Multidimensional Spatial Indexing & LSH for Healthcare Reviews: A Comparative Study on U.S. Hospital Satisfaction

Project
2024-2025

Φίλιππος-Παρασκευάς Ζυγούρης

A.M: 1084660

Περιεχόμενα

1. Εισαγωγή

Ο ψηφιακός κόσμος βασίζεται σε έξυπνες υπολογιστικές στρατηγικές που υπερβαίνουν την αδύνατη, εξαντλητική αναζήτηση ένα, η οποία απαιτεί δυσανάλογους υπολογιστικούς πόρους. Στην παρούσα εργασία αξιολογείται η απόδοση τεσσάρων πολυδιάστατων δομών δεδομένων k-d trees, octrees, range trees και R-trees σε συνδυασμό με την τεχνική Locality-Sensitive Hashing (LSH). Η συγκεκριμένη μεθοδολογία ακολουθεί τις αλγοριθμικές αρχές της εκφώνησης και εφαρμόζεται σε πραγματικά και πλούσια δεδομένα υγείας, τα οποία απαιτούν εκτεταμένη προεπεξεργασία ώστε να υποστηρίζονται πολυπαραγοντικά και ρεαλιστικά σενάρια ανάλυσης. Επιπλέον, η αξιολόγηση επικεντρώνεται σε κρίσιμες μετρικές, όπως ο χρόνος εκτέλεσης, η κατανάλωση μνήμης και η ακρίβεια των αποτελεσμάτων, επιτρέποντας αντικειμενική σύγκριση των μεθόδων ως προς την ευρετηρίαση και την επεξεργασία πολυδιάστατων δεδομένων. Κύριος στόχος είναι να αναδειχθεί ο βέλτιστος συνδυασμός δομής και τεχνικής για αποδοτική διαχείριση και αναζήτηση πληροφοριών, περιορίζοντας την ανάγκη για γραμμική σάρωση και μεγιστοποιώντας την αποτελεσματικότητα της ανάκτησης. Κατευθυντήρια αρχή είναι ότι, στον κόσμο των δεδομένων, η ισχυρότερη δεξιότητα είναι η ικανότητα να γνωρίζουμε τι μπορούμε με ασφάλεια να αγνοήσουμε.

2. Δεδομένα

Α. Σύνολο Δεδομένων

Στην υπάρχουσα εργασία χρησιμοποιήθηκε το σύνολο δεδομένων **U.S. Hospital Customer Satisfaction 2016-2020**, το οποίο περιλαμβάνει αξιολογήσεις και βαθμολογίες ικανοποίησης ασθενών για νοσοκομεία στις Ηνωμένες Πολιτείες κατά την περίοδο 2016-2020. Το συγκεκριμένο σύνολο δεδομένων προέρχεται από το πρόγραμμα Hospital Compare του CMS (Centers for Medicare & Medicaid Services), το οποίο συγκεντρώνει στοιχεία για περισσότερα από 4.000 νοσοκομεία και χαρακτηριστικά που καλύπτουν γεωγραφία (ZIP Code), χρόνο (Year), δείκτες εμπειρίας/ποιότητας (π.χ. HCAHPS Linear Mean Value) και κείμενα αξιολόγησης (HCAHPS Question/Answer Description). Επίσης, το σύνολο αυτό υποστηρίζει πολυδιάστατη ευρετηρίαση και κειμενική ομοιότητα, επιτρέποντας ρεαλιστικά ερωτήματα στον χώρο της υγείας (π.χ. εύρεση των πιο παρόμοιων reviews σε συγκεκριμένη περίοδο/περιοχή με υψηλή ικανοποίηση). Για την υλοποίηση των συγκεκριμένων ερωτημάτων ορίζονται τρεις διαστάσεις ευρετηρίασης

(k=3: ZIP Code, Year, HCAHPS Score) και αξιοποιείται LSH στα κείμενα για ταχύ εντοπισμό ομοίων περιγραφών, ενώ τίθενται φίλτρα περιόδου (2018–2020) και ποιότητας (HCAHPS > 94) με προαιρετικό περιορισμό σε ίδια/όμορη γεωγραφική περιοχή.

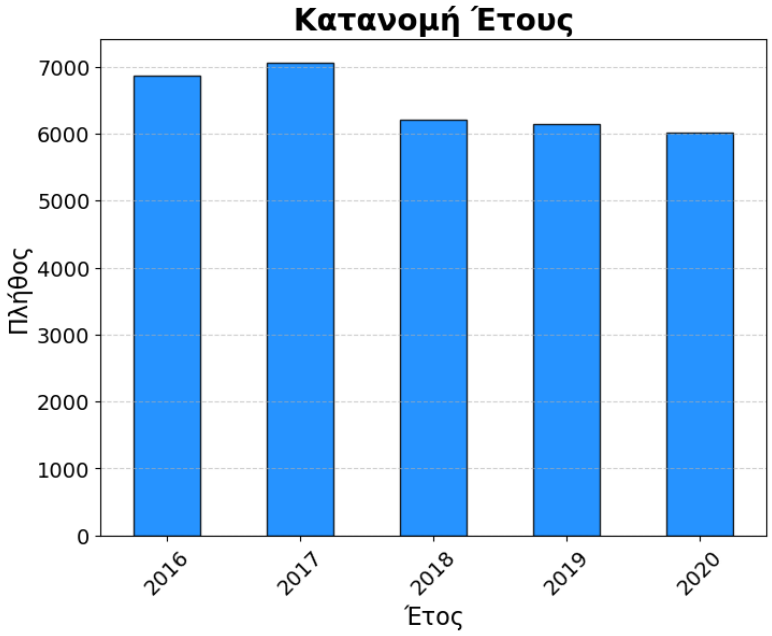
Β. Συγχώνευση Δεδομένων

Για τη συγχώνευση των ετήσιων δεδομένων ικανοποίησης ασθενών των νοσοκομείων για τα έτη 2016 έως 2020, υλοποιήθηκε το script combine_csv.py, το οποίο ενοποιεί όλα τα επιμέρους αρχεία CSV σε ένα ενιαίο, διαχειρίσιμο αρχείο μεγέθους περίπου 197 MB. Πιο αναλυτικά, το script διαβάζει τα διαθέσιμα αρχεία, τα συνενώνει σε ένα DataFrame και αποθηκεύει το τελικό αποτέλεσμα, υπολογίζοντας παράλληλα το μέγεθός του. Σε περίπτωση που το παραγόμενο αρχείο υπερβαίνει τα 199MB, εφαρμόζεται ένας μηχανισμός επαναληπτικής δειγματοληψίας, ο οποίος επιλέγει τυχαία μικρότερα υποσύνολα των δεδομένων μέχρι να επιτευχθεί το επιθυμητό μέγεθος εντός ανοχής ±1MB. Η διαδικασία αυτή επαναλαμβάνεται έως και δέκα φορές, καταγράφοντας σε κάθε βήμα το τρέχον μέγεθος του αρχείου και τον αριθμό των γραμμών, ώστε να διασφαλίζεται η δημιουργία ενός δείγματος που πληροί τις τεχνικές απαιτήσεις μνήμης και απόδοσης του συστήματος. Με αυτόν τον τρόπο εξασφαλίζεται ότι το τελικό αρχείο είναι κατάλληλο τόσο για αξιόπιστη αξιολόγηση της απόδοσης των δομών δεδομένων, όσο και για την απρόσκοπτη υλοποίηση των αλγορίθμων πολυδιάστατης ευρετηρίασης και ερωτημάτων ομοιότητας.

Γ. Κατανόηση & Προετοιμασία Δεδομένων

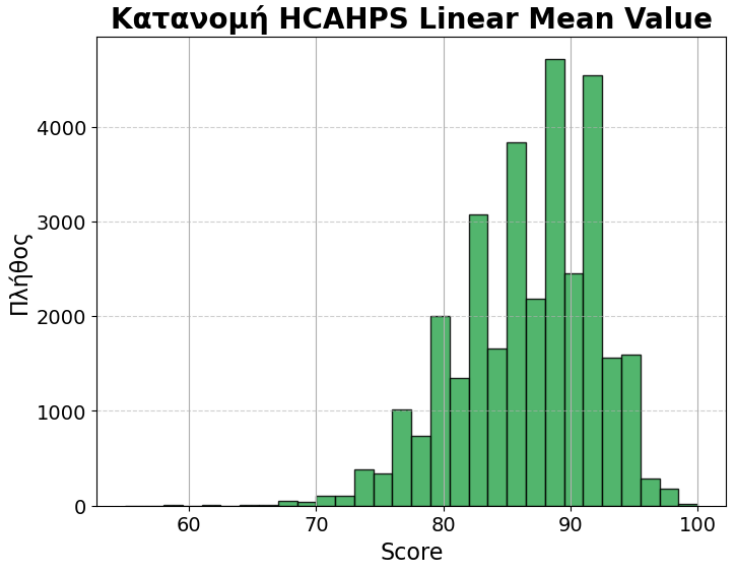
Στα αποτελέσματα της περιγραφικής ανάλυσης το Σχήμα 1 δείχνει ότι οι περισσότερες καταγραφές εντοπίζονται το 2016–2017 και ακολουθεί ήπια μείωση στα έτη 2018–2020. Παρ' όλα αυτά, ο όγκος δεδομένων παραμένει υψηλός σε όλες τις χρονιές, υποστηρίζοντας αξιόπιστες συγκρίσεις. Το Σχήμα 2, από την άλλη, καταδεικνύει συγκέντρωση βαθμολογιών περίπου στο 88–92, με την πλειονότητα των τιμών να βρίσκονται στο εύρος 80–95 και λίγες χαμηλότερες τιμές, που παραπέμπει σε αρνητική ασυμμετρία, γεγονός που επιβεβαιώνει γενικά υψηλά επίπεδα ικανοποίησης με περιορισμένα χαμηλά outliers. Επίσης, το Σχήμα 3 δείχνει ότι τα ενδιάμεσα σκορ διατηρούνται σχεδόν σταθερά (~88–90) σε όλη την περίοδο με μικρές διακυμάνσεις, ενώ οι ακραίες χαμηλές τιμές είναι μεμονωμένες και δεν μεταβάλλουν τη συνολική εικόνα ποιότητας. Επιπλέον, οι συχνότερες κατηγορίες ερωτήσεων/απαντήσεων αφορούν κρίσιμες πτυχές της εμπειρίας φροντίδας (επικοινωνία ιατρών/νοσηλευτών, ανταπόκριση προσωπικού, καθαριότητα, σύσταση νοσοκομείου), επιτρέποντας στο LSH να εκμεταλλευτεί θεματικές ομοιότητες για αναζήτηση παρόμοιων σχολίων. Συνολικά, τα ευρήματα

τεκμηριώνουν σταθερά υψηλή ικανοποίηση των ασθενών σε βάθος χρόνου, επαρκή στατιστική κάλυψη ανά έτος και κατάλληλο υπόβαθρο για πιο σύνθετα και πολυκριτηριακά ερωτήματα.



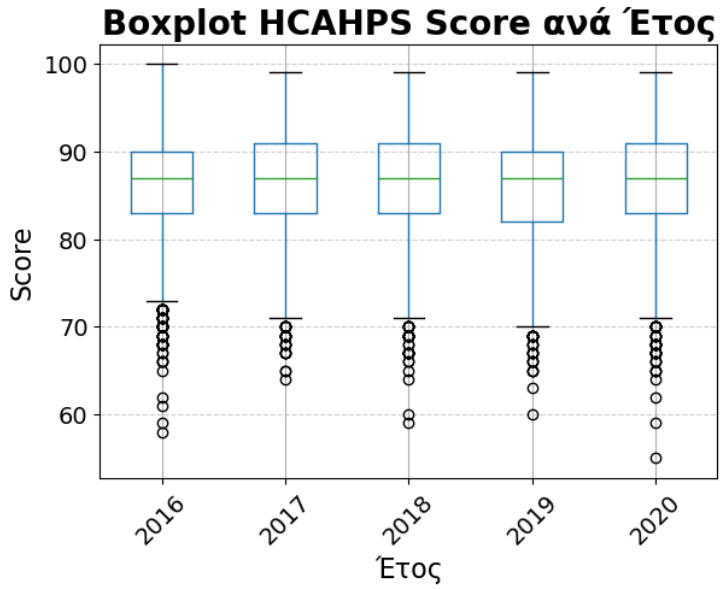
Σχήμα 1: Κατανομή εγγραφών ανά έτος (2016–2020)

Η προετοιμασία των δεδομένων ενοποιεί χωρικά και κειμενικά γνωρίσματα ώστε να υποστηριχθούν πολυδιάστατη ευρετηρίαση και LSH. Ο δείκτης HCAHPS Linear Mean Value μετατρέπεται σε αριθμητικό μέσω της βοηθητικής συνάρτησης safe_float (μη έγκυρες τιμές → NaN), ενώ τα ZIP Code και Year μετατρέπονται με pd.to_numeric(errors=»coerce»). Εφαρμόζεται φίλτρο ποιότητας/χρόνου για το υποσύνολο 2018–2020 και για τιμές HCAHPS < 98, και δημιουργείται ενιαίο πεδίο κειμένου feedback_text με συνένωση των HCAHPS Question και HCAHPS Answer Description. Στη συνέχεια απορρίπτονται εγγραφές με ελλείψεις στα βασικά αριθμητικά πεδία και εξάγεται ο πίνακας χαρακτηριστικών features σχήματος (n, 3) \— (ZIP, Year, Score)—μαζί με τη λίστα κειμένων feedbacks και τους αρχικούς δείκτες γραμμών indices, ώστε να διατηρείται πλήρης ιχνηλασιμότητα.



Σχήμα 2: Κατανομή του δείκτη HCAHPS Linear Mean Value

Για το κειμενικό σκέλος, η κλάση PerformanceEvaluator πραγματοποιεί εκ των προτέρων υπολογισμούς MinHash/LSH μέσω της _precompute_text_structures(), όπου κάθε κείμενο κανονικοποιείται και γίνεται tokenization σε character 3-grams (προεπιλογή n=3), παράγοντας λίστες tokens και σύνολα tokens (χρήσιμα για ακριβή Jaccard). Έπειτα, δημιουργείται ένα αντικείμενο MinHash ανά έγγραφο (num_perm=128) και αποθηκεύεται στα doc_minhash, μαζί με doc_tokens/doc_token_sets. Στο χωρικό σκέλος, υλοποιείται σταθερή αντιστοίχιση τρισδιάστατων σημείων σε αναγνωριστικά εγγράφων, ώστε οι τριπλέτες (ZIP, Year, Score) να ευθυγραμμίζονται με τις συντεταγμένες που επιστρέφουν οι χωρικοί δείκτες (KD-/Octree/Range/R-Tree). Τέλος, το αποτέλεσμα είναι καθαρά αριθμητικά χαρακτηριστικά, συνεπή MinHash embeddings και αξιόπιστη χαρτογράφηση εγγραφών–σημείων, προσφέροντας στιβαρή βάση για πολυδιάστατη ευρετηρίαση και ερωτήματα ομοιότητας.



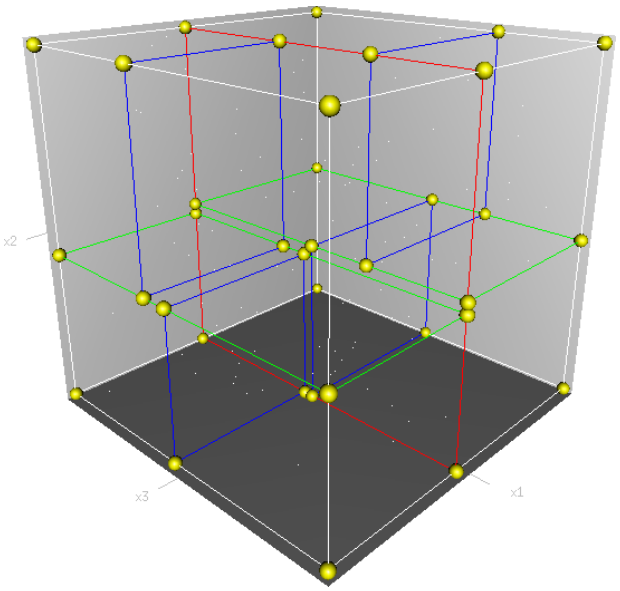
Σχήμα 3: Διακύμανση HCAHPS Score ανά έτος (Boxplot).

3. Υλοποίηση

Η υλοποίηση πραγματοποιήθηκε σε Jupyter Notebook. Για αναπαραγωγή των αποτελεσμάτων, εκτελέστε αρχικά το script combine_csv.py για τη συγχώνευση των δεδομένων και στη συνέχεια προχωρήστε στην ανάλυση μέσα από το Jupyter Notebook.

A. K-D Tree

Το KD-Tree είναι μια δομή δεδομένων για οργάνωση σημείων σε χώρο k διαστάσεων, παρόμοια με δυαδικό δέντρο αναζήτησης αλλά με κριτήριο σύγκρισης που εναλλάσσεται κυκλικά ανά επίπεδο: στο βάθος 0 συγκρίνουμε π.χ. τη συντεταγμένη x (δείκτης 0), στο βάθος 1 τη y (δείκτης 1), στο βάθος 2 τη z (δείκτης 2) και ξανά από την αρχή. Κάθε κόμβος αποθηκεύει ένα σημείο και ορίζει ένα υπερεπίπεδο παράλληλο στους άξονες που χωρίζει τον χώρο σε δύο υποπεριοχές. Έτσι, η αναδρομική διαίρεση δημιουργεί ολοένα μικρότερες περιοχές που ομαδοποιούν κοντινά σημεία. Σε πιο ισορροπημένα δέντρα ο μέσος χρόνος αναζήτησης είναι λογαριθμικός ως προς το πλήθος των σημείων, αν και στη χειρότερη περίπτωση μπορεί να εκφυλιστεί, ενώ για τις πολύ υψηλές διαστάσεις η απόδοση μειώνεται λόγω της κατάρας της διαστατικότητας. Χρησιμοποιείται ευρέως σε ρομποτική, 3D γραφικά, γεωχωρικές βάσεις και συστήματα αναζήτησης ομοιότητας.



1.1 Κλάση KDTreeNode

Η κλάση KDTreeNode αναπαριστά έναν κόμβο σε KD-Tree. Κάθε κόμβος αποθηκεύει το σημείο του χώρου (point) ως πλειάδα συντεταγμένων, τον άξονα διαχωρισμού (axis) που καθορίζει σε ποια διάσταση γίνεται ο διαχωρισμός στο συγκεκριμένο επίπεδο, καθώς και δείκτες στα θυγατρικά υποδέντρα (left, right) και αν δεν υπάρχουν τιμές, προεπιλεγμένα ορίζονται ως None, υποδηλώνοντας ότι ο κόμβος είναι φύλλο ή δεν έχει ακόμη υποδενδροποιηθεί. Επιπλέον, το πεδίο depth κρατά το βάθος του κόμβου στο δέντρο και χρησιμοποιείται για τον υπολογισμό του άξονα διαχωρισμού με κυκλική εναλλαγή διαστάσεων (axis = depth % k). Με αυτή τη δομή, κάθε κόμβος κόβει σωστά τον χώρο κατά μήκος του axis, επιτρέποντας στο k-d tree να οργανώνει αποδοτικά πολυδιάστατα δεδομένα και να υποστηρίζει γρήγορες αναζητήσεις, όπως πλησιέστερου γείτονα και ερωτήματα περιοχής σε οποιοδήποτε πλήθος διαστάσεων.

1.2. Κλάση KDTree

1.2.1 Κατασκευή του Δέντρου

Η κλάση KDTree ξεκινάει αποθηκεύοντας το πλήθος των διαστάσεων k από το πρώτο σημείο της λίστας, εφόσον υπάρχουν σημεία. Στην περίπτωση που δεν δοθούν σημεία, το k τίθεται σε 0 και η ρίζα παραμένει None. Όταν δοθεί η λίστα σημείων, καλείται αμέσως η μέθοδος build(points), η οποία αναλαμβάνει την αναδρομική κατασκευή του δέντρου. Η build υλοποιεί την κλασική αναδρομική διαδικασία κατασκευής KD-Tree. Σε κάθε επίπεδο, επιλέγεται ένας άξονας διαχωρισμού κυκλικά με βάση το βάθος. Τα σημεία ταξινομούνται ως προς αυτή τη διάσταση και επιλέγεται το διάμεσο στοιχείο (median) ως ρίζα του αντίστοιχου υποδέντρου. Έπειτα, η build καλείται ξανά για τα στοιχεία πριν από το median (αριστερό υποδέντρο) και για τα στοιχεία μετά το median (δεξί υποδέντρο), αυξάνοντας το βάθος κατά 1, ενώ η διαδικασία

επαναλαμβάνεται για κάθε υποσύνολο. Έτσι δημιουργείται ένας κόμβος KDTreeNode που περιέχει το σημείο, τον άξονα και τους δείκτες στα δύο υποδέντρα, ενώ γεωμετρικά ορίζει ένα υπερεπίπεδο κάθετο στον άξονά του που χωρίζει τον χώρο στα δύο. Η διαδοχική αυτή διάσπαση του χώρου σε ολοένα και μικρότερες περιοχές οδηγεί σε ισορροπημένη δομή και επιταχύνει τις αναζητήσεις.

1.2.2 Εισαγωγή Κόμβου

Η μέθοδος insert_point υλοποιεί την αναδρομική εισαγωγή ενός νέου σημείου σε ένα K-D Tree, ξεκινώντας από έναν δεδομένο κόμβο (ή None αν το δέντρο είναι άδειο) και αν δεν υπάρχει κόμβος δημιουργεί νέο KDTreeNode με άξονα axis = depth % k. Διαφορετικά, χρησιμοποιεί τον αποθηκευμένο άξονα διαχωρισμού του τρέχοντος κόμβου (node.axis) και συγκρίνει μόνο αυτή τη διάσταση. Η εισαγωγή συνεχίζεται αριστερά αν η τιμή του νέου σημείου είναι μικρότερη ή δεξιά διαφορετικά, αυξάνοντας το βάθος κατά 1. Η μέθοδος insert απλώς εκκινεί τη διαδικασία από τη ρίζα ενημερώνοντας το δέντρο. Η λογική μοιάζει με δυαδικό δέντρο αναζήτησης, αλλά το κριτήριο σύγκρισης εναλλάσσεται κυκλικά ανά διάσταση, χωρίζοντας σταδιακά τον χώρο σε υποπεριοχές.

1.2.3 Διαγραφή Κόμβου

Η διαγραφή ενός κόμβου σε ένα K-D Tree πρέπει να διατηρεί την ιδιότητα διαχωρισμού, ότι αριστερά είναι τα μικρότερα και δεξιά τα μεγαλύτερα ως προς τον συγκεκριμένο άξονα. Η βοηθητική find_min(node, d) επιστρέφει τον κόμβο με τη μικρότερη τιμή στη συγκεκριμένη διάσταση d. Εάν ο τρέχων κόμβος διαχωρίζει ως προς τη διάσταση d, το ελάχιστο βρίσκεται στον ίδιο τον κόμβο ή στο αριστερό του υποδέντρο, οπότε η αναζήτηση περιορίζεται εκεί. Αν ο άξονας διαχωρισμού είναι διαφορετικός, το ελάχιστο μπορεί να βρίσκεται είτε στον ίδιο τον κόμβο είτε σε οποιοδήποτε από τα δύο υποδέντρα, οπότε εξετάζονται όλα και επιλέγεται το μικρότερο.

Η μέθοδος delete_point διαγράφει ένα σημείο ξεκινώντας από τη ρίζα, εντοπίζοντας πρώτα τον κόμβο-στόχο με σύγκριση μόνο στην τρέχουσα διάσταση (node.axis) όπως σε δυαδικό δέντρο αναζήτησης. Αν ο κόμβος είναι φύλλο επιστρέφεται None, αν έχει δεξί παιδί το σημείο του αντικαθίσταται με το ελάχιστο ως προς τον άξονά του από το δεξί υποδέντρο (find_min) και στη συνέχεια το ελάχιστο διαγράφεται από εκεί, ενώ αν δεν υπάρχει δεξί αλλά υπάρχει αριστερό παιδί, το σημείο αντικαθίσταται με το ελάχιστο του αριστερού υποδέντρου, όπου αυτό διαγράφεται από το αριστερό και το υπόλοιπο αριστερό μεταφέρεται ως δεξί (με το αριστερό να αδειάζει). Όταν το προς διαγραφή σημείο δεν ταυτίζεται με τον τρέχοντα κόμβο, η αναζήτηση συνεχίζεται αριστερά ή δεξιά ανάλογα με τη σύγκριση στη διάσταση axis. Η

delete(point) απλώς εκκινεί τη διαδικασία από τη ρίζα. Η στρατηγική αντικατάστασης μέσω find_min και τοπικής αναδόμησης διατηρεί την ιδιότητα του KD-Tree, ότι δηλαδή, ως προς τον άξονα κάθε κόμβου, τα αριστερά σημεία είναι μικρότερα και τα δεξιά μεγαλύτερα, εξασφαλίζοντας έτσι ορθή γεωμετρία και αποδοτικές αναζητήσεις.

1.2.4 Ενημέρωση Κόμβου

Η μέθοδος update σε ένα K-D Tree υλοποιεί τη δυναμική αντικατάσταση ενός υπάρχοντος σημείου με ένα νέο, συνδυάζοντας τις διαδικασίες διαγραφής και εισαγωγής. Αρχικά διαγράφει το old_point διατηρώντας τη δομή και την ιεραρχία του δέντρου, και στη συνέχεια εισάγει το new_point στην κατάλληλη θέση βάσει της κυκλικής εναλλαγής των αξόνων. Αυτή η προσέγγιση επιτρέπει την ενημέρωση των δεδομένων χωρίς να απαιτείται ολική ανακατασκευή της δομής.

1.2.5 Αναζήτηση k-Πλησιέστερων Γειτόνων

Η μέθοδος distance_sq υπολογίζει την τετραγωνική Ευκλείδεια απόσταση μεταξύ δύο σημείων χωρίς τετραγωνική ρίζα, ώστε οι συγκρίσεις των αποστάσεων να γίνονται γρηγορότερα. Η kNN(target, k) εντοπίζει τους k πλησιέστερους γείτονες με αναδρομική αναζήτηση στο KD-Tree. Σε κάθε βήμα, υπολογίζεται η τετραγωνική απόσταση του τρέχοντος σημείου από το στόχο και χρησιμοποιείται ένα max-heap για να διατηρούνται τα k καλύτερα αποτελέσματα μέχρι στιγμής, ταξινομημένα ως προς τη μεγαλύτερη απόσταση. Στη συνέχεια, επιλέγεται κοντινό και μακρινό υποδέντρο με βάση τη διαφορά στη διάσταση διαχωρισμού (diff = target[axis] - point[axis]) και γίνεται πρώτα αναζήτηση στο κοντινό, δηλαδή στο υποδέντρο που περιέχει το σημείο-στόχο. Καθώς επιστρέφουμε (backtracking), εξετάζουμε το μακρινό υποδέντρο μόνο αν ο heap δεν έχει γεμίσει ή αν το τετράγωνο της απόστασης από το υπερεπίπεδο ($diff^2$) είναι μικρότερο από τη χειρότερη απόσταση που κρατά ο heap, οπότε υπάρχει πιθανότητα για καλύτερο γείτονα. Τέλος, επιστρέφονται τα k σημεία ταξινομημένα κατά απόσταση. Επομένως, με αυτό το κλάδεμα το πλήθος των συγκρίσεων μειώνεται σημαντικά σε σχέση με την αφελή γραμμική αναζήτηση.

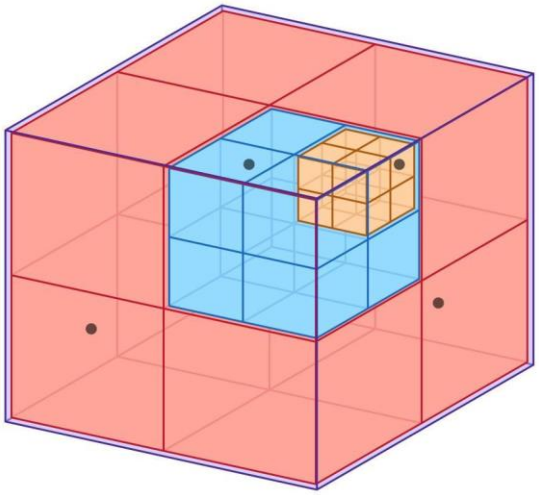
1.2.6 Ερώτημα Εύρους

Η range_query επιστρέφει όλα τα σημεία που ανήκουν σε ένα ορθογώνιο ερώτημα σε πολυδιάστατο χώρο, αξιοποιώντας τη δομή του δέντρου και μειώνοντας σημαντικά τον αριθμό των σημείων που χρειάζεται να ελεγχθούν συγκριτικά με μία αφελή προσέγγιση. Σε κάθε κόμβο ελέγχεται αν το σημείο του βρίσκεται εντός των δοσμένων ορίων (is_point_in_range) κάθε διάστασης. Παράλληλα, διατηρείται για κάθε υποδέντρο μια περιοχή (region)

που αν αυτή περιέχεται πλήρως στο ερώτημα (is_region_contained), τότε προστίθενται μαζικά όλα τα σημεία του υποδέντρου (add_subtree) χωρίς περαιτέρω ελέγχους, αλλιώς, αν υπάρχει έστω και μερική τομή (regions_intersect), η αναζήτηση συνεχίζεται μόνο στα σχετικά κλαδιά. Η αρχική περιοχή είναι άπειρη σε όλες τις διαστάσεις και σφίγγει σε κάθε βήμα με βάση τον άξονα διαχωρισμού του κόμβου. Με αυτό τον τρόπο απορρίπτονται ολόκληρες υποπεριοχές που δεν τέμνουν το ερώτημα.

B. Oct Tree

Η έννοια του οκταδένδρου (octree) βασίζεται στη χωρική διαμέριση και αποτελεί τρισδιάστατη επέκταση του τετραδικού δέντρου (quadtree). Στο quadtree, μια δισδιάστατη περιοχή υποδιαιρείται επαναληπτικά σε τέσσερα ίσα τμήματα έως ότου ο αριθμός των σημείων ανά υποπεριοχή υποχωρήσει κάτω από προκαθορισμένο όριο στο octree η ίδια λογική εφαρμόζεται σε τρισδιάστατο χώρο, ξεκινώντας από έναν κύβο που διαιρείται σε οκτώ ίσους υποκύβους (οκτάντες) και υποδιαιρείται αναδρομικά μόνο όπου απαιτείται. Τα δεδομένα αποθηκεύονται τυπικά στους κόμβους-φύλλα, ενώ οι εσωτερικοί κόμβοι λειτουργούν ως συνδετικοί κόμβοι της ιεραρχίας, επιτρέποντας μια αραιή (sparse) δομή που κατασκευάζεται δυναμικά κατά την εισαγωγή σημείων. Η διαδικασία εκκινεί από τον κυβικό χώρο της ρίζας και τερματίζει όταν ικανοποιούνται κριτήρια όπως μέγιστος αριθμός σημείων ανά φύλλο, ελάχιστο επιτρεπτό μέγεθος κελιού ή μέγιστο βάθος δέντρου.



2.1 Κλάση OctreeNode

Η κλάση OctreeNode αναπαριστά έναν κυβικό όγκο (AABB) στον 3D χώρο με κέντρο (center=[x,y,z]) και μήκος ακμής (size). Κάθε κόμβος διαθέτει λίστα children οκτώ θέσεων (μία ανά οκτάντη), που αρχικοποιούνται ως None, και ακολουθεί το μοντέλο «δεδομένα μόνο στα φύλλα». Όταν is_leaf=True ο κόμβος δεν έχει παιδιά και αποθηκεύει τα σημεία της περιοχής του στη λίστα data (ως λεξικά με κλειδί coords=[x,y,z]), ενώ όταν χρειαστεί να υποδιαιρεθεί,

παύει να είναι φύλλο (is_leaf=False) και τα σημεία του μετακινούνται στα κατάλληλα παιδιά. Επίσης, δεν υπάρχει ξεχωριστή κλάση «κενού» κόμβου, καθώς ένα φύλλο με κενό data λειτουργεί ως κενό και οι εσωτερικοί κόμβοι δημιουργούνται μόνο όπου απαιτείται από την παρουσία δεδομένων, με αποτέλεσμα να δημιουργείται μια αραιή (sparse) δομή που αποφεύγει περιττή δέσμευση μνήμης. Σε αντίθεση με τα παραπάνω, ένα πλήρες οκτάδενδρο θα είχε και τα οκτώ παιδιά παρόντα σε κάθε εσωτερικό κόμβο.

2.2. Κλάση Octree

2.2.1 Αρχικοποίηση

Κατά την αρχικοποίηση, δημιουργείται ο ριζικός κόμβος OctreeNode(root_center, root_size) που οριοθετεί τον αρχικό πεπερασμένο 3D χώρο (AABB) με κέντρο root_center και μήκος ακμής root_size. Αναλυτικότερα, ορίζονται τρεις παράμετροι ελέγχου της αναδρομικής υποδιαίρεσης, το max_points_per_node (ανώτατος αριθμός σημείων ανά φύλλο πριν από υποδιαίρεση σε 8 οκτάντες), το min_size (ελάχιστο επιτρεπτό μήκος ακμής για αποφυγή υπερβολικών/άπειρων υποδιαιρέσεων) και το max_depth (ανώτατο βάθος δέντρου), ενώ η τιμή self.dim = 3 δηλώνει ρητά ότι η υλοποίηση λειτουργεί σε τρισδιάστατο χώρο. Οι παράμετροι αυτές λειτουργούν ως «φρένα» στην ανάπτυξη του δέντρου, ώστε η υποδιαίρεση να σταματά είτε όταν τα σημεία είναι λίγα για τον κόμβο, είτε όταν ο κύβος έχει μικρύνει αρκετά, είτε όταν έχει επιτευχθεί το μέγιστο επιτρεπτό βάθος, διατηρώντας έτσι μια ισορροπία ανάμεσα σε κόστος μνήμης και ταχύτητα αναζήτησης.

Είναι σημαντικό να σημειωθεί το γεγονός ότι η ιεραρχία του octree δεν προκατασκευάζεται, αλλά ξεκινά μόνο από τη ρίζα και, κατά την εισαγωγή σημείων (insert), ένας κόμβος-φύλλο υποδιαιρείται σε οκτώ οκτάντες μόνο όταν απαιτείται (π.χ. αν ξεπεραστεί το max_points_per_node, εκτός αν όλα τα σημεία ανήκουν στον ίδιο οκτάντη). Έτσι, δημιουργούνται μόνο τα αναγκαία παιδιά ακριβώς όπου υπάρχει δεδομένο, επιτυγχάνοντας οικονομία μνήμης, μικρότερο κόστος κατασκευής και καλύτερη προσαρμογή στην πραγματική κατανομή των σημείων.

2.2.2 Εισαγωγή και Κατασκευή

Η μέθοδος insert δεν περιορίζεται στην απλή εισαγωγή του σημείου, αλλά ταυτόχρονα οικοδομεί δυναμικά το octree ως αναδρομική διαδικασία που ξεκινά από τη ρίζα και κατεβαίνει μέχρι την κατάλληλη θέση. Αν δεν δοθεί κόμβος, χρησιμοποιείται η ρίζα και λαμβάνονται οι συντεταγμένες coords από το point για να γίνει έλεγχος ότι έχουν τη σωστή διάσταση (3D) και ότι

βρίσκονται εντός ορίων του τρέχοντος κόμβου με την _in_bounds (χρήση half=size/2 και μικρού eps για αριθμητική σταθερότητα). Αν ο κόμβος είναι φύλλο, ελέγχονται οι συνθήκες τερματισμού: (α) δεν έχει ξεπεραστεί το max_points_per_node, (β) ο κύβος είναι πολύ μικρός (size <= min_size), ή (γ) έχει επιτευχθεί το μέγιστο βάθος (depth >= max_depth). Σε μία από αυτές, το σημείο προστίθεται στη data και η διαδικασία τερματίζει.

Διαφορετικά, ελέγχει αν όλα τα (παλαιά + νέο) σημεία ανήκουν στον ίδιο οκτάντη με _octant, οπότε αποφεύγεται το split, αλλιώς ο κόμβος μετατρέπεται σε εσωτερικό, αδειάζει η data και όλα τα σημεία επαναεισάγονται στα κατάλληλα παιδιά μέσω _insert_child. Η _insert_child υπολογίζει τον οκτάντη (bitmasking), δημιουργεί παιδί όπου λείπει με νέο κέντρο center[i] + (((octant >> i) & 1)*half - half/2) και μισό μέγεθος (size=half), και συνεχίζει αναδρομικά αυξάνοντας το depth. Με αυτόν τον τρόπο εξασφαλίζεται γεωμετρικά ορθή, αποδοτική και προσαρμοστική κατασκευή: δημιουργούνται μόνο τα αναγκαία παιδιά, ακριβώς όπου υπάρχουν δεδομένα.

2.2.3 Διαγραφή

Η μέθοδος delete_by_coords(coords) υλοποιεί αναδρομική διαγραφή με κλάδεμα. Αρχικά, ξεκινάει από τη ρίζα και απορρίπτει αμέσως κλαδιά όπου το ζητούμενο σημείο βρίσκεται εκτός ορίων του AABB του κόμβου (_in_bounds). Εάν ο κόμβος είναι φύλλο, πραγματοποιείται γραμμική αναζήτηση στη data και, σε ακριβές ταίριασμα συντεταγμένων, το σημείο αφαιρείται (pop) και καταγράφεται επιτυχία. Εάν ο κόμβος είναι εσωτερικός, υπολογίζεται ο οκτάντης του στόχου με _octant και η αναζήτηση συνεχίζει μόνο στο αντίστοιχο παιδί. Σε περίπτωση που το παιδί απουσιάζει, ενημερώνει ότι το σημείο δεν υφίσταται. Μετά από επιτυχή διαγραφή σε υποδέντρο, εφαρμόζεται συγχώνευση (collapse), δηλαδή αν όλα τα υπάρχοντα παιδιά είναι φύλλα και το συνολικό πλήθος των σημείων τους δεν υπερβαίνει το max_points_per_node, ο γονέας μετατρέπεται σε φύλλο (is_leaf=True), συγκεντρώνει τα σημεία τους στη node.data και μηδενίζει τον πίνακα παιδιών, διατηρώντας έτσι ένα αραιό octree χωρίς περιττή πολυπλοκότητα όταν οι υποκλάδοι αδειάζουν. Τέλος, η μέθοδος επιστρέφει True/False ανάλογα με το αν ολοκληρώθηκε η διαγραφή.

2.2.4 Ενημέρωση

Η μέθοδος update_point(old_coords, new_point) αρχικά επιχειρεί τη διαγραφή του παλαιού σημείου με ακριβές ταίριασμα συντεταγμένων μέσω της delete_by_coords(old_coords) και, εφόσον αυτή επιτύχει, εισάγει το νέο σημείο μέσω της insert(new_point), ώστε να τοποθετηθεί στον ορθό οκτάντη βάσει της νέας θέσης και να διατηρηθεί το octree συνεπές χωρίς πλήρη

ανακατασκευή. Η διαδικασία επηρεάζει μόνο τους γειτονικούς κόμβους: το σημείο αφαιρείται από το παλιό φύλλο και εισάγεται στο νέο, με αποτέλεσμα να αλλάζει μόνο το μονοπάτι από τη ρίζα στο αντίστοιχο φύλλο. Αν σε κάποιο φύλλο ξεπεραστεί το όριο σημείων, εκτελείται υποδιαίρεση (split), ενώ αν τα παιδιά ενός εσωτερικού κόμβου συγκεντρώνουν πλέον λίγα σημεία συνολικά, εκτελείται συγχώνευση (collapse). Έτσι, οι ενημερώσεις παραμένουν τοπικές, ακολουθούν τους κανόνες split/collapse χωρίς να επηρεάζουν άσχετα τμήματα του δέντρου και, σε ισορροπημένες περιπτώσεις, επιτυγχάνουν κατά μέσο όρο πολυπλοκότητα περίπου O(logN).

2.2.5 Αναζήτηση k-Πλησιέστερων Γειτόνων

Η μέθοδος kNN(target, k) υλοποιεί αναζήτηση των k πλησιέστερων γειτόνων με χρήση σωρού (max-heap) όπου αποθηκεύονται προσωρινά οι καλύτεροι υποψήφιοι με αρνητικές αποστάσεις για σταθερότητα. Η απόσταση μεταξύ σημείων υπολογίζεται με τετραγωνική ευκλείδεια (χωρίς τετραγωνική ρίζα για βελτίωση απόδοσης), ενώ η συνάρτηση bbox_min_sqdist υπολογίζει το ελάχιστο δυνατό κατώτερο φράγμα της απόστασης μεταξύ του στόχου και του κυβικού όγκου (AABB) που καλύπτει κάθε κόμβος. Αν αυτό το κατώτερο φράγμα είναι μεγαλύτερο από την τρέχουσα χειρότερη απόσταση στο heap, τότε το αντίστοιχο υποδέντρο κλαδεύεται, καθώς δεν μπορεί να περιέχει πιο κοντινό σημείο. Επιπλέον, στους εσωτερικούς κόμβους, τα παιδιά ταξινομούνται με βάση το bbox_min_sqdist και επισκέπτονται πρώτα τα υποδέντρα με το μικρότερο θεωρητικό φράγμα, υλοποιώντας στην πράξη μια στρατηγική best-first που μεγιστοποιεί το pruning. Παράλληλα,στους κόμβους-φύλλα, η μέθοδος ελέγχει κάθε σημείο, υπολογίζει την πραγματική απόσταση και ενημερώνει τον σωρό με heappush ή heappushpop ώστε να διατηρούνται πάντα μόνο τα k καλύτερα σημεία. Στο τέλος, τα αποτελέσματα ταξινομούνται από το πιο κοντινό στο πιο μακρινό και επιστρέφονται ως λίστα dictionaries με κλειδί coords, διευκολύνοντας την επέκταση με πρόσθετη πληροφορία.

Η διαδικασία αυτή μειώνει σημαντικά τον αριθμό των άσκοπων ελέγχων, καθώς αποφεύγεται η πλήρης διερεύνηση του δέντρου και εξετάζονται μόνο οι περιοχές που είναι αποδεδειγμένα σχετικές με τον στόχο. Το pruning που είναι βασισμένο σε κατώτερα φράγματα και η χρήση τετραγωνικών αποστάσεων καθιστούν τη μέθοδο πιο αποδοτική, μετατρέποντας την πολυπλοκότητα από O(n), που απαιτεί πλήρη γραμμικό έλεγχο όλων των σημείων, σε περίπου O(logN+ klogk) σε καλά ισορροπημένα δέντρα και για τρισδιάστατο χώρο, επιτυγχάνοντας έτσι σημαντική βελτίωση απόδοσης.

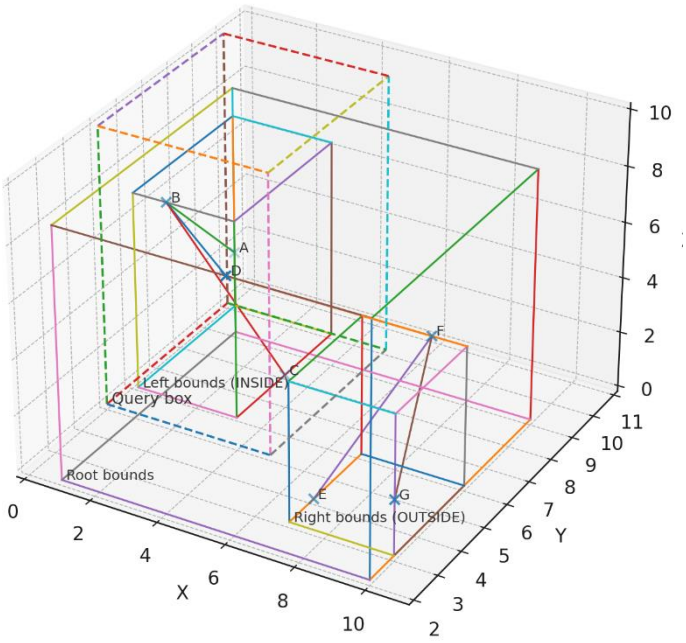
2.2.6 Ερώτημα Εύρους

Η συνάρτηση _intersects(node, mn, mx) υλοποιεί τον τυπικό έλεγχο τομής δύο AABB με inclusive όρια. Για κόμβο με κέντρο node.center και μήκος ακμής node.size, υπολογίζεται το μισό μήκος half = size/2 και εξετάζεται, σε κάθε άξονα x,y,z, αν υπάρχει διαχωρισμός: αν ισχύει είτε center[i] + half < mn[i] είτε center[i] - half > mx[i], δεν υπάρχει επικάλυψη στον συγκεκριμένο άξονα και συνεπώς δεν υπάρχει τομή συνολικά διαφορετικά, θεωρείται ότι υπάρχει τομή (περιλαμβανομένης της οριακής επαφής).

Η μέθοδος range_query(mn, mx, node=None, out=None) πραγματοποιεί αναζήτηση σημείων εντός παραλληλεπιπέδου ορισμένου από τα διανύσματα ορίων mn=[xmin,ymin,zmin] και mx=[xmax,ymax,zmax]. Αρχικά, ξεκινάει από τη ρίζα όταν δεν δοθεί κόμβος, αρχικοποιεί δοχείο αποτελεσμάτων αν απαιτείται και εφαρμόζει pruning: αν το AABB του κόμβου δεν τέμνεται με το [mn,mx] (σύμφωνα με το _intersects), το αντίστοιχο υποδέντρο απορρίπτεται άμεσα. Όταν υπάρχει τομή και ο κόμβος είναι φύλλο, ελέγχονται γραμμικά τα σημεία του και προστίθενται στο αποτέλεσμα όσα ικανοποιούν mn[i]≤coords[i]≤mx[i]. Επίσης, για εσωτερικούς κόμβους, η αναζήτηση συνεχίζεται αναδρομικά μόνο στα μη κενά παιδιά. Ο συνδυασμός ιεραρχικού κλαδέματος (AABB–AABB) και τοπικών γραμμικών ελέγχων αποφεύγει άσκοπες διερευνήσεις κλαδιών που δεν μπορούν να περιέχουν λύσεις, βελτιώνοντας σημαντικά την απόδοση έναντι του αφελούς ελέγχου όλων των σημείων.

Γ. Range Tree

Το Range Tree είναι μια πολυεπίπεδη δομή δεδομένων της υπολογιστικής γεωμετρίας για ορθογώνια ερωτήματα εύρους σε k διαστάσεις. Σε πιο πρακτικούς όρους, κάθε φίλτρο ενός ερωτήματος (π.χ. ημερομηνία πρόσληψης, μισθός) αντιστοιχίζεται σε μία διάσταση, και το ερώτημα ορίζει ένα εύρος τιμών ανά διάσταση, σχηματίζοντας έτσι ένα k-διάστατο άξονα-ευθυγραμμισμένο υπερορθογώνιο (AABB). Αναλυτικότερα, η δομή οργανώνεται ιεραρχικά ως δέντρο από δέντρα, όπου το κύριο δέντρο ταξινομεί τα σημεία ως προς μία διάσταση, ενώ κάθε κόμβος του διατηρεί ένα πλήρες δευτερεύον δέντρο που οργανώνει τα σημεία του υποδέντρου του ως προς δεύτερη διάσταση, και ούτω καθεξής για όλες τις k διαστάσεις. Επομένως, η επαύξηση αυτή επιτρέπει την ταυτόχρονη εφαρμογή πολλαπλών φίλτρων και την απάντηση ερωτημάτων μέσω κανονικών υποσυνόλων που καλύπτουν μαζικά ομάδες σημείων.



3.1 Κλάση RangeTreeNode

Η κλάση RangeTreeNode αναπαριστά έναν κόμβο πολυεπίπεδου Range Tree και αποθηκεύει το median σημείο στη διάσταση axis (πεδίο point), το οποίο λειτουργεί ως κλειδί για τον τοπικό δυαδικό διαχωρισμό, κατά τον οποίο, τα σημεία με μικρότερη τιμή τοποθετούνται στο left, ενώ τα υπόλοιπα στο right, διατηρώντας έτσι ισορροπημένη τη δομή ως προς τη συγκεκριμένη διάσταση. Παράλληλα, ο κόμβος διατηρεί δείκτη subtree προς συσχετισμένη δομή (associated structure), δηλαδή ένα νέο RangeTree για τις υπόλοιπες k−1 διαστάσεις, χτισμένο πάνω σε όλα τα σημεία του υποδέντρου του κόμβου, ώστε τα πολυδιάστατα ερωτήματα εύρους να επιλύονται μέσω των κανονικών υποσυνόλων (canonical subsets) που ενεργοποιούνται στο πρώτο επίπεδο και εξειδικεύονται στα επόμενα.

Για επιθετικό κλάδεμα κατά την αναζήτηση, ο κόμβος διατηρεί επίσης το υπερ ορθογώνιο περίβλημα του υποδέντρου (bounds_min, bounds_max), επιτρέποντας σε O(1) χρόνο να διαπιστωθεί αν το query πλαίσιο τέμνει, περιέχει ή αποκλείει ολόκληρο το υποδέντρο, ώστε να παραλείπονται μεγάλα τμήματα του δέντρου. Ο συνδυασμός BST ως προς μία διάσταση, της συσχετισμένης δομής για τις υπόλοιπες και το ορθογώνιο παραλληλεπίπεδο καθώς γίνεται η χρήση σε 3D, συγκροτεί τον πυρήνα της αποδοτικότητας του Range Tree που υλοποιείται από την κλάση.

3.2. Κλάση RangeTree

3.2.1 Κατασκευή

Η build ταξινομεί τα σημεία ως προς τη διάσταση axis και εφαρμόζει διάσπαση με τη διάμεσο (median split), εξασφαλίζοντας περίπου ισορροπημένο δυαδικό δέντρο αναζήτησης (BST), όπου τα σημεία με μικρότερη τιμή στη διάσταση axis τοποθετούνται αριστερά και τα λοιπά δεξιά. Επιπλέον, ορίζεται η compute_bounds για τον υπολογισμό του k-διάστατου υπερορθογώνιου περιβλήματος (bounding box) κάθε υποσυνόλου, το οποίο αποθηκεύεται σε κάθε κόμβο και επιτρέπει επιθετικό κλάδεμα σε ερωτήματα εύρους. Για κάθε κόμβο δημιουργείται επίσης συνδεδεμένη δομή subtree, δηλαδή ένα νέο Range Tree πάνω σε όλα τα σημεία του συγκεκριμένου υποδέντρου και ως προς τον επόμενο άξονα (axis+1), υλοποιώντας το δέντρο από δέντρα για k διαστάσεις. Συνεπώς, έχουμε ένα (k−1)-διάστατο range tree ανά κόμβο.

Παράλληλα, για την αναδρομική κατασκευή σε βάθος χρησιμοποιείται η build_rec(pts, axis), η οποία για κενό σύνολο επιστρέφει None, ενώ στη γενική περίπτωση επιλέγει τη διάμεσο median_point, διαχωρίζει τα υποσύνολα σε αριστερό και δεξί, υπολογίζει τα bounds_min/bounds_max για όλα τα σημεία του κόμβου και, εφόσον axis+1 < k, κατασκευάζει το αντίστοιχο subtree με RangeTree(pts, axis=axis+1). Τελικά, επιστρέφει έναν πλήρως ορισμένο RangeTreeNode που περιέχει το median_point, δείκτες left/right, το subtree και τα προαναφερθέντα όρια. Η ρίζα root προκύπτει από την αρχική κλήση της build_rec στα ταξινομημένα σημεία. Το αποτέλεσμα είναι ένα πολυεπίπεδο, όσο γίνεται ισορροπημένο δέντρο, που υποστηρίζει αποδοτικό κλάδεμα βάσει ορίων και ένθετη αναζήτηση στις υπόλοιπες διαστάσεις μέσω των subtree.

3.2.2 Εισαγωγή

Η insert(point) ελέγχει πρώτα την συμβατότητα διαστάσεων (το νέο σημείο οφείλει να έχει μήκος ίσο με το k του δέντρου) και, εφόσον περάσει ο έλεγχος, το προσθέτει στη λίστα points πριν καλέσει εκ νέου τη build για πλήρη ανακατασκευή της δομής. Η επιλογή της στατικής ανακατασκευής, αντί δυναμικών in-place ενημερώσεων σε όλα τα επίπεδα, απλοποιεί δραστικά την υλοποίηση και εγγυάται ισορροπημένο BST, ακριβή bounding boxes και συνεπείς συσχετισμένες δομές στις υπόλοιπες διαστάσεις, διατηρώντας υψηλή αποδοτικότητα στα ερωτήματα εύρους, όμως το τίμημα είναι αυξημένο κόστος ανά εισαγωγή, O(n log^(d−1) n).

3.2.3 Διαγραφή

Η διαγραφή σημείου γίνεται με την βοήθεια της delete(point), στην οποία γίνεται έλεγχος και εφόσον το σημείο υπάρχει στη λίστα points, αφαιρείται και καλείται εκ νέου η build ώστε να αναδομηθεί ολόκληρη η δομή, αποκαθιστώντας ισορροπία στο κύριο BST ως προς τον τρέχοντα άξονα,

επανασυνθέτοντας τις συσχετισμένες δομές (k−1) διαστάσεων (subtrees) και επανυπολογίζοντας τα bounds_min/bounds_max για ορθό και επιθετικό pruning στα ερωτήματα εύρους. Όμως αν το σημείο δεν βρεθεί, δεν πραγματοποιείται καμία αλλαγή.

Η επιλογή αυτή καθιστά τη διαγραφή εννοιολογικά απλή, καθώς δεν απαιτούνται οι κλασικές, λεπτομερείς επεμβάσεις των δυναμικά ισορροπημένων δομών, όπως τοπικές περιστροφές (π.χ. AVL/Red-Black για αποκατάσταση ισορροπίας) ή συγχωνεύσεις κόμβων (π.χ. σε B-trees). Πιο συγκεκριμένα, σε ένα πολυεπίπεδο range tree, τέτοιες in-place επεμβάσεις θα έπρεπε να εφαρμοστούν σε πολλαπλά επίπεδα και στις συσχετισμένες δομές (k−1) διαστάσεων, ενημερώνοντας επιπλέον και τα bounding boxes. Αυτή η πλήρης ανακατασκευή τις αποφεύγει, κρατώντας τον κώδικα απλό και τη συνέπεια εγγυημένη, με τίμημα την πολυπλοκότητα πλήρους αναδόμησης ανά διαγραφή, τυπικά O(n log^{d−1} n), η οποία όμως διασφαλίζει συνέπεια και ορθότητα σε όλες τις διαστάσεις.

3.2.4 Ενημέρωση

Η update(old_point, new_point) αφαιρεί πρώτα το παλιό σημείο μέσω της delete και στη συνέχεια εισάγει το νέο μέσω της insert, ώστε το σημείο να επανατοποθετηθεί στη σωστή θέση του κύριου BST ως προς τον ενεργό άξονα και να αναδομηθούν αυτομάτως οι συσχετισμένες δομές (k−1) διαστάσεων (subtrees) μαζί με τα bounding boxes, ώστε να αποφευχθούν εξειδικευμένοι αλγόριθμοι επανατοποθέτησης σε πολλαπλά επίπεδα.

3.2.5 Αναζήτηση k-Πλησιέστερων Γειτόνων

Η kNN(target, k) υλοποιεί αναζήτηση k πλησιέστερων γειτόνων με συνδυασμό σωρού και γεωμετρικού κλαδέματος. Αρχικά, ελέγχει την εγκυρότητα (κενό δέντρο, k≤0, συμβατότητα διαστασιμότητας στόχου) και διατηρεί τους τρέχοντες k καλύτερους υποψηφίους σε σωρό που συμπεριφέρεται ως max-heap μέσω αρνητικών τετραγωνικών αποστάσεων. Κάθε κόμβος αξιολογείται με την τετραγωνική απόσταση του σημείου του από τον στόχο και εφαρμόζεται pruning συγκρίνοντας την ελάχιστη δυνατή απόσταση από το bounding box του υποδέντρου (bbox_min_dist_sq) με τη χειρότερη απόσταση του heap. Αν η πρώτη είναι μεγαλύτερη ή ίση, το υποδέντρο παραλείπεται. Η διάσχιση αυτή δίνει προτεραιότητα στο κλαδί όπου πέφτει ο στόχος στη διάσταση διαχωρισμού (axis), όπως στα kd-trees και επισκέπτεται το αντίθετο κλαδί μόνο αν το περίβλημά του μπορεί να βελτιώσει την τρέχουσα ακτίνα. Στο τέλος, οι υποψήφιοι από τον σωρό ταξινομούνται ώστε να επιστραφούν από τον πλησιέστερο στον μακρινότερο εντός των top-k, επιτυγχάνοντας αποδοτική αναζήτηση με ελάχιστο κόστος σε περιττούς ελέγχους.

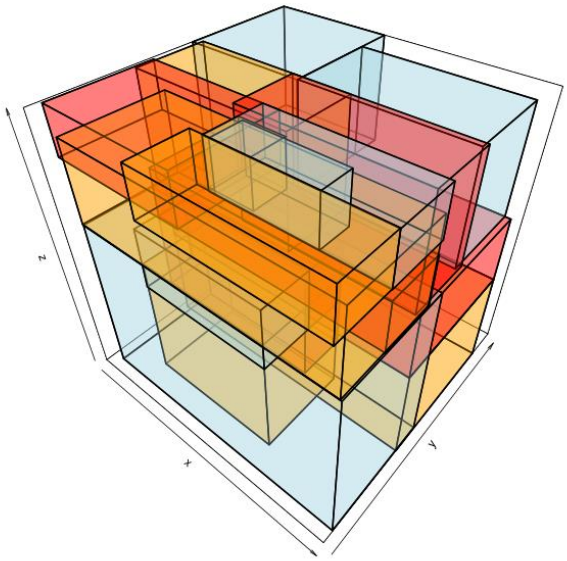
3.2.6 Ερώτημα Εύρους

Η range_query(query_ranges) υλοποιεί ερώτημα εύρους σε k διαστάσεις λαμβάνοντας ανά διάσταση ένα διάστημα [low,high] και επιστρέφοντας όλα τα σημεία εντός του αντίστοιχου AABB. Επίσης, πραγματοποιεί επιθετικό κλάδεμα μέσω των βοηθητικών rect_outside/rect_inside, που αποφασίζουν σε O(1) αν το υπερορθογώνιο του υποδέντρου (bounds_min, bounds_max) αποκλείεται πλήρως ή περιέχεται εξ ολοκλήρου στο ερώτημα. Στη δεύτερη περίπτωση, αποφεύγεται η διάσχιση ανά σημείο, καθώς αν υπάρχει συσχετισμένη δομή (subtree) για τις υπόλοιπες (k−1) διαστάσεις, της αναθέτουμε μαζικά το υπόλοιπο φίλτρο (ένα (k−1)-διάστατο range query πάνω σε όλα τα σημεία του υποδέντρου) και παίρνουμε απευθείας τα αποτελέσματα, διαφορετικά αν δεν υπάρχει subtree (δηλ. είμαστε στην τελευταία διάσταση), τότε απλώς συλλέγουμε όλα τα σημεία του υποδέντρου με ενδοδιάταξη (collect_all), αφού λόγω πλήρους κάλυψης είναι εξ ορισμού εντός του query. Αυτό, επομένως, αποφεύγει περιττές διελεύσεις και επιταχύνει την εκτέλεση.

Όταν υπάρχει μερική επικάλυψη, ελέγχεται το median σημείο του τρέχοντος κόμβου (point_in_all_dims) και η αναζήτηση κατέρχεται αναδρομικά και στα δύο παιδιά, καθώς δυνητικές λύσεις μπορεί να βρίσκονται οπουδήποτε εντός της τομής. Τέλος, αφαιρούνται τυχόν διπλοεγγραφές από τα συγκεντρωμένα αποτελέσματα, διασφαλίζοντας ορθότητα χωρίς επανάληψη, ενώ η χρήση των bounds σε κάθε κόμβο επιτρέπει εκτεταμένο pruning στα ανώτερα επίπεδα και αποτελεσματικό συνδυασμό ιεραρχικού φιλτραρίσματος με τοπικούς ελέγχους σημείων.

Δ. R Tree

Το R-tree είναι μια ιεραρχική δομή ευρετηρίασης χωρικών δεδομένων που ομαδοποιεί γειτονικά αντικείμενα μέσα σε ελάχιστα περιβάλλοντα ορθογώνια (MBR: Minimum Bounding Rectangles) και στη συνέχεια οργανώνει τα MBR σε μεγαλύτερα MBR έως τη ρίζα, σχηματίζοντας κουτιά μέσα σε κουτιά. Έτσι, σε ένα ερώτημα εύρους (π.χ. καφέ κοντά μου) που εκτελείται αποδοτικά με κλάδεμα από τη ρίζα, εξετάζονται μόνο τα MBR που τέμνονται με την περιοχή ενδιαφέροντος και αγνοούνται μαζικά όσα δεν τέμνονται. Η δομή προτάθηκε από τον Antonin Guttman το 1984 και από τότε αποτελεί θεμελιώδη μηχανισμό σε Γεωγραφικά Συστήματα Πληροφοριών (GIS), ψηφιακούς χάρτες, χωρικές βάσεις δεδομένων και γενικότερα σε εφαρμογές όπου τα ερωτήματα του τύπου «τι βρίσκεται κοντά;» απαιτούν αποδοτικό ιεραρχικό φιλτράρισμα.



4.1 Κλάση RTreeEntryK

Η κλάση RTreeEntryK μοντελοποιεί ένα ελάχιστο περιβάλλον ορθογώνιο (MBR) σε k διαστάσεις ως @dataclass, αποθηκεύοντας για κάθε διάσταση τα κατώτερα και ανώτερα όρια (mins, maxs) και προαιρετικά το ωφέλιμο φορτίο (data), δηλαδή το ίδιο το k-διάστατο σημείο όταν η καταχώρηση ανήκει σε φύλλο. Πιο αναλυτικά, παρέχει τρεις βασικές λειτουργίες, την enlarge(other), που επιστρέφει νέο RTreeEntryK με ανά συνιστώσα ελάχιστα/μέγιστα για την ένωση δύο MBRs (χρήσιμο για τη συμπύκνωση στους γονικούς κόμβους) χωρίς να μεταβάλλει το payload, την volume(), που υπολογίζει τον όγκο ως γινόμενο μηκών πλευρών, επιβάλλοντας μικρό κατώφλι eps ώστε να αποφεύγονται εκφυλισμένες μηδενικές πλευρές (με χρήση της βοηθητικής _prod) και την intersects(qmins, qmaxs), που ελέγχει σε χρόνο O(k) αν δύο MBR τέμνονται, απορρίπτοντας άμεσα την τομή όταν σε κάποια διάσταση ισχύει max < qmin ή min > qmax. Σημασιολογικά, η RTreeEntryK χρησιμοποιείται είτε ως καταχώριση φύλλου (MBR + δεδομένα) είτε ως συνοπτικό MBR υποδέντρου σε εσωτερικούς κόμβους, υλοποιώντας το σχήμα κουτιά μέσα σε κουτιά.

4.2. Κλάση RTreeNodeK

Η RTreeNodeK υλοποιεί κόμβο R-tree με δύο τρόπους, είτε σε φύλλο (leaf=True), όπου η λίστα entries περιέχει αντικείμενα RTreeEntryK (δηλ. MBR + ενδεχόμενο payload), είτε σε εσωτερικό κόμβο (leaf=False), όπου η λίστα περιέχει δείκτες σε παιδικούς κόμβους (RTreeNodeK). Και στις δύο περιπτώσεις, το πεδίο bounding αποθηκεύει το MBR που περικλείει ολόκληρο το υποδέντρο και ισούται με την ένωση των MBR των καταχωρίσεων ή των παιδιών. Επιπλέον, η μέθοδος update_bounding() επανυπολογίζει αυτό το γονικό MBR, στα φύλλα υπολογίζει τα mins/maxs όλων των RTreeEntryK, ενώ στους εσωτερικούς κόμβους ενοποιεί τα bounding των παιδιών. Η μέθοδος

enlarge_bounding(entry) παρέχει ταχεία, αθροιστική ενημέρωση κατά την εισαγωγή, αν δεν υπάρχει ήδη bounding, το αρχικοποιεί με το MBR του νέου στοιχείου διαφορετικά, το επεκτείνει συγκρίνοντας συνιστωσά προς συνιστωσά τα ελάχιστα και μέγιστα με το υπάρχον.

4.3. Κλάση RTreeK

4.3.1 Αρχικοποίηση

Ο κατασκευαστής της κλάσης RTreeK ελέγχει την ορθότητα των παραμέτρων, απαιτώντας τουλάχιστον μία διάσταση ($k \geq 1$) και ελάχιστη χωρητικότητα κόμβου δύο καταχωρίσεις ($max\_entries \geq 2$), καθώς μικρότερες τιμές θα καθιστούσαν τη δομή μη λειτουργική. Στη συνέχεια, αποθηκεύει τον αριθμό διαστάσεων και τη μέγιστη χωρητικότητα, ενώ δημιουργεί τη ρίζα ως κενό φύλλο χωρίς bounding box, ώστε το δέντρο να ξεκινά ρηχό και να φιλοξενεί απευθείας καταχωρίσεις. Το ύψος του δέντρου αυξάνεται μόνο όταν η ρίζα υπερχειλίσει, οπότε διασπάται σε δύο φύλλα και δημιουργείται νέα ρίζα που τα συνδέει, διατηρώντας έτσι όλα τα φύλλα στο ίδιο βάθος. Η σχεδιαστική αυτή επιλογή, σύμφωνη με την αρχιτεκτονική των B-trees, επιτρέπει οι διασπάσεις να παραμένουν τοπικές και να ανεβαίνουν μόνο όσο απαιτείται, διασφαλίζοντας ισορροπία.

4.3.2 Κατασκευή & Εισαγωγή

Η μέθοδος insert() δημιουργεί ένα νέο αντικείμενο RTreeEntryK με ελάχιστα και μέγιστα όρια ίσα με τις συντεταγμένες του σημείου (σημειακό MBR) και εκκινεί την αναδρομική εισαγωγή από τη ρίζα. Σε εσωτερικούς κόμβους, η διαδικασία βασίζεται στη _choose_subtree(), η οποία επιλέγει το παιδί που υφίσταται τη μικρότερη αύξηση όγκου αν ενταχθεί το νέο entry, ενώ σε περίπτωση ισοπαλίας προτιμάται εκείνο με τον μικρότερο τρέχοντα όγκο. Σε φύλλα, το νέο entry απλώς προστίθεται και το bounding ενημερώνεται με enlarge_bounding(), διατηρώντας τη συνέπεια της ιεραρχίας. Αν ο κόμβος υπερχειλίσει, ενεργοποιείται η _split_node(), όπου αρχικά εντοπίζεται seeds pair, δηλαδή τα δύο πρώτα entries (MBRs) που επιλέγονται για να τραβήξουν τα υπόλοιπα entries σε δύο ομάδες με μικρή επικάλυψη και σφιχτά κουτιά, με μέγιστη L1 απόσταση κέντρων για βέλτιστο διαχωρισμό, και στη συνέχεια τα υπόλοιπα entries κατανέμονται στον κόμβο που προκαλεί τη μικρότερη διόγκωση, με εύλογες ισοβαθμίες σε όγκο ή πληθικότητα. Η ίδια λογική εφαρμόζεται και στη ρίζα μέσω της _split_root(), ώστε το δέντρο να ψηλώνει μόνο όταν χρειάζεται.

4.3.3 Διαγραφή

Η delete(*coords) υλοποιεί διαγραφή σημείου με ακριβές ταίριασμα συντεταγμένων και αναδρομική διάσχιση με κλάδεμα βάσει τομής, όπου αντιμετωπίζει το στόχο ως σημειακό MBR (mins=maxs=coords) και κατέρχεται μόνο σε παιδιά των οποίων το bounding τέμνει αυτό το MBR. Σε περίπτωση φύλλου, αν εντοπιστεί entry.data == target, το entry αφαιρείται και ο κόμβος επαναϋπολογίζει το δικό του MBR μέσω update_bounding(). Η ένδειξη επιτυχίας διαχέεται προς τα πάνω, ώστε κάθε πρόγονος που συμμετείχε στην αναζήτηση να ανανεώσει το συνοπτικό του MBR, διατηρώντας συνέπεια στη ιεραρχία. Μετά την ολοκλήρωση, αν η ρίζα δεν έχει πλέον περιεχόμενο, αντικαθίσταται από νέο κενό φύλλο ώστε η δομή να παραμένει έγκυρη.

4.3.4 Ενημέρωση

Η μέθοδος update(old_coords, new_coords) πρώτα καλεί delete(*old_coords) ώστε να αφαιρεθεί η παλαιά καταχώριση και να ανανεωθούν αναδρομικά τα γονικά MBRs των εμπλεκόμενων κόμβων, και στη συνέχεια καλεί insert(*new_coords) για να εισαχθεί η νέα θέση, τοποθετώντας την στο κατάλληλο υποδέντρο βάσει του κριτηρίου ελάχιστης διόγκωσης (minimal enlargement) και επικαιροποιώντας εκ νέου τα συνοπτικά MBRs μέσω των υπαρχόντων μηχανισμών (enlarge_bounding, update_bounding).

4.3.5 Αναζήτηση k-Πλησιέστερων Γειτόνων

Η kNN(target, k) υλοποιεί αναζήτηση k-πλησιέστερων γειτόνων με best-first διάσχιση πάνω σε ελάχιστο σωρό (min-heap), όπου συνδυάζονται στον ίδιο σωρό τόσο κόμβοι του R-tree όσο και τελικά σημεία, με κάθε εγγραφή να φέρει κλειδί προτεραιότητας την τετραγωνική απόσταση. Πριν ξεκινήσει, ελέγχει τη διάσταση του target και το ακέραιο k, και αρχικοποιεί τον σωρό με τη ρίζα μέσω push_node, η οποία υπολογίζει κατώτερο φράγμα απόστασης σημείου→κουτί (MBR) με τη _bbox_min_dist_sq και ωθεί το ζεύγος (lb, id(node), node, False).

Για πραγματικά σημεία, η push_point υπολογίζει την πραγματική τετραγωνική ευκλείδεια απόσταση d2 και ωθεί (d2, id(p), p, True). Ο βρόχος εξάγει κάθε φορά το στοιχείο με το μικρότερο κλειδί, αν είναι κόμβος, επεκτείνει το αντικείμενο ωθώντας είτε τα παιδιά (εσωτερικός κόμβος) είτε τα σημεία των φύλλων, διαφορετικά αν είναι σημείο, το προσθέτει στους καλύτερους υποψηφίους και ενημερώνει το τρέχον κατώφλι best_kth (τη μέγιστη από τις μέχρι τώρα k μικρότερες d2). Εφαρμόζεται ισχυρό κλάδεμα, όταν έχουμε ήδη k σημεία και το ελάχιστο διαθέσιμο κάτω όριο στον σωρό είναι ≥ best_kth, καμία περαιτέρω επέκταση δεν μπορεί να βελτιώσει το αποτέλεσμα και η αναζήτηση τερματίζει. Παράλληλα, η χρήση τετραγωνικών αποστάσεων είναι σημαντική (χωρίς ρίζες), id(...) tie-breaker ώστε να επιβάλλεται μια σταθερή

και ορισμένη σειρά σε ισοβαθμίες και κάτω φραγμάτων από MBRs. Επιπλέον, επιτρέπει να αποφεύγονται μεγάλα άσχετα υποδέντρα, προσφέροντας αποδοτική κλιμάκωση και επιστρέφοντας τα k πλησιέστερα σημεία ταξινομημένα κατά αυξανόμενη απόσταση.

4.3.6 Ερώτημα Εύρους

Η range_query(*ranges) υλοποιεί ερώτημα εύρους σε R-tree k διαστάσεων. Αρχικά, ελέγχει ότι δόθηκε ένα ζεύγος (min,max) για κάθε διάσταση, εξάγει τα διανύσματα qmins/qmaxs και εκκινεί μια αναδρομική DFS από τη ρίζα. Για κάθε κόμβο λαμβάνει το γονικό MBR (node.bounding) και, αν λείπει ή δεν τέμνει το ορθογώνιο του query (intersects), κλαδεύει ολόκληρο το υποδέντρο (early exit). Επίσης, αν ο κόμβος είναι φύλλο, ελέγχει γραμμικά τα MBR των entries και, όπου υπάρχει τομή, προσθέτει το συσχετισμένο payload (entry.data) στα αποτελέσματα, διαφορετικά αν είναι εσωτερικός, κατέρχεται αναδρομικά στα παιδιά που πέρασαν το φίλτρο του γονικού MBR. Στο τέλος επιστρέφει τη λίστα των payload σημείων που τέμνουν το ορθογώνιο του query.

4.4. Κλάση RTree3D

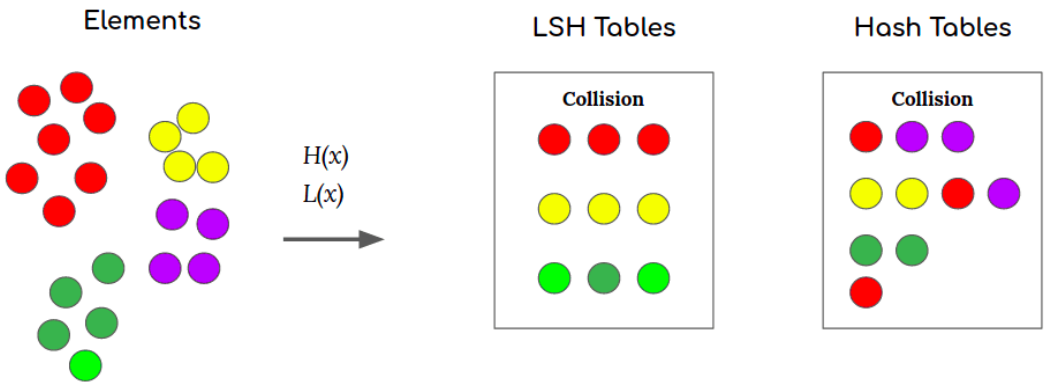
Η κλάση RTree3D υλοποιεί ένα εξειδικευμένο περιτύλιγμα της RTreeK με σταθερή διάσταση k=3, προσφέροντας απλοποιημένο API για τρισδιάστατα δεδομένα μέσω των μεθόδων insert(x,y,z), delete(x,y,z), update(old_x,old_y,old_z,new_x,new_y,new_z), range_query(x_range,y_range,z_range) και kNN(target,k). Κάθε μέθοδος προωθείται εσωτερικά στο RTreeK (self._rt = RTreeK(k=3, ...)) και τα αποτελέσματα επιστρέφονται ως τριπλέτες (x,y,z), εξασφαλίζοντας συνεκτική διαχείριση σημείων στον τρισδιάστατο χώρο. Η παράμετρος max_entries λειτουργεί ως κρίσιμος ρυθμιστής χωρητικότητας κόμβων, καθορίζοντας το μέγιστο πλήθος MBRs/παιδιών ανά κόμβο και, κατ' επέκταση, την ισορροπία του δέντρου. Παράλληλα, με υψηλές τιμές το δέντρο παραμένει ρηχό, απαιτώντας λιγότερες διασπάσεις κατά την εισαγωγή και μειώνοντας το κόστος κατασκευής/συντήρησης, αλλά με συνέπεια μεγαλύτερη επικάλυψη MBRs και υποβάθμιση του pruning στα ερωτήματα. Αντίθετα, με χαμηλές τιμές οι κόμβοι διασπώνται συχνότερα, το δέντρο αποκτάει μεγαλύτερο ύψος και τα MBRs παραμένουν πιο συμπαγή, με περιορισμένη επικάλυψη και βελτιωμένη απόδοση στα queries.

E. Locality-Sensitive Hashing (LSH)

Η Locality Sensitive Hashing (LSH) σχεδιάστηκε για να επιταχύνει την αναζήτηση ομοιοτήτων σε τεράστια σύνολα δεδομένων, μετατρέποντας

περίπλοκα αντικείμενα (π.χ. τραγούδια, έγγραφα, γονιδιώματα) σε μικρές υπογραφές σταθερού μεγέθους που λειτουργούν σαν ψηφιακά αποτυπώματα, ώστε η ομοιότητα δύο υπογραφών να προσεγγίζει την ομοιότητα των αρχικών δεδομένων χωρίς να απαιτείται πλήρης σύγκριση. Το κρίσιμο χαρακτηριστικό του LSH είναι ότι, αντί να αποφεύγει τις συγκρούσεις όπως το παραδοσιακό hashing, τις επιδιώκει για τα παρόμοια αντικείμενα, φροντίζοντας αυτά να καταλήγουν στον ίδιο κάδο με μεγάλη πιθανότητα. Αυτό επιτυγχάνεται μέσω της τεχνικής banding, όπου οι υπογραφές χωρίζονται σε ζώνες και αρκεί μία μόνο ζώνη να συμπέσει για να χαρακτηριστούν δύο στοιχεία ως υποψήφιο ζεύγος.

Επιπλέον, με την κατάλληλη ρύθμιση των παραμέτρων (πλήθος και μέγεθος ζωνών), το LSH δημιουργεί ένα απότομο όριο, στο οποίο στοιχεία που μοιάζουν λίγο αγνοούνται, ενώ όσα μοιάζουν πολύ συγκρούονται σχεδόν σίγουρα. Έτσι, αποφεύγονται τρισεκατομμύρια συγκρίσεις, η διαδικασία γίνεται κλιμακώσιμη, και εφαρμογές όπως το Spotify, το Netflix, η Google, το Shazam, ακόμη και τα σύγχρονα LLMs, μπορούν να εντοπίζουν παρόμοιο περιεχόμενο σε πραγματικό χρόνο. Επομένως, η δύναμη του LSH βρίσκεται στην ανταλλαγή απόλυτης ακρίβειας με εντυπωσιακή ταχύτητα, προσφέροντας αρκετά καλές απαντήσεις αμέσως, αντί για τέλειες απαντήσεις που δεν έρχονται ποτέ.



5.1. MinHash & Υπογραφές Jaccard

Η συνάρτηση _minhash_from_tokens υλοποιεί την τεχνική MinHash, κατασκευάζοντας ένα αντικείμενο datasketch.MinHash με num_perm ανεξάρτητες συναρτήσεις κατακερματισμού που προσομοιώνουν τυχαίες μεταθέσεις. Για κάθε σύνολο tokens εκτελείται μία ενημέρωση ανά token, η οποία εφαρμόζεται εσωτερικά σε όλες τις μεταθέσεις, κρατώντας το ελάχιστο hash σε κάθε περίπτωση. Οι παραγόμενες υπογραφές αποθηκεύονται σε cache (doc_minhash) ώστε να επαναχρησιμοποιούνται αποδοτικά σε μεταγενέστερα ερωτήματα ή διαδικασίες backfill. Με τον τρόπο αυτό, η ομοιότητα Jaccard μεταξύ δύο συνόλων προσεγγίζεται μέσω του ποσοστού σύμπτωσης στις θέσεις των υπογραφών τους, μειώνοντας δραστικά το υπολογιστικό κόστος

σύγκρισης από συνάρτηση του μεγέθους των αρχικών συνόλων σε σταθερό χρόνο O(num_perm). Το αποτέλεσμα είναι προβλέψιμες και χαμηλού κόστους συγκρίσεις, καθώς και η δυνατότητα μαζικής προκατάταξης υποψηφίων πριν από τυχόν ακριβέστερη επαλήθευση, χαρακτηριστικά που καθιστούν το MinHash θεμελιώδες εργαλείο για συστήματα ανάκτησης πληροφορίας μεγάλης κλίμακας.

5.2 Τεχνική Ζωνών (Banding) και Ρύθμιση Ευαισθησίας

Η υλοποίηση με datasketch.MinHashLSH αξιοποιεί την τεχνική banding ώστε να ελέγξει την ευαισθησία της διαδικασίας ανάκτησης υποψηφίων. Αναλυτικότερα, κάθε υπογραφή MinHash μήκους num_perm διαιρείται σε b ζώνες των r γραμμών, και κάθε ζώνη κατακερματίζεται ανεξάρτητα. Σε περίπτωση που δύο υπογραφές συγκρουστούν σε κάποια ζώνη, τότε χαρακτηρίζονται ως υποψήφιες για περαιτέρω έλεγχο. Παράλληλα, η παράμετρος threshold συνοψίζει την καμπύλη ευαισθησίας σε ένα απλό όριο, δηλαδή μόνο ζεύγη με MinHash–Jaccard ομοιότητα μεγαλύτερη ή ίση από το threshold ανακτώνται. Επίσης, η επιλογή μεγάλου b ή μικρού r καθιστά το φίλτρο πιο χαλαρό, αυξάνοντας την ανάκληση αλλά και τα ψευδώς θετικά, ενώ αντίστροφα μικρό b ή μεγάλο r οδηγεί σε αυστηρότερο φίλτρο με μεγαλύτερη ακρίβεια, αλλά κίνδυνο ψευδώς αρνητικών. Κατά συνέπεια, οι παράμετροι num_perm και threshold προσφέρουν έναν ελεγχόμενο μηχανισμό trade-off ανάμεσα σε ταχύτητα, ανάκτηση και ακρίβεια, καθιστώντας την τεχνική banding θεμελιώδη για αποδοτικά LSH σχήματα μεγάλης κλίμακας.

5.3. Κλάση LSHIndex

Η κλάση LSHIndex υλοποιεί ένα ολοκληρωμένο περιτύλιγμα γύρω από τον μηχανισμό LSH/MinHash, ενσωματώνοντας όλα τα απαραίτητα στάδια για αποδοτική και κλιμακώσιμη αναζήτηση ομοιότητας. Συγκεκριμένα, υποστηρίζει tokenization, μαζική εισαγωγή εγγράφων, εισαγωγή με προϋπολογισμένα MinHash, καθώς και προαιρετικές διαδικασίες ακριβούς επανακατάταξης (exact rerank) ή backfill για ενίσχυση της ανάκλησης όταν οι πίνακες LSH επιστρέφουν λίγους υποψηφίους. Το index διατηρεί in-memory caches για tokens, σύνολα tokens και MinHash ανά doc_id, μειώνοντας το υπολογιστικό κόστος μέσω αποφυγής επαναϋπολογισμών. Επιπλέον, ο πυρήνας της λειτουργίας βασίζεται στο datasketch.MinHashLSH(threshold, num_perm), με τη ροή ενός ερωτήματος να περιλαμβάνει (α) εξαγωγή υποψηφίων από τους πίνακες LSH, (β) προαιρετική συμπλήρωση με παγκόσμια κατάταξη βάσει MinHash–Jaccard (backfill), και (γ) τελικό φιλτράρισμα με exact Jaccard σε περιορισμένη δεξαμενή για καθαρό top-N. Με αυτόν τον τρόπο, η LSHIndex προσφέρει έναν επεκτάσιμο και παραμετροποιήσιμο μηχανισμό

προσεγγιστικής αναζήτησης ομοιότητας, συνδυάζοντας την ταχύτητα του LSH με την ακρίβεια της Jaccard επαλήθευσης.

5.3.1 Αρχικοποίηση

Κατά την αρχικοποίηση της κλάσης LSHIndex, ο κατασκευαστής δέχεται τις παραμέτρους num_perm (μήκος υπογραφής) και threshold (κατώφλι ανάκτησης υποψηφίων από το LSH) και δημιουργεί εσωτερικά ένα αντικείμενο datasketch.MinHashLSH με τις αντίστοιχες τιμές. Παράλληλα, προετοιμάζονται τρεις in-memory caches, το doc_tokens (λίστα tokens ανά έγγραφο), το doc_token_sets (σύνολο tokens για ακριβή υπολογισμό Jaccard) και το doc_minhash (υπογραφή MinHash). Η παράμετρος tokenizer είναι επεκτάσιμη (pluggable), που επιτρέπει στον σχεδιαστή του συστήματος να ορίσει το επίπεδο λεπτομέρειας με το οποίο θα αναπαριστάται το κείμενο πριν μπει στη διαδικασία του MinHash/LSH.

Από προεπιλογή, χρησιμοποιείται ένας απλός tokenizer που μετατρέπει το κείμενο σε πεζά γράμματα, ώστε οι συγκρίσεις να είναι ανεξάρτητες από κεφαλαία/πεζά, και το διαχωρίζει σε tokens με βάση τα κενά (whitespace). Ωστόσο, ο χρήστης έχει τη δυνατότητα να τον αντικαταστήσει με πιο εξειδικευμένες τεχνικές, όπως είναι το shingling, όπου παράγονται επικαλυπτόμενες ακολουθίες χαρακτήρων ή λέξεων για μια πιο λεπτομερή ανίχνευση ομοιοτήτων, ή τον καθαρισμό κειμένου, όπου αφαιρούνται stopwords, σημεία στίξης ή ειδικοί χαρακτήρες ώστε τα tokens να είναι πιο καθαρά και συγκρίσιμα. Αυτός ο σχεδιασμός διαχωρίζει με σαφήνεια το επίπεδο ευρετηρίασης (MinHash/LSH) από το επίπεδο δεδομένων (tokenization), ώστε να προσαρμόζεται στις εκάστοτε ανάγκες. Τέλος, η επιλογή του num_perm καθορίζει τη στατιστική ακρίβεια στην εκτίμηση της ομοιότητας Jaccard, ενώ το threshold ελέγχει την ευαισθησία του πρώτου σταδίου ανάκτησης, προσφέροντας ισορροπία ανάμεσα σε ανάκληση και ακρίβεια.

5.3.2 Βοηθητικά

Η συνάρτηση _to_tokens εξασφαλίζει την ομοιομορφία της εισόδου, μετατρέποντας είτε ακατέργαστο κείμενο μέσω του ορισμένου tokenizer, είτε προϋπάρχουσες ακολουθίες tokens σε λίστα συμβολοσειρών, για να είναι συνεπή ανεξάρτητα από τη μορφή των δεδομένων. Αντίστοιχα, η _minhash_from_tokens κατασκευάζει αντικείμενο MinHash πραγματοποιώντας num_perm ενημερώσεις, μία για κάθε token, με χρήση κωδικοποίησης UTF-8, ενώ σε περίπτωση κενής εισόδου επιστρέφει None, αποτρέποντας άκυρες καταχωρήσεις και διασφαλίζοντας την ακεραιότητα του ευρετηρίου. Οι δύο αυτές συναρτήσεις συνιστούν θεμελιώδη στοιχεία για τη σταθερότητα και την αξιοπιστία της διαδικασίας δημιουργίας υπογραφών MinHash.

5.3.3 Εισαγωγή Εγγράφων

Οι ροές καταχώρισης στο ευρετήριο υλοποιούνται μέσω της add_document(doc_id, text_or_tokens), που ελέγχει την ύπαρξη διπλότυπων αναγνωριστικών, μετατρέπει σε token και τυποποιεί την είσοδο με τη βοήθεια της _to_tokens, απορρίπτει κενές εισόδους, δημιουργεί υπογραφή με την _minhash_from_tokens και, εφόσον αυτή είναι έγκυρη, την εισάγει στο MinHashLSH, ενημερώνοντας παράλληλα τις τρεις in-memory caches (doc_tokens, doc_token_sets, doc_minhash) για ταχεία πρόσβαση και υποστήριξη ακριβούς υπολογισμού Jaccard. Από την άλλη, η add_documents_batch αποτελεί περιτύλιγμα της ίδιας διαδικασίας για μαζικές εισαγωγές, ενώ η add_document_minhash(doc_id, m, tokens=None) επιτρέπει την καταχώρηση εγγράφων με προϋπολογισμένο αντικείμενο MinHash, ελέγχοντας τον τύπο και την ύπαρξη διπλοτύπων, και εισάγοντάς το στο LSH. Σε περίπτωση που παρέχονται tokens, ενημερώνονται επιπλέον τα doc_tokens και doc_token_sets ώστε να είναι δυνατή η ακριβής επανακατάταξη, ενώ σε διαφορετική περίπτωση διατηρείται μόνο η προσεγγιστική βαθμολόγηση βάσει υπογραφών.

5.3.4 Αφαίρεση & Μέγεθος

Η remove_document(doc_id) απομακρύνει το έγγραφο αποκλειστικά από τις τοπικές δομές cache (doc_tokens, doc_token_sets, doc_minhash), ενώ το MinHashLSH δεν υποστηρίζει εγγενή διαγραφή, γεγονός που καθιστά αναγκαία την πλήρη ανακατασκευή του ευρετηρίου για την οριστική απομάκρυνση από τους πίνακες LSH. Συμπληρωματικά, η size() επιστρέφει το πλήθος εγγράφων βάσει του λεξικού doc_minhash. Η συγκεκριμένη σχεδιαστική επιλογή διασφαλίζει την αποφυγή ασυνεπειών μεταξύ cache και LSH δομής, θέτοντας σαφή όρια, ώστε οι αφαιρέσεις πραγματοποιούνται σε O(1) χρόνο στις cache, ενώ η διατήρηση της συνολικής συνέπειας απαιτεί περιοδικό rebuild.

5.3.5 Διαδικασία Ερωτήματος

Η query_similar αρχικά μετατρέπει το ερώτημα σε tokens και παράγει το αντίστοιχο MinHash. Στη συνέχεια, πραγματοποιεί ανάκτηση υποψηφίων μέσω των πινάκων LSH (με την _minhash_query_candidates να καλεί με ασφάλεια τη self.lsh.query(q_m) και, σε περίπτωση σφάλματος, να επιστρέφει κενή λίστα με σχετική προειδοποίηση), ενώ σε περίπτωση που οι υποψήφιοι υπολείπονται του ορίου min_candidates, εφαρμόζεται backfill με την _global_candidates_by_minhash, η οποία υπολογίζει την ομοιότητα MinHash-Jaccard του ερωτήματος με όλα τα έγγραφα του doc_minhash, ταξινομεί τα αποτελέσματα σε φθίνουσα σειρά και περιορίζει την αναζήτηση βάσει του backfill_budget για ελεγχόμενο κόστος.

Παράλληλα, ακολουθεί προαιρετικό rerank με ακριβή Jaccard σε exact_pool για βελτίωση της τελικής κατάταξης, και τέλος επιστρέφονται τα κορυφαία top_n αποτελέσματα, είτε ως αναγνωριστικά εγγράφων, είτε ως ζεύγη (doc_id, score) μέσω της επιλογής return_pairs. Η χρήση της μετρικής MinHash-Jaccard σε όλα τα στάδια εξασφαλίζει συνέπεια ανάμεσα στην ανάκτηση και την τελική αξιολόγηση, ενώ ο έλεγχος μέσω του backfill_budget επιτυγχάνει ισορροπία ανάμεσα σε υψηλή ανάκτηση και περιορισμένο υπολογιστικό κόστος.

5.3.6 Ακριβής Επανακατάταξη

Η μέθοδος _exact_rerank εφαρμόζει στοχευμένη ανακατάταξη υψηλής ακρίβειας, υπολογίζοντας την ομοιότητα Jaccard σε περιορισμένη δεξαμενή κορυφαίων υποψηφίων (exact_pool). Αρχικά, τα tokens του ερωτήματος μετατρέπονται σε σύνολο και συγκρίνονται με τα αποθηκευμένα σύνολα tokens των εγγράφων, ώστε να προκύψει το ακριβές σκορ μέσω του λόγου τομής προς ένωση. Σε περιπτώσεις όπου απουσιάζουν tokens για κάποιο έγγραφο, διατηρείται το προσεγγιστικό σκορ MinHash, εξασφαλίζοντας συνεπή και σταθερή συμπεριφορά. Τα αποτελέσματα ταξινομούνται εκ νέου κατά φθίνουσα ομοιότητα και επιστρέφονται, υλοποιώντας ένα υβριδικό σχήμα κατάταξης που συνδυάζει ταχύ αρχικό φιλτράρισμα με LSH/MinHash και ακριβή έλεγχο σε επιλεγμένο υποσύνολο, επιτυγχάνοντας βελτιωμένη precision@N με περιορισμένο υπολογιστικό κόστος.

5.3.7 Backfill για Ανάκτηση

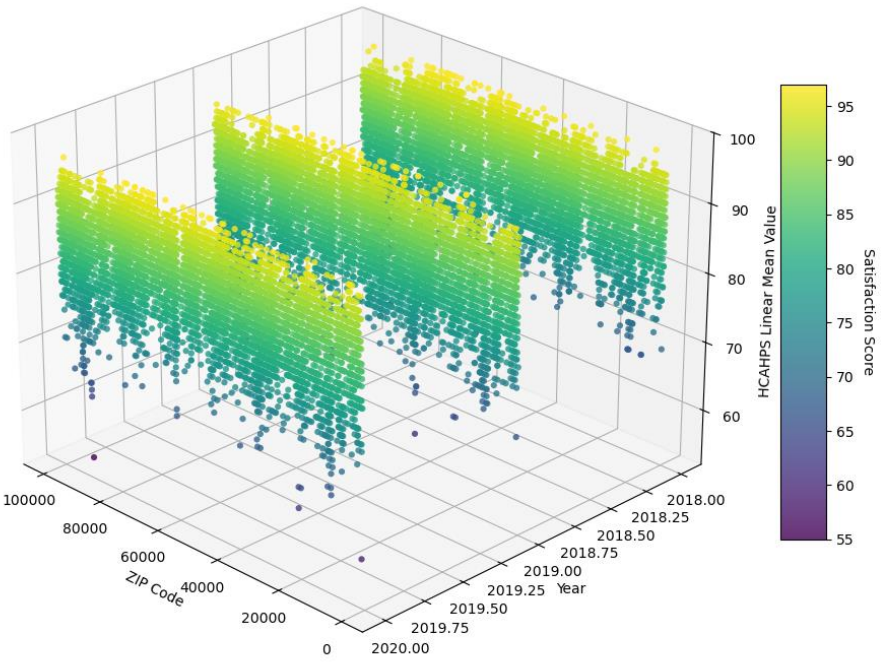
Η _global_candidates_by_minhash υλοποιεί μηχανισμό backfill για ενίσχυση της ανάκτησης, εκτελώντας παγκόσμιο σκανάρισμα όλων των εγγράφων με βάση τη μετρική MinHash-Jaccard, ταξινομώντας τα αποτελέσματα κατά φθίνουσα ομοιότητα και επιστρέφοντας τα καλύτερα, είτε χωρίς περιορισμό, είτε βάσει του ορίου budget. Στη συνέχεια, η query_similar συγχωνεύει τα αποτελέσματα του backfill με τους αρχικούς υποψηφίους από τους πίνακες LSH μέχρι να ικανοποιηθεί το κατώφλι min_candidates. Επομένως, το σχήμα αυτό λειτουργεί ως δίχτυ ασφαλείας, αποτρέποντας την επιστροφή κενών αποτελεσμάτων σε δύσκολα ερωτήματα ή αυστηρά κατώφλια.

4. Αξιολόγηση

Η αξιολόγηση πραγματοποιήθηκε σε 18324 εγγραφές με 3D σημεία (ZIP, Year, Score) και αντίστοιχες κειμενικές περιγραφές. Πιο συγκεκριμένα, για την αποτύπωση της κειμενικής ομοιότητας εφαρμόστηκε MinHash (num_perm = 128) πάνω σε character 3-grams, ενώ για το χωρικό σκέλος δοκιμάστηκαν

τέσσερις πολυδιάστατες δομές δεδομένων, KD-Tree, Octree, Range Tree και R-Tree. Παράλληλα, η υβριδική ροή spatial pruning → LSH bucket backfill → exact rerank πέτυχε recall@5 = 1.0 και Jaccard = 1.0 για όλες τις δομές, δηλαδή ανέκτησε τα ίδια top-5 αποτελέσματα με το baseline (LSH σε ολόκληρο το corpus). Το αποτέλεσμα αυτό επιβεβαιώνει τη θεωρία ότι το LSH λειτουργεί ως γρήγορος μηχανισμός παραγωγής υποψηφίων, το χωρικό κλάδεμα περιορίζει με ασφάλεια τον χώρο αναζήτησης και το ακριβές Jaccard rerank διασφαλίζει την τελική ορθότητα της κατάταξης.

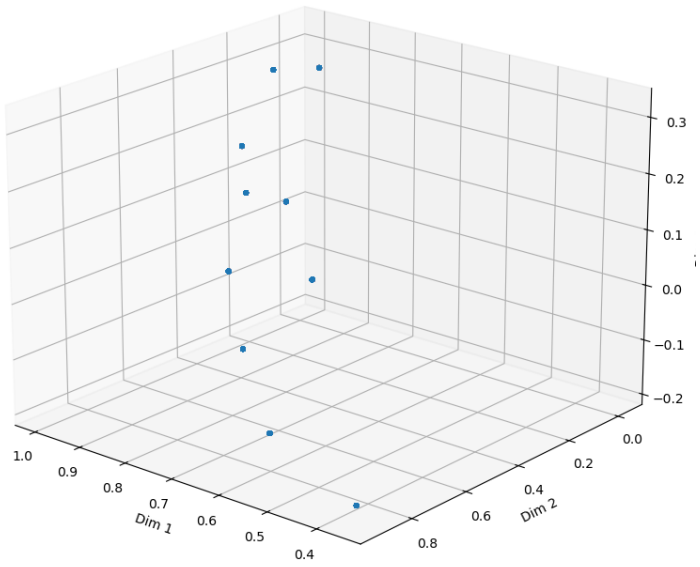
Στο Σχήμα 4, κάθε σημείο αντιπροσωπεύει μια εγγραφή νοσοκομείου, στον άξονα X απεικονίζεται ο ZIP Code, στον άξονα Y το Year (2018–2020) και στον άξονα Z το HCAHPS Linear Mean Value. Αναλυτικότερα, το χρώμα αποτυπώνει την τιμή του score, με κύρια συγκέντρωση γύρω από το εύρος 75–95, ενώ οι χαμηλότερες τιμές εμφανίζονται ως ψυχρότερα χρώματα. Οπτικά, διακρίνονται τρεις λωρίδες (μία ανά έτος) και κατακόρυφες στήλες κατά ZIP, σχηματίζοντας μια σχεδόν ορθογωνική γεωμετρία. Επομένως, η διάταξη αυτή εξηγεί γιατί οι δομές που βασίζονται σε ορθογώνια εύρη ή κυβοειδή κελιά (Octree, Range Tree, KD-Tree) μπορούν να κλαδέψουν αποτελεσματικά τον χώρο, καθώς μεγάλα τμήματα αποκλείονται άμεσα πριν την εφαρμογή του LSH για το κειμενικό backfill.



Σχήμα 4: 3D Scatter: ZIP vs Year vs Score

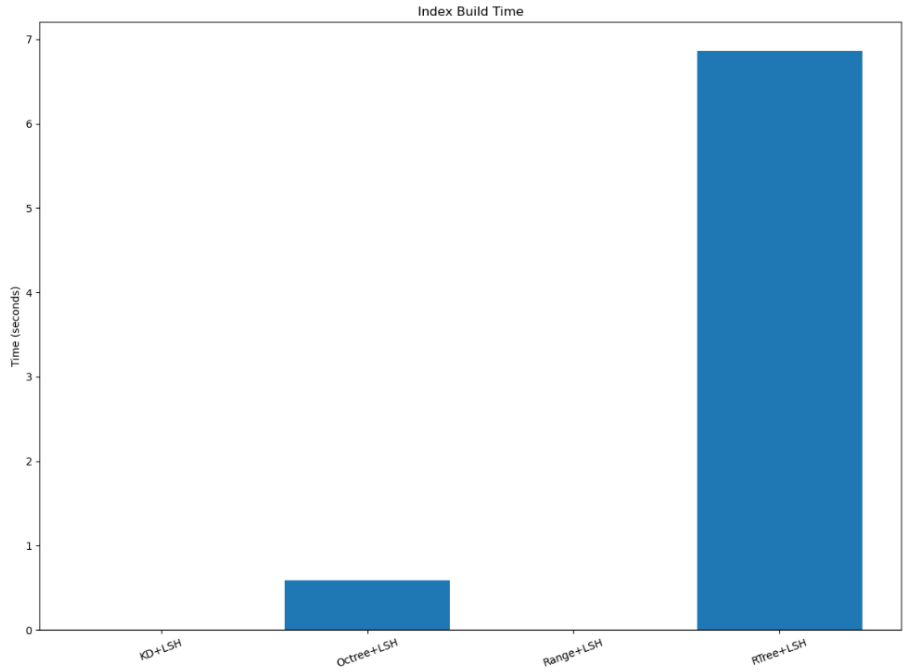
Το Σχήμα 5 παρουσιάζει μια ενδεικτική τρισδιάστατη απεικόνιση «hash-embeddings» για ένα μικρό δείγμα κειμένων. Κάθε κουκίδα αντιπροσωπεύει

ένα feedback, το οποίο έχει χαρτογραφηθεί σε 3D με κατεύθυνση που καθορίζεται μέσω hash ανά token. Οι άξονες Dim 1–3 δεν φέρουν καμία σημασιολογική ερμηνεία, οι μικρές συστάδες αντανακλούν κοινά tokens ή character n-grams, χωρίς να υποδηλώνουν απαραίτητα νοηματική ομοιότητα. Επιπλέον, το γράφημα έχει καθαρά οπτικό χαρακτήρα για την αποτύπωση της διασποράς και τον γρήγορο έλεγχο της ποιότητας των δεδομένων (π.χ. ύπαρξη συστάδων, outliers, κενών κειμένων ή διπλοτύπων). Η πραγματική αναζήτηση ομοιότητας πραγματοποιείται μέσω MinHash + LSH πάνω σε character n-grams, με ακριβές rerank βάσει Jaccard στο επιτρεπτό σύνολο. Συνεπώς, δεν πρέπει να αποδοθεί υπερβολική σημασία στις αποστάσεις ή στις θέσεις των σημείων στο διάγραμμα.



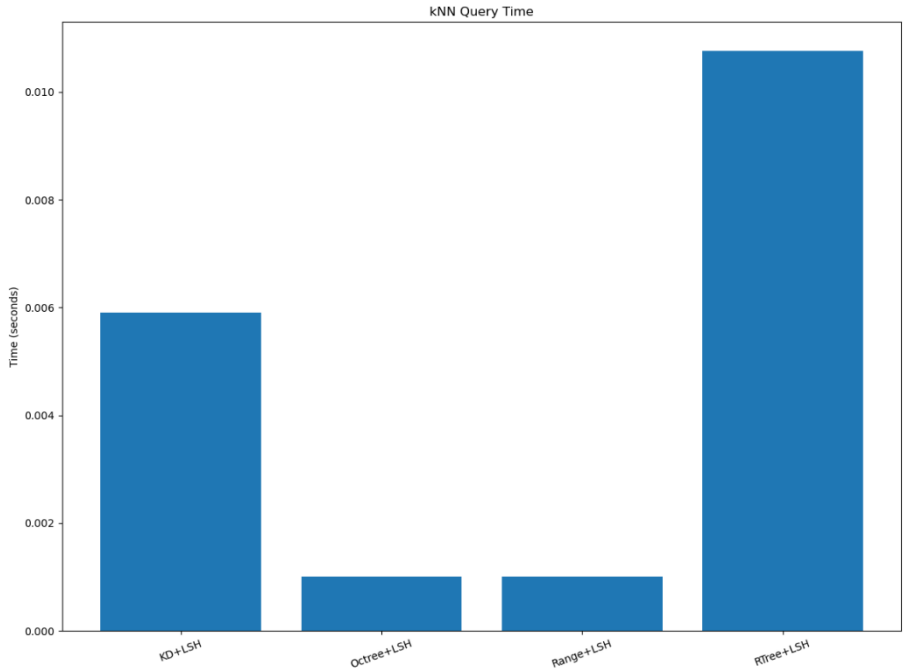
Σχήμα 5: 3D Text Embedding (hash-based)

Στο Σχήμα 6 αποτυπώνεται ο χρόνος δόμησης των χωρικών ευρετηρίων ανά συνδυασμό δομής. Το RTree+LSH είναι σαφώς το ακριβότερο, με χρόνο κατασκευής 6.86 s, λόγω των πολλαπλών εισαγωγών με split, MBRs και διαδικασίες εξισορρόπησης. Από την άλλη, το Octree+LSH έχει ενδιάμεσο κόστος (0.59 s) καθώς απαιτεί σημείο προς σημείο εισαγωγή σε οκτάντες, ενώ τα KD+LSH και Range+LSH εμφανίζουν πρακτικά μηδενικό χρόνο κατασκευής (~0.00 s), καθώς χτίζονται μαζικά στον constructor χωρίς ακριβά splits ανά σημείο. Παράλληλα, ο χρόνος κατασκευής του LSH παραμένει σχεδόν σταθερός σε όλες τις περιπτώσεις (~1.45–1.62 s) και δεν ευθύνεται για τις διαφορές του γραφήματος, το οποίο αντικατοπτρίζει κυρίως το κόστος του spatial build. Τέλος, τα αποτελέσματα είναι συνεπή με τη θεωρία, το R-Tree έχει μεγαλύτερο overhead αλλά προσφέρει δυναμικές ενημερώσεις, ενώ KD-Tree και Range Tree είναι πιο ελαφριές δομές για σημειακά δεδομένα χαμηλής διάστασης.



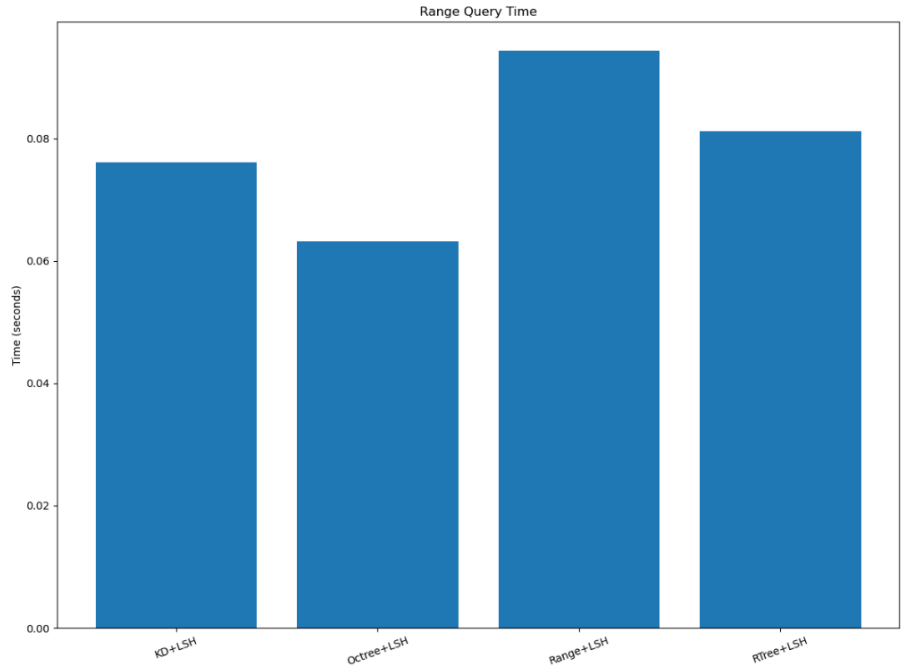
Σχήμα 6: Index Build Time

Το Σχήμα 7 παρουσιάζει τους χρόνους απόκρισης για k-NN ερωτήματα ανά ευρετήριο. Οι δομές Octree+LSH και Range+LSH είναι οι ταχύτερες (~0.001 s), χάρη στην ιεραρχική και ορθογώνια διαίρεση του χώρου που επιτρέπει αποτελεσματικό κλάδεμα σε 3D. Στη συνέχεια, το KD+LSH ακολουθεί (~0.006 s), με ελαφρώς υψηλότερο κόστος λόγω median splits που συχνά απαιτούν οπισθοδρόμηση κατά την αναζήτηση. Τέλος, το RTree+LSH εμφανίζει τον μεγαλύτερο χρόνο (~0.011 s), καθώς οι επικαλύψεις των MBRs οδηγούν σε διερεύνηση πολλαπλών κλάδων. Παρότι οι απόλυτοι χρόνοι είναι μικροί, η σχετική κατάταξη συμφωνεί πλήρως με την θεωρία ότι οι δομές που ευθυγραμμίζονται φυσικά με τον 3D χώρο (όπως το Octree) έχουν το χαμηλότερο κόστος διάσχισης, ενώ τα R-Trees υστερούν λόγω των επικαλύψεων.



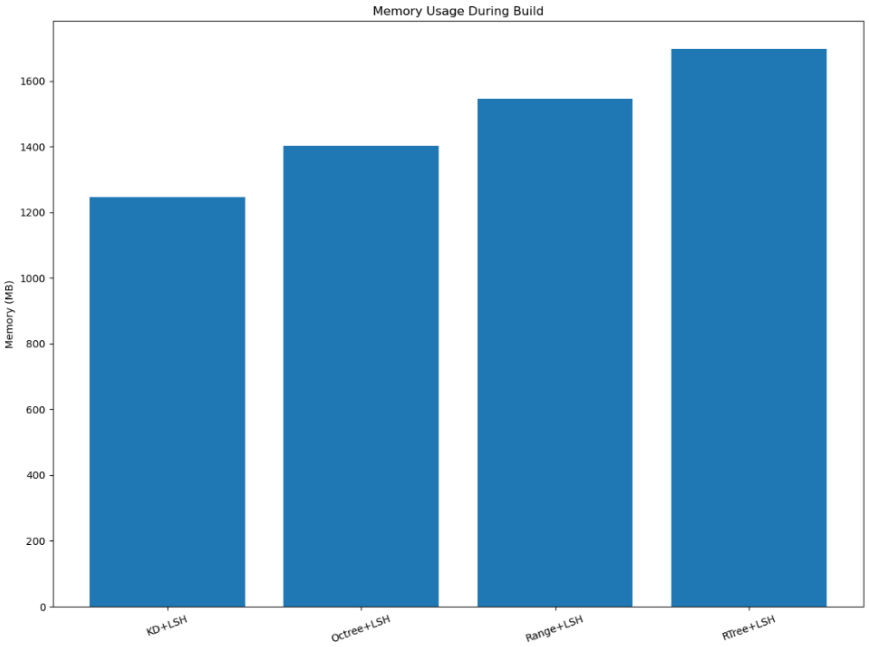
Σχήμα 7: kNN Query Time

Στο Σχήμα 8 φαίνονται οι χρόνοι εκτέλεσης για ορθογώνια ερωτήματα σε 3D. Το Octree+LSH είναι το ταχύτερο (~0.063 s), και είναι επωφελούμενο από το αποτελεσματικό κλάδεμα με κυβοειδή κελιά. Έπειτα, ακολουθεί το KD+LSH (~0.076 s) και στη συνέχεια το RTree+LSH (~0.081 s), ενώ το Range+LSH καταγράφει τον μεγαλύτερο χρόνο (~0.094 s). Η διαφορά αυτή οφείλεται σε μεγαλύτερους σταθερούς συντελεστές της υλοποίησης, αλλά και στο γεγονός ότι η δοκιμή αφορά full-range queries, που καλύπτουν μεγάλο μέρος του χώρου. Παρότι το Range-Tree διαθέτει άριστη θεωρητική πολυπλοκότητα για ορθογώνια εύρη, στην πράξη το μικρό μέγεθος του dataset (18k σημεία) και τα σταθερά κόστη καθιστούν πιο αποδοτικές τις ελαφρύτερες δομές. Σε κάθε περίπτωση, όλοι οι χρόνοι είναι μικρότεροι από 0.1 s, γεγονός που σημαίνει ότι οι διαφορές είναι περιορισμένες σε απόλυτους όρους αλλά παραμένουν συμβατές με τις θεωρητικές προσδοκίες.



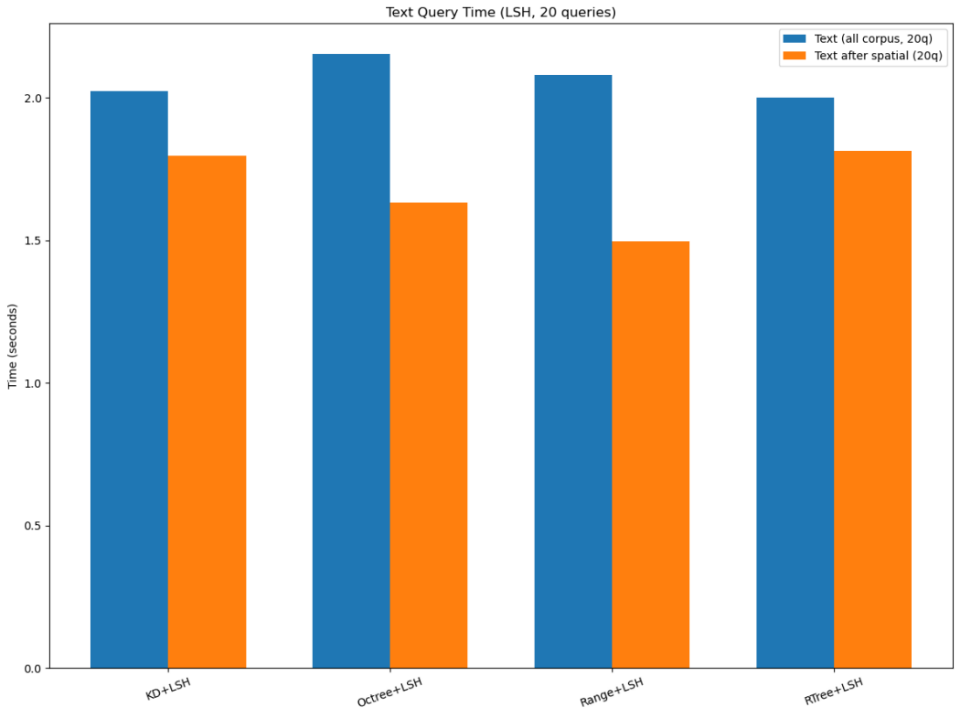
Σχήμα 8: Range Query Time

Το Σχήμα 9 παρουσιάζει την αιχμή κατανάλωσης μνήμης κατά τη δόμηση των χωρικών ευρετηρίων. Το RTree+LSH εμφανίζει τη μεγαλύτερη χρήση (~1697 MB), εξαιτίας των MBRs, των δεικτών και του overhead από τα split, ενώ το Range+LSH ακολουθεί (~1546 MB) λόγω των επιπρόσθετων δέντρων εύρους. Το Octree+LSH καταναλώνει ~1403 MB, αντανακλώντας τη δομή με κόμβους οκταντών, ενώ το KD+LSH είναι το πιο «ελαφρύ» (~1247 MB), καθώς περιορίζεται σε πίνακες και διχοτομήσεις. Το κοινό «πάτωμα» μνήμης προέρχεται κυρίως από το MinHash/LSH (18324 έγγραφα × 128 permutations), με τις διαφορές να οφείλονται στο εκάστοτε χωρικό index. Για μείωση κατανάλωσης μνήμης, μπορούν να μειωθούν οι τιμές του num_perm, να εφαρμοστεί συμπίεση στα tokens ή να προτιμηθούν πιο ελαφριές δομές όπως KD ή Octree. Συνολικά, η χρήση μνήμης ακολουθεί την πολυπλοκότητα της δομής από ~1.25 GB για KD-Tree έως ~1.70 GB για R-Tree, με το αποτύπωμα του MinHash/LSH να κυριαρχεί.



Σχήμα 9: Memory Usage

Το Σχήμα 10 συγκρίνει τον χρόνο κειμενικών ερωτημάτων με MinHash-LSH σε όλο το corpus έναντι της υβριδικής ροής spatial pruning → LSH bucket backfill → exact rerank. Και οι τέσσερις δομές παρουσιάζουν σαφή βελτίωση, KD+LSH από 2.025 σε 1.796 s (-11.3%), Octree+LSH από 2.153 σε 1.632 s (-24.2%), Range+LSH από 2.079 σε 1.496 s (-28.0%, η μεγαλύτερη μείωση), ενώ το RTree+LSH από 2.002 σε 1.813 s (-9.4%). Αναλυτικότερα, το όφελος προκύπτει καθώς το χωρικό κλάδεμα περιορίζει το allowed pool περίπου στο 50% του corpus και το backfill από τα LSH buckets ανακτά αποδοτικά τους λίγους ελλείποντες υποψηφίους, πριν την ακριβή επανακατάταξη, διατηρώντας έτσι recall@5 = 1.0 και Jaccard = 1.0. Η θεωρία αυτή επιβεβαιώνεται από τα εξής: το LSH δρα ως γρήγορος generator υποψηφίων, αλλά ο συνδυασμός του με αποτελεσματικό spatial pruning (ιδίως Range και Octree που διατηρούν καλή τοπικότητα σε 3D) μειώνει σημαντικά τον φόρτο χωρίς απώλεια ακρίβειας. Επομένως, τα μικρότερα κέρδη του R-Tree εξηγούνται από το υψηλότερο overhead και τις επικαλύψεις των MBRs, που περιορίζουν την αποδοτικότητα του pruning.



Σχήμα 10: Text Query Time

Συνεπώς, η υβριδική ροή αποδίδει εξαιρετικά, καθώς διατηρεί recall@5 = 1.0 και μειώνει τον χρόνο κειμενικών ερωτημάτων έναντι του καθαρού LSH κατά ~10–30%. Με βάση τα αποτελέσματα, το Range+LSH προσφέρει το καλύτερο συνολικό trade-off για text retrieval (μετά το spatial pruning, 2.079→1.496 s, ~−28%), ενώ το Octree+LSH ακολουθεί πολύ κοντά (1.632 s, ~−24%) και τα δύο παρέχουν ταχύτερα k-NN (~1 ms). Το KD+LSH είναι η ελαφρύτερη επιλογή (ουσιαστικά μηδενικός χρόνος δόμησης, μνήμη ~1.25 GB) και χρησιμοποιείται όταν προέχει η απλότητα. Αντίθετα, το R-Tree+LSH εμφανίζει το υψηλότερο κόστος δόμησης (≈6.86 s) και μνήμης (~1.70 GB) και ενδείκνυται κυρίως όταν απαιτούνται συχνές δυναμικές ενημερώσεις ή όταν τα δεδομένα περιγράφονται μέσω εκτεταμένων MBRs.