikos.** Reverse Nearest Neighbors Search in Ad-hoc Subspaces. *IEEE Transactions on Knowledge and Data Engineering.* March 2007, Vol. 19, 3, pp. 412-426.

16. *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects.* **Sellis, Timos, Roussopoulos, Nick and Faloutsos, Christos.** San Francisco, CA, USA : Morgan Kaufmann Publishers Inc, 1987. International Conference on Very Large Data Bases. pp. 507-518.

17. *Efficient and self-tuning incremental query expansion for top-k query processing.* **Theobald, Martin, Schenker, Ralf and Weikum, Gerhard.** Salvador, Brazil : ACM Press, 2005. 28th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 242-249.

18. *Towards systematic design of distance functions for data mining applications.* **Aggrawal, Charu.** Washington, D.C. : ACM Press, 2003. 9th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 9-18.

19. **Silberschatz, Abraham, Korth, Henry and Sudarschan, S.** *Database Systems Concepts.* s.l. : McGraw-Hill Education - Europe, 2001.