Μάθημα: Διαχείριση μη Παραδοσιακών Δεδομένων

Project: Quad Trees vs R-Trees

Ονοματεπώνυμο: Κωνσταντίνος Βαρδάκας

AM: 522

Email: pcs0522@uoi.gr

Δομή Κώδικα

Αρχεία Point.h και Point.cpp

Η κλάση Point αποτελεί τη βασική δομή δεδομένων για την αναπαράσταση σημείων στον δισδιάστατο χώρο και υλοποιείται στα αρχεία Point.h (αρχείο header) και Point.cpp (αρχείο υλοποίησης). Περιλαμβάνει δύο μεταβλητές τύπου float, τις x και y, οι οποίες εκφράζουν τις συντεταγμένες του σημείου στο επίπεδο.

Η κλάση παρέχει δύο βασικές συναρτήσεις για τον υπολογισμό αποστάσεων. Η συνάρτηση distance_to_point() υπολογίζει την Ευκλείδεια απόσταση του σημείου από ένα άλλο σημείο και χρησιμοποιείται στον αλγόριθμο των k-NN ερωτημάτων τόσο για τα QuadTrees όσο και για τα R-Trees, εφαρμόζοντας τον τύπο του Πυθαγορείου θεωρήματος:

$$distance = \sqrt{(x_2 - x_2)^2 + (y_1 + y_2)^2}$$

Η συνάρτηση distance_to_rectangle() υπολογίζει την ελάχιστη Ευκλείδεια απόσταση του σημείου από ένα παραλληλόγραμμο (rectangle) και χρησιμοποιείται για τα αντίστοιχα range queries των δύο δομών. Αν το σημείο βρίσκεται εντός των ορίων του παραλληλογράμμου, η απόσταση είναι μηδενική. Σε διαφορετική περίπτωση, η συνάρτηση υπολογίζει την απόσταση από τη πλησιέστερο άκρο ή γωνία του ορθογωνίου και επιστρέφει την αντίστοιχη ευκλείδεια τιμή.

Τέλος, γίνεται υπερφόρτωση του τελεστή εξόδου <<, έτσι ώστε ένα αντικείμενο τύπου Point να μπορεί να εκτυπωθεί απευθείας σε μορφή Point(x, y).

Αρχεία Rectangle.h και Rectangle.cpp

Η κλάση Rectangle, η οποία υλοποιείται στα αρχεία Rectangle.h και Rectangle.cpp, αποτελεί βασική δομή για την αναπαράσταση ορθογώνιων περιοχών στον δισδιάστατο χώρο. Χρησιμοποιείται τόσο για την υλοποίηση των χωρικών ερωτημάτων τύπου range query, όσο και για τον ορισμό των ορίων του χώρου που καταλαμβάνει κάθε κόμβος σε δομές δέντρων όπως τα R-Tree και QuadTree. Κάθε αντικείμενο της κλάσης περιγράφεται από τις συντεταγμένες του κέντρου του (x, y), καθώς και από το πλάτος (w) και το ύψος (h) του. Από αυτές τις τιμές υπολογίζονται και αποθηκεύονται τα άκρα του παραλληλογράμμου: left, right, top και bottom, ώστε να διευκολύνεται η εκτέλεση χωρικών ελέγχων.

Η κλάση περιλαμβάνει τρεις βασικές μεθόδους. Η συνάρτηση contains() ελέγχει αν ένα σημείο βρίσκεται εντός των ορίων του ορθογωνίου, και χρησιμοποιείται είτε κατά την εισαγωγή ενός νέου σημείου σε κόμβο (στην περίπτωση του QuadTree), είτε κατά τον έλεγχο συμμετοχής ενός σημείου σε δεδομένο range query. Η intersects() εξετάζει αν δύο ορθογώνια επικαλύπτονται, και χρησιμοποιείται πάλι στα range queries με σκοπό να παραλείπονται περιοχές χωρίς επικάλυψη και έτσι να μειώνεται το υπολογιστικό κόστος. Τέλος, η μέθοδος union_with() της κλάσης Rectangle υπολογίζει το ελάχιστο ορθογώνιο που περιέχει τόσο το αρχικό ορθογώνιο (this) όσο και ένα δεύτερο ορθογώνιο που δίνεται ως παράμετρο. Επιστρέφει ένα νέο Rectangle που περιλαμβάνει πλήρως και τα δυο Rectangle, και χρησιμοποιείται για τον υπολογισμό των MBR των R-trees.

Επιπλέον, υπάρχει υπερφόρτωση του τελεστή << ώστε τα αντικείμενα της κλάσης Rectangle να μπορούν να εκτυπωθούν.

Αρχείο HeapEntry.h

Το αρχείο HeapEntry.h ορίζει μία γενική (template) δομή δεδομένων HeapEntry, η οποία χρησιμοποιείται για την αποθήκευση και διαχείριση εγγραφών μέσα σε μια ουρά προτεραιότητας (priority queue). Η δομή αυτή είναι παραμετροποιημένη ως προς τον τύπο κόμβου NodeType, ώστε να μπορεί να χρησιμοποιηθεί τόσο για την QuadTree όσο και για την RTree.

Κάθε HeapEntry περιέχει τρεις μεταβλητές. Αρχικά, περιέχει την απόσταση που υπολογίζεται είτε μέσω της συνάρτησης distance_to_point, είτε μέσω της distance_to_rectangle. Αυτή η απόσταση χρησιμοποιείται για τη σύγκριση των στοιχείων μέσα στην ουρά προτεραιότητας, η οποία υλοποιείται στα k-NN queries κάθε δομής δέντρου. Ο τελεστής < είναι υπερφορτωμένος ώστε τα στοιχεία με τη μικρότερη απόσταση να έχουν την υψηλότερη προτεραιότητα. Σε περίπτωση ισοπαλίας ως προς την απόσταση, η σύγκριση γίνεται βάσει ενός μετρητή (counter), ο οποίος εισάγεται αποκλειστικά για αυτόν τον λόγο. Τέλος, η τρίτη μεταβλητή είναι μια variant που μπορεί να αποθηκεύσει είτε ένα σημείο, είτε έναν δείκτη σε έναν κόμβο τύπου NodeType, ο οποίος μπορεί να είναι είτε QuadTree είτε RTree. Οι δύο constructors της δομής επιτρέπουν τη δημιουργία αντικειμένων HeapEntry είτε με δεδομένα σημείου είτε με δείκτη κόμβου, αντιστοίχως.

Αρχεία RTree.h και RTree.cpp

Η κλάση RTree, υλοποιείται στα αρχεία RTree.h και RTree.cpp. Στην υλοποίηση του R-Tree, τα δεδομένα οργανώνονται ιεραρχικά σε φύλλα και εσωτερικούς κόμβους, όπως υποδεικνύεται από τη μεταβλητή is_leaf. Τα φύλλα αποθηκεύουν σημεία (vector<Point>), ενώ οι εσωτερικοί κόμβοι περιέχουν δείκτες σε υποκόμβους (vector<RTree*> children). Τα σημεία ταξινομούνται αρχικά βάσει των Z-order (Morton) κωδικών, ώστε να διατηρηθεί χωρική εγγύτητα. Στη συνέχεια, ομαδοποιούνται σε ομάδες μεγέθους έως max_entries, από τις οποίες κατασκευάζονται τα φύλλα, και ανοδικά προκύπτουν οι ενδιάμεσοι κόμβοι μέχρι τη ρίζα. Οι MBRs υπολογίζονται με χρήση της union_with.

RTree::insert

Για την κατασκευή του R-Tree εφαρμόζεται στρατηγική bulk loading, παρόμοια με εκείνη που χρησιμοποιήθηκε στη δεύτερη άσκηση. Τα δεδομένα αρχικά ταξινομούνται είτε βάσει της Z-order κανονικοποίησης (Morton order), είτε ακολουθώντας τη μεθοδολογία STR (Sort Tile Recursive), όπου πρώτα διαχωρίζονται σε λωρίδες κατά μήκος του άξονα x και στη συνέχεια κατά μήκος του άξονα y. Έπειτα,

τα ταξινομημένα σημεία ομαδοποιούνται σταδιακά σε κόμβους, ξεκινώντας από τα φύλλα και προχωρώντας ιεραρχικά προς τα ανώτερα επίπεδα μέχρι την κατασκευή της ρίζας του δέντρου. Ο υπολογισμός της Z-order τιμής για κάθε σημείο πραγματοποιείται με χρήση της εξωτερικής βιβλιοθήκης libmorton*.

*Για να τρέξει ο κώδικας πρέπει να γίνει clone η βιβλιοθηκη στο directory του κωδικα ως εξής git clone https://github.com/Forceflow/libmorton, περιλαμβάνεται ήδη στο φάκελο src.

Μετά τον υπολογισμό, πραγματοποιείται ταξινόμηση των σημείων, και στη συνέχεια χωρίζονται σε διαδοχικές ομάδες, καθεμία από τις οποίες περιέχει έως και max_entries στοιχεία. Για κάθε τέτοια ομάδα δημιουργείται ένας κόμβος φύλλου (με is_leaf = true), ο οποίος περιλαμβάνει τα αντίστοιχα σημεία. Το MBR του κόμβου υπολογίζεται μέσω της συνάρτησης compute_boundary, η οποία εντοπίζει τις ελάχιστες και μέγιστες τιμές των συντεταγμένων x και y και σχηματίζει το μικρότερο δυνατό ορθογώνιο που περικλείει όλα τα σημεία. Σε περιπτώσεις όπου το τελευταίο φύλλο που δημιουργείται περιέχει λιγότερες εγγραφές από το ελάχιστο επιτρεπτό όριο min_entries μεταφέρονται ορισμένα σημεία από το προηγούμενο φύλλο, εφαρμόζεται επιπλέον επεξεργασία, ώστε να καλυφθεί το κατώτατο όριο.

Μετά τη δημιουργία των φύλλων, ξεκινά η ανοδική κατασκευή του δέντρου με την ομαδοποίηση των φύλλων σε εσωτερικούς κόμβους. Η διαδικασία ακολουθεί την ίδια λογική, με κάθε parent Node να περιέχει έως max_entries παιδιά. Για κάθε κόμβο, υπολογίζεται το MBR που περικλείει τα MBR όλων των παιδιών του, χρησιμοποιώντας και πάλι τη compute_boundary. Η διαδικασία επαναλαμβάνετα, δημιουργώντας επίπεδα εσωτερικών κόμβων, μέχρις ότου όλοι οι κόμβοι ενωθούν κάτω από έναν κοινό κόμβο.

Τελικά, όταν όλα τα επίπεδα έχουν κατασκευαστεί, ο κόμβος που απομένει αποτελεί τη ρίζα του δέντρου. Αν αυτός περιέχει σημεία απευθείας, τότε είναι και φύλλο, διαφορετικά αποτελεί εσωτερικό κόμβο που εποπτεύει ολόκληρη τη δομή.

RTree::range_query

Η συνάρτηση range_query() πραγματοποιεί αναζήτηση όλων των σημείων που βρίσκονται εντός μιας συγκεκριμένης περιοχής (ορθογώνιο range_rect) και επιστρέφει ένα διάνυσμα vector<Point> με τα σημεία που ικανοποιούν το ερώτημα.

Αρχικά, γίνεται έλεγχος αν η περιοχή που καλύπτει ο τρέχων κόμβος (boundary) τέμνεται με την περιοχή αναζήτησης range_rect, μέσω της intersects(). Αν δεν υπάρχει επικάλυψη, τότε αποκλείεται να υπάρχουν σχετικά σημεία στον υποκόμβο, οπότε επιστρέφεται απευθείας ένα κενό διάνυσμα. Αυτός ο έλεγχος αποτελεί βήμα βελτιστοποίησης, καθώς αποφεύγει περιττές αναδρομές και συγκρίσεις.

Στη συνέχεια, εφόσον υπάρχει επικάλυψη εάν ο κόμβος είναι φύλλο, γίνεται έλεγχος σε όλα τα σημεία που περιέχει για το αν βρίσκονται εντός του range_rect μέσω της contains(). Όσα σημεία ανήκουν στην περιοχή, προστίθενται στο αποτέλεσμα. Αντίθετα, εάν ο κόμβος είναι εσωτερικός , η συνάρτηση καλείται αναδρομικά για κάθε παιδί του. Τα αποτελέσματα από τα υποδέντρα συγκεντρώνονται στο διάνυσμα found μέσω insert(), ώστε να διατηρείται ενιαίο αποτέλεσμα. Τέλος, επιστρέφεται το συνολικό διάνυσμα σημείων που βρίσκονται εντός της περιοχής αναζήτησης.

RTree::knn_query

Η συνάρτηση knn_query() εκτελεί αναζήτηση των k πλησιέστερων γειτόνων ενός σημείου query μέσα στο R-Tree. Επιστρέφει ένα διάνυσμα από ζεύγη (pair<Point, float>) που περιλαμβάνουν το σημείο και την απόστασή του από το query.

Η αναζήτηση βασίζεται σε ένα min-heap, το οποίο υλοποιείται μέσω της priority_queue με στοιχεία τύπου HeapEntry<RTree>. Ο τελεστής < της HeapEntry, όπως αναφέρθηκε, έχει υπερφορτωθεί ώστε το στοιχείο με τη μικρότερη απόσταση να έχει τη μεγαλύτερη προτεραιότητα. Η πρώτη εγγραφή του heap είναι το ίδιο το root του δέντρου, συνοδευόμενο από την απόσταση του query από το boundary του root, μέσω της συνάρτησης distance_to_rectangle. Επίσης προστίθεται και ο counter, ο οποίος εξασφαλίζει σταθερή σειρά ανάμεσα σε αντικείμενα με ίσες αποστάσεις, όπως έχει αναφερθεί στη δομή HeapEntry.

Η κύρια λογική της αναζήτησης υλοποιείται μέσα σε έναν βρόχο while, ο οποίος συνεχίζει να εκτελείται μέχρι να εντοπιστούν k κοντινότερα σημεία ή μέχρι να

αδειάσει το heap. Κάθε στοιχείο που αφαιρείται από την κορυφή του heap μπορεί να είναι είτε ένα σημείο, είτε ένας κόμβος του δέντρου, κάτι που διαχειρίζεται η μεταβλητή variant entry.data. Ο έλεγχος γίνεται μέσω της holds_alternative<Point>(), ώστε να διαπιστωθεί τι είδος δεδομένων περιέχει το entry.

Έτσι, εάν το στοιχείο entry.data είναι σημείο, προστίθεται απευθείας στα αποτελέσματα results μαζί με την απόστασή του, ενώ εάν είναι κόμβος (const RTree*) ελέγχεται εάν είναι φύλλο ή εσωτερικός κόμβος. Στην περίπτωση που είναι φύλλο, κάθε σημείο προστίθεται στο heap, μαζί με το distance_to_point και τον counter που του αντιστοιχεί. Στην περίπτωση που είναι εσωτερικός, τότε υπολογίζεται η απόσταση του query από το boundary κάθε παιδιού, και κάθε παιδί προστίθεται στο heap αντίστοιχα. Ο βρόχος σταματά μόλις βρεθούν k σημεία ή αδειάσει τελείως το heap, και συνεπώς έχουν ελεγχθεί όλα τα σημεία πριν συμπληρωθούν οι k γείτονες. Τελικά, η συνάρτηση επιστρέφει ένα vector<pair<Point, float>> που περιέχει τα k πλησιέστερα σημεία του query.

RTree::print_tree

Η συνάρτηση print_tree() υλοποιεί μια αναδρομική εκτύπωση του δέντρου R-Tree, με σκοπό την απεικόνιση της δομής του. Αρχικά, εκτυπώνει αν ο τρέχων κόμβος είναι φύλλο (is_leaf) καθώς και το όριο του boundary που καλύπτει. Αν πρόκειται για φύλλο, εκτυπώνει όλα τα σημεία που περιέχει. Σε διαφορετική περίπτωση, η συνάρτηση καλεί αναδρομικά τον εαυτό της για κάθε παιδί, αυξάνοντας το επίπεδο βάθους.

Αρχεία QuadTree.h και QuadTree.cpp

Η κλάση QuadTree υλοποιείται στα αρχεία QuadTree.h και QuadTree.cpp. Κάθε κόμβος του δέντρου καλύπτει μια ορθογώνια περιοχή, όπως ορίζεται από τη μεταβλητή boundary, και μπορεί να περιέχει μέχρι έναν συγκεκριμένο αριθμό σημείων, που καθορίζεται από τη μεταβλητή capacity. Η αποθήκευση των σημείων γίνεται αρχικά στον ίδιο τον κόμβο, μέσω της μεταβλητής points. Όταν ξεπεραστεί το όριο χωρητικότητας, ο κόμβος υποδιαιρείται σε τέσσερα επιμέρους τεταρτημόρια,

northwest, northeast, southwest και southeast, τα οποία υλοποιούνται με δείκτες προς αντίστοιχους νέους υποκόμβους. Η μεταβλητή divided υποδεικνύει εάν ένας κόμβος έχει ήδη υποδιαιρεθεί. Όλη η υλοποίηση της λειτουργικότητα querying του QuadTree (range_query, knn_query), είναι παρόμοια με την αντίστοιχη του R-tree

QuadTree::insert

Η συνάρτηση insert είναι υπεύθυνη για την εισαγωγή ενός νέου σημείου στη δομή QuadTree. Η διαδικασία ξεκινά με τον έλεγχο του αν το σημείο βρίσκεται εντός του ορθογωνίου (boundary) που ορίζει την περιοχή του συγκεκριμένου κόμβου. Αν το σημείο βρίσκεται εκτός των ορίων αυτών, η εισαγωγή απορρίπτεται και η συνάρτηση επιστρέφει false.

Σε αντίθετη περίπτωση, εάν ο κόμβος δεν έχει ακόμη διασπαστεί (δηλαδή divided == false) και διαθέτει ελεύθερο χώρο (δηλαδή points.size() < capacity), τότε το σημείο μπορεί να προστεθε. Πριν από την προσθήκη, πραγματοποιείται έλεγχος για αποφυγή duplicate σημείων, συγκρίνοντας τις συντεταγμένες του σημείου με τα ήδη αποθηκευμένα. Αν το σημείο είναι μοναδικό, προστίθεται και η συνάρτηση επιστρέφει true.

Όταν ο κόμβος φτάσει τη χωρητικότητά του και δεν έχει ακόμη διασπαστεί, καλείται η συνάρτηση subdivide, η οποία διασπά τον τρέχοντα κόμβο σε τέσσερις επιμέρους τεταρτημόρια: northwest, northeast, southwest και southeast. Κάθε νέος υποκόμβος δημιουργείται με νέο ορθογώνιο, το οποίο καλύπτει το αντίστοιχο τεταρτημόριο του αρχικού χώρου. Αμέσως μετά τη διάσπαση, όλα τα προϋπάρχοντα σημεία του κόμβου επανεισάγονται στους νέους υποκόμβους, και η λίστα points αδειάζει.

Αφού ολοκληρωθεί η διάσπαση και ανακατανομή, επιχειρείται η εισαγωγή του νέου σημείου στους τέσσερις υποκόμβους. Το σημείο εισάγεται στον πρώτο υποκόμβο που το περιέχει, και αν η εισαγωγή ολοκληρωθεί επιτυχώς, επιστρέφεται true. Αλλιώς, στην περίπτωση που δεν εντοπιστεί κατάλληλος υποκόμβος ή αποτύχει η εισαγωγή, η συνάρτηση επιστρέφει false. Στην πραγματικότητα το τελευταίο return false είναι unreachable, και έχει εισαχθεί για τον compiler.

QuadTree::range_query

Η λογική της αναζήτησης περιοχής είναι σχεδόν πανομοιότυπη: ελέγχεται αν το boundary του τρέχοντος κόμβου τέμνεται με το ορθογώνιο αναζήτησης και αν ναι, αναζητούνται σημεία είτε στα φύλλα είτε στους υποκόμβους. Η μόνη διαφορά εντοπίζεται στο ότι το QuadTree έχει πάντα τέσσερις σταθερούς υποκόμβους αντί για μεταβλητό πλήθος όπως στο RTree.

QuadTree::knn_query

Η εύρεση των k πλησιέστερων γειτόνων είναι σχεδόν ίδια σε δομή: γίνεται χρήση min-heap (με HeapEntry), εναλλαγή μεταξύ κόμβων και σημείων μέσω std::variant, και τερματισμός όταν έχουν βρεθεί k σημεία. Η διαφορά είναι μόνο στο traversal: το QuadTree εξετάζει πάντα τέσσερα προκαθορισμένα παιδιά, ενώ το RTree χρησιμοποιεί έναν γενικό κατάλογο υποκόμβων.

Δομή του Dataset

Τα Dataset που χρησιμοποιήθηκαν αφορούν γεωγραφικές τοποθεσίες από περιοχές των Ηνωμένων Πολιτειών της Αμερικής. Περιέχουν παρόμοια σημεία, αλλά διαφέρουν στο μέγεθος, το πρώτο αρχείο T2 διαθέτει 2.280.427 δεδομένα ενώ το T5 διαθέτει 5.043.188 δεδομένα. Τα σημεία και στις δύο περιπτώσεις ανήκουν στο ίδιο εύρος:

- T2: $x \in [-124.7595, -66.9875]$, $y \in [24.5219, 49.1668]$
- T5: $x \in [-124.73, -66.9887]$, $y \in [24.5635, 49.1611]$

Οι κατανομές δεν είναι uniform και δεν παρατηρείται κάποια σαφής διάταξη ανά άξονα. Εντοπίζονται μερικά clusters, αλλά συνολικά η διάταξη είναι αρκετά ακανόνιστη και διάσπαρτη, όπως παρατηρείται και στο ακόλουθο διάγραμμα:

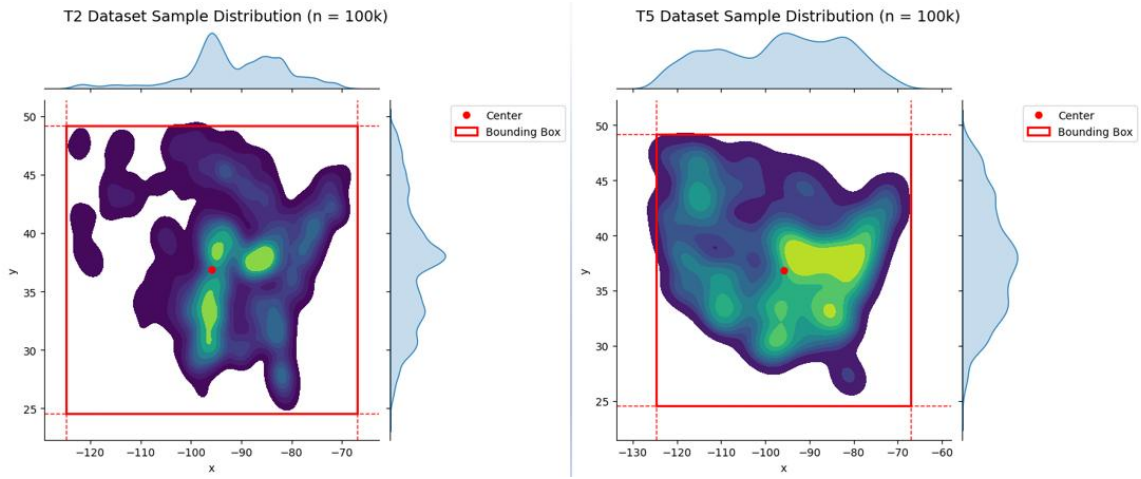

Figure 1: Κατανομή των Δεδομένων των δυο Dataset

Αναφορικά πλέον με τη δομή των query, είναι πιο uniform κατανεμημένα και απαρτίζονται από 5 διαφορετικά αρχεία τα οποία διαφέρουν στο μέγεθος του εύρους που καλύπτει το range query. Κάθε τέτοιο αρχείο αποτελείτε από 10000 queries, και ως point για το k-NN query, χρησιμοποιήθηκε το κέντρο κάθε ορθογωνίου. Η κατανομή των κέντρων αυτών φαίνεται στο παρακάτω σχήμα.

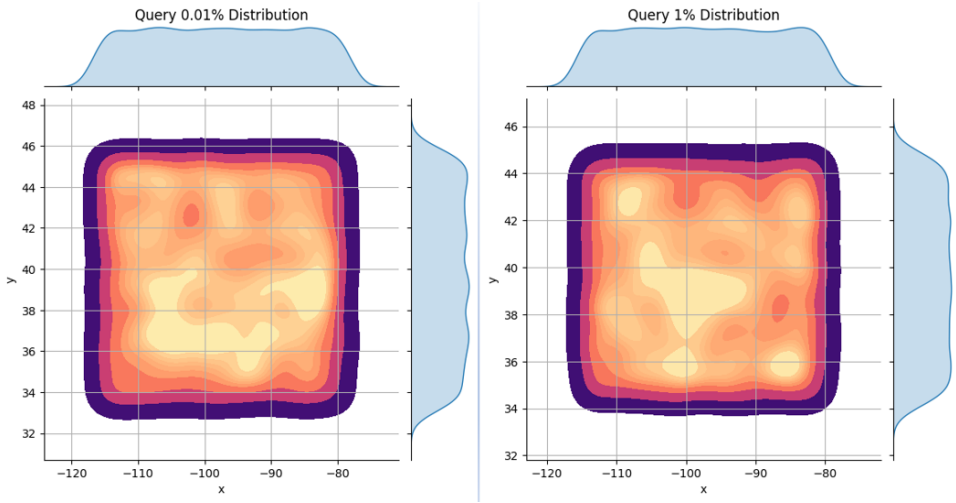

Figure 2: Κατανομή των κέντρων των range queries ορθογωνίων

Τα αντίστοιχα ορθογώνια ορίζονται και από το πλάτος και ύψος τους πέρα από το κέντρο τους, και για κάθε αρχείο queries, το εμβαδόν κυμαίνεται γύρω από μια διαφορετική μέση τιμή

- Αρχείο 0.01%: μέσο εμβαδόν 0.051
- Αρχείο 0.05%: μέσο εμβαδόν 0.25
- Αρχείο 0.1%: μέσο εμβαδόν 0.51
- Αρχείο 0.5%: μέσο εμβαδόν 2.53
- Αρχείο 1%: μέσο εμβαδόν 5.07

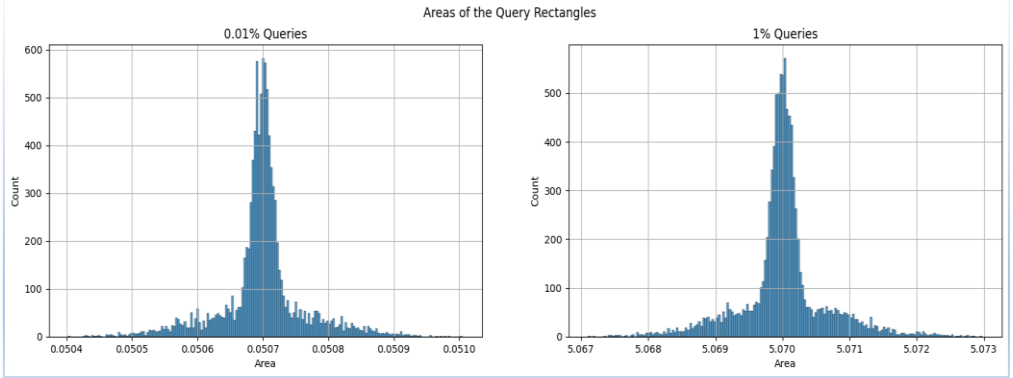

Figure 3: Κατανομές Εμβαδού του αρχείου με τα μικρότερα queries σε σύγκριση με αυτό με τα μεγαλύτερα

Κάθε query διαφέρει σε αναλογίες αλλά το εμβαδόν σε κάθε περίπτωση κυμαίνεται στο ίδιο εύρος, όπως φαίνεται και από το παρακάτω διάγραμμα. Όσο μεγαλύτερο είναι το ύψος, τόσο μικρότερο είναι το πλάτος και αντίστροφα.

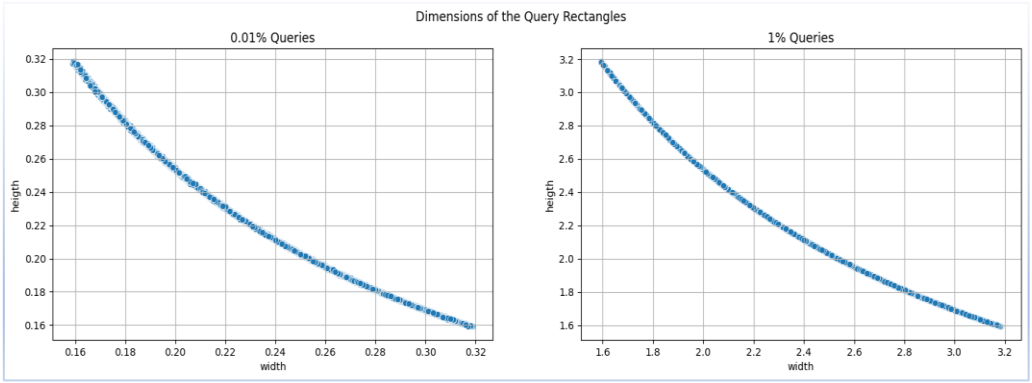

Figure 4: Ύψος x Πλάτος του εύρους κάθε query για τα αρχεία με τα μικρότερα και μεγαλύτερα εύρη

Αποτελέσματα

Βελτιστοποίηση τιμής Capacity στο Quad Tree

Αρχικά, πραγματοποιήθηκε αναζήτηση της κατάλληλης τιμής του capacity για το Quad Tree, δηλαδή του μέγιστου αριθμού σημείων που μπορεί να περιέχει ένα φύλλο πριν αυτό διασπαστεί. Για αυτές τις δοκιμές χρησιμοποιήθηκε το T2 dataset με k = 3 και σκοπός ήταν η βελτίωση του χρόνου εκτέλεσης του k-NN query του αρχείου 0.01%. Η βέλτιστη τιμή επιλέχθηκε με βάση τη μέση απόδοση σε πολλαπλά queries. Το αποτέλεσμα αυτού του tuning φαίνονται στον παρακάτω πίνακα.

capacity	**mean time μs/query**	**Max depth**	**Min depth**	**Avg points per leaf**	**q1**	**q3**	**total leaves**	**internal nodes**
4	86.7533	23	2	1.64924	1	3	1382389	460796
8	68.1143	20	2	3.32551	2	5	685636	228545
16	58.9574	17	2	6.71635	4	10	339502	113167
32	59.8129	15	2	13.30900	8	18	171334	57111
64	72.5485	15	2	26.86070	15	37	84895	28298
128	94.5242	12	2	52.21230	31	71	43675	14558
256	127.5290	11	2	106.27800	61	148	21457	7152

Table 1: Πίνακας αποτελεσμάτων χρόνου και δομής του δέντρου, για διάφορες τιμές capacity

Για το συγκεκριμένο dataset και αριθμό k, τα καλύτερα αποτελέσματα φαίνεται να λαμβάνονται όταν το capacity είναι μεταξύ 16 και 32.

Βελτιστοποίηση τιμών Min και Max Entries στο R-Tree

Αντίστοιχη διαδικασία βελτιστοποίησης εφαρμόστηκε και για τις υπερπαραμέτρους min και max entries του R-Tree, εξετάζοντας και τις δύο υλοποιημένες μεθόδους εισαγωγής δεδομένων. Και στις δύο περιπτώσεις, οι καλύτεροι χρόνοι για k-NN queries επιτεύχθηκαν με min = 4 και max = 8. Επιπλέον, διαπιστώθηκε

ότι η μέθοδος STR εισαγωγής παρουσίασε καλύτερη ομαδοποίηση των δεδομένων, οδηγώντας σε γενικά πιο αποδοτική απόκριση σε ερωτήματα k-NN σε σχέση με την Z-order ταξινόμηση. Στον πίνακα 2 παρουσιάζονται οι σχετικοί χρόνοι για z-order inserted.

Max entries	**2*min**	**3*min**	**4*min**
Min entries			
4	129.46	165.18	175.72
8	178.472	203.38	244.55
16	271.552	313.52	348.91
32	321.905	409.44	568.93
64	584.56	717.74	884.31

Table 2: Πίνακας αποτελεσμάτων χρόνου του k-NN, για διάφορες τιμές min και max entries για z-order inserted

Στον ακόλουθο πίνακα φαίνονται και τα αντίστοιχα αποτελέσματα, όταν τα δεδομένα ομαδοποιούνται σύμφωνα με την ταξινόμηση STR.

Max entries	**2*min**	**3*min**	**4*min**	**5*min**
Min entries				
4	110.748	115.385	112.657	129.127
8	114.713	124.251	142.86	171.055
16	145.113	167.538	199.031	216.589
32	199.24	259.802	296.271	316.802
64	279.377	322.112	349.591	408.514

Table 3: Πίνακας αποτελεσμάτων χρόνου του k-NN, για διάφορες τιμές min και max entries για STR inserted

Σύγκριση k-NN με baseline μεθοδολογία

Για την αξιολόγηση των αποτελεσμάτων, χρησιμοποιήθηκε και μια *naïve* μεθοδολογία ως baseline. Συγκεκριμένα, για κάθε query point, υπολογίστηκαν οι αποστάσεις από όλα τα σημεία του dataset T2, έγινε ταξινόμηση των αποστάσεων και επιστράφηκαν τα *k* πλησιέστερα σημεία. Η απόδοση της κάθε μεθόδου συγκρίθηκε ως προς τον μέσο χρόνο ανά query (σε μικροδευτερόλεπτα):

Method	**Naïve**	**Quad Tree***	**R-Tree (z)***	**R-Tree (STR)***
Time (μsec/query)	240.33	44.59	151.38	99.84

*Table 4: Σύγκριση χρόνου ανά query, για κάθε μεθοδολογία. *Οι χρόνοι που αναδεικνύονται για τα δέντρα είναι median τιμές από 10 επαναλήψεις*

Από τα αποτελέσματα γίνεται σαφές ότι το Quad Tree πέτυχε την καλύτερη απόδοση, όντας περίπου 5.39 φορές ταχύτερο από τη baseline μέθοδο. Ακολούθησε το R-Tree με STR εισαγωγή, που ήταν 2.41 φορές ταχύτερο, ενώ το R-Tree με Z-order ταξινόμηση εμφάνισε μικρότερη βελτίωση, περίπου 1.59 φορές ταχύτερο από τη naïve προσέγγιση.

Ανάλυση Περιοχών ως προς τη Δυσκολία KNN Query

Ακολούθησε μια ανάλυση με σκοπό να διερευνηθεί η ύπαρξη “εύκολων” και “δύσκολων” περιοχών στον χώρο, δηλαδή περιοχών όπου τα KNN queries εκτελούνται είτε γρηγορότερα είτε πιο αργά. Η μελέτη βασίστηκε στο dataset T2, με k = 3, χρησιμοποιώντας τα queries από το αρχείο 0.01%.

Για κάθε query σημείο, πραγματοποιήθηκαν 10 επαναλήψεις, ώστε να υπολογιστούν ο στατιστικός μέσος χρόνος εκτέλεσης και η τυπική απόκλιση (mean και std) και να εντοπιστούν πιθανά μοτίβα σχετιζόμενα με τη γεωγραφική θέση των queries ή την τοπική πυκνότητα των δεδομένων. Στο παρακάτω διάγραμμα φαίνεται ο μέσος όρος και η διακύμανση για κάθε query point του αρχείου 0.01%.

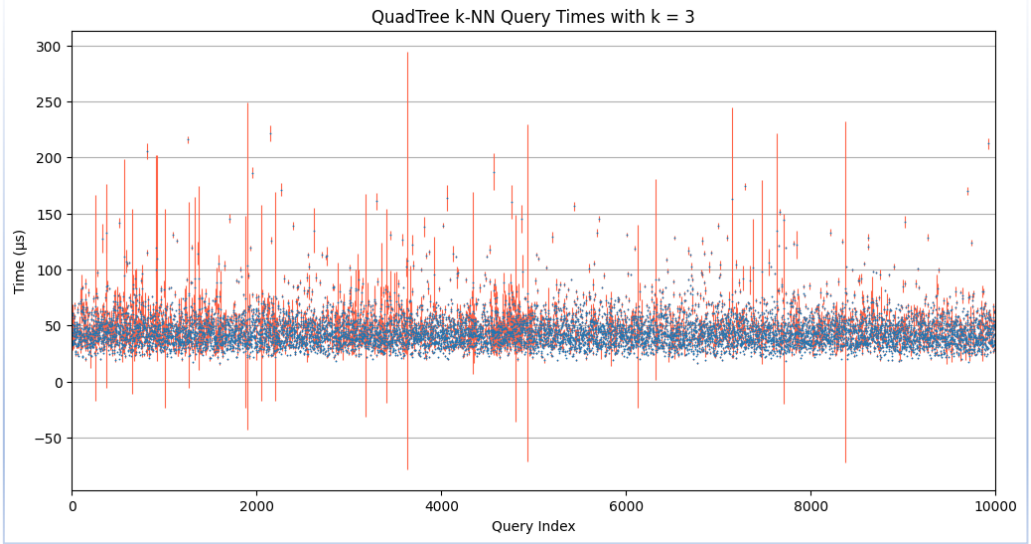

Figure 5: Μέσος χρόνος και τυπική απόκλιση για κάθε query του αρχείου 0.01%, στο T2 dataset

Καθώς όμως ο αλγόριθμος είναι ντετερμινιστικός, τα σημεία με μεγάλο std προκύπτουν από outlier λόγω του μηχανήματος, όπως φαίνεται και από το σημείο 3630, το οποίο είναι αυτό με τη μεγαλύτερη διακύμανση.

επανάληψη	1	2	3	4	5	6	7	8	9	10
χρόνος (μsec)	638	77	46	45	46	46	47	47	46	45

Table 5: Χρόνος του query 3630 στο T2 dataset, για κάθε επανάληψη

Συνεπώς λαμβάνεται η τιμή median, ως αντιπροσωπευτική τιμή για χρόνου για δεδομένο query.

Η κατανομή των median τιμών των query, φαίνεται να ακολουθούν right-skewed κατανομή, γεγονός που υποδηλώνει την ύπαρξη περιοχών στον χώρο όπου τα queries απαιτούν περισσότερο χρόνο για να απαντηθούν, δηλαδή υπάρχουν «πιο δύσκολες» περιοχές.

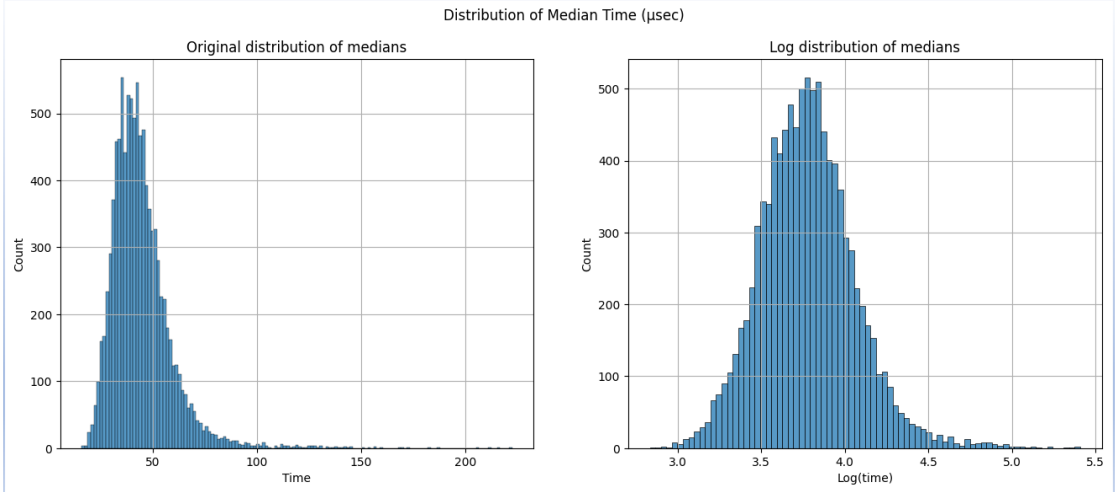

Figure 6: Κατανομή των median χρόνων των query και κατανομή των log(median χρονων)

Για την καλύτερη οπτικοποίηση αυτής της πληροφορίας, εφαρμόστηκε λογαριθμικός μετασχηματισμός (log transformation) στις τιμές χρόνου και η μετασχηματισμένη τιμή χρησιμοποιήθηκε ως χρωματική κλιμάκωση για τα αντίστοιχα σημεία του query set (αρχείο 0.01%). Έτσι, καθίσταται εφικτή η εντόπιση των περιοχών που συνδέονται με υψηλότερο κόστος υπολογισμού.

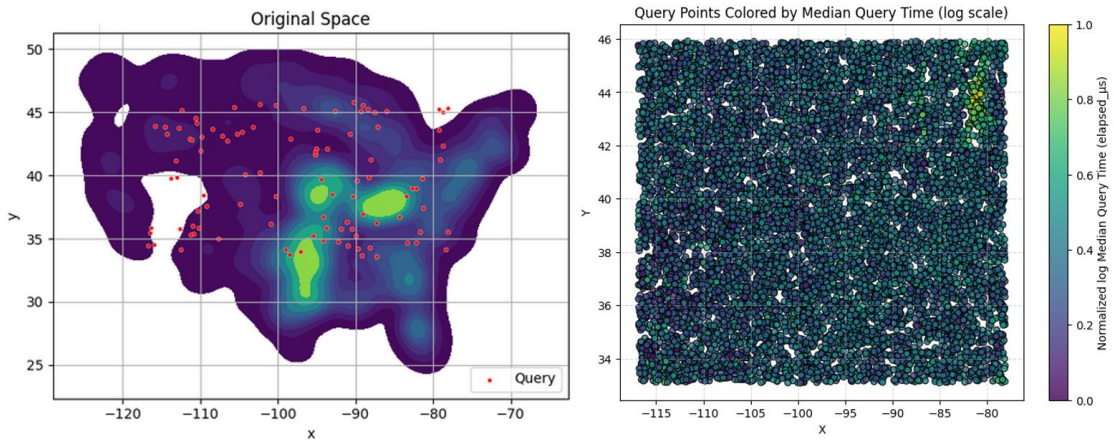

Figure 7: Αριστερά ο χώρος των δεδομένων του T2 dataset με sample query points και δεξιά τα query points, με μωβ τα πιο γρήγορα και ανοιχτό πράσινο τα πιο αργά

Από αυτό το δεξί σχήμα της εικόνας 7, φαίνεται ότι η περιοχή που οφείλεται για το right skewness της κατανομής χρόνων ανήκει προσεγγιστικά στο εύρος $x \in [-81, -80]$, $y \in [42, 45]$.

Μετασχηματισμός Χώρου

Στη συνέχεια επιχειρήθηκε η βελτιστοποίηση των χρόνων k-NN αναζητήσεων στο Quad Tree μέσω κατάλληλου μετασχηματισμού του χώρου, με στόχο τη βελτίωση της κατανομής των δεδομένων στο δέντρο. Επιλέχθηκε ένας τύπος μετασχηματισμού που δεν επηρεάζει τις αποστάσεις μεταξύ των σημείων, ώστε να διατηρείται η ακρίβεια των αποτελεσμάτων. Συγκεκριμένα, χρησιμοποιήθηκαν περιστροφές (rotations) του χώρου.

Ο στόχος ήταν να προκύψει ένα πιο ομοιόμορφα ισορροπημένο Quad Tree, γεγονός που θα μπορούσε να μειώσει τον μέσο χρόνο αναζήτησης. Αρχικά εφαρμόστηκε PCA (Principal Component Analysis) για την ευθυγράμμιση των των αξόνων με την μεγαλύτερη διακύμανση με τους άξονες διαχωρισμού του δέντρου ($x = 0$, $y = 0$). Ωστόσο, καθώς ο πίνακας περιστροφής που προέκυψε από το PCA ήταν πολύ κοντά στον identity, η ευθυγράμμιση που προσέφερε ήταν ελάχιστη. Συνεπώς, για να διαπιστωθεί αν υπάρχει όντως διαφορά με rotation, μετά το PCA έγινε rotation κατά 45° (original → PCA → 45°).

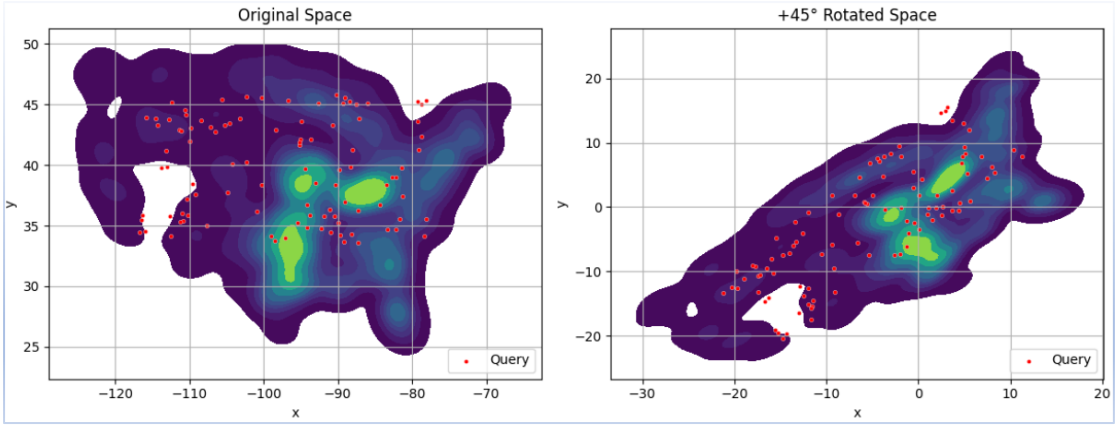

Figure 8: Μετασχηματισμός του χώρου σε 45º μακριά από το PCA transformation

Ο μετασχηματισμός των σημείων query πραγματοποιήθηκε με τον ίδιο transformation matrix που προέκυψε από το PCA των δεδομένων, ώστε να διατηρείται η σχετική γεωμετρία μεταξύ queries και dataset. Στη συνέχεια, εφαρμόστηκε επιπλέον περιστροφή κατά 45° μέσω αντίστοιχου transformation matrix.

Τα αποτελέσματα αυτής της περιστροφής δεν έδειξαν σημαντική διαφορά στον χρόνο εκτέλεσης των k-NN. Συγκεκριμένα, ο μέσος χρόνος ανά query μειώθηκε οριακά, από 44.59 μsec σε 43.43 μsec (βελτίωση ~2.5%). Για να αξιολογηθεί αν η διαφορά αυτή είναι στατιστικά σημαντική, οι χρόνοι μετασχηματίστηκαν με λογαριθμική κλίμακα ώστε η κατανομή τους να προσεγγίζει την κανονική. Στη συνέχεια, εφαρμόστηκε paired t-test, το οποίο απέδωσε p-value = 4.4×10^{-15}, υποδεικνύοντας ότι, αν και η διαφορά είναι μικρή, είναι στατιστικά σημαντική. Ωστόσο, η πρακτική της σημασία παραμένει αμελητέα.

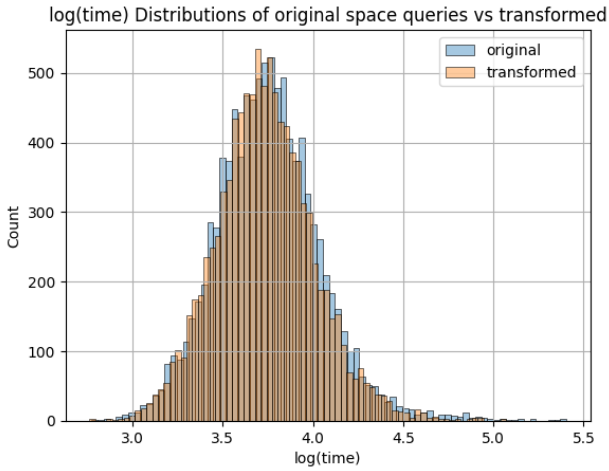

Figure 9: Κατανομές των log(time) πριν και μετά τον μετασχηματισμό του χώρου

Πειραματική Αξιολόγηση Περιστροφών

Ως συνέχεια της προηγούμενης ανάλυσης, πραγματοποιήθηκε περιστροφή του χώρου από 0° έως 90° ως προς τους άξονες του PCA, με βήμα 1°. Για κάθε γωνία περιστροφής, κατασκευάστηκε νέο Quad Tree και συλλέχθηκαν στατιστικά χαρακτηριστικά της δομής του, όπως: min depth, max depth, μέσος αριθμός σημείων ανά φύλλο, τυπική απόκλιση πληθυσμού φύλλων, καθώς και τα 1ο και 3ο τεταρτημόρια (Q1, Q3) του μεγέθους των φύλλων.

Στόχος αυτής της διαδικασίας ήταν η διερεύνηση ύπαρξης συσχέτισης ανάμεσα στα στατιστικά χαρακτηριστικά του δέντρου και τον μέσο χρόνο εκτέλεσης ενός k-NN query. Εφόσον εντοπιστεί τέτοια σχέση, σε δεύτερο στάδιο εξετάζεται κατά πόσο η γωνία περιστροφής μπορεί να προβλέψει ή να ελέγξει τα στατιστικά αυτά, ώστε να εντοπιστούν επιθυμητές γωνίες περιστροφής του χώρου που βελτιστοποιούν τη δομή του δέντρου και κατ' επέκταση την αποδοτικότητα των queries.

Η συνολική προσέγγιση στοχεύει στο να κατασκευαστεί μια στρατηγική ευθυγράμμισης του χώρου πριν την κατασκευή του δέντρου, η οποία να οδηγεί σε πιο ρηχά και ισορροπημένα δέντρα, προσφέροντας ταχύτερους χρόνους απάντησης για επαναλαμβανόμενα k-NN queries.

Μια αρχική ανάλυση των δεδομένων, δείχνει ότι υπάρχει συσχέτιση ορισμένων χαρακτηριστικών με το χρόνο εκτέλεσης των query. Συγκεκριμένα, όσο μειώνεται η ανομοιογένεια του δέντρου (π.χ. αυξάνεται το min depth και μειώνεται το max depth) αυξάνεται ο μέσος όρος των query και μειώνονται τα outliers.

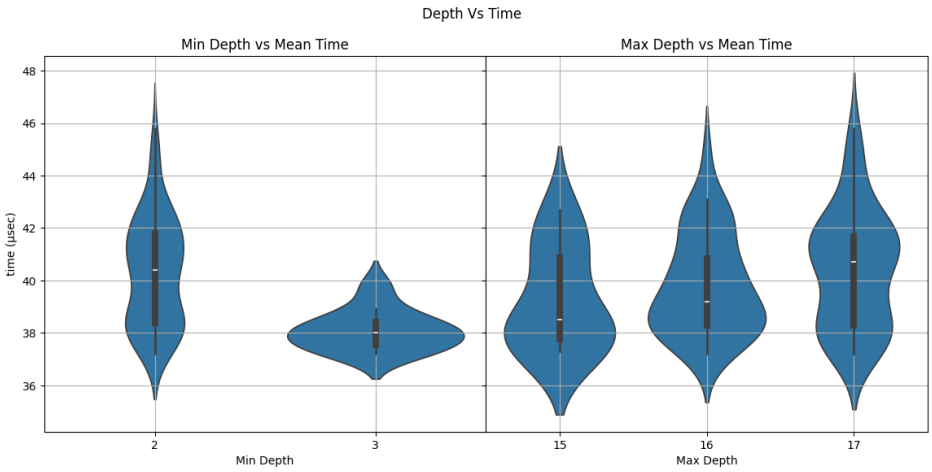

Figure 10: Violin plots του min depth και max depth ως προς τον χρόνο σε μsec

Στο επόμενο στάδιο εφαρμόστηκε μία εξαντλητική αναζήτηση χαρακτηριστικών για πρόβλεψη χρόνου k-NN, χρησιμοποιώντας ένα από μοντέλο Decision Tree με max_depth = 4. Ως μετρική αξιολόγησης χρησιμοποιήθηκε το R^2, καθώς ένα θετικό R^2 δείχνει μείωση της αβεβαιότητας της πρόβλεψης, σε σύγκριση με τη χρήση του μέσου όρου της κατανομής χρόνων. Μετά από αυτό το search, το καλύτερο υποσύνολο βρέθηκε να είναι το max depth, min depth και το τρίτο τεταρτημόριο του μεγέθους των φύλλων. Το αντίστοιχο R^2 που επετεύχθη ήταν 0.161, το οποίο είναι χαμηλό, αλλά επιβεβαιώνει ότι η χρήση αυτών των χαρακτηριστικών βελτιώνει την προβλεψιμότητα σε σχέση με το baseline μοντέλο (χρήση του μέσου όρου χωρίς χαρακτηριστικά).

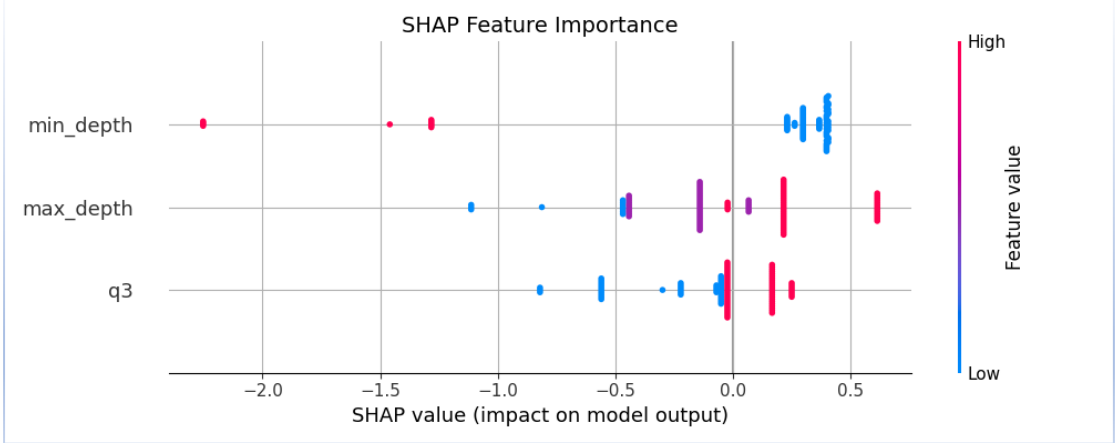

Figure 11: SHAP Values για το Decision Tree με το καλύτερο υποσύνολο των χαρακτηριστικών.

Η ανάλυση των SHAP values ανέδειξε ως σημαντικότερο χαρακτηριστικό το min depth, το οποίο συνδέεται με μείωση του χρόνου εκτέλεσης όταν η τιμή του είναι υψηλότερη (βάθος 3 αντί για 2). Ακολουθεί το max depth, το οποίο έχει αντίστροφη επίδραση: όσο μεγαλύτερο είναι το μέγιστο βάθος, τόσο αυξάνεται ο αναμενόμενος χρόνος για ένα query. Τέλος, το χαρακτηριστικό Q3, φαίνεται επίσης να συσχετίζεται θετικά με τον χρόνο, όσο αυξάνεται η τιμή του, τόσο αυξάνεται και ο προβλεπόμενος χρόνος ανά query.

Quad Tree vs R-Tree k-NN Queries

Για τη σύγκριση των δύο δομών ως προς το χρόνο των k-NN queries, πραγματοποιήθηκαν πειράματα με το 0.01% αρχείο των κέντρων, και στα δύο αρχεία δεδομένων. Αρχικά στο T2 dataset, φαίνεται ότι για κάθε αριθμό k, το Quad Tree είναι πιο γρήγορο από τα δύο R-Tree, και σε μεγάλες τιμές k, η ομαδοποίηση z-order είναι καλύτερη από την ομαδοποίηση STR. Επίσης, ακολουθούν γραμμική σχέση σε αυτό το εύρος που μελετήθηκαν τα δέντρα.

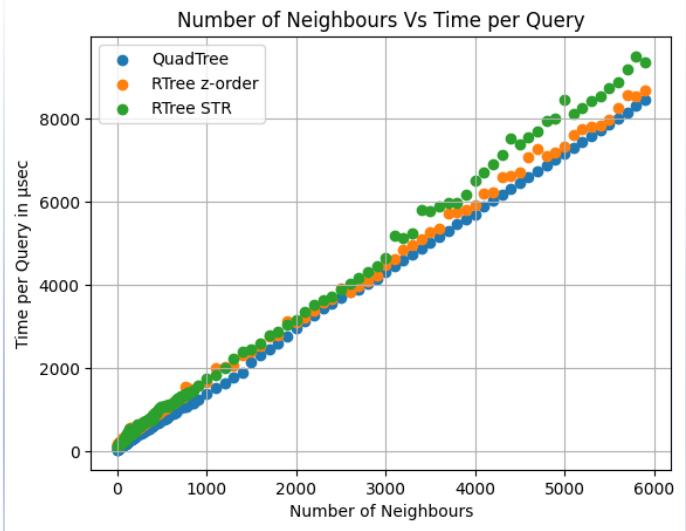

Figure 12: Αριθμός k vs Χρόνος ανά query για το T2 dataset

Από τον παρακάτω πίνακα, φαίνεται το ίδιο αποτέλεσμα του υποκεφαλαίου «Βελτιστοποίηση τιμών Min και Max Entries στο R-Tree», στους πίνακες 2 και 3, όπου σε χαμηλές τιμές k (κοντά στο intercept), η ομαδοποίηση STR ήταν καλύτερη από την ομαδοποίηση z-order. Καθώς όμως οι κλίσεις ακολουθούν αντίθετη σειρά, σε μεγάλα k, αντιστρέφονται οι σειρές ταχύτητας των query των R-Trees.

	Quad Tree	R-Tree (z-order)	R-Tree (STR)
Intercept (μsec/query)	61.878	228.218	181.087
Slope (μsec/query/k)	1.418	1.440	1.572

Table 6: Πίνακας με τις κλίσεις και τα intercept των αποτελεσμάτων κάθε δέντρου στο T2 dataset

Παρόμοια αποτελέσματα λήφθηκαν και κατά την ίδια μελέτη στο T5 Dataset, με την διαφορά ότι το STR inserted R-Tree ήταν καλύτερο και σε χαμηλά k από το αντίστοιχο δέντρο με εισαγωγή των δεδομένων με z-order.

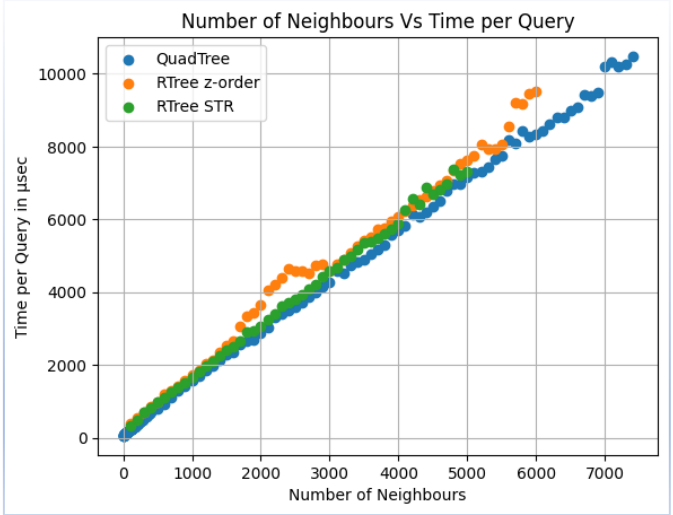

Figure 13: Αριθμός k vs Χρόνος ανά query για το T5 dataset

Για το γραμμικό fit των αποτελεσμάτων R-Tree z-order, εξαιρέθηκαν τα σημεία ~1700-3000 και 5600+, καθώς φαίνεται να είναι outliers και να έχουν προκύψει από τον υπολογιστή.

	Quad Tree	R-Tree (z-order)	R-Tree (STR)
Intercept (μsec/query)	89.71	526.07	181.087
Slope (μsec/query/k)	1.397	1.417	1.448

Table 7: Πίνακας με τις κλίσεις και τα intercept των αποτελεσμάτων κάθε δέντρου στο T2 dataset

Σύγκριση Range Query (T2 dataset, all ranges)

Αναφορικά με τα Range Queries, πραγματοποιήθηκε και μια naïve μεθοδολογία, κατά την οποία, κάθε σημείο του dataset ελέγχεται εάν βρίσκεται στο range query. Έτσι, για κάθε αρχείο με queries (0.01%, 0.05%, 0.1%, 0.5%, 1%), ο χρόνος είναι σταθερός (με τυχαία διακύμανση), καθώς ο αριθμός πράξεων είναι ίδιος (για κάθε query range, ελέγχονται όλα τα σημεία). Σε αντίθεση με την naïve μεθοδολογία, οι υπόλοιπες tree-based μεθοδολογίες, παρουσιάζουν αύξηση του χρόνου εκτέλεσης με την αύξηση του εύρους του query.

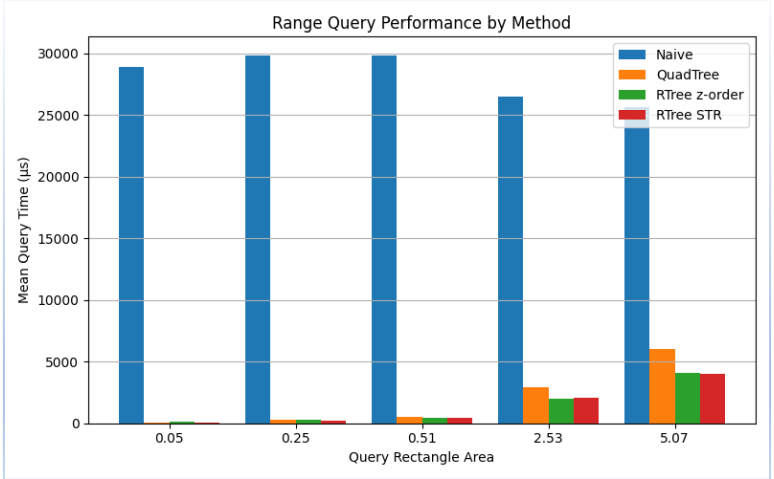

Figure 14: Σύγκριση χρόνων range queries κάθε μεθοδολογίας σε κάθε αρχείο query

Quad Tree vs R-Tree Range Queries

Αναφορικά πλέον με τη διαφορά των δύο δέντρων, λαμβάνεται μια αντίθετη εικόνα από αυτή που λήφθηκε για την ταχύτητα των k-NN queries. Αρχικά στο T2 dataset, φαίνεται ότι το quadtree αργεί περισσότερο με την αύξηση του εύρος του query, σε σχέση με τα δύο R-Trees. Και στη συγκεκριμένη περίπτωση των Range Queries, φαίνεται ότι ακολουθεί γραμμική σχέση σε αυτό το εύρος.

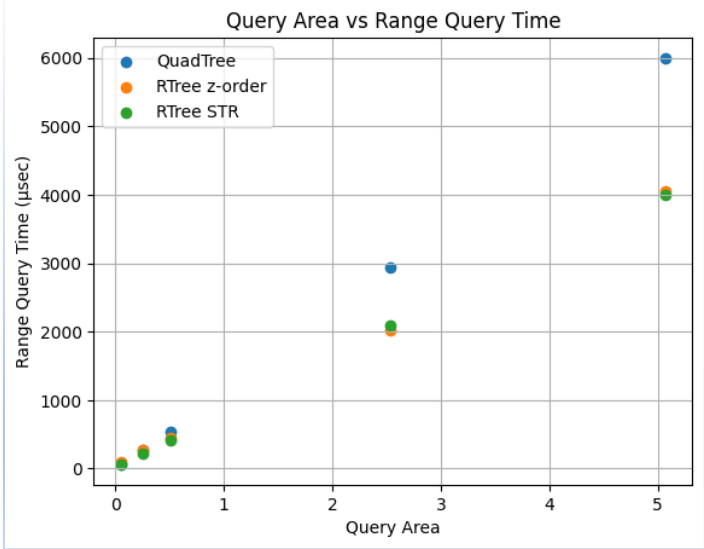

Figure 15: Εύρος query vs Χρόνος ανά query για το T2 dataset

Η κλίση του Quad Tree είναι πολύ μεγαλύτερη από την κλίση των R -Trees, ενώ σε πολύ μικρά εύρη query, φαίνεται να είναι λίγο ταχύτερο από τα δύο R-Trees.

	Quad Tree	R-Tree (z-order)	R-Tree (STR)
Intercept (μsec/query)	-29.62	57.02	32.73
Slope (μsec/query/k)	1184.67	785.89	787.73

Table 8: Πίνακας με τις κλίσεις και τα intercept των αποτελεσμάτων κάθε δέντρου στο T2 dataset

Παρόμοια αποτελέσματα λήφθηκαν και κατά την μελέτη των χρόνων στο T5 dataset.

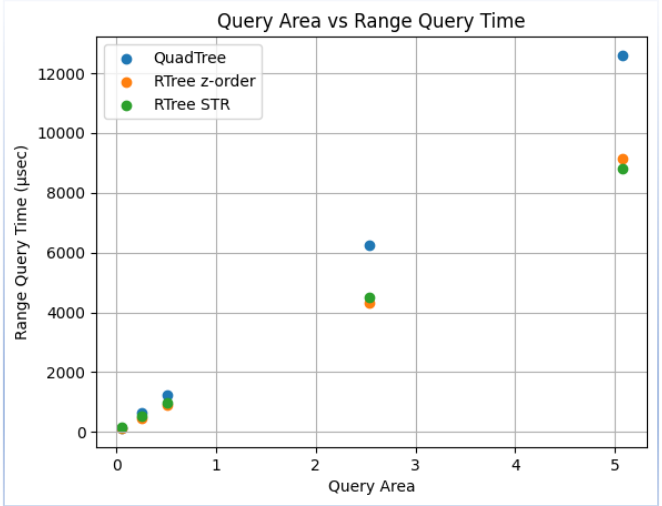

Figure 16: Εύρος query vs Χρόνος ανά query για το T5 dataset

	Quad Tree	R-Tree (z-order)	R-Tree (STR)
Intercept (μsec/query)	-12.57	-16.03	108.02
Slope (μsec/query/k)	2483.09	1787.79	1717.63

Table 9: Πίνακας με τις κλίσεις και τα intercept των αποτελεσμάτων κάθε δέντρου στο T5 dataset

Συμπεράσματα

Η παρούσα εργασία ανέδειξε την αποτελεσματικότητα των χωρικών δομών (Quad Tree και R-Tree) σε σύγκριση με τις naive μεθοδολογίες αναζήτησης. Συγκεκριμένα, επιτεύχθηκαν σημαντικές βελτιώσεις χρόνου εκτέλεσης, με το Quad Tree να είναι έως και 5.39 φορές ταχύτερο από τη baseline προσέγγιση, και το R-Tree με STR insertion να ακολουθεί με 2.41 φορές καλύτερη επίδοση για συγκεκριμένες συνθήκες dataset και αριθμό γειτόνων.

Παρατηρήθηκε ότι οι naïve μέθοδοι έχουν σταθερό χρόνο εκτέλεσης ανεξάρτητα από το μέγεθος του query, ενώ αντίθετα, οι δομές δέντρου εμφανίζουν γραμμική συσχέτιση μεταξύ του μεγέθους του query (σε εμβαδόν) και του χρόνου εκτέλεσης. Σε επίπεδο επιμέρους μεθοδολογιών, τα R-Trees αποδίδουν καλύτερα σε range queries, ιδιαίτερα όσο μεγαλώνει το εμβαδόν του query, ενώ τα Quad Trees υπερέχουν οριακά στα k-NN queries, κυρίως λόγω της απλούστερης και σταθερότερης τοπολογίας τους.

Η προσπάθεια μετασχηματισμού του χώρου μέσω περιστροφών ανέδειξε η διαφορά στους χρόνους ήταν μικρή αλλά στατιστικά σημαντική (p-value ≈ 0.05). Ωστόσο, καταγράφηκαν συστηματικές τάσεις στους χρόνους που συσχετίζονται με στατιστικά χαρακτηριστικά του δέντρου, όπως το min depth (μεγαλύτερες τιμές σχετίζονται με χαμηλότερους χρόνους), το max depth (μεγαλύτερες τιμές οδηγούν σε αύξηση του χρόνου) και το Q3 (υψηλότερες τιμές οδηγούν επίσης σε μικρή αύξηση του χρόνου).

Τέλος, η χρήση μοντέλων πρόβλεψης (Decision Trees) για τον χρόνο εκτέλεσης ενός query, με βάση τα στατιστικά του δέντρου, παρουσίασε χαμηλό R^2 (~0.16), αλλά ανέδειξε μία μικρή βελτίωση στην αβεβαιότητα της πρόβλεψης από την πρόβλεψη απουσία των χαρακτηριστικών αυτών.