Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τμημα Τεχνολογιας Πληροφορικης και Υπολογιστων

# A Data-Driven Approach to the Approximate Nearest Neighbor Problem

Διπλωματικη Εργασια

του

**ΑΝΔΡΕΑ Γ. ΚΑΛΑΒΑ**

**Επιβλέπων:** Δημήτρης Φωτάκης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τμήμα Τεχνολογίας Πληροφορικής και Υπολογιστών

# A Data-Driven Approach to the Approximate Nearest Neighbor Problem

## Διπλωματικη Εργασια

του

**ΑΝΔΡΕΑ Γ. ΚΑΛΑΒΑ**

**Επιβλέπων:** Δημήτρης Φωτάκης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουλίου 2024.

| *(Υπογραφή)* | *(Υπογραφή)* | *(Υπογραφή)* |
|---|---|---|
| ..... | ..... | ..... |
| Δημήτρης Φωτάκης | Αριστείδης Παγουρτζής | Ιωάννης Εμίρης |
| Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Κ.Π.Α. |

Αθήνα, Ιούλιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τμήμα Τεχνολογίας Πληροφορικής και Υπολογιστών



**Υπεύθυνη Δήλωση**

Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Τεχνολογίας Πληροφορικής και Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

*(Υπογραφή)*

.....
Ανδρέας Καλαβάς

# Περίληψη

Το πρόβλημα της εύρεσης κοντινότερου γείτονα και οι παραλλαγές του απασχολούν την επιστημονική κοινότητα τα τελευταία πενήντα χρόνια. Εμφανίζεται σε εφαρμογές όπως συμπίεση δεδομένων, εξόρυξη δεδομένων και μηχανική μάθηση. Εκ των πολλαπλών προτεινόμενων λύσεων, λίγες μόνο παρέχουν θεωρητικές εγγυήσεις, βελτιστοποιώντας παράλληλα τη δομή για τα δεδομένα εισόδου. Αυτή η πρόκληση οφείλεται στο γεγονός ότι η προσαρμογή της δομής σε συγκεκριμένο σύνολο σημείων την καθιστά ευάλωτη σε κακόβουλα (adversarial) ερωτήματα, τα οποία επιδεινώνουν την απόδοση.

Στην παρούσα εργασία, παρουσιάζουμε ένα νέο μοντέλο για τη λύση του προβλήματος του κατά προσέγγιση κοντινού γείτονα (που είναι η εκδοχή απόφασης του προβλήματος του κοντινότερου γείτονα), στοχεύοντας να ισορροπήσουμε θεωρητικές εγγυήσεις με προσαρμοσιμότητα στο σύνολο δεδομένων. Η προσέγγισή μας είναι να αποθηκεύσουμε το σύνολο σημείων εισόδου σε μια δομή δυαδικού δέντρου, η οποία είναι βελτιστοποιημένη για συγκεκριμένο σύνολο δεδομένων και κατανομή ερωτημάτων. Η αναζήτηση ερωτημάτων γίνεται διασχίζοντας το δέντρο από τη ρίζα προς ένα η περισσότερα φύλλα. Η απόφαση για το αν η αναζήτηση θα ακολουθήσει ένα ή και τα δύο παιδιά γίνεται με βάση διαχωριστές που βρίσκονται στις κορυφές. Επιπλέον, παρουσιάζουμε μεθόδους βέλτιστης εύρεσης αυτών των διαχωριστών.

Η κεντρική ιδέα της προσέγγισής μας έγγειται στη λήψη χρήσιμης πληροφορίας από το σύνολο σημείων για τη βελτίωση της δομής μας, αλλά στη διακοπή της διαδικασίας αυτής όταν η πληροφορία μπορεί να γίνει επιβλαβής, οπότε και εφαρμόζουμε μια υπάρχουσα τεχνική με θεωρητικές εγγυήσεις. Αυτή η στρατηγική μας επιτρέπει να βελτιώσουμε το μοντέλο μας, αποφεύγοντας καταστάσεις οι οποίες θα υποβιβάζαν την επίδοσή του. Έτσι, η δομή μας παραμένει data-driven ενώ παράλληλα διατηρεί θεωρητικές εγγυήσεις.

Τέλος, διεξάγουμε πειράματα για να δείξουμε τη δυνατότητα του αλγορίθμου μας να προσαρμόζεται σε ένα σύνολο δεδομένων, ενώ παράλληλα να διατηρεί τις εγγυήσεις. Συγκεκριμένα, δοκιμάζουμε το μοντέλο μας στο σύνολο δεδομένων MNIST, εκτελώντας ερωτήματα σε μοντέλα δημιουργημένα πάνω σε διαφορετικού μεγέθους δείγμα, και ακολούθως συγκρίνουμε τα αποτελέσματά μας με αυτά της σειριακής αναζήτησης.

## Λέξεις Κλειδιά

προσεγγιστικοί αλγόριθμοι, κοντινότερος γείτονας, κοντινός γείτονας, δομές δεδομένων, αλγόριθμοι καθοδηγούμενοι από δεδομένα, βελτιστοποίηση, υπολογιστική γεωμετρία

# Abstract

The nearest neighbor search (NNS) problem and its variants have captivated scientists for the past fifty years. This problem is prevalent in applications such as data compression, data mining, and machine learning. Although numerous solutions have been proposed, few offer theoretical guarantees while simultaneously optimizing the structure for the input data. This challenge arises because adapting the structure for a specific dataset can expose vulnerabilities to adversarial queries, leading to suboptimal performance.

In this thesis, we propose a new model to solve the approximate near neighbor problem (which is the decision version of the nearest neighbor problem), aiming to balance theoretical guarantees with dataset adaptability. Our approach involves storing the input point set in a binary tree structure, optimized for performance on a fixed dataset and query distribution. Queries are processed by traversing from the root to one or more leaves. The decision to follow one or both child nodes is determined by separators located at the vertices. Additionally, we present methods for identifying those separators optimally.

The core idea of our approach is to extract useful information from the point set to enhance our structure, but to halt this extraction when it becomes potentially harmful. When this happens, we transition to an existing technique that offers theoretical guarantees. This strategy allows us to leverage the efficiency of our model while avoiding elements that could degrade performance. Thus, our structure remains data-driven while maintaining theoretical guarantees.

Finally, we conduct experiments to demonstrate our algorithm's adaptability to a dataset while preserving its theoretical guarantees. Specifically, we assess our model on the MNIST dataset, by performing queries on model instances built on different sized samples. We then compare our results with those of linear search.

## Keywords

approximation, nearest neighbor, near neighbor, data structures, data-driven algorithms, optimization, computational geometry

*στους γονείς μου, Γιώργο και Κυριακή*

# Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω τον κ. Δημήτρη Φωτάκη, επιβλέποντα καθηγητή της διπλωματικής μου εργασίας, για την ευκαιρία που μου έδωσε να ασχοληθώ με τη θεωρητική πληροφορική και για την εισαγωγή μου στον κόσμο της έρευνας. Τον ευχαριστώ για την υποστήριξη που μου παρείχε, τον χρόνο που αφιέρωσε, τις γνώσεις που μου μετέδωσε, τις γνωριμίες που μου προσέφερε, καθώς και την αντίληψη που μου ανέπτυξε σε θέματα θεωρητικής πληροφορικής. Επιπλέον, θα ήθελα να ευχαριστήσω αυτόν και τους καθηγητές κκ. Άρη Παγουρτζή και Ιωάννη Εμίρη, , οι οποίοι αποτελούν τα μέλη της επιτροπής αξιολόγησης της εργασίας μου.

Ιδιαίτερα, θα ήθελα να ευχαριστήσω τον Γιάννη Ψαρρό, με τον οποίο συνεργαστήκαμε στενά για την ολοκλήρωση αυτής της εργασίας. Οι εξειδικευμένες γνώσεις του ήταν καθοριστικές για τη δημιουργία της. Τον ευχαριστώ για την υπομονή και την εμπιστοσύνη που μου έδειξε, καθώς και για την στήριξή του καθ' όλη τη διάρκεια του τελευταίου χρόνου. Ανυπομονώ για τη συνεργασία μας και εκτός πανεπιστημίου.

Θα ήθελα επίσης να ευχαριστήσω τον κ. Πάνο Ηρακλέους, η καθοδήγηση του οποίου από τα χρόνια του γυμνασίου, ήταν καθοριστική στην απόφασή μου να ακολουθήσω σπουδές στην Πληροφορική. Οι γνώσεις του, σε συνδυασμό με την έγνοια που δείχνει για τους μαθητές του, τον καθιστούν πρότυπο καθηγητή.

Στη συνέχεια, θα ήθελα να ευχαριστήσω όλα τα παιδιά του corelab, για την προθυμία τους να μοιραστούν τις γνώσεις τους και να δώσουν συμβουλές, αλλά κυρίως για τη δημιουργία ενός φιλικού κλίματος που με βοήθησε να νιώσω μέλος μιας πολύ όμορφης κοινότητας. Επιπλέον, θα ήθελα να ευχαριστήσω τους συμφοιτητές και φίλους μου, εντός και εκτός corelab, για τις όμορφες στιγμές που μου χάρισαν, οι οποίες θα αποτελούν αναμνήσεις ζωής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για όλη τη στήριξη που μου παρείχε, και κυρίως τους γονείς μου, Γιώργο και Κυριακή. Αν και βρίσκονται στην Κύπρο, κάνουν ό,τι μπορούν για να σιγουρευτούν ότι δεν θα λείψει τίποτα σε μένα και στα αδέρφια μου, ώστε να μπορούμε να ζήσουμε χωρίς ανησυχίες και να χαρούμε τα φοιτητικά μας χρόνια. Σας αγαπώ πολύ.

Ανδρέας Καλαβάς,

Αθήνα, 9η Ιουλίου 2024

# Περιεχόμενα

# Εκτεταμένη Ελληνική Περίληψη

Το βασικό σκέλος της παρούσας διπλωματικής εργασίας είναι γραμμένο στην αγγλική γλώσσα. Σε αυτή την ενότητα, παρέχουμε μια περίληψη του περιεχομένου της, επικεντρώνοντας στους βασικούς ορισμούς, τις μεθοδολογίες και τα θεωρήματα, παραλείποντας τις μαθηματικές αποδείξεις.

## 0.1 Το Πρόβλημα

Το πρόβλημα της αναζήτησης του κοντινότερου γείτονα (Nearest Neighbor Search - NNS) είναι ουσιαστικά η δημιουργία μιας δομής για την αποθήκευση ενός συνόλου σημείων $P$, επιτρέποντας την αποτελεσματική αναζήτηση του κοντινοτερου σημείου $p^* \in P$ όταν εισάγονται αργότερα ερωτήματα $q$. Το πρόβλημα θεωρείται θεμελιώδες λόγω των διαδεδομένων εφαρμογών του σε διάφορους τομείς, όπως η συμπίεση δεδομένων, οι βάσεις δεδομένων, η εξόρυξη δεδομένων, η ανάκτηση πληροφοριών, οι βάσεις δεδομένων εικόνας και βίντεο, η μηχανική μάθηση και η επεξεργασία σημάτων.

Η εύρεση του ακριβούς κοντινότερου γείτονα αποτελεί πρόκληση επειδή απαιτεί την αναζήτηση ολόκληρου του συνόλου των σημείων στη χειρότερη περίπτωση ή τη χρήση μιας δομής δεδομένων που απαιτεί σημαντικό χώρο. Αυτή η προσέγγιση είναι αναποτελεσματική για πολλές εφαρμογές που απαιτούν ταχύτερο χρόνους αναζήτησης ή ελάχιστο χώρο αποθήκευσης. Το πρόβλημα του κατά προσέγγιση κοντινότερου γείτονα εισάγει ένα προσεγγιστικό παράγοντα $c = 1 + \epsilon$ στην απόσταση, επιτρέποντας υπογραμμικούς χρόνους αναζήτησης σε σχέση με το μέγεθος του συνόλου σημείων. Δηλώνει ότι, εάν το πλησιέστερο σημείο βρίσκεται σε απόσταση $r^*$, τότε μπορούμε να αναφέρουμε οποιοδήποτε σημείο σε απόσταση το πολύ $cr^*$.

Δεδομένου ότι το πρόβλημα του κατά προσέγγιση πλησιέστερου γείτονα είναι ένα πρόβλημα βελτιστοποίησης, πολλές λύσεις επικεντρώνονται στο αντίστοιχο πρόβλημα απόφασης, γνωστό ως το πρόβλημα του κατά προσέγγιση κοντινού γείτονα, το οποίο δηλώνει ότι εάν υπάρχει ένα σημείο σε απόσταση το πολύ $r$ από το ερώτημα, τότε ανάφερε οποιοδήποτε σημείο σε απόσταση το πολύ $cr$. Αυτό καθιστά το πρόβλημα του κατά προσέγγιση κοντινού γείτονα πιο εύχρηστο και προσιτό σε αποδοτικούς αλγορίθμους. Επιπλέον, η αναγωγή από το πρόβλημα του κοντινότερου γείτονα στο πρόβλημα του κοντινού γείτονα προσθέτει μόνο έναν πολυλογαριθμικό παράγοντα στις εγγυήσεις.

## 0.2 Προϋπάρχουσες Εργασίες (Previous Work)

Από τις πολλαπλές λύσεις που έχουν προταθεί, εμείς αναφέρουμε τρεις πολύ βασικές, και μία πιο πρόσφατη η οποία βελτιστοποιεί μία από τις προηγούμενες ακολουθώντας μια data-driven προσέγγιση. Συγκεκριμένα παρουσιάζουμε τις μεθόδους Bucketing, Approximate Voronoi Diagram, και Locality-Sensitive Hashing, γνωστή και ως LSH.

Το Bucketing χρησιμοποιείται για να λύσει το πρόβλημα του κατά προσέγγιση κοντινού γείτονα. Σχεδιάζουμε ένα τετραγωνικό πλέγμα στο χώρο, με μήκος ακμής τέτοιο ώστε η μέγιστη απόσταση μεταξύ δύο σημείων σε ίδιο κελί να είναι $\epsilon r$. Ακολούθως, για κάθε σημείο $p$ του σημειοσυνόλου $P$, σημειώνουμε σε όλα τα κελιά τα οποία τέμνονται με την μπάλα $B(p, r)$ ότι το $p$ είναι κατά προσέγγιση κοντινός γείτονας. Όταν έρθει ερώτημα $q$, απλά υπολογίζουμε σε πιο κελί πέφτει, και απαντάμε κάποιο σημείο το οποίο είναι κατά προσέγγιση κοντινός γείτονας αν υπάρχει, αλλιώς δεν κάνουμε τίποτα. Αυτή η μέθοδος δίνει πολύ καλό χρόνο ερωτήματος $O(d)$, όμως απαιτεί εκθετικό χώρο στη διάσταση $O(n) \times O(1/\epsilon)^d$.

Ένα προσεγγιστικό διάγραμμα Voronoi λύνει απευθείας το πρόβλημα του κατά προσέγγιση κοντινότερου γείτονα. Πρόκειται για μια πιο περίπλοκη δομή, η οποία ουσιαστικά συνδυάζει πολλά πλέγματα σαν αυτά του Bucketing (με ακτίνες σε δυνάμεις του 2). Τελικά, τα κελιά του διαγράμματος Voronoi (τα οποία δεν είναι κατ'ανάγκη (υπερ)κύβοι) αντιστοιχούν σε συγκεκριμένο σημείο του σημειοσυνόλου, το οποίο είναι κατά προσέγγιση κοντινότερος γείτονας. Η δομή χτίζεται σε $O((n(C/\epsilon)^{d+1}) \log^3 n)$, και απαιτεί $O((n(C/\epsilon)^{d+1}) \log^2 n)$ χώρο, όπου $C$ μια απόλυτη σταθερά.

Η μέθοδος LSH βασίζεται σε οικογένειες συναρτήσεων κατακερματισμού (hash functions) για τις οποίες όσο πιο κοντά είναι δύο σημεία, τόσο πιο πιθανό είναι το ενδεχόμενο να έχουν ίδιους κώδικες κατακερματισμού. Συνήθως τέτοιες οικογένειες πηγάζουν από κάποια τυχαία διαμέριση του χώρου (π.χ. μοίρασμα χώρου από τυχαίο υπερεπίπεδο). Αυτό που κάνουμε, είναι να ενώσουμε τους κώδικες πολλαπλών τέτοιων συναρτήσεων και να χρησιμοποιούμε τον νέο συνενωμένο κώδικα για να πούμε αν το ερώτημά μας έχει κοντινό γείτονα. Για να διασφαλίσουμε ότι αυτό γίνεται με σταθερή πιθανότητα, πρέπει να κρατήσουμε αρκετά αντίγραφα με διαφορετικές συναρτήσεις. Αυτή η διαδικασία δίνει χρόνο ερωτήματος $O(n^\rho)$ και απαιτεί χώρο $O(n^{1+\rho})$, όπου το $\rho < 1$ εξαρτάται από τις συναρτήσεις κατακερματισμού.

Μια κλασσική εφαρμογή της παραπάνω μεθόδου είναι στον χώρο Hamming, όπου τα σημεία είναι συμβολοσειρές από bits και η απόσταση μεταξύ δύο σημείων ορίζεται ως ο αριθμός από bits στα οποία διαφέρουν. Εδώ ως συναρτήσεις κατακερματισμού μπορούμε να πάρουμε τις προβολές σε ένα συγκεκριμένο bit. Εφαρμόζοντας τη μέθοδο παίρνουμε $O(n^{\frac{1}{1+\epsilon}})$ χρόνο ερωτήματος και $O(n^{1+\frac{1}{1+\epsilon}})$ χώρο. Σε αυτή την εφαρμογή έχει πρόσφατα γίνει βελτίωση χρησιμοποιώντας μια data-driven προσέγγιση [1]. Στην αρχική μέθοδο, η επιλογή της συνάρτησης γίνεται με βάση την ομοιόμορφη κατανομή. Η βελτίωση έγγυται στον υπολογισμό μιας κατανομής πιθανοτήτων πάνω στις συναρτήσεις, λαμβάνοντας υπόψη το σύνολο σημείων, και μετά επιλογή της συνάρτησης από αυτή την κατανομή. Έτσι η νέα μέθοδος πετυχαίνει καλύτερα αποτελέσματα σε σημειοσύνολα που έχουν μορφή που ευνοεί καλύτερες διαμερίσεις.

## 0.3 Το Μοντέλο Μας (Our Model)

Η λύση που προτείνουμε για το πρόβλημα του κατά προσέγγιση κοντινού γείτονα (το συμβολίζουμε ως $(\epsilon, r)$-ΚΓ), είναι ουσιαστικά ένα δυαδικό δέντρο το οποίο μοιραζει το σημειοσύνολο σε κάθε εσωτερικό κόμβο, μέχρι το μέγεθος των υποσυνόλων να γίνει αρκετά μικρό ($O(1)$). Το μοίρασμα του σημειοσυνόλου γίνεται βάση διαχωριστών, οι οποίοι πληρούν κάποια κριτήρια για να διασφαλίσουν την απόδοση της δομής.

Η κατασκευή του δέντρου γίνεται αναδρομικά από τη ρίζα και σε κάθε κόμβο βρίσκουμε έναν κατάλληλο διαχωριστή ανεξάρτητα, δηλαδή δεν λαμβάνουμε υπόψη ποιοι ήταν οι διαχωριστές στα προηγούμενα επίπεδα, ή πώς μπορεί να μοιάζουν στα επόμενα. Θέλοντας να χτίσουμε ένα δέντρο το οποίο είναι κατάλληλο για το σημειοσύνολο εισόδου $P$, αλλά και για την κατανομή των ερωτημάτων $\mathcal{D}_Q$, η διαδικασία εύρεσης διαχωριστή βλέπει και το $P$ και το $\mathcal{D}_Q$. Αυτό φαίνεται και στον αλγόριθμο 1.

**Algorithm 1** Build Tree

**procedure** PREPROCESS(*pointset* $P$, *query distribution* $\mathcal{D}_Q$)
  N $\leftarrow$ **new** Node
  N.$P = P$
  **if** $|\text{N}.P| = O(1)$ **then**
    **return** N
  N.$separator \leftarrow findSeparator(\text{N}.P, \mathcal{D}_Q)$
  $P.in, P.out \leftarrow split(\text{N}.P, \text{N}.separator)$
  N.$lchild \leftarrow$PREPROCESS$(P.in, \mathcal{D}_Q)$
  N.$rchild \leftarrow$PREPROCESS$(P.out, \mathcal{D}_Q)$
  **return** N

Η αναζήτηση γείτονα κάποιου ερωτήματος ξεκινά από τη ρίζα του δέντρου και ανάλογα με τον διαχωριστή θα ακολουθεί ένα ή και τα δύο παιδιά. Αυτό γίνεται διότι θέλουμε να διασφαλίσουμε την πιθανότητα επιτυχίας να είναι ίση με 1, δηλαδή αν υπάρχει κοντινός γείτονας τότε σίγουρα να τον βρούμε. Τα ερωτήματα στα οποία η αναζήτηση συνεχίζει και στα δύο παιδιά, είναι αυτά που βρίσκονται αρκετά κοντά στον διαχωριστή και άρα ο κοντινός γείτονας μπορεί να βρίσκεται σπό την άλλη πλευρά. Για να το αντιμετωπίσουμε αυτό, ορίζουμε τους διαχωριστές ως διαφορά ομόκεντρων μπαλών με διαφορά ακτίνας $2r$ (ισοδύναμα λέμε δαχτυλίδια πάχους $2r$). Τότε, η αναζήτηση ακολουθεί κια τα δύο παιδιά αν το ερώτημα πέφτει πάνω στον διαχωριστή. Η διαδικασία φαίνεται και στον αλγόριθμο 2.

Οπότε, για να είναι αποδοτική η αναζήτηση θα πρέπει το ύψος του δέντρου και η πιθανότητα ένα ερώτημα να πέφτει πάνω στον διαχωριστή να ελαχιστοποιηθούν. Το ύψος του δέντρου θέλουμε να είναι μέχρι πολυλογαριθμικό (αλλιώς έχουμε χρόνους ίσους με τη σειριακή αναζήτηση). Αυτό πετυχαίνεται αν οι διαχωριστές μοιράζουν το σημειοσύνολο ισορροπημένα, δηλαδή κανένα από τα υποσύνολα δεν είναι εξαιρετικά μικρό (ή μεγάλο). Επιπλέον, η πιθανότητα ένα ερώτημα να πέφτει πάνω στον διαχωριστή ισούται με τη μάζα πιθανότητας της κατανομής πάνω στον διαχωριστή. Πρακτικά, αφού δεν έχουμε πρόσβαση στην κατανομή των

**Algorithm 2** Query Search

```
1: procedure QUERY(query point q, starting node N)
2:     if N is a leaf Node then
3:         p* ← null
4:         for each p ∈ N.P do
5:             if ‖p − q‖ ≤ (1 + ϵ)r then
6:                 p* ← p
7:                 break
8:         return p*
9:     ϱ(o, r_s) ← N.separator
10:    if ‖q − o‖ < r_s then
11:        return QUERY(q, N.lchild)
12:    else if ‖q − o‖ > r_s + 2r then
13:        return QUERY(q, N.rchild)
14:    else
15:        left ← QUERY(q, N.lchild)
16:        if left ≠ null then
17:            return left
18:        else
19:            return QUERY(q, N.rchild))
```

ερωτημάτων, δουλεύουμε με ένα δείγμα από αυτήν. Ο αριθμός των σημείων του δείγματος που πέφτουν πάνω στον διαχωριστή μας δίνει μια καλή αντίληψη για την πραγματική πιθανότητα και έτσι προσπαθούμε να τον ελάχιστοποιήσουμε.

Η εύρεση του καλύτερου διαχωριστή φαίνεται να είναι ένα δύσκολο πρόβλημα βελτιστοποίησης, αφού δεν καταφέραμε να εφαρμόσουμε κάποια από τις γνωστές μεθόδους (γραμμικό προγραμματισμό, gradient descent, local search, δυναμικό προγραμματισμό). Από τους θεωρητικά άπειρους πιθανούς διαχωριστές, εμείς λαμβάνουμε υπόψη μόνο αυτούς που ορίζονται από σημεία του σημειοσυνόλου $P$ και του δείγματος της κατανομής $S_Q$. Αποδεικνύουμε ότι ο καλύτερος από όλους αυτούς είναι αρκετά κοντά στον βέλτιστο. Ο αλγόριθμος 3 περιγράφει τη διαδικασία εύρεσης πλήρους αναζήτησης.

Δυστυχώς αυτή η μέθοδος δίνει μη ρεαλιστικούς χρόνους ($O(n)^{O(d)}$) και καθιστά αδύνατη ουσιαστικά την εφαρμογή του. Εναλλακτικά, προτείνουμε μια ευριστική μέθοδο (αλγόριθμος 4), βασισμένη στην μέθοδο local search. Ξεκινάμε αρχικοποιώντας τυχαία ένα διαχωριστή (πάλι λαμβάνοντας υπόψη μόνο αυτούς που ορίζονται από τα σημεία), και προσπαθούμε να βρούμε κάποιον καλύτερο μεταξύ αυτών που ορίζονται από τα ίδια σημεία εκτός ενός. Αυτούς τους λέμε γειτονικούς διαχωριστές. Αφού φτάσουμε σε ένα τοπικά βέλτιστο διαχωριστή, δηλαδή όλοι οι γειτονικοί έχουν χειρότερη μάζα ή ισορροπία, αρχικοποιούμε ξανά τυχαία τον διαχωριστή και ξαναβρίσκουμε άλλον τοπικά βέλτιστο. Επαναλαμβάνουμε τη διαδικασία $m$ φορές, όπου $m$ δική μας παράμετρος του αλγορίθμου. Μια καλή τιμή θα ήταν $m = \log n$.

**Algorithm 3** Find Separator - Brute Force

```
1: procedure SEPARATOR(pointset P, query sample S_Q)
2:     min_mass ← n
3:     for each subset A of P ∪ S_Q of size d + 1 do
4:         for each possible ring separator ϱ defined by A do
5:             mass ← |R(ϱ) ∩ S_Q|
6:             if mass < min_mass and ϱ is P − balanced then
7:                 min_mass ← mass
8:                 ans ← ϱ
9:     return ans
```

Για να υπολογίσουμε την πολυπλοκότητα αυτού του αλγορίθμου, βρίσκουμε πόσο χρόνο χρειάζεται ένα βήμα και πόσα βήματα μπορούν να γίνουν μετά από μια αρχικοποίηση. Αφού δοκιμάζουμε όλους τους γειτονικούς, σε κάθε βήμα δοκιμάζουμε $O(dn)$ διαχωριστές, όπου $d$ η διάσταση και $n$ ο αριθμός των σημείων στο $P$ (Κάθε ένα από τα $O(d)$ που ορίζουν ένα διαχωριστή, ανταλλάζεται με όλα τα υπόλοιπα). Επιπλέον, σε κάθε βήμα η μάζα θα μειώνεται το λιγότερο κατά 1, άρα φτάνουμε σε ελάχιστο το πολύ μετά από $O(n)$ βήματα. Τελικά η πολυπλοκότητα του αλγορίθμου είναι $O(mn^2d)$.

**Algorithm 4** Find Separator - Locality Heuristic

```
1: procedure SEPARATOR(pointset P, query sample S_Q)
2:     min_mass ← n
3:     for i in [m] do
4:         A ← random subset of P ∪ S_Q of size d + 1
5:         while true do
6:             temp_mass ← min_mass
7:             for each possible ring separator ϱ defined by neighborhood(A) do
8:                 mass ← |R(ϱ) ∩ S_Q|
9:                 if mass < temp_mass and ϱ is P − balanced then
10:                    temp_mass ← mass
11:                    t_ans ← ϱ
12:            if temp_mass < min_mass then
13:                min_mass ← temp_mass
14:                ans ← t_ans
15:            else
16:                break
17:    return ans
```

Η παραπάνω μέθοδος κατασκευάζει μια καθαρά data-driven δομή. Δυστυχώς, αν την αφήσουμε έτσι, τότε πιθανώς να υπάρχουν κακόβουλα ερωτήματα που να αναγκάζουν την αναζήτηση να κατέβει σε όλα τα φύλλα και άρα να καταλήξει να γίνεται σειριακή αναζήτηση.

Για να αντιμετωπίσουμε αυτό το φαινόμενο, εφαρμόζουμε την μέθοδο Bucketing όταν δεν μπορούμε πλέον να εξασφαλίσουμε την απόδοση της δομής μας.

Η ιδιότητα που θα μας χρειαστεί για να καταλάβουμε πότε πρέπει να αλλάζουμε σε Bucketing είναι η αραιότητα (sparsity) του σημειοσυνόλου. Πιο αυστηρά έχουμε:

**Ορισμός 0.1.** *Έστω $f$ η συνάρτηση μάζας πιθανότητας μιας κατανομής $\mathcal{D}$ στον $\mathbb{R}^d$. Λέμε ότι η $\mathcal{D}$ είναι $(\alpha, r)$-sparse αν για κάθε Ευκλείδια μπάλα $B$ ακτίνας $r$, ισχύει $\int_B f(x)\mathrm{d}x \leq \alpha$.*

Σε τέτοιες κατανομές αποδεικνύουμε την ύπαρξη διαχωριστών οι οποίοι ικανοποιούν συνθήκες ισορροπίας και αραιότητας. Συγκεκριμένα έχουμε:

**Λήμμα 0.2.** *Έστω $\mathcal{D}_Q$ ότι είναι $(\alpha, r)$-sparse κατανομή και έστω $f$ η σ.μ.π. Επίσης, έστω $P$ το σημειοσύνολο στον $\mathbb{R}^d$ το οποίο ακολουθεί την κατανομή $\mathcal{D}_P$, με $|P| = n$. Τέλος, έστω $c$ η σταθερά διπλασιασμού (doubling constant) του $\mathbb{R}^d$. Υπάρχει σημείο $p \in \mathbb{R}^d$ και ακτίνα $r_s$ που ορίζουν τον διαχωριστή $\varrho(p, r_s)$ τ.ω για το δαχτυλίδι $R = \{x \in \mathbb{R}^d \mid \|x-p\| \in [r_s, r_s + 2r]\}$ ισχύει ότι:*

1. $|\{x \in P \mid \|x - p\| \leq r_s\}| \geq \frac{n}{c+1}$ *και* $|\{x \in P \mid \|x - p\| \geq r_s + 2r\}| \geq \frac{n}{c+1}$
2. $\int_R f(x)\ \mathrm{d}x \leq 2\alpha^{O(1/d)}$

*Αυτός ο διαχωριστής λέμε ότι είναι $P - balanced - \mathcal{D}_Q - sparse$.*

Αν η δομή μας χρησιμοποιεί μόνο τέτοιους διαχωριστές, τότε αποδεικνύουμε το κεντρικό θεώρημα της παρούσας εργασίας που μας δίνει τις θεωρητικές εγγυήσεις της δομής. Ονομαστικά μπορούμε να πετύχουμε (με κατάλληλη επιλογή του $\alpha$) υπογραμμικό χρόνο αναζήτησης ερωτήματος, γραμμικό χώρο και σχεδόν τετραγωνικό χρόνο προεπεξεργασίας. Σημειώνεται επίσης, πώς η αραιότητα της κατανομής είναι ικανή συνθήκη για την ύπαρξη διαχωριστή και όχι αναγκαία, δηλαδή μπορεί να υπάρχουν διαχωριστές που να ικανοποιούν τις συνθήκες του λήμματος χωρίς να είναι αραιές.

**Θεώρημα 0.3.** *Ένα δέντρο που είναι χτισμένο για ένα σημειοσύνολο $P$ και μια $(\alpha, r)$-sparse κατανομή ερωτημάτων $\mathcal{D}_Q$, και χρησιμοποιεί $P-balanced-\mathcal{D}_Q-sparse$ διαχωριστές, είναι λύση για το πρόβλημα $(\epsilon, r)$-ΚΓ και παρέχει τις ακόλουθες θεωρητικές εγγυήσεις:*

1. *Ο αναμενόμενος χρόνος αναζήτησης ερωτήματος είναι $O(n\alpha^{O(1/d)} c \log n)$*
2. *Η χωρική πολυπλοκότητα είναι $O(n)$*
3. *Ο χρόνος προεπεξεργασίας είναι $O(c \log n) \times \mathcal{F}(\varrho)$, όπου $\mathcal{F}(\varrho)$ ο απαιτούμενος χρόνος εύρεσης $P - balanced - \mathcal{D}_Q - sparse$ διαχωριστή $\varrho$, όταν $\mathcal{F}(\varrho)$ είναι $\Omega(n^2)$.*

*Εδώ $c = 2^{O(d)}$ είναι η σταθερά διπλασιασμού του $\mathbb{R}^d$.*

Αντικαθιστώντας τον χρόνο του ευριστικού αλγορίθμου με $m = \log n$ παίρνουμε χρόνο επεξεργασίας ίσο με $O(n^2 dc \log^2 n)$.

Στην περίπτωση όπου δεν υπάρχει κατάλληλος διαχωριστής, τότε δεν μπορούμε να είμαστε σίγουροι για την απόδοση της δομής, οπότε κάνουμε τον κόμβο φύλλο και εφαρμόζουμε τη

μέθοδο *Bucketing*. Εναλλακτικά, θα μπορούσαμε να εφαρμόσουμε άλλη μέθοδο με εγγυήσεις, απλά η μέθοδος *Bucketing* φαίνεται να λειτουργεί καλύτερα με πυκνά σημειοσύνολα, που είναι η περίπτωση που αντμετωπίζουμε. Έτσι μπορούμε να διατηρήσουμε τις εγγυήσεις ενώ παράλληλα να εκμεταλλευτούμε την ωφέλιμη πληροφορία του σημειοσυνόλου και της κατανομής ερωτημάτων. Καταλαβαίνουμε πως αυτό δίνει στη δομή μας μια δϋική φύση, την data-driven και των θεωρητικών εγγυήσεων.

Η εργασία εστιάζει κυρίως στον Ευκλείδιο χώρο, με την απόσταση να είναι η $l_2$ νόρμα. Ο τρόπος που ορίζουμε τους διαχωριστές όμως μας δίνει τη δυνατότητα να επεκτείνουμε τη δομή και σε άλλους μετρικούς χώρους και μετρικές. Η μόνη ιδιότητα που θα πρέπει να ικανοποιείται είναι να έχουν σταθερά διπλασιασμού $c \leq 2^{O(d)}$.

Τέλος, αναφέρουμε ένα μάλλον απαισιόδοξο αποτέλεσμα, που έχει να κάνει με τη βελτιστότητα της δομής. Με τον όρο βέλτιστη δομή, εννοούμε τη δομή η οποία ελαχιστοποιεί τον χρόνο αναζήτησης ερωτήματος, μεταξύ όλων των πιθανών δομών που μπορούν να χτιστούν με τέτοιους διαχωριστές. Δυστυχώς, ο τρόπος που χτίζουμε τη δομή (θα μπορούσαμε να πούμε και άπληστα, αφού δεν μας ενδιαφέρει τι γίνεται στα παρακάτω επίπεδα) φαίνεται να μην μπορεί να μας δώσει κάποιο λόγο προσέγγισης. Αυτό μπορεί να γίνει αν η βέλτιστη επιλογή διαχωριστή σε ένα επίπεδο αναγκάζει την επιλογή κακών διαχωριστών στα επόμενα, χειροτερεύοντας κατά πολύ έτσι τη συνολική απόδοση.

## 0.4 Πείραμα (Experiment)

Για το πείραμα χρησιμοποιούμε το σύνολο δεδομένων MNIST, το οποίο εμφανίζεται συχνά στον τομέα της Τεχνητής Νοημοσύνης. Αποτελείται από 70,000 ασπρόμαυρες εικόνες (60,000 για εκπαίδευση και 10,000 για επαλήθευση) που απεικονίζουν χειρόγραφα ψηφία. Εμείς ασχολούμαστε με τις 60,000 της εκπαίδευσης.

Αρχικά πρέπει να διαμορφώσουμε το σύνολο εικόνων σε σύνολο σημείων $P$. Αναπαριστούμε την εικόνα ως διάνυσμα στον 784-διάστατο χώρο, όπου κάθε διάσταση αντιστοιχεί στην τιμή ενός pixel. Ακολούθως εφαρμόζουμε τον μετασχηματισμό Johnson-Lindenstrauss, για να μειώσουμε τις διαστάσεις σε 15. Επιπλέον, δημιουργούμε τα ερωτήματα παράγοντας 2 τυχαία σημεία που είναι κοντινοί γείτονες για κάθε σημείο του συνόλου $P$. Τέλος υποθέτουμε ότι η κατανομή των ερωτημάτων είναι η ίδια με αυτή του σημειοσυνόλου $P$, και άρα αυτό αποτελεί καλή αναπαράσταση της κατανομής.

Στη συνέχεια, χτίζουμε δέντρα με διαφορετικά μεγέθη σημειοσυνόλου (όλα δείγματα από το $P$), και συγκρίνουμε τα διάφορα χαρακτηριστικά τους. Συγκεκριμένα, τα μεγέθη που δοκιμάζουμε είναι 1,000, 3,000, 6,000, 10,000 και 20,000, και τα μεγέθη που συγκρίνουμε είναι ο χρόνος κατασκευής των δέντρων, ο μέσος χρόνος ερωτήματος για τα ερωτήματα που αντιστοιχούν στο δείγμα που χτίζεται το δέντρο, και ο μέσος χρόνος ερωτήματος όλων των ερωτημάτων αφού προσθέσουμε στη δομή και τα υπόλοιπα σημεία. Τέλος, συγκρίνουμε τα αποτελέσματά μας με αυτά της σειριακής αναζήτησης.

Τα αποτελέσματα αναγράφονται στον παρακάτω πίνακα.

| Sample Size | Preprocessing Time ($min$) | Sample Tree Query Time ($\mu s$) | Tree Query Time ($\mu s$) | Linear Search Query Time ($\mu s$) |
|---|---|---|---|---|
| 1,000 | 6 | 4.092 | 152.220 | 127.183 |
| 3,000 | 33 | 5.760 | 50.145 | 381.731 |
| 6,000 | 138 | 6.638 | 24.955 | 763.587 |
| 10,000 | 366 | 7.239 | 14.758 | 1272.08 |
| 20,000 | 1352 | 8.738 | 10.050 | 2543.72 |
| 30,000 | - | - | - | 3817.48 |
| 60,000 | - | - | - | 7706.68 |

Παρατηρούμε ότι, ο χρόνος κατασκευής αυξάνεται ταχύτερα από γραμμικά με το μέγεθος του δείγματος, όπως αναμένεται λόγω της πολυπλοκότητάς του. Συγκεκριμένα, παρατηρούμε ότι με μέγεθος δείγματος 1.000, η προεπεξεργασία διαρκεί μόνο λίγα λεπτά. Ωστόσο, όταν το μέγεθος του δείγματος αυξάνεται σε 20.000, ο χρόνος προεπεξεργασίας επεκτείνεται σε αρκετές ώρες, καθιστώντας το μη πρακτικό για μεγαλύτερα σύνολα δεδομένων.

Ο χρόνος αναζήτησης παρουσιάζει υπογραμμική αύξηση σε σχέση με το μέγεθος του συνόλου σημείων. Καθώς αυξάνεται το μέγεθος, αυξάνεται και το ύψος του δέντρου, οδηγώντας σε μεγαλύτερους χρόνους αναζήτησης. Αυτή η συμπεριφορά είναι αναμενόμενη, επειδή ένας μεγαλύτερος αριθμός σημείων οδηγεί σε αυξημένους χρόνους αναζήτησης. Ενσωματώνοντας τα υπόλοιπα σημεία στα δέντρα που κατασκευάστηκαν με διαφορετικά μεγέθη δείγματος, αξιολογούμε την απόδοσή τους σε ολόκληρο το σύνολο δεδομένων, παρατηρώντας ότι οι χρόνοι αναζήτησης μειώνονται ραγδαία με την αύξηση του μεγέθους του δείγματος, υποδεικνύοντας αποτελεσματική προσαρμογή του δέντρου πάνω στο σημειοσύνολο.

Ένα μέγεθος δείγματος γύρω στις 10.000 αναδεικνύεται ως το κατάλληλο για το χτίσιμο δέντρου. Πέρα από αυτό, ο χρόνος προεπεξεργασίας μεγαλώνει υπερβολικά, με οριακές μόνο βελτιώσεις στο χρόνο αναζήτησης ερωτημάτων. Ο διπλασιασμός του μεγέθους του δείγματος από τα 10.000 οδηγεί σε μια μέση μείωση του χρόνου αναζήτησης μόνο κατά 4 microseconds, ενώ ο χρόνος προεπεξεργασίας αυξάνεται κατά περισσότερο από 16 ώρες.

Τέλος, καθώς αυξάνεται το μέγεθος του δείγματος, οι μέσοι χρόνοι ερωτήματος για το πλήρες σύνολο ερωτημάτων και για τα ερωτήματα που αντιστοιχούν στο δείγμα, συγκλίνουν στην ίδια τιμή. Τα αποτελέσματά μας υποδηλώνουν ότι ο μέσος χρόνος ερωτήματος για ένα δέντρο που έχει κατασκευαστεί με βάση ολόκληρο το σύνολο κυμαίνεται μεταξύ 8,7 microseconds και 10,0 microseconds. Επομένως, ένα δέντρο που κατασκευάζεται σε 10.000 σημεία είναι αρκετό για να επιτύχει αποδοτικές επιδόσεις ερωτημάτων για ολόκληρο το σύνολο δεδομένων.

# Κείμενο στα Αγγλικά

# Chapter 1

# Introduction

The Nearest Neighbor Search (NNS) problem is basically the creation of a structure to store a point set $P$, allowing for efficient searches and reporting of the nearest point $p^* \in P$ when query points $q$ are later introduced. The problem is considered fundamental due to its prevalent applications in various fields, including data compression, databases, data mining, information retrieval, image and video databases, machine learning, and signal processing.

Finding the Exact Nearest Neighbor is challenging because it requires searching the entire point set in the worst-case scenario or using a data structure that consumes significant space. This approach is inefficient for many applications that demand faster query times or minimal storage space. The Approximate Nearest Neighbor approach allows for an approximation $c > 1$ in the distance, enabling sublinear query times relative to the size of the point set. It states that if the nearest point is at distance $r^*$, then we can report any point at distance at most $cr^*$.

Since the Approximate Nearest Neighbor problem is an optimization problem, many solutions focus on its decision version, known as the Approximate Near Neighbor problem, which states that if there is a point at distance at most $r$ from the query, then report any point at distance at most $cr$. This makes the Approximate Near Neighbor problem more tractable and amenable to efficient algorithms. Moreover, the reduction from the Approximate Nearest Neighbor problem to the Approximate Near Neighbor problem only incurs a polylogarithmic factor in the guarantees.

Many solutions have been proposed for the NNS problem, which has been studied across various settings using different techniques. Research includes work in fixed dimensions [2], low dimensions [3], high dimensions [4]–[6], and for growth-restricted metrics [7]. Techniques employed include product quantization [8], embeddings [9], hyperplane-based structures [10], and tree structures [11], [12]. Additionally, studies have explored the space-time tradeoffs for these techniques [13], [14]. A recent survey summarizing these advancements can be found here [15].

Generally, NNS solutions can be classified into two categories: those that come with theoretical guarantees and those that aim to construct the optimal structure for a given

point set. Algorithms in the first category include Locality-Sensitive Hashing ([16]–[18]) and its derivatives, methods based on randomized space partitions, Bucketing ([17]), which offers very fast query times but at the cost of increased space overhead, and Approximate Voronoi Diagrams ([17], [19]), which directly solve the Approximate Nearest Neighbor problem but are more sophisticated and complex to implement.

In the second category, we find methods such as PCA trees [20], which use Principal Component Analysis for partitioning the point set, graph algorithms ([21]–[25]), which build a graph on the dataset and perform graph exploration to reach the nearest neighbor, and ANNOY [26], which constructs multiple hierarchical 2-means trees and was used in Spotify's recommendation system. Although these methods are often more efficient in practice, they typically lack worst-case guarantees. Adapting the structure to a specific dataset can expose vulnerabilities to adversarial queries, resulting in suboptimal performance and potentially degrading to trivial linear search. To address this, we can establish guarantees while exploiting potential structural properties of the point set and query distribution. This is the objective of this thesis.

## 1.1 Theoretical Results

The model we propose is essentially a binary tree that recursively divides the point set into two disjoint subsets from the root downwards until the subsets are sufficiently small, i.e. of cardinality $O(1)$. Each division is made independently, without considering the divisions on other levels. As we want to optimize the data structure for the given point set $P$ and query distribution $\mathcal{D}_Q$, this division is not random, but derived from the input data.

Specifically each division is guided by a ring separator of width $2r$ (a set difference of concentric balls of radii that differ by $2r$). The point set is divided according to the sphere of same center and radius equal to the mean of the inner and outer radii of the separator; The points in the sphere constitute the inner subset, and the rest the outer. When queries arrive, the search begins at the root and traverses down to one or more leaves. This happens to ensure that the success probability is equal to 1. The queries that force the search to follow both children are those that fall on the separator, and thus the near neighbor can be in the opposite subset of the one that the search would follow.

The separators aim to minimize the probability of descending to both child nodes while maintaining balanced subsets, thus maintaining the height of the tree logarithmic. The structural property that our model focuses on to optimally adapt is mainly the sparsity of the query distribution. Specifically we consider distributions that are $(\alpha, r)$-sparse, which means that for every ball of radius $r$, the accumulated probability is at most $\alpha$. This is a sufficient condition for the existence of a separators that ensure good performance of our data structure.

The data-driven aspect of our data structure involves applying this method as long as the point set at a vertex remains sufficiently sparse. When we encounter a vertex

with a dense point set, we switch to an established solution, specifically a bucketing-based implementation. This approach enables us to extract useful information from the point set to enhance our structure without compromising our worst-case guarantees, ensuring that we only apply the method when it is beneficial.

Finding good separators that yield efficient query times in our structure is another optimization challenge we address. Due to the nature of the problem - characterized by infinite possible solutions, continuity of solutions, and non-convexity - no known efficient algorithmic paradigm seems applicable. To tackle this, we first propose a discretization of the problem's domain and prove that an exhaustive search on this discretized space yields efficient results. Specifically, we only consider separators that are defined by $d+1$ points of the point set $P$ and query sample $S_Q$ (which is the only representation of the query distribution that we can work with as we cannot access the distribution).

However, the computation time for this approach is prohibitively large $(O(n)^{O(d)})$, making it impractical. Therefore, we propose a heuristic algorithm inspired by the local search technique, which is significantly faster. Firstly, we randomly pick a separator out of all possible that are defined by points. Then we compare it with the "neighboring" separators, which are those defined by the same points except one, and update to the best out of those. If no "neighboring" separator is better, then we arrived in a local optimum. We repeat this process a number of times and keep the best out of all local optimums.

By combining all of the above we build a data-driven model that can achieve the following guarantees:

- The expected query time is $O(n^{1-1/d}c\log n)$.
- The space complexity is $O(n)$.
- The preprocessing time is $O(n^2dc\log^2 n)$.

Note that these results pertain to the purely data-driven model when the query distribution satisfies the sparsity property. The hybrid structure, which also incorporates bucketing, requires slightly more space.

## 1.2 Experimental Results

For our experiments, we use the MNIST dataset and perform some preprocessing to ensure it is suitable for our model. We then build instances of our tree, using different sample sizes to observe the model's adaptability to the dataset. Additionally, we compare our results with those obtained using the trivial linear search method.

We observe that preprocessing time increases faster than linearly with sample size, as expected due to its complexity. Specifically, we observe that with a sample size of 1,000, the preprocessing takes only a few minutes. However, when the sample size increases to 20,000, the preprocessing time extends to several hours, making it impractical for larger datasets.

The query time exhibits sublinear growth relative to the point set size. As the size increases, the height of the tree also increases, leading to longer query times. This behavior is expected because a larger number of points results in increased query times. By incorporating the remaining points into the trees constructed with different sample sizes, we evaluate their performance on the entire dataset, noting that query times decrease rapidly with increasing sample sizes, indicating efficient tree adaptation.

A sample size of around 10,000 emerges as the sweet spot. Beyond this point, preprocessing time became excessively large, with only marginal improvements in query time. Doubling the sample size from 10,000 results in a mean query time reduction of only 4 microseconds, while preprocessing time increases by over 16 hours.

Additionally, as sample size increases, the mean query time for the full query set and the sample queries converges to the same value. Our results suggest that the mean query time for a tree built on the entire dataset would likely fall between 8.7 microseconds and 10.0 microseconds. Therefore, a tree built on 10,000 points is sufficient to achieve efficient query performance for the whole dataset.

# Chapter 2

# Technical Background

In this chapter, we delve into some foundational concepts and methodologies that will be used in the research presented in this thesis. While understanding these technical elements is not strictly essential, it significantly aids in comprehending the analysis and, particularly, the rationale behind the experimental choices discussed in the subsequent chapters. This chapter provides a brief examination of three pivotal concepts: Ball Separators, Dimensionality Reduction, and Sampling Theorems. Each of these topics plays a significant role in the field of computatonal geometry, which is central to this research.

Firstly, we explore Ball Separators, a geometric tool essential for the partitioning of high-dimensional data sets. Ball separators ensure balanced partitioning and offer guarantees regarding the number of points that lie within them. Additionally, they can be applied to any metric space, enhancing their versatility and utility in various applications.

The second focus is on the Johnson-Lindenstrauss Lemma, a fundamental result in dimensionality reduction, along with some of its variations. This lemma asserts that it is possible to project a set of high-dimensional points into a much lower-dimensional space while preserving the distances between the points with high accuracy. This property is immensely valuable in processing and analyzing high-dimensional data, enabling more manageable and faster computations without significant loss of information.

Finally, we delve into Sampling Theorems, which are crucial for capturing continuous distributions from discrete data. These theorems provide guarantees about the amount of information retained from a distribution and the accuracy of its reconstruction. Additionally, they inform us of the number of samples required to preserve essential information about the distribution. Key results include the VC dimension and the theorems that utilize it.

In this research, ball separators will be applied to the data structure of our model to facilitate efficient and balanced partitioning of high-dimensional data sets. The results from dimensionality reduction, specifically those derived from the Johnson-Lindenstrauss Lemma, will be used to decrease the number of dimensions of the experiment data without significant loss of information. This reduction will make the data more manageable and enhance the performance of our computations. Additionally, sampling theorems will help us

reduce the time complexity of our algorithms, thereby accelerating their performance and making their implementation feasible for applications involving large datasets. Together, these techniques will play a crucial role in the design and execution of our experiments, ensuring robust and reliable results.

These three topics - Ball Separators, Dimensionality Reduction, and Sampling Theorems - collectively provide a technical background that supports the methodologies and analyses presented in this thesis. Each section will offer an exploration of the theoretical foundations, and relevance of these concepts to the broader context of this research. By understanding these technical components, readers will be better equipped to appreciate the techniques and complexities presented in the following chapters.

## 2.1 Ball Separators

Here we mainly present the work of Har-Peled in [27, Chapter 16], about the sphere separators on a set of balls $\mathcal{B}$ in $\mathbb{R}^d$, when no point of $\mathbb{R}^d$ belongs to more than $k$ of those balls. Formally, we say that $\mathcal{B}$ is *k-ply*.

**Definition 2.4.** *The **doubling constant** of a metric space is the smallest number of balls of the same radius needed to cover a ball of twice the radius (formally, we take the maximum such number over all possible balls to be covered). The doubling constant of* $\mathbb{R}^d$ *is* $c \leq 2^{O(d)}$

In our work, we will use the idea of the proof of the following theorem, to prove the existence of light-balanced separators that are used in our data-structure.

**Theorem 2.5** ([27])**.** *Let $\mathcal{B}$ be a set of $n$ balls that is $k$-ply in $\mathbb{R}^d$. Then, there exists a sphere $\mathbb{S}^{(d)}$ that intersects $4k^{1/d}n^{1-1/d}$ balls of $\mathcal{B}$. Furthermore, the number of balls of $\mathcal{B}$ that are completely inside (resp. outside) $\mathbb{S}^{(d)}$ is $\geq n/(c+1)$.*

*Proof.* Let $P$ be the set of centers of the balls in $\mathcal{B}$, and let $b$ be the smallest ball that contains $n/(1+c)$ points of $P$. We assume w.l.o.g. that this ball is centered at the origin and has radius 1. In addition, let $\mathbb{S}^{(d)}$ be a sphere with center the origin as well, and random radius $x$ picked uniformly from the range $[1, 2]$.

By the definition of the doubling constant, $\mathbb{S}^{(d)}$ will contain at most $cn/(1+c)$ points from $P$, and thus there will be at least $1 - cn/(1+c) = n/(c+1)$ points op $P$ outside $\mathbb{S}^{(d)}$. This concludes the first part of the proof.

The second part focuses on the expected number of balls intersecting $\mathbb{S}^{(d)}$. Let $v_d r^d$ be the volume of a ball of radius $r$ in $\mathbb{R}^d$, where $v_d$ is a constant that depends on the dimension. We then clip the balls of $\mathcal{B}$ to the ball centered at the origin and has radius 2, and then replace these clipped balls (formally lens) with a ball of the same volume. Let $r_i$ be the radius of the $i$th such ball $f_i$, for $i = 1, \ldots, n$. By the k-ply property, we have that:

$$\sum_i r_i^d = \frac{1}{v_d}\left(\sum_i v_d r_i^d\right) \leq \frac{1}{v_d}(v_d 2^d) \leq k2^d,$$

where $(v_d 2^d)$ is the volume of a ball of radius 2 in $\mathbb{R}^d$.

The probability of the the $i$th ball to intersect $\mathbb{S}^{(d)}$ is bounded by the probability that the radius of $\mathbb{S}^{(d)}$ lies in $[\|p_i\| - r_i, \|p_i\| + r_i]$, where $p_i$ is the center of the $i$th ball. This is equal to $2r_i/(2-1) = 2r_i$ because the radius of $\mathbb{S}^{(d)}$ is picked uniformly from $[1, 2]$.

Let $S$ be the set of balls of $\mathcal{B}$ that intersects $\mathbb{S}^{(d)}$. We have, by Hölder's inequality, that:

$$\begin{aligned}\mathbb{E}\left[|S|\right] &= \sum_i P\left[f_i \cap \mathbb{S}^{(d)} \neq \emptyset\right] \leq \sum_i 2r_i = 2\sum_i 1 \cdot r_i \leq 2\left(\sum_{i=1}^{n} 1^{d/(d-1)}\right)^{(d-1)/d}\left(\sum_{i=1}^{n} r_i^d\right)^{1/d} \\ &\leq 2n^{1-1/d}\left(k2^d\right)^{1/d} \leq 4n^{1-1/d}k^{1/d},\end{aligned} \tag{2.1}$$

which concludes the second part of the proof.

□

We will adjust these separators for points and incorporate them in our structure. Also, we will try to minimize the number of points that fall on the separator, while maintaining the balance of it. There will be a detailed analysis in Chapter 4 - Our Model.

## 2.2 Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of dimensions of the space under consideration, which simplifies the computational processes involved and often leads to more efficient algorithms. This reduction is crucial in many applications, such as machine learning, data visualization and signal processing, where high-dimensional data can be cumbersome and computationally expensive to work with. By mapping data to a lower-dimensional space, we can achieve faster computations, reduce storage requirements and often even enhance the performance of certain algorithms by mitigating the curse of dimensionality.

In this discussion, we will primarily focus on the Johnson-Lindenstrauss Lemma [28], [29], a cornerstone result in the field of dimensionality reduction. The Johnson-Lindenstrauss Lemma provides a powerful guarantee that points in high-dimensional space can be embedded into a lower-dimensional space while preserving the pairwise distances between the points within a small margin of error. This lemma has profound implications for the efficiency of algorithms in various domains, including machine learning, optimization and numerical linear algebra.

Additionally, we will briefly explore the contributions of Achlioptas [30] and Chazelle [31]. Achlioptas is known for his work on randomized algorithms and their applications to dimensionality reduction. He introduced simpler and more computationally efficient methods for achieving the Johnson-Lindenstrauss guarantees, making these techniques more accessible for practical applications. Chazelle's work has also significantly impacted

the field, particularly through his $Fast-Johnson-Lindenstrauss-Transform$ (FJLT), which makes use of an idea similar to Heisenberg's uncertainty principle.

By examining these foundational results and contributions, we will gain a deeper understanding of the principles and applications of dimensionality reduction, demonstrating its crucial role and effectiveness in our experiment.

### 2.2.1 Johnson-Lindenstrauss Lemma

**Lemma 2.6** (JL lemma)**.** *For any $\epsilon \in (0,1)$ and any $X \subset \mathbb{R}^d$ for $|X| = n$ finite, there exists an embedding $f : X \to \mathbb{R}^m$ for $m = O\left(\epsilon^{-2} \log n\right)$ such that:*

$$\forall x, y \in X,\ (1-\epsilon)\|x-y\|_2^2 \leq \|f(x)-f(y)\|_2^2 \leq (1+\epsilon)\|x-y\|_2^2.$$

A typical application of the Johnson-Lindenstrauss (JL) lemma is in creating approximate algorithms for high-dimensional computational geometry problems. The concept is that, given an input $X$ consisting of a set of high-dimensional vectors, we can solve the computational problem on $f(X)$, where $f$ is an embedding as defined by the JL lemma. Because $f(X)$ resides in a lower-dimensional space, the algorithm is expected to run faster. Significantly, when the time complexity of an algorithm is exponential in the dimension $d$, then the complexity is improved to linear in the number of samples $n$.

Proofs of the JL lemma first prove the following "Distributional Johnson-Lindenstrauss lemma":

**Lemma 2.7** (DJL lemma)**.** *For any $\epsilon, \delta \in (0, 1/2)$ and integer $d > 1$, there exists a distribution $\mathcal{D}_{\epsilon,\delta}$ over matrices $\Pi \in \mathbb{R}^{m \times d}$ for $m = O(\epsilon^{-2} \log(1/\delta))$ such that for any fixed $z \in \mathbb{R}^d$ with $\|z\|_2 = 1$,*

$$\mathop{\mathbb{P}}_{\Pi \sim \mathcal{D}_{\epsilon,\delta}} \left( \left| \|\Pi z\|_2^2 - 1 \right| > \epsilon \right) < \delta.$$

The Johnson-Lindenstrauss (JL) lemma follows as a corollary from the Distributional Johnson-Lindenstrauss (DJL) lemma for the following reason: by setting $\delta < 1/n^2$ and choosing a random $\Pi$ as in the DJL lemma, we can consider any $x \neq y \in X$ and define $z_{x,y} := (x-y)/\|x-y\|_2$. The DJL lemma implies that $\mathbb{P}\left( \left| \|\Pi z_{x,y}\|_2^2 - 1 \right| > \epsilon \right) < \delta$, which is equivalent to $\mathbb{P}\left( \left| \|\Pi(x-y)\|_2^2 - \|x-y\|_2^2 \right| > \epsilon \|x-y\|_2^2 \right) < \delta$. By applying the union bound, the probability that there exists some $x \neq y \in X$ such that $\|\Pi(x-y)\|_2^2 \notin \left[ (1-\epsilon)\|x-y\|_2^2, (1+\epsilon)\|x-y\|_2^2 \right]$ is at most $\binom{n}{2}\delta < 1$. Thus there exists a $\Pi^*$ such that $\|\Pi^*(x-y)\|_2^2 \in \left[ (1-\epsilon)\|x-y\|_2^2, (1+\epsilon)\|x-y\|_2^2 \right]$ for all $x \neq y \in X$. We define $f(x) = \Pi^* x$.

Now that we understand how the lemma operates, we need to determine how to construct a matrix $\Pi$ that satisfies the conditions of the lemma.

The originally used [32] $\Pi$ is an orthogonal projection onto a random m-dimensional subspace of $\mathbb{R}^d$. Thus we would like to pick a random basis of $m$ orthonormal vectors. We can accomplish this by picking a gaussian vector $g_1 \sim N(0, I_d)$ then letting the first row of $\Pi$ be $r_1 := g_1/\|g_1\|_2$. For $r_2$, we pick another $g_2 \sim N(0, I_d)$ independently and define $g_2' = g_2 - \langle g_2, g_1 \rangle g_1$ then $r_2 := g_2'/\|g_2'\|_2$. That is, we first subtract its projection onto $g_1$

before normalizing to form $g_2$. We then continue doing this for the rest of the rows (the "Gram-Schmidt process"). In the end, $\Pi$ is a dense, unstructured matrix, so computing $\Pi z$ takes $O(md)$ time.

This method has some drawbacks: it requires a significant amount of time for the calculations. More critically, it demands sufficient memory to preserve the accuracy of the floating-point entries. An alternative approach proposed by Achlioptas [30], involves filling the entries independently with $-1$ or 1, each with a probability of $1/2$. This approach allows us to store $\Pi$ in exactly $md$ bits and enables much more efficient computation.

For our experiments, we will use the method that fills the matrix $\Pi$ with random and independent entries of $\{-1, +1\}$. Using this method the computation of $f(z) = \Pi z$ takes $O(md)$ time. For completeness we will briefly present two techniques to accelerate the JL transforms.

### 2.2.2 Faster JL Transforms

While a matrix filled with $\{-1, +1\}$ entries is straightforward to compute and suitable for one-time applications, it sometimes falls short in terms of efficiency. Consequently, faster methods are often needed. Achlioptas [30] also observed that such a matrix is dense and proposed a technique to make it sparse, thereby enhancing its efficiency.

Specifically, Achlioptas suggested filling each entry of the matrix with 0 with a probability of $2/3$, with $\pm 1$ with a probability of $1/6$ each, and then normalizing it by multiplying it with $1/\sqrt{m/3}$. This approach makes the matrix two-thirds sparser. By reducing the number of non-zero entries, the computational cost of matrix-vector multiplications decreases significantly, leading to faster performance while still maintaining the desired properties for the JL transform.

In addition to improving computation time, this sparse matrix approach reduces memory usage and can be more efficient for repeated applications, such as in real-time data processing or large-scale machine learning tasks. This method strikes a balance between simplicity and performance, making it a valuable enhancement for the JL transform.

Another significant improvement was the $Fast{-}Johnson{-}Lindenstrauss{-}Transform$ (FJTL) by Ailon and Chazelle [31]. Here the matrix is obtained as a product of three matrices $\Pi = PFD$. The matrices $P$ and $D$ are random and $F$ is deterministic:

- **P** is a $m \times d$ sampling matrix with replacement (each row has a 1 in a uniformly random location and zeroes elsewhere, and the rows are independent).
- **F** is a $d \times d$ normalized discrete Fourier transform (and can be computed fast using the FFT algorithm). In fact, the matrix only needs to be a normalized Hadamard matrix. We utilize a Discrete Fourier Transform (DFT) matrix since it functions as a Hadamard matrix and will facilitate our reasoning later.
- **D** is a $d \times d$ diagonal matrix, where each entry (in the diagonal) is drawn independently from $\{-1, +1\}$ with probability $1/2$.

We will not delve into the proof of the FJLT, but we will provide some intuition for its construction. To begin with, the matrix $P$ is sparse, meaning that, on average, only a small fraction of its elements are non-zero. However, $P$ alone cannot serve as an effective JL transform because the variance of the estimator $\|Px\|_2$ is too high for sparse inputs $x$ (consider a vector $x$ with only one non-zero coordinate). This is where the mapping $FD$ comes into play. The transformation $FD$ ensures that, with overwhelming probability, the vector $FDx$ is smoothed out. Consequently, when applying $P$ to $FDx$, the resulting vector $PFDx$ exhibits good concentration properties.

To conclude, the key idea behind this transform is to achieve smooth input vectors. When the input vector is not smooth, the Fourier transform tends to spread out with high probability. This spreading effect is used to smooth the sparse input vector. This concept (that a vector and its Fourier transform cannot both be sparse simultaneously) is analogous to Heisenberg's uncertainty principle. This principle ensures that the Fast Johnson-Lindenstrauss Transform (FJLT) remains an efficient and reliable method for dimensionality reduction.

## 2.3 VC Dimension and Sampling

The sheer volume of data and the massive size of some datasets often make it impractical for algorithms to run efficiently on the entire dataset. As a result, we frequently rely on smaller samples to approximate and generalize the results for the entire dataset, whether we are developing algorithms, data structures or other models. However, determining the optimal sample size for this purpose is crucial. This is where the concept of VC dimension, or Vapnik-Chervonenkis dimension, comes into play. VC dimension provides a measure of the capacity and complexity of a statistical model, offering insights into its ability to capture and describe the underlying patterns in the data. By understanding the VC dimension, we can make more informed decisions about sample sizes and improve the generalization performance of our models. The definitions presented in this section are taken from [33], if not cited otherwise.

There are two primary definitions of VC dimension. The first, from a geometric perspective, describes the capturing capability of a set family $\mathcal{S}$. The second, from a machine learning perspective, explains the capturing capability of a classification model.

In this discussion, we will focus on the geometric definition. The VC dimension, in this context, measures the largest set of points $T$ that can be shattered by the set family $\mathcal{S}$. A set of points is said to be shattered if, for every possible subset of points $T'$, there exists a subset in S which when intersected with $T$ equals $T'$. Formally:

**Definition 2.8** (VC Dimension)**.** *A set $T$ is shattered by a set family $\mathcal{S}$ which consists of a collection of subsets $S_1, S_2, \ldots$ of $[n]$, if for every $T' \subset T$ there is some $S_i \in \mathcal{S}$ such that $T \cap S_i = T'$. The VC-dimension of $\mathcal{S}$ is the cardinality of the largest set $T$ shattered by $\mathcal{S}$.*

Here, we will also present an important lemma regarding the impact on the VC di-

mension when performing set operations on two family sets $\mathcal{S}_1$ and $\mathcal{S}_2$, taken from [27, Chapter 6] and adjusted to go along with our current notation.

**Lemma 2.9** (Union and Intersection of family sets)**.** *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two family sets with VC dimension $d_1$ and $d_2$ respectively, where $d_1, d_2 > 1$. Let $\mathcal{S}_1' = \{S_1 \cup S_2 \,|\, S_1 \in \mathcal{S}_1,\, S_2 \in \mathcal{S}_2\}$ and $\mathcal{S}_2' = \{S_1 \cap S_2 \,|\, S_1 \in \mathcal{S}_1,\, S_2 \in \mathcal{S}_2\}$. Both family sets $\mathcal{S}_1'$ and $\mathcal{S}_2'$ have VC Dimension equal to $O(d_1 + d_2)$.*

Next we present the concept of $\epsilon$-samples:

**Definition 2.10** ($\epsilon$-sample)**.** *A set $W$ is an $\epsilon$-sample with respect to a collection $\mathcal{S}$ of subsets $S_1, S_2, \ldots$ of $[n]$, if for every set $S_i \in \mathcal{S}$,*

$$\left(\frac{|S_i|}{n} - \epsilon\right)|W| \leq |W \cap S_i| \leq \left(\frac{S_i}{n} + \epsilon\right)|W|$$

The above definition indicates that an $\epsilon$-sample includes points from every subset $S_i$, ensuring that the proportion of these points in the sample is close to the proportion of the size of $S_i$ to the total number of points $n$. The following lemma gives us a guarantee for the size of an $\epsilon$-sample:

**Lemma 2.11** ([33])**.** *For some universal $c$, for every set system $\mathcal{S}$ over $[n]$ of VC dimension $d$, a random set $W \subset [n]$ of size*

$$\frac{c}{\epsilon^2} d \log \frac{1}{\epsilon} + \log \frac{1}{\delta}$$

*has probability at least $1 - \delta$ of being an $\epsilon$-sample for $\mathcal{S}$.*

Feige and Mahdian provide a proof sketch for the lemma, which we omit here. The core idea of their proof involves selecting a random sample of size $t$. By assuming that this sample is not an $\epsilon$-sample, they derive an inequality involving $t$. Therefore, $t$ must satisfy this inequality, leading to the specific size value stated in the lemma.

Another application of known VC bounds [34] is to constrain the deviation of our sample estimation from the true distribution.

**Lemma 2.12** (VC bounds)**.** *For any subset $S$, and an i.i.d. drawn sample $W$, we have that with probability at least $1 - \delta$,*

$$\left|\frac{|S \cap W|}{|W|} - \frac{|S|}{n}\right| \leq \sqrt{\frac{\ln(1/\delta)}{|W|}}.$$

Thus from the last lemma we can now bound the error $\epsilon$, by $\epsilon \leq \sqrt{\frac{\ln(1/\delta)}{|W|}}$, where $\delta$ is the probability of failure, and $W$ the sample.

All the aforementioned concepts will be particularly useful later, especially during the experiment, where we will need tools to make various decisions (such as determining the sample size) and to quantify the total error that may result.

# Chapter 3

# Previous Work

The Approximate Nearest Neighbor (ANN) problem is fundamental due to its prevalent applications in various fields, including data compression, databases, data mining, information retrieval, image and video databases, machine learning and signal processing. In this chapter, we present a reduction from the approximate nearest neighbor problem to the approximate near neighbor problem and then explore several foundational solutions to it. Specifically, we will examine Locality-Sensitive Hashing (LSH) and its derivatives, as well as Bucketing and Voronoi Diagrams.

Moreover, we will discuss a recent advancement by Andoni and Beaglehole [1], who optimized the LSH solution for the Hamming metric using a data-driven approach. In this thesis, we aim to achieve a similar optimization, albeit using different methods, and extend the applicability to any metric space.

Formally, the Approximate Nearest Neighbor and the Approximate Near Neighbor problems are defined below:

**Definition 3.13.** *Let $P$ be a set of points in metric space (X,D). The $\epsilon$-approximate nearest neighbor problem (or $\epsilon$-NN) is to construct a data structure that given any fixed query point $q \in X$, it reports a point $p \in P$ for which is true that $\forall\, p' \in P\ D(q,p) \leq (1+\epsilon) \cdot D(q,p')$.*

**Definition 3.14.** *Let $P$ be a set of points in metric space (X,D). The ($\epsilon$,r)-approximate near neighbor problem (or ($\epsilon$,r)-NN) is to construct a data structure that given any fixed query point $q \in X$, if $\exists\, p \in P$ for which $D(q,p) \leq r$ then it reports a point $p' \in P$ for which is true that $D(q,p') \leq (1+\epsilon) \cdot r$.*

The methods we will analyze in this Chapter almost all address the Approximate Near Neighbor problem. This focus is primarily because the Approximate Near Neighbor problem is fundamentally a decision problem, which simplifies certain aspects of its solution. This makes the Approximate Near Neighbor problem more tractable and amenable to efficient algorithms. Moreover, the reduction from the Approximate Nearest Neighbor problem to the Approximate Near Neighbor problem only incurs a polylogarithmic factor in the guarantees. Consequently, this justifies the extensive research and effort dedicated

to developing robust solutions for the Approximate Near Neighbor problem, as it provides a foundational approach that can be effectively extended to address the Approximate Nearest Neighbor problem with minimal additional computational cost.

Another reason why solving the Approximate Near Neighbor problem is often sufficient is that, in many practical scenarios, finding the exact nearest neighbor is not necessary. Instead, it is acceptable to find a neighbor that is sufficiently close. For example, in recommendation systems where items (such as movies) are embedded in a metric space, the similarity between two items is indicated by their distance (closer items are more similar than those further apart). In such systems, it is not imperative to recommend the exact nearest neighbor to a query (such as a previously watched movie); rather, it is adequate to recommend an item that is near enough to be considered similar. This approach simplifies the problem and still meets the practical requirements of the application, making the Approximate Near Neighbor problem a valuable and efficient alternative.

## 3.1 From the Approximate Nearest to Near Neighbor

By reducing the Approximate Nearest Neighbor problem to the Approximate Near Neighbor problem, we transform an optimization problem into its decision version, which is typically much easier to address directly. In this section, we will present a simple reduction from the $\epsilon$-NN problem to the $(\epsilon, r)$-NN problem to demonstrate the main idea. However, this basic reduction has some limitations. Therefore, we will also provide insights into a more robust reduction that addresses the issues encountered by the simpler approach.

Let $R$ be the ratio of the smallest and the largest inter-point distances in the point set $P$. For each $r \in \{(1+\epsilon)^0, (1+\epsilon)^1, (1+\epsilon)^2, \ldots, R\}$, we build structures for the $(\epsilon, r)$-NN problem. Upon the arrival of any query point $q$, we binary search on the structures for the minimal $r$, for which the structure returns a near neighbor $p_i$, which we report as an approximate nearest neighbor. This reduction yields a query time overhead factor $O(\log \log R)$ and a space overhead factor $O(\log R)$.

The simplicity of this reduction is very useful in practice. However, the $O(\log R)$ overhead factor can become problematic for point sets with a large, or even unbounded $R$. Next, we describe some ideas used in the exact reduction, which while it addresses this weakness, it is more sophisticated and complex.

We partition the point set $P$ into components $P_1, P_2, \ldots, P_k$ based on a radius $r$, ensuring that any point within a component $P_i$ is at most distance $r$ from another point in the same component $P_i$. Then, we create a subset $P'$ by picking a representative $p'_i$ from each component. Subsequently, we recursively repeat this procedure for the subsets $P_1, P_2, \ldots, P_k, P'$, until their size is small enough $O(1)$. For this technique to be optimal, we must set the radius $r$, to a value such that the largest component of $P$, will have at least $n/2 + 1$ points (more than half).

Given a query $q$, we estimate the distance to the nearest neighbor. If the estimated distance is sufficiently small, we recursively search within the nearest component $P_i$. If

the estimated distance is large, we recursively search within the representative subset $P'$. Otherwise, we report the point $p'$ identified by our structure during the estimation process.

In the end, the exact reduction achieves a query time overhead factor $O(\log n)$ and a space overhead factor $O(\frac{1}{\gamma} \cdot \log^2 n)$, where $\gamma \in (1/n, 1)$ is a prescribed parameter. Overall, this reduction is effective for any input point set $P$, without relying on any favorable properties it may possess.

## 3.2 Locality-Sensitive Hashing

Locality-Sensitive Hashing ($LSH$) operates on the principle that data and query points in close proximity are more likely to be assigned the same hash, thereby increasing the likelihood of collisions, compared to points that are farther apart. Formally, we require the following.

**Definition 3.15** (locality-sensitive family)**.** *A family $\mathcal{H} = \{h : X \to U\}$ is $(r_1, r_2, p_1, p_2)$-sensitive for $(X, D)$ if for any $q, p \in X$ we have*

- *if $D(p, q) \leq r_1$ then $\mathbb{P}_{\mathcal{H}}[h(q) = h(p)] \geq p_1$,*
- *if $D(p, q) > r_2$ then $\mathbb{P}_{\mathcal{H}}[h(q) = h(p)] \leq p_1$.*

In order to be useful, a locality-sensitive family must satisfy the inequalities $p_1 > p_2$ and $r_1 < r_2$.

Now, by using a locality-sensitive family, we can solve the $(\epsilon, r)$-NN problem as follows: Firstly we set $r_1 = r$ and $r_2 = (1 + \epsilon) \cdot r$. We then amplify the gap between the probabilities $p_1$ and $p_2$ by concatenating several functions from $\mathcal{H}$. Particularly, we define the function family $\mathcal{G} = \{g : X \to U^k\}$, where k is to be specified later, such that $g(p) = (h_{i_1}(p), \ldots, h_{i_k}(p))$, where $h_{i_t} \in \mathcal{H}$, for $I = \{i_1, \ldots, i_k\} \subset \{1, \ldots, |\mathcal{H}|\}$, and repetitions are allowed. For an integer $L$ we choose $L$ functions $g_1, \ldots, g_L$ from $\mathcal{G}$ independently and uniformly at random and let $I_j$ denote the multi-set defining $g_j$. During preproccessing, we store a pointer to each $p \in P$ in the buckets $g_1(p), \ldots, g_L(p)$. Since the total number of buckets may be large ($O(L2^k)$), we maintain only the ones that contain points by applying "standard" hashing.

When a query point $q$ arrives, the procedure we follow is carrying a brute-force search for a neighbor of $q$ in buckets $g_1(q), \ldots, g_L(q)$. After inspecting the first $3L$ points (including duplicates), we terminate the procedure as the total number of points in those buckets may be large and consequently cost linear time to search them all. If there is any point $p$ such that $D(p, q) \leq r_2$ we return it, else we return null.

Before demonstrating the correctness of the procedure, it is important to first provide some intuition behind the parameters $k$ and $L$. As mentioned before, $k$ is the number of concatenated hash functions $h$ which point out the bucket in one of the L instances of the structure. We understand that when $k$ is low, then each of the buckets correspond to a larger area of the space, thus a larger number of points will have the same concatenated

hash function $g$. This may lead to cases where significantly far points end up in the same bucket. On the other hand, as $k$ increases, not only the points farther apart but also near points may end up in different buckets, because the probability of having none of the hash functions $h_i$ different decreases. Here is where $L$ is important. Having multiple instances, with different $g_j$ functions, decreases the probability that this happens, as it is fairly unlikely for 2 near points to end up in different buckets *many* times.

**Definition 3.16** (Correctness)**.** *We choose $k$ and $L$ ensuring that with constant probability the following two events hold. For any $p^*$ we define:*

- *$E_1(q, p^*)$ occurs iff either $p^* \notin B(q, r)$ or $g_j(p^*) = g_j(q)$ for some $j = 1, \ldots, L$.*
- *$E_2(q)$ occurs iff the total number of collisions of $q$ with points from $P - B(q, r_2)$ is less than $3L$:*

$$\sum_{j=1}^{L} |(P - B(q, r_2)) \cap g_j^{-1}(g_j(q))| < 3L.$$

Lets explain what these 2 events are. The first one refers to the case where $q$ has no near neighbors (and thus the structure does not have to return anything) or it collides with $p^*$ under $g_j$ ($g_j(p^*) = g_j(q)$)for some $j = 1, \ldots, L$. We could say that $p^*$ in this case is either *true negative* or *true positive* respectively. The second event refers to the case where the number of points in the same buckets as $q$ ($g_j^{-1}(g_j(q))$) which are not approximate near neighbors ($P - B(q, r_2)$) is under $3L$, in order for a true near neighbor to be returned. We could call these points *false positive*.

We now present the main theorem:

**Theorem 3.17** ([17])**.** *Suppose there is a $(r, cr, p_1, p_2)$-sensitive family $\mathcal{H}$ for $(X, D)$, where $p_1, p_2 \in (0, 1)$ and let $\rho = \log(1/p_1)/\log(1/p_2)$. Then there exists a fully dynamic data structure for $(c - 1, r)$-NN over a set $P \subset X$ of at most $n$ points, such that:*

- *The query time complexity is $O(n^\rho/p_1) \cdot \lceil \log_{1/p_2} n \rceil$; $O(n^\rho/p_1)$ distance computations and evaluations of hash function from $\mathcal{H}$, each taking $\lceil \log_{1/p_2} n \rceil$.*
- *The space complexity is $O(n^{1+\rho}/p_1)$.*
- *The failure probability $f$ has the upper bound: $f < 1/3 + 1/e$.*

We do not provide the full proof, only a proof sketch and include the result values of $k$ and $L$, that the structure must have to satisfy the theorem. We only consider the case where there exists $p^* \in B(q, r_1)$, for the other we have nothing to prove. We have to prove that the events $E_1(q, p^*)$ and $E_2(q)$ hold with probabilities strictly greater than half.

Firstly we set $k = \lceil \log_{1/p_2} n \rceil$, so that the expected number of *false positives* under fixed $g_j$ is at most 1. Then the expected number of *false positives* under any $g_j$ is at most $L$. By Markov's inequality, as we try $3L$ collision points, the probability of $E_2(q)$ is at least $2/3$. For $E_1(q, p^*)$ we set $L = n^\rho/p_1$ in order to bound from below the probability that a $g_j$ exists so that $g_j(p^*) = g_j(q)$. This value of $L$ gives a probability of at least $1 - 1/e$. Lastly, by a Union bound the theorem follows.

### 3.2.1 Hamming metric

In this subsection we present the application of the LSH Theorem on the Hamming Space with binary alphabet, also called the hypercube. The Hamming Space of dimension $d$ is defined as the set of all $2^d$ binary strings of length $d$ (from now on referred to as points), and the Hamming metric distance of two such points $p_1 = (b_1^1, b_1^2, \ldots, b_1^q)$ and $p_2 = (b_2^1, b_2^2, \ldots, b_2^d)$, is the number of bits that they differ $D(p_1, p_2) = |\{i : b_1^i \neq b_2^i\}|$.

**Proposition 3.18.** *Let $D(p, q)$ be the Hamming metric for $p, q \in \{0, 1\}^d$. Then for any $r, \epsilon > 0$, the family*

$$\mathcal{H} = \{h_i : h_i((b^1, b^2, \ldots, b^d)) = b^i,\ i = 1, \ldots, d\}$$

*is $(r, rc, 1 - r/d, 1 - rc/d)$-sensitive, where $c = 1 + \epsilon$.*

We can get a direct corollary from the theorem by applying the proposition

**Corollary 3.19.** *For any $\epsilon > 0$, there exists a fully dynamic data structure for $(\epsilon, r)$-NN in Hamming metric over $\{0, 1\}^d$, such that:*

- *The query time complexity is $O(n^{1/(1+\epsilon)} \cdot d/r \cdot \log n)$; $O(n^{1/(1+\epsilon)})$ distance computations and evaluations of hash functions from $\mathcal{H}$, each taking $O(d/r \cdot \log n)$.*
- *The space complexity is $O(n^{1+1/(1+\epsilon)})$.*

We will further examine the LSH technique on the Hamming metric in section 5 of this chapter, focusing on the work of Andoni in optimizing it through a data-driven approach.

### 3.2.2 LSH Derivatives

We have analyzed the fundamental approach of Locality-Sensitive Hashing (LSH). However, LSH has numerous variations, as it accepts adjustments, can be integrated with various techniques, and applied to a wide range of problems. Here, we highlight some of these variants.

Several studies have explored the theoretical aspects of LSH. Lower bounds for LSH in $l_1$ space are established by [35], [36], and for Hamming and Euclidean spaces by [37], with optimizations presented in [38]. The use of LSH for angular distance is discussed in [39], while [6] explores its application in high-dimensional spaces. LSH is also employed based on p-stable distributions using $l_p$ norms as shown in [40]. Additionally, [14] discusses time-space tradeoffs for LSH, and [41] introduces a new data structure with lower bounds that bypass those of traditional LSH.

Various adaptive and specialized methods have also been developed. PUFFIN [42] utilizes an adaptive approach based on an LSH forest trie data structure [43]. PM-LSH [44] focuses on $L_p$ norms and incorporates PM-trees [45], with a query procedure similar to the SRS approach [46]. FARGO [47] addresses the Maximum Inner Product Search

problem (MIPS) using LSH. LSH-CO-SUBSTRING [48] introduces a more efficient indexing method, while LSH-APG [49] combines LSH with graph-based techniques. DB-LSH [50] implements a dynamic bucketing scheme, and HD-INDEX [51] focuses on finding approximate $k$ nearest neighbors to reduce space requirements.

## 3.3 Bucketing

In this section we solve the $(\epsilon, 1)$-NN with bucketing. We set $r = 1$ as we can reduce any case to this by scaling up or down our space. At first, we impose a uniform grid with side length equal to $\epsilon/\sqrt{d}$ on $\mathbb{R}^d$. This way, the furthest (euclidean) distance between two points in the same grid is at most $\epsilon$. Then we define the balls $B_i = B(p_i, 1)$, for every $p_i \in P$, $\overline{B_i}$ to be the set of grid cells intersecting $B_i$, and store all elements of $\bigcup_i \overline{B_i}$ in a hash table, along with the information about the corresponding ball(s); for each cell we store in the table we have to *remember* to which points $p_i$ it corresponds to.

To answer a query $q$, we just find the cell which contains $q$, and check if it is stored in the table. If it is stored then we return a corresponding point $p_i$, else we return *null*.

We can see that for $0 < \epsilon < 1$, $\overline{B_i} = O(1/\epsilon)^d$ (recall that $v_d r^d$ is the volume of a ball of radius $r$ in $\mathbb{R}^d$, where $v_d$ is a constant that depends on the dimension). Thus for every point $p_i$ we need $O(1/\epsilon)^d$ space to store the cells. Finally we can write formally the theorem.

**Theorem 3.20.** *For any $0 < \epsilon < 1/2$, there exists a fully dynamic data structure for $(\epsilon, r)$-NN in $l_2$ norm over $\mathbb{R}^d$, such that:*

- *The query time complexity is $O(d)$.*
- *The space complexity is $O(n) \times O(1/\epsilon)^d$.*

We note that we can extend this method to any $l_s$ norm, by setting the grid side length to $\epsilon/d^{1/s}$. This way $\overline{B}$ is still bounded by the same number, and thus the theorem also holds.

Bucketing will be used in our model as well. It will be the method in which we resort when the form of the point set is not the desired and by applying the data driven method (that will be explained later) could result in times worst than the guarantees.

## 3.4 Voronoi Diagram

For completeness, in this section we present the Voronoi Diagram solution of the $\epsilon$-NN problem, which consists a very powerful tool and is one of the most widely used. It can also be applied to many other problems in computational geometry such as surface reconstruction.

Section 3.1 reduces the $\epsilon$-NN problem to performing $O(\log n)$ searches in appropriate instances of $(\epsilon, r)$-NN structures. It is natural to ask if it is possible to only have one

structure, and obtain a natural geometric representation of the input point set, resulting in solving directly the $\epsilon$-NN problem. It has been proven that it is indeed possible to do that by adjusting the bucketing technique to get a Voronoi diagram.

**Theorem 3.21.** *There is an absolute constant $C > 0$ such that given a set $P$ of $n$ points in $\mathbb{R}^d$ and a parameter $\epsilon \in (0, 1/2)$, one can compute, in $O((n(C/\epsilon)^{d+1}) \log^3 n)$ time, an approximate Voronoi diagram of $P$ of size $O((n(C/\epsilon)^{d+1}) \log^2 n)$, such that*

- *every cell in this diagram is either a cube or a set difference of two cubes, and*
- *every such cell has a point of $P$ associated with it that it is a $(1+\epsilon)$-approximate nearest neighbor of all the points of the cell.*

We now explain how to collapse all the bucketing grids of the $\log n$ *near* neighbor structure. First, we assume that the grid side lengths are powers of 2 (achieved by scaling), and that all grids are centered at the origin. This ensures that the grids are nested, meaning cells of finer grids are not split across different cells of coarser grids.

Let $Z_1, Z_2, \ldots, Z_k$ be the structure instances for the *near* neighbor. Also, let $r_i$ and $t_i$ be the radius and the grid side length of $Z_i$ accordingly. For every $Z_i$ and every point $p$, we label every grid cell intersecting the ball $B(p, r_i)$ with $p$. The decomposition can be obtained by superimposing all grids onto the space $\mathbb{R}^d$, where any point takes the label of the smartest grid cell containing it.

Upon the arrival of a query point $q$, we find the grid cell of the smallest side length containing $q$. Then we know that the label of that cell is a correct approximate nearest neighbor for $q$. Regarding finding the approximate nearest neighbor, one can apply a search algorithm in the instances, or use a compressed d-dimensional quadtree.

This concludes the description of the approximate Voronoi diagram. For a more detailed description see [17].

## 3.5 Data-Driven LSH

In this section, we discuss recent work by Andoni and Beaglehole [1], which focuses on optimizing the LSH method for the Hamming metric using a data-driven approach. We present this work because it attempts to bridge the gap between the data-driven aspect and worst-case guarantees. As stated in their paper, algorithms can be divided into two categories: those with theoretical guarantees and those that aim to find (or learn) the best possible space partition for a given dataset.

To provide an overview of their solution, we think of the LSH method (see section 3.2)and the procedure of a concatenated hash function $g_j$ as a tree. At the root, we randomly select (uniformly) one of the possible hash functions $h_i$, which corresponds to a specific bit. Based on that bit, we split the dataset. We then repeat this process for the subsequent levels of the tree $k$ times, where $k$ is the number of hash functions $h_i$ in one concatenated function $g_j$.

The paper's contribution lies in replacing the uniform random selection of hash functions with a method that finds a distribution over the hash functions $h_i$ to maximize the recall on the least performing queries. These least performing queries are defined as those exactly at distance $r$ from the nearest point in the point set, making them the furthest near neighbors. To find that distribution, they solve a two-player zero-sum game, in which one player is the hash player who chooses a distribution over the hash functions $h_i$, and the other is the query player who selects a query/nearest neighbor pair adversarially for the least probability of success at the end of the tree.

For proof of concept, they demonstrate that in point sets generated from mixture models, their method achieves an improvement factor of $\Omega\left(exp(\Omega(\sqrt{\ln d}))\right)$ on the minimum query compared to standard LSH techniques. They also conducted experiments, primarily classification tasks on the ImageNet and MNIST datasets, showing significantly better success probabilities than those achieved with uniform LSH. Additionally, they illustrated the distribution of the hashes over the MNIST dataset images, revealing that the model assigns higher probabilities to the hashes (pixels) in the center than to those in the corners.

Their solution successfully maintained the correctness and performance of the LSH method while making it data-adaptive, which means the algorithm adapts to a given fixed dataset, achieving even better success probabilities for datasets with favorable characteristics. They also backed up their theoretical results with experiments.

# Chapter 4

# Our Model

In this chapter, we introduce the data structure we developed, and explain its dual nature; it is a data-driven model (it adapts according to the point set) while simultaneously upholding worst-case guarantees, a feature rarely achieved in previous work. One reason for this could be that developing a data-driven model is inherently non-randomized and typically involves a degree of greediness. As a result, there's always the potential for adversarial point sets that can cause the model to perform poorly.

The key concept of our model is to cease utilizing information from the point set to build a data-driven model once it is no longer safe to ensure worst-case guarantees. At that point, we switch to an established solution for the problem, namely the bucketing technique. This approach allows us to leverage as much information from the point set as possible without compromising worst-case guarantees.

Our model is essentially a binary tree that divides the point set into two disjoint subsets from the root downwards, until the size of them becomes small enough ($O(1)$). The division is guided by a separator with specific properties designed to ensure the model's efficiency and that separator is computed while taking into consideration the point set. This data-driven method is applied as long as the point set at a vertex remains sufficiently sparse (more details to follow). When encountering a vertex with a dense point set, we designate it as a leaf (i.e., no further splitting occurs) and implement a preprocessing technique inspired by bucketing.

Upon the arrival of a query point $q$, we traverse the tree starting from the root, proceeding to either one or both children based on the position of $q$ relative to the separator. Then when we reach a leaf, we iterate through the points until we find a near neighbor. If the leaf has undergone bucketing preprocessing, we return a point based on the bucketing technique.

In the following sections, we will conduct an in-depth analysis of the model's structure as well as the vertex separators. As we will see, the computation of the separators constitutes an optimization problem in itself (and in fact a challenging one). We will also propose two algorithms for identifying these separators and, finally, discuss the overall optimality of the model.

## 4.1 Structure

Before defining our structure, we need to introduce the following notation.

**Definition 4.22.** *A ball $B(o,r)$ is defined as $B(o,r) = \{x \in \mathbb{R}^d \mid \|o-x\|_2 \leq r\}$ and a ring $R(o,r_1,r_2)$ is defined as $R(o,r_1,r_2) = \{x \in \mathbb{R}^d \mid r_1 \leq \|o-x\|_2 \leq r_2\}$*

**Definition 4.23.** *A ring separator $\varrho(p,r_s)$ is a ring $R(p,r_s,r_s+2r)$, where $r$ is the search radius of the Near Neighbour problem instance. We say that $R(\varrho(p,r_s))$ is the ring of the separator $\varrho(p,r_s)$. This separator splits the point set $P$ into 2 subsets, $P \cap B(p,r_s+r)$ and $P \cap \overline{B(p,r_s+r)}$. We call these subsets inner and outer respectively.*

Now we are able to define our data-driven structure.

**Definition 4.24.** *Let $RT_\rho$ be a data structure that builds a binary tree as follows: Starting with the whole point set $P$ at the root, divide it in two according to the ring separator $\varrho \in \rho$ procedure defined later on, where $\rho$ is the collection of the separators used to build the particular tree. Repeat this on every new node, until the size of all the leaf sets is $O(1)$. We call this structure a Ring Tree.*

The tree is such constructed that at each vertex, we must identify a suitable separator. As we will later demonstrate, we solve this problem independently for each vertex, without considering the outcomes at other vertices. The building algorithm can be seen below.

**Algorithm 1** Build Tree

```
1:  procedure PREPROCESS(point set P, query distribution D_Q)
2:      N ← new Node
3:      N.P = P
4:      if |N.P| = O(1) then
5:          return N
6:      N.separator ← findSeparator(N.P, D_Q)
7:      P.in, P.out ← split(N.P, N.separator)
8:      N.lchild ←PREPROCESS(P.in, D_Q)
9:      N.rchild ←PREPROCESS(P.out, D_Q)
10:     return N
```

For a query point $q \in \mathbb{R}^d$, the procedure of finding a near neighbor is starting from the root and checking where $q$ falls on the ring separator. If it is inside the ring (meaning $\|o-q\|_2 < r_s$), then follow the child corresponding to the inner subset, else if $q$ is outside the separator (meaning $\|o-q\|_2 > r_s + 2r$), then follow the child corresponding to the outer subset. If the query falls on the separator (meaning $r_s \leq \|o-q\|_2 \leq r_s+2r$), then search for a near neighbor to both children. Upon arriving to a leaf node, we check the points in that point set, and return the first $p_0$ that satisfies $||q-p_0||_2 \leq r$. This way we can be sure that we will find a near neighbour (if it exists) with probability 1. The query algorithm can also be seen below.

**Algorithm 2** Query Search

```
1: procedure QUERY(query point q, starting node N)
2:     if N is a leaf Node then
3:         p* ← null
4:         for each p ∈ N.P do
5:             if ‖p − q‖ ≤ (1 + ϵ)r then
6:                 p* ← p
7:                 break
8:         return p*
9:     ϱ(o, r_s) ← N.separator
10:    if ‖q − o‖ < r_s then
11:        return QUERY(q, N.lchild)
12:    else if ‖q − o‖ > r_s + 2r then
13:        return QUERY(q, N.rchild)
14:    else
15:        left ← QUERY(q, N.lchild)
16:        if left ≠ null then
17:            return left
18:        else
19:            return QUERY(q, N.rchild))
```

For this model to be efficient and the query procedure to be fast, two main properties must be satisfied. First, the tree's height must be kept relatively small. This can be achieved by ensuring that the point set splits at the inner vertices are balanced; otherwise, the height could become linear on the number of points. Second, we must avoid instances where the query point falls on the separator, necessitating traversal to both children. This can be accomplished by minimizing the probability of a query point landing on the separator.

The general case is that the point set $P$ is drawn from a distribution $\mathcal{D}_P$ and the queries $q_i$ come from a distribution $\mathcal{D}_Q$. We notice that the balance of the separator depends only on the point set $P$, and the probability of a query $q$ falling on the separator depends only on the distribution $\mathcal{D}_Q$. Therefore, it is natural to ask for the separator to be balanced on $P$ and sparse on $\mathcal{D}_Q$.

The following analysis does not assume that $\mathcal{D}_P$ and $\mathcal{D}_Q$ distributions are identical. However, if the they are the same, applying the same distribution will yield similar results.

Recall that in Section 2.1 we needed the $k - ply$ property for the set of balls $\mathcal{B}$. As we now are on the Euclidean space and use the $l_2$ norm, we need a similar notion for probability distributions. This property will ensure that the probability distribution does not have *dense* areas, and that is rather smoothed out.

**Definition 4.25.** *Let $f$ be the PDF of a distribution $\mathcal{D}$ over $\mathbb{R}^d$. We say that $\mathcal{D}$ is*

*$(\alpha, r)$-sparse if for any Euclidean ball $B$ of radius $r$, it holds $\int_B f(x)\mathrm{d}x \leq \alpha$.*

We also define the $P - balanced - \mathcal{D}_Q - sparse$ separators which will be needed for the construction of the model.

**Definition 4.26.** *Let $\mathcal{D}_Q$ be an $(\alpha, r)$-sparse distribution and $f$ be its PDF. In addition, let $P$ be a point set in $\mathbb{R}^d$ drawn from $\mathcal{D}_P$, with $|P| = n$. Lastly, let $c$ be the doubling constant of $\mathbb{R}^d$. A ring separator $\varrho(p, r_s)$ is called $P - balanced - \mathcal{D}_Q - sparse$ if for the ring $R = \{x \in \mathbb{R}^d \mid \|x - p\| \in [r_s, r_s + 2r]\}$ it holds*

$$\int_R f(x)\ \mathrm{d}x \leq 2\alpha^{O(1/d)},$$

*and for the inner ball $\mathcal{B} = B(p, r_s + r)$ and the outer ball $\overline{\mathcal{B}} = \{x \in \mathbb{R}^d \mid \|x - p\| \geq r_s + r\}$, it holds*

$$|P \cap \mathcal{B}| \geq \frac{n}{c+1} \text{ and } |P \cap \overline{\mathcal{B}}| \geq \frac{n}{c+1}.$$

The existence of such separators is proven in Subsection 4.1.1.

We are now able to show the following theorem.

**Theorem 4.27.** *A Ring Tree Structure that is built for a point set $P$ and an $(\alpha, r)$-sparse query distribution $\mathcal{D}_Q$, by using $P - balanced - \mathcal{D}_Q - sparse$ separators, is a solution for the $(\epsilon, r)$-NN problem and comes along with the following guarantees:*

1. *The expected query time is $O(n\alpha^{O(1/d)} c \log n)$*
2. *The space complexity is $O(n)$*
3. *The preprocessing time is $O(c \log n) \times \mathcal{F}(\varrho)$, where $\mathcal{F}(\varrho)$ is the time needed to find a $P - balanced - \mathcal{D}_Q - sparse$ separator $\varrho$. This is for the case that $\mathcal{F}(\varrho)$ is $\Omega(n^2)$.*

*Here $c = 2^{O(d)}$ is the doubling constant of $\mathbb{R}^d$.*

*Proof.* Let $b$ be the number of nodes in the tree that the query follows both children to search for a near neighbor. We know that the probability that a query follows both children is equal to the probability that the query falls on the ring separator. This probability is known and is less than $2\alpha^{O(1/d)}$.

Thus, the expected number of bad nodes is $\mathbb{E}[b] = n \cdot 2\alpha^{O(1/d)}$, because we have at most n separators in the structure. Finally, the expected query time is less than the product of the leaves visited (which is equal to the expected number of bad nodes plus 1) and the height of the tree:

$$\mathbb{E}[T_q] \leq (\mathbb{E}[b] + 1) \cdot h = (n \cdot 2\alpha^{O(1/d)} + 1) \cdot h$$

We have that $h = \log_{\frac{c+1}{c}} n = \frac{\log n}{\log \frac{c+1}{c}} = O(c \log n)$. We have $\frac{c+1}{c}$ as the base of the logarithm, as the bigger in size child has at most $\frac{c}{c+1}$ portion of its parent points. This is true, as we use $P - balanced - \mathcal{D}_Q - sparse$ separators. This concludes the proof of the first point.

Towards proving the second point, we simply need to observe that each point is stored in only one leaf. Consequently, the required space includes the space needed to store the points and the space needed to store at most $n$ inner vertices of the model along with their separators. Thus the space needed is linear in the number of points $n$.

Lastly for the third point, during preprocessing we need to find $O(n)$ $P-balanced-\mathcal{D}_Q-sparse$ separators, as we have $O(n)$ inner vertices. Assuming worst possible balance and $\mathcal{F}(\varrho)$ is $\Omega(n^2)$ we have that for the root we need $\Omega(n^2)$ time. Then for the second layer we have:

$$\frac{n^2}{(c+1)^2}+\frac{c^2n^2}{(c+1)^2}=n^2\frac{1+c^2}{(1+c)^2}\leq n^2$$

Thus, for each subsequent layer, the time required will be less than the time needed for the root. Consequently, the total time required to find all separators is $O(h)\times\mathcal{F}(\varrho)$, where $h=O(c\log n)$ as shown before.

Also, after finding each separator, we need to split the point set on that vertex. The splitting can be calculated as the size of the initial point set multiplied by the height of the tree, which is $O(c\log n)$. The total preprocessing time is the sum of the times of those two operations, but since $\mathcal{F}(\varrho)>O(n)$, the final complexity is $O(c\log n)\times\mathcal{F}(\varrho)$. □

Currently, the query time still includes the parameter $a$. To achieve a sublinear query time, we can set the parameter $a$ to be equal to $n^{-1}$ to obtain a desired query performance. This can be seen in the follwing corollary.

**Corollary 4.28.** *A Ring Tree Structure that is built for a point set $P$ and an $(n^{-1},r_n)$-sparse query distribution $\mathcal{D}_Q$, by using $P-balanced-\mathcal{D}_Q-sparse$ separators, is a solution for the $(\epsilon,r_n)$-NN and has expected query time equal to $O(n^{1-1/d}c\log n)$, which is sublinear.*

### 4.1.1 Existence of Separators

In the proof of Theorem 4.27 we assumed the existence of $P-balanced-\mathcal{D}_Q-sparse$ separators. In this subsection, we demonstrate that these separators do indeed exist.

**Lemma 4.29.** *Let $\mathcal{D}_Q$ be an $(\alpha,r)$-sparse distribution and let $f$ be its PDF. In addition, let $P$ be a point set in $\mathbb{R}^d$ drawn from $\mathcal{D}_P$, with $|P|=n$. Lastly, let $c$ be the doubling constant of $\mathbb{R}^d$. There is a point $p\in\mathbb{R}^d$ and a radius $r_s$ defining a ring separator $\varrho(p,r_s)$ such that for the ring $R=\{x\in\mathbb{R}^d\mid\|x-p\|\in[r_s,r_s+2r]\}$ is true that:*

1. $|\{x\in P\mid\|x-p\|\leq r_s\}|\geq\frac{n}{c+1}$ *and* $|\{x\in P\mid\|x-p\|\geq r_s+2r\}|\geq\frac{n}{c+1}$
2. $\int_R f(x)\,\mathrm{d}x\leq 2\alpha^{O(1/d)}$

*Proof.* Let $\mathcal{B}_1$ be the Euclidean ball of minimum radius such that $|\mathcal{B}_1\cap P|=\frac{n}{c+1}$ and let $\mathcal{B}_2$ be the concentric ball of twice the radius. Separator $\varrho$ will be somewhere in between. By the definition of $\mathcal{B}_1$ and the doubling constant we have $|P\setminus\mathcal{B}_2|\geq\frac{n}{c+1}$.

Towards proving point 2, we first scale the space so that the radius of the ball $\mathcal{B}_1$ is equal to 1 (and the radius of $\mathcal{B}_2$ is equal to 2). Due to the scaling, $r$ changes to $r'$. The

separator will be concentric to the balls, and will have inner radius so that it is whole in $\mathcal{B}_2 \setminus \mathcal{B}_1$. Thus, we can assume that $\int_{\mathcal{B}_2 \setminus \mathcal{B}_1} f(x)\, \mathrm{d}x > \alpha^{O(1/\log c)}$, otherwise the statement is trivially true. By the definition of the doubling constant, we can cover $\mathcal{B}_2$ with $c^{\Theta(\log(1/r'))}$ balls of radius $r'$. Hence,

$$\alpha^{O(1/\log c)} \leq \int_{\mathcal{B}_2 \setminus \mathcal{B}_1} f(x)\, \mathrm{d}x \leq c^{\Theta(\log(1/r'))}\alpha \implies r' \leq \alpha^{\Theta(1/\log c)}. \tag{4.1}$$

A more careful and attentive analysis for Equation 4.1 is provided in Section A.1 of the Appendix. This equation implies that there are at least $\frac{\alpha^{-\Theta(1/d)}}{2}$ annulii of width $2r'$ in $\mathcal{B}_2 \setminus \mathcal{B}_1$, having overall density at most 1. Hence one of those annulii has density at most $2\alpha^{O(1/d)}$. □

This proof is inspired by Har-Peled's proof for Ball Separators. Actually, the argument about the balance is the same. Now, instead of the $k - ply$ property, we have the $(\alpha, r)$-sparsity property for the probability distribution. This is what ensures us that the $r$-the search radius of the Near Neighbor instance, as well as the separator half width- will be under $1/2$ when scaled ($r' < 1/2$), so that the separator lies fully in $\mathcal{B}_2 \setminus \mathcal{B}_1$.

Also, notice that the search radius $r$ of the Near Neighbor instance, is in both the separator and the sparsity property. This means that for every instance of the structure, the sparsity requirement changes; for larger radii it is more strict, as the probability mass of a larger area must be less than the same $\alpha$. Because of this, it is essential to switch to another technique, in order not to let $\alpha$ be very large, and thus render useless our guarantees. The switch of technique is described in Subsection 4.1.3.

### 4.1.2 Generalization to Other Metric Spaces

Up until now, all the proofs were for the Euclidean space and the $l_2$ norm. However, notice that for the proofs to hold, the only requirement for the metric space in which the point set resides is that it has a doubling constant bounded from above by $2^{O(d)}$. Thus, we get the following proposition.

**Proposition 4.30.** *If the metric space $(X, D)$ has doubling constant $c < 2^{O(d)}$ then we can build a Ring Tree for any point set residing in that space.*

With this in mind, we note that this applies to the data-driven structure. The bucketing method, as we know, can be applied in Euclidean space, but with any $l_s$ norm. However, this does not prevent us from using a different technique suitable for that space. Additionally, one might choose to sacrifice the guarantees and continue finding *suboptimal* separators that do not satisfy the sparsity condition.

### 4.1.3 Switch to Bucketing

Up to this point, we have discussed $(\alpha, r)$-sparse distributions. Now, we consider distributions that may also contain dense areas. Given that the total density is 1, there

cannot be many non-overlapping dense regions. Therefore, we can apply the bucketing method, incorporating the extra information that we now have about the distribution.

**Lemma 4.31.** *Let $\mathcal{D}_Q$ be a distribution that is not $(\alpha, r)$-sparse and let $f$ be its PDF. In addition, let $P$ be a point set in $\mathbb{R}^d$ drawn from $\mathcal{D}_P$, with $|P| = n$. Lastly, let $c$ be the doubling constant of $\mathbb{R}^d$. For $0 < \epsilon < 1/2$, by applying the grid bucketing technique, we can solve the $(\epsilon, r)$-NN problem in $l_2$ norm over $\mathbb{R}^d$, such that:*

- *The query time complexity is $O(d)$.*
- *The space complexity is $O(1/\alpha) \times O(1/\epsilon)^d$.*

*Proof.* The only thing changed from the classic bucketing method is the space needed. As the distribution $\mathcal{D}_Q$ is not $(\alpha, r)$-sparse, we know that there is at most $\frac{1}{\alpha} \cdot V_B$, where $V_B$ is the volume of a ball with radius $r$. According to the bucketing method, we also know that a ball with that radius intersects $O(1/\epsilon)^d$ grid cells. Thus, the theorem follows. □

In practice, it is difficult to check whether a distribution is $(\alpha, r)$-sparse or not. Thus, what we can do is try to find a $P - balanced - \mathcal{D}_Q - sparse$ separator anyways, since the $(\alpha, r)$-sparsity is a sufficient and not necessary condition for its existence. If the separator satisfies the balance and sparsity conditions, then we can continue building the tree. Otherwise, we know that the distribution must not be $(\alpha, r)$-sparse, and in that case we apply the bucketing technique.

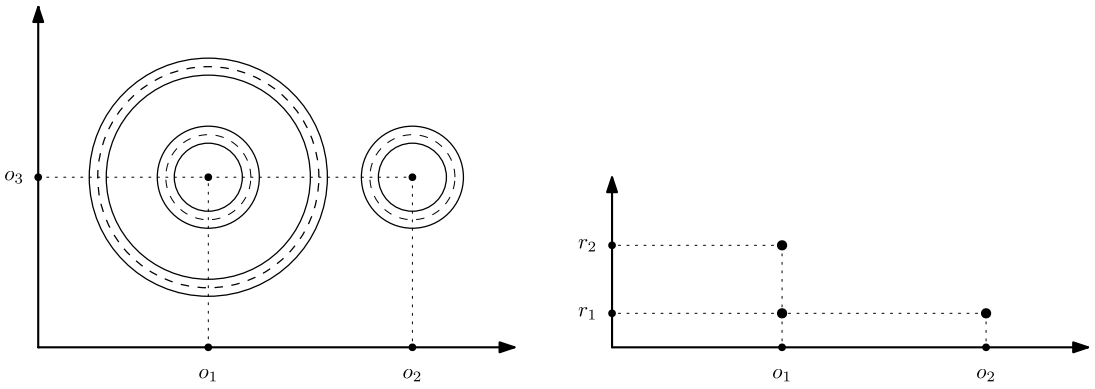
## 4.2 Finding a Separator

In the previous section, we proved the existence of separators that provide worst-case guarantees. In this section, our goal is to find not just any separator that meets the minimum conditions, but the optimal one. Given that balance and sparsity are the two characteristics defining a separator, optimality can be defined in various ways. For our purposes, we define optimality as minimizing sparsity while maintaining balance, where balance only needs to meet the required threshold rather than being optimized.

Finding a separator is challenging, as it constitutes a difficult optimization problem. Defining a separator requires specifying two components: the center $o$ and the inner radius $r_s$. Therefore, $d+1$ variables are needed: $d$ for the center and 1 for the radius. The width of the separator is constant and is equal to $2r$, the double of the search radius of the Near Neighbor instance. So lets imagine a $(d+1)$-dimensional space, where all the separators reside (a point in this space corresponds to a unique separator).

**Definition 4.32.** *Let the Separator Space ($\mathbb{S}_S$) be the $\mathbb{R}^d \times \mathbb{R}^+$ space that describes all possible separators of $\mathbb{R}^d$, inthe following way:*

- *The first $d$ dimensions correspond to the center of the separator $o$.*
- *The last dimension corresponds to the inner radius of the separator $r_s$.*




(a) Euclidean Space $\mathbb{R}^2$ (b) Separator Space $\mathbb{S}_S$

Figure 4.1: Illustration of the Separator Space. Here three separators $\varrho_1((o_1, o_3), r_1)$, $\varrho_2((o_1, o_3), r_2)$, $\varrho_3((o_2, o_3), r_1)$, where $r_2 = 3r_1$, are shown in the Euclidean Space $\mathbb{R}^2$ and the Separator Space $\mathbb{S}_S$ (as $\mathbb{S}_S$ is 3-dimensional, we illustrate the projection plane where the second dimension is equal to $o_3$).

In this space, we need to define two functions that describe the balance and sparsity of the separator. We'll start by discussing balance. We want the separator to split the point set in a balanced manner, ensuring that the size of each child falls within a specific range. Therefore, we can define the balance function as the size of one of the two children and verify that it is above a lower threshold and below an upper threshold.

**Definition 4.33.** *Let $Bl(\varrho) : \mathbb{S}_S \to \mathbb{N}$ be the balance function that returns the size of the inner subset when the point set $P$ in the initial space $\mathbb{R}^d$ is split by a given separator $\varrho \in \mathbb{S}_S$. Let $o$ be the $d$-dimensional center, and $r_s$ the inner radius of $\varrho$. Also let $2r$ be the width of the separator. Then,*

$$Bl(\varrho(o, r_s)) = |\{p \in P \ : \ \|p - o\| \leq r_s + r\}|$$

*We will say that the separator $\varrho(o, r_s) \in \mathbb{S}_S$ is $P-$balanced if:*

$$\frac{n}{c+1} \leq Bl(\varrho(o, r_s)) \leq \frac{cn}{c+1},$$

*where $n = |P|$, and $c$ is the doubling constant of $\mathbb{R}^d$.*

Alternatively, we could define the balance as the size of the smaller subset of the two, and then simply check if it is above the lower threshold. This definition is equivalent.

Now lets talk about sparsity. As it is difficult to have the $PDF$ of the query distribution $\mathcal{D}_Q$, we draw a sample from it $S_Q \sim \mathcal{D}_Q$, and we work with that. This sample can be the first queries that arrive (assuming that they are i.i.d), or if we know that the query distribution is the same with the distribution of point set $P$, we can work with $P$ itself. Then the sparsity function becomes obvious, and is the number of points that lie on the separator. For a separator to be considered sparse, the sparsity function must be below an upper threshold, as defined in Definition 4.26. With this in mind, we also aim to find the optimal separator, which has the minimum sparsity.

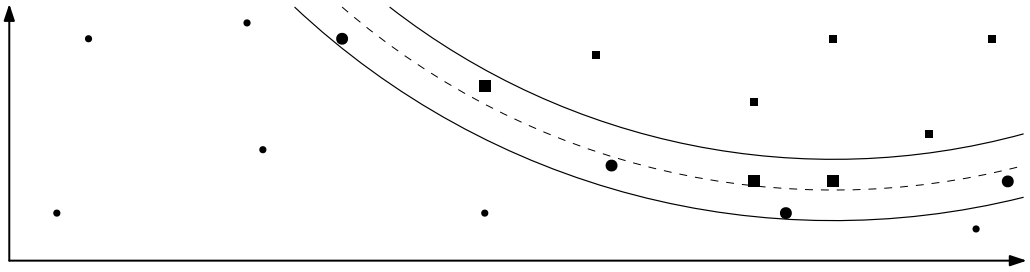


Figure 4.2: Example of separator split. The square points are the inner subset, and the circle points the outer. Also the bold points are counted to estimate the sparsity of the separator.

**Definition 4.34.** *Let $Sp(\varrho) : \mathbb{S}_S \to \mathbb{N}$ be the sparsity function that returns the number of sample queries of $S_Q$ drawn from $\mathcal{D}_Q$ that lie on the input separator $\varrho \in \mathbb{S}_S$. Let $o$ be the $d$-dimensional center, and $r_s$ the inner radius of $\varrho$. Also let $2r$ be the width of the separator. Then,*

$$Sp(\varrho(o, r_s)) = |\{p \in P \ : \ r_s \leq \|p - o\| \leq r_s + 2r\}|$$

*We will say that the separator $\varrho(o, r_s) \in \mathbb{S}_S$ is $S_Q - Sparse$ if:*

$$Sp(\varrho(o, r_s)) \leq |S_Q| \cdot 2\alpha^{O(1/d)},$$

*where $\alpha$ is the parameter in the wanted $(\alpha, r)$-sparsity condition.*

Figure 4.2 shows en example of a separator split. We can now formally define our optimization problem:

**Problem 4.35.** *Given a point set $P$, and a query sample $S_Q$, find the separator $\varrho^*(o, r_s) \in \mathbb{S}_S$, such that $\varrho^*(o, r_s)$ is $P - balanced$ and $Sp(\varrho^*(o, r_s))$ is minimized.*

A natural approach to solving this problem would be to formulate it as a linear (or integer) program. This could involve setting the objective function as the sparsity function and incorporating the balance function as constraints. Unfortunately, this approach appears to be unfeasible. Observe that both the balance and sparsity functions are piecewise constant. This means that while their domains are continuous, their codomains are discrete. Both functions operate by counting points that have a particular property. However, expressing this counting process within a linear programming framework is challenging.

An alternative approach could involve using gradient descent or local search. For these algorithms to guarantee optimality, the objective function must be convex. However, in our scenario, both the balance and sparsity functions are highly non-convex. This non-convexity arises because moving a separator in the initial space $\mathbb{R}^d$ causes the relative positions of points to change in an almost random manner. Therefore, gradient descent and local search algorithms seem not to be suitable for this problem.

Lastly, we consider the option of dynamic programming, which involves simplifying a complex problem by breaking it down into simpler sub-problems solved recursively. This

approach might require preprocessing to organize the data of the sub-problems for efficient execution of the recursive formula. However, this method appears impractical for our case due to the difficulty in simultaneously modeling all constraints.

Considering all the above, we conjecture that the problem of finding the sparsest balanced separator is NP-hard. Therefore, in order to achieve optimality we resort to a brute-force approach, where we extensively test many separators and select the one with the best results.

### 4.2.1 Brute-Force Approach

As previously explained, brute-force appears to be the only method that can guarantee an optimal solution for finding the sparsest separator $\varrho$. However, this approach is still challenging due to the infinite number of possible separators to consider. To address this, we must first discretize the Separator Space $\mathbb{S}_S$. Our goal is to include a sufficient number of points to ensure the algorithm can find a good separator, while also avoiding multiple separators that split the point set $P$ identically.

We propose the following algorithm:

**Algorithm 3** Find Separator - Brute Force

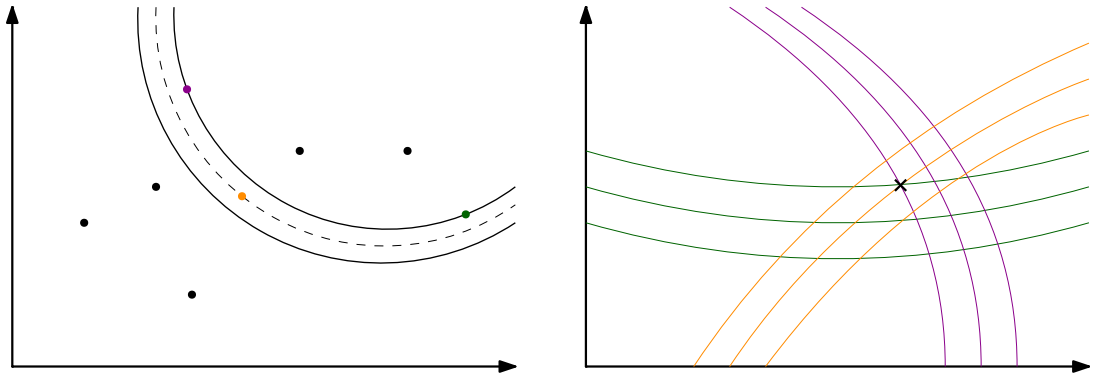
```
1: procedure SEPARATOR(point set P, query sample S_Q)
2:     min_mass ← n
3:     for each subset A of P ∪ S_Q of size d + 1 do
4:         for each possible ring separator ϱ defined by A do
5:             mass ← |R(ϱ) ∩ S_Q|
6:             if mass < min_mass and ϱ is P − balanced then
7:                 min_mass ← mass
8:                 ans ← ϱ
9:     return ans
```

It is known that $d+1$ points in $\mathbb{R}^d$ uniquely define a $d$-dimensional sphere. In a similar manner, $d+1$ points when distributed on the three different radii ($r_s$, $r_s + r$ and $r_s + 2r$) can uniquely define a separator $\varrho$.

For example, lets say that we have $d+1$ points $p_1, \ldots, p_{d+1}$ and we want to find a separator $\varrho(o, r_s)$ that has the first point on the inner radius, the second point on the middle radius and all the rest on the outer radius. Formally, we want:

- $\|p_1 - o\| = r_s$
- $\|p_2 - o\| = r_s + r$
- $\|p_i - o\| = r_s + 2r$, $\forall i \in \{3, \ldots, d+1\}$

There exists only one separator $\varrho(o, r_s)$ which satisfies all the conditions (under some constraints as well). This separator can be found using linear algebra. For a full analysis on how to find a separator from $d+1$ points see Section A.2 from the Appendix.



(a) Euclidean Space $\mathbb{R}^2$ (b) Separator Space $\mathbb{S}_S$

Figure 4.3: Discretization of the Separator Space. The separator that has the purple and the green point on the inner radius, and the orange on the middle radius, corresponds to the point "X" in the Separator Space $\mathbb{S}_S$ (as $\mathbb{S}_S$ is 3-dimensional, we illustrate the projection plane where the three polynomials intersect).

By discretizing the Separator Space in this manner, we limit our consideration to separators that intersect with $d+1$ points. Consequently, if the optimal separator does not intersect any points, it will not be considered by our algorithm. However, the following theorem demonstrates that our algorithm will still identify a sufficiently good separator.

**Theorem 4.36.** *Let $\varrho^*$ be the optimal separator for a given point set $P$ and a query sample $S_Q$. Algorithm 3 will find a separator $\varrho^+$ which will satisfy the following:*

$$|Bl(\varrho^*) - Bl(\varrho^+)| + |Sp(\varrho^*) - Sp(\varrho^+)| \leq d+1$$

*This means that Algorithm 3 finds the optimal separator with an additive error up to $d+1$.*

*Proof.* Let's visualize the Separator Space once more. Within this space, we draw the following hypersurfaces for every $i \in [|P \cup S_Q|]$:

- $\|p_i - o\|^2 = r_s^2$
- $\|p_i - o\|^2 = (r_s + r)^2$
- $\|p_i - o\|^2 = (r_s + 2r)^2$

The points on these hypersurfaces represent the separators that intersect with each point $p_i \in P \cup S_Q$ on the inner, the middle and the outer radii respectively (observe that the three hypersurfaces of each point do not intersect, as a point cannot be at the same time in multiple radii). Therefore, we can retrieve the relative position of $p_i$ to any separator $\varrho(o, r_s)$, by looking on which side of those hypersurfaces it resides. To do that, we can look the sign of the expressions:

- $\|p_i - o\|^2 - r_s^2$

- $\|p_i - o\|^2 - (r_s + r)^2$
- $\|p_i - o\|^2 - (r_s + 2r)^2$

Specifically, regarding balance we have:

- if $\|p_i - o\|^2 - (r_s + r)^2 < 0$, then $p_i$ will be in the inner subset after the split.
- if $\|p_i - o\|^2 - (r_s + r)^2 = 0$, then $p_i$ will be on the middle radius, and thus in the inner subset after the split.
- if $\|p_i - o\|^2 - (r_s + r)^2 > 0$, then $p_i$ will be in the outer subset after the split.

Similarly, regarding the sparsity we have:

- if $\|p_i - o\|^2 - (r_s)^2 > 0$ and $\|p_i - o\|^2 - (r_s + 2r)^2 < 0$, then $p_i$ will lay on the separator.
- if $\|p_i - o\|^2 - (r_s)^2 = 0$ or $\|p_i - o\|^2 - (r_s + 2r)^2 = 0$, then $p_i$ will intersect the inner or the outer radius, and thus lay on the separator.
- if $\|p_i - o\|^2 - (r_s)^2 < 0$ or $\|p_i - o\|^2 - (r_s + 2r)^2 > 0$, then $p_i$ will not lay on the separator.

All the hypersurfaces build a grid in the Separator Space $\mathbb{S}_S$, in which there are cells, (hyper)edges and vertices. A cell can be described as a $(d+1)$-dimensional region in $\mathbb{S}_S$, in which no expression is equal to 0, and for different points in that region, the expressions have the same sign. A (hyper)edge can be described as a $d'$-dimensional region in $\mathbb{S}_S$, where $d' \in \{1, \ldots, d\}$, in which the same $d - d'$ expression(s) are equal to 0, and for different points in that region, the rest of the expressions have the same sign. Lastly, a vertex can be described as a point in $\mathbb{S}_S$, where $d + 1$ expressions are equal to 0.

The separators that we consider are represented with points in the Separator Space that are intersections of $d + 1$ hypersurfaces. Specifically, these hypersurfaces correspond to the $d + 1$ points, and the respective radius. This is demonstrated in Figure 4.3. As we go through all possible subsets of $d + 1$ points, and check all possible separators defined by those points, we basically check all the separators that correspond to all the vertices of the grid in $\mathbb{S}_S$.

Now, we assume, without loss of generality, that the point representing the optimal separator, $\varrho^*$, is located within a cell in $\mathbb{S}_S$. This implies that all expressions for the points in that cell are non-zero and have specific signs. As we evaluate the separators corresponding to all vertices, we will also consider the vertices adjacent to that cell. For these vertices, the expressions that are non-zero will share the same sign as the expressions for the points within the cell, and consequently, the point corresponding to the optimal separator. Consequently, except the $d + 1$ points that define the separator, the relative position of the rest of the points is the same with the optimal separator. □

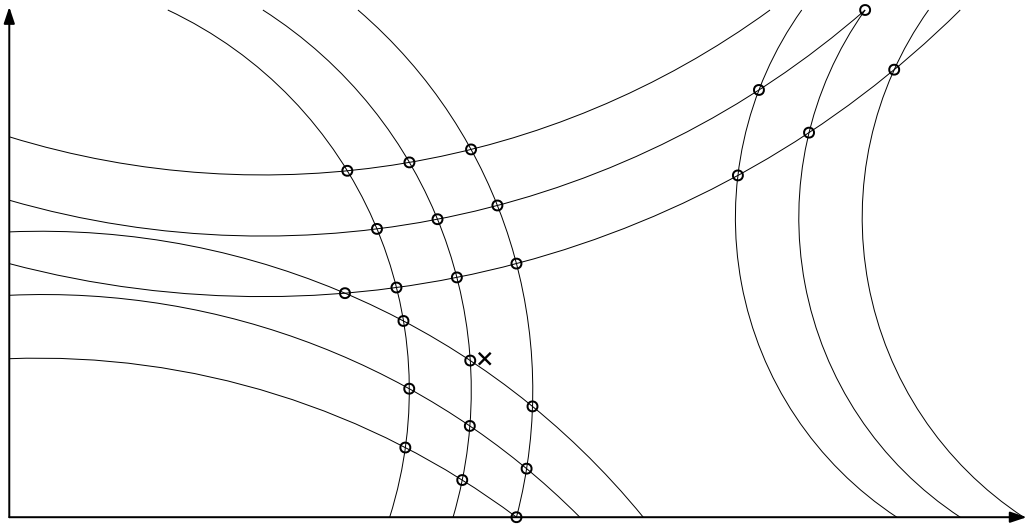


Figure 4.4: Absurd illustration in the Separator Space $\mathbb{S}_S$, where the lines represent the hypersurfaces. The separators that are considered by the brute-force algorithm are the circles, and the optimal separator is the 'x'.

The core idea of the proof is that every possible separator is *almost* equivalent to one that touches $d+1$ points. By evaluating all possible separators defined in this way, we are guaranteed to find a separator that is *almost* optimal.

A rough estimation of the algorithm's time complexity is the number of subsets of $P \cup S_Q$ of size $d+1$, multiplied by the number of ways they can be distributed on the three radii $r_s$, $r_s+r$ and $r_s+2r$. We assume that the size of the sample $S_Q$ is lower than $n$. Therefore, the complexity is equal to $O(n)^{d+1} \times 3^{d+1} = O(n)^{O(d)}$. Also this is multiplied by the time it takes to solve a $d$-dimensional system of linear equations, but we omit it as it is negligible.

Considering this complexity, we can say that it is highly problematic. Even having 100 points, and 10 dimensions gives us $10^{20}$ computations already. This means that technically the algorithm cannot be applied. The main factor for this is that we have the dimension $d$ on the exponent. This phenomenon is known as curse of dimensionality as well.

What we could do to reduce the complexity, is sample from the point set $P$ as well. Specifically, it is known that a $d$-sphere has $VC$ dimension equal to $\delta = d+1$. Also by Lemma 2.9, we get that the $VC$ dimension of a separator, as the set difference of two concentric balls is $O(2\delta) = O(d)$. Thus, we can set the sample size to be $O(d)$. Consequently, the complexity is reduced to $O(d)^{O(d)}$, which is still bad, as $d$ is still on the exponent.

In conclusion, while the brute-force approach offers reliable guarantees, its inefficiency and poor running times render it impractical for real-world applications. Therefore, implementing a heuristic method is the most viable option, as it will significantly improve the efficiency of the original brute-force approach.

### 4.2.2 Locality Heuristic

In this subsection, we will outline our reasoning that led to the development of a heuristic designed to accelerate the process of identifying the sparsest separator. This heuristic is grounded in locality arguments and is inspired by the concept of local search. However, as is often the case with heuristics, it does not offer reliable guarantees regarding the optimality of the resulting separator.

The algorithm with the locality heuristic is shown below:

**Algorithm 4** Find Separator - Locality Heuristic

```
1:  procedure SEPARATOR(point set P, query sample S_Q)
2:      min_mass ← n
3:      for i in [m] do
4:          A ← random subset of P ∪ S_Q of size d + 1
5:          while true do
6:              temp_mass ← min_mass
7:              for each possible ring separator ϱ defined by neighborhood(A) do
8:                  mass ← |R(ϱ) ∩ S_Q|
9:                  if mass < temp_mass and ϱ is P − balanced then
10:                     temp_mass ← mass
11:                     t_ans ← ϱ
12:             if temp_mass < min_mass then
13:                 min_mass ← temp_mass
14:                 ans ← t_ans
15:             else
16:                 break
17:     return ans
```

As $neighborhood(A)$ we define the family of subsets of the same size as $A$, that have at most 1 different point. Formally,

**Definition 4.37.** *As $neighborhood(A)$ we define the family:*

$$neighborhood(A) = \{A' \subset P \cup S_Q \ : \ |A'| = |A|, |A' \cap A| \geq |A| - 1\}$$

The algorithm begins by selecting a random separator (again considering only those corresponding to vertices in the Separator Space $\mathbb{S}_S$). From this starting point, we attempt to find a better separator by exploring options defined by the same points except for one. This process continues until we reach a local optimum, where all neighboring separators have worse sparsity. Additionally, the parameter $m$ in the algorithm specifies how many times the set of points will be randomly initialized to restart the search.

To gain a better understanding of how this heuristic works, let's revisit the Separator Space $\mathbb{S}_S$. We observe that neighboring cells differ by one in terms of sparsity or balance.

Similarly, the vertices exhibit this smooth property, which we refer to as strict locality - where neighboring cells or vertices change singularly in balance or sparsity.

Now, consider the vertex representing the current separator. By allowing only one point to change, we effectively focus on a 1-dimensional edge (basically a curve in the $(d+1)$-dimensional space) out of the $d+1$ that go through the vertex, and examine all vertices on that edge. This approach enables us to check not only neighboring vertices but also make jumps to distant vertices. This broader sense of locality prevents us from getting stuck in local optima, which could occur if we adhered strictly to strict locality. However, this method remains a variation of local search. While it can avoid some local optima, it does not guarantee that the solution returned is a global optimum.

To determine the time complexity of the algorithm, we multiply the size of a subset's neighborhood by the number of possible steps and the number of initializations. The size of the neighborhood is $O(nd)$, as there are $d+1$ points in the subset that can be swapped with the $n-(d+1)$ points that are not used. The number of possible steps is $O(n)$, as the maximum sparsity is $n$, and in each step we reduce it by at least 1. Lastly, by definition of the algorithm, the number of initializations is $m$. Therefore the complexity is $O(mn^2d)$. A reasonable choice for $m$ would be $\log n$, which gives us $O(n^2 d \log n)$ complexity.

Now, by substituting this result to Theorem 4.27, we get that the preprocessing time of our model is $O(n^2 dc \log^2 n)$. This results in a much more manageable complexity, making it feasible for real-world applications. The algorithm is also demonstrated through the experiment in the next chapter.

## 4.3 Model Optimality

In this section, we present a critical evaluation of our model. Our objective is to determine whether our tree construction method (Algorithm 1) is optimal for the given point set $P$ and query distribution $D_Q$. By optimal we mean that it yields the tree with the minimum expected query time. Formally,

**Definition 4.38.** *We say that $RT_\rho$ is the optimal for a point set $P$ and a query distribution $D_Q$, if out of all possible Ring Trees it minimizes the expected query time $\mathbb{E}_{P,D_Q}[T_q]$*

Unfortunately, we were unable to establish any guarantees. Additionally, we suspect that this construction scheme may have an unbounded approximation ratio. This is likely because we determine optimal separators independently for each vertex, without taking into consideration the separators of other vertices. This greedy approach prevents us from avoiding the selection of a separator that may appear optimal at a certain vertex but proves problematic at deeper levels. In addition, the fact that we have additive error on the balance and/or the sparsity of the separator that we find, can be really bad for optimal separators that have 0 sparsity.

We can demonstrate this with an example. Consider a point set $P$ and a query distribution $D_Q$, for which there exists a tree where, for all possible queries, the search always

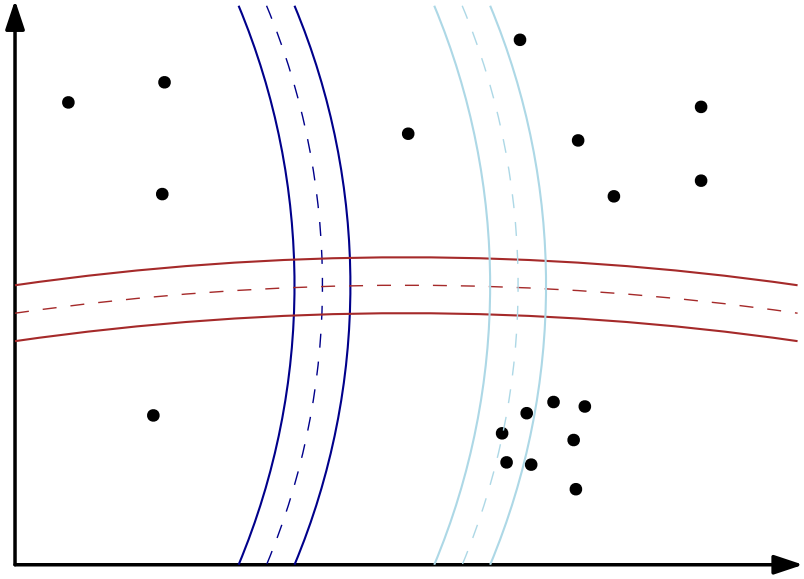


Figure 4.5: Example point set, where the tree we build may have bad approximation from the optimal. If the blue separator is picked for the parent node, then the brown separator is okay for both child nodes. However if the brown separator is picked for the parent node, the blue does not satisfy the balance condition for the lower subset. Thus, another separator (e.g. light-blue) must be picked, but because the points are really close together, it cannot have 0 sparsity.

leads to a single leaf. Now, suppose that at the first split of the point set (at the root of the tree), we do not use the specific separator of the optimal tree. Instead, we use another separator that also has zero sparsity. This choice may result in the separators at the lower nodes having non-zero sparsity, thereby significantly worsening the approximation ratio for our tree. This can also be seen in figure 4.5.

In conclusion, we developed a data-driven model that offers worst-case guarantees. However, it does not provide assurances about the extent to which its performance deviates from the optimal.

# Chapter 5

# Experiment

For our experiments, we use the MNIST dataset, which is often used for AI models. Firstly, we perform some preprocessing to ensure it is suitable for our model. Namely we apply the Johnson-Lindenstrauss transform to reduce the number of dimensions, and prepare the queries so that they have one near neighbor. We then build instances of our tree, using different sample sizes to observe the model's adaptability to the dataset. Additionally, we compare our results with those obtained using the trivial linear search method. For implementation details see section A.3 from the Appendix. In the following sections we give a thorough analysis of the process.

## 5.1 Dataset Preparation

The MNIST dataset is a widely recognized benchmark for training and testing AI models. The dataset consists of grayscale 28x28 images of handwritten digits, with pixel values ranging from 0 to 255. It contains a total of 70,000 images, with 60,000 designated for training and the remaining 10,000 for testing.

We can interpret the pixel values of each image as coordinates in a 784-dimensional Euclidean space, $\mathbb{R}^{784}$. By doing so, each image is represented as a point, transforming the image dataset into a point set. This allows us to apply our algorithm to the dataset effectively. But first, we will apply the Johnson-Lindenstrauss transform (see Section 2.2) to reduce the number of dimensions, in order to decrease the runtime.

We choose $m = 15 \sim \log 60,000$ as the number of new dimensions, and we use a transform matrix $\Pi$ filled with $\pm 1$ values, each with a probability of $1/2$. Theoretically, this ensures with overwhelming probability that the new distances will be less than double the original distances.

For the queries, we first determine the minimum distance between any two points in $P$, denoted as $dist$. We then set the search radius to $r = dist/2$ to ensure our distribution is sufficiently sparse for our guarantees to hold. Next, we generate query points that are within a distance $r' \leq r$ from a point in the point set. We create two query points for each point in the point set, resulting in a total of 120,000 queries.

Additionally, we assume that the query distribution $D_Q$ is the same as the point set distribution. This is a fair assumption since the queries we generate are near points in the point set. Therefore, we do not need to sample the query distribution, as $P$ serves as a fair representation of it.

## 5.2 Benchmark

Typically, a model's performance is mainly evaluated on its query success probability, as seen in most previous work. However, our model is different because its success probability is always 1, making it incomparable to other probabilistic models. Therefore, we need to evaluate our model using alternative methods.

Our approach involves constructing multiple trees with varying sample sizes and then comparing their characteristics. Given the mixed nature of the MNIST dataset, we can select leading points as samples (for example, for a sample size of 1000, we can use the first 1000 data points). We anticipate differences in several characteristics across these trees, particularly in preprocessing time and query times. Specifically, we expect that the preprocessing time will increase quadratically on the sample size, and the mean query time of the full query set will decrease. This comprehensive comparison will help us understand the trade-offs involved in choosing different sample sizes and guide us in optimizing our tree construction strategy for the desired performance.

Furthermore, we will compare the performance of our data structure against that of a simple linear search. Since linear search guarantees a success probability of 1, it serves as an ideal benchmark for evaluating our results. This comparison will allow us to assess the efficiency and effectiveness of our approach relative to a straightforward and reliable method.

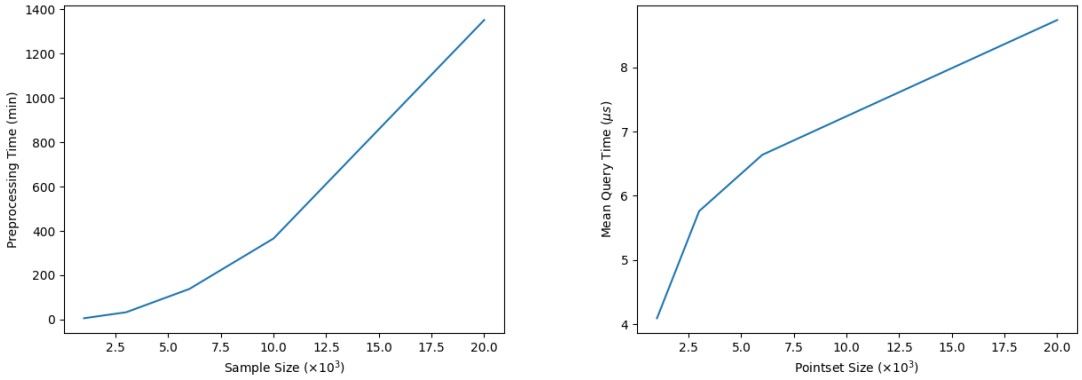
## 5.3 Results

We constructed trees using sample sizes of 1,000, 3,000, 6,000, 10,000, and 20,000 data points. In the following subsections, we will discuss the performance of various aspects of these trees. Additionally, we include plots to visually represent the performance times.

### 5.3.1 Preprocessing Times

Firstly, we discuss the preprocessing times. As we can see from figure 5.1a, preprocessing time increases faster than linearly as the sample size increases. This was expected as the complexity of the preprocessing time is $O(n^2 dc \log^2 n)$. The times can be seen on the table below.

| Ring Tree Preprocessing Times | | | | | |
|---|---|---|---|---|---|
| Sample Size | 1,000 | 3,000 | 6,000 | 10,000 | 20,000 |
| Time ($min$) | 6 | 33 | 138 | 366 | 1352 |




(a) Preprocessing Times (b) Ring Tree Query Times

Figure 5.1: Preprocessing times and query times for different point set sizes.

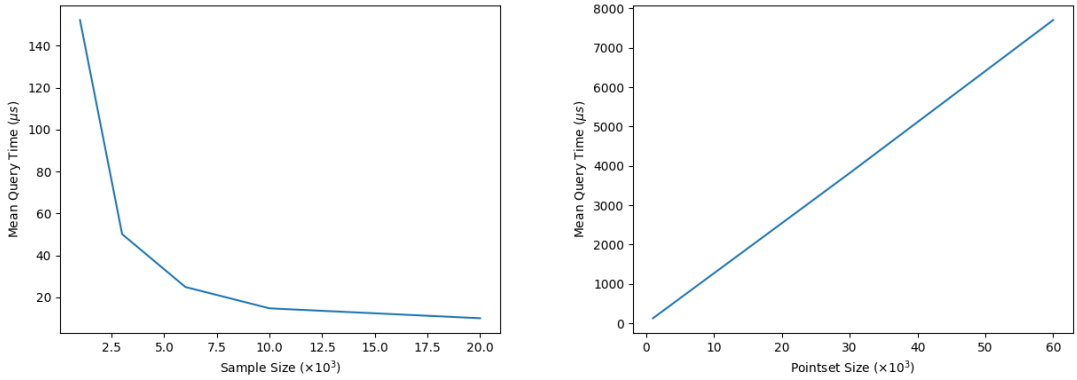
Upon examining the numbers closely, we can deduce that the preprocessing times are actually better than the theoretical complexity suggests. This discrepancy arises because the proven complexity accounts for the worst possible balance and the slowest separator-finding running time. In practice, with a large number of points, it is highly unlikely to encounter the worst-case scenario for every separator, leading to more efficient preprocessing times.

We observe that with a sample size of 1,000, the preprocessing takes only a few minutes. However, when the sample size increases to 20,000, the preprocessing time extends to several hours, making it impractical for larger datasets. Despite this, the query search times show converging performance across different sample sizes (as it will be demonstrated in the next subsections), indicating that constructing the tree on a sufficiently large sample is adequate for maintaining efficient query times.

### 5.3.2 Query Times

The expected query time has a complexity of $O(n^{1-1/d}c \log n)$, which reflects the expected sublinear curve observed in Figure 5.1b. For each tree constructed with different point set sizes (sampled from the original point set), we only make queries corresponding to the points within those point sets. As the size increases, the height of the tree also increases, leading to longer query times. This behavior is expected because a larger number of points results in increased query times. In the following table we see the query times for the corresponding trees.

| Ring Tree Query Times | | | | | |
|---|---|---|---|---|---|
| Pointset Size | 1,000 | 3,000 | 6,000 | 10,000 | 20,000 |
| Time ($\mu s$) | 4.092 | 5.760 | 6.638 | 7.239 | 8.738 |




(a) Ring Tree Query Times

(b) Linear Search Query Times

Figure 5.2: Ring Tree query times for different sample sizes and Linear Search query times for different point set sizes.

### 5.3.3 Adaptability of Our Model

Building upon the trees constructed with different sample sizes from the previous subsection, we now incorporate the remaining points into these trees. This allows us to perform queries on the entire dataset and evaluate the performance of each tree comprehensively. The query times on each tree can be seen on the table on the next page.

| Ring Tree Query Times | | | | | |
|---|---|---|---|---|---|
| Sample Size | 1,000 | 3,000 | 6,000 | 10,000 | 20,000 |
| Time ($\mu s$) | 152.220 | 50.145 | 24.955 | 14.758 | 10.050 |

As the sample size increases, we expect the query time to decrease rapidly. This is because the tree structure adapts to the underlying distribution of the dataset, resulting in a taller tree that more effectively splits the dataset into a greater number of leaves. This is actually the case as it can be observed in Figure 5.2a. We can see that for small sample sizes the difference in query times is significantly bigger, than the difference observed at bigger ones.

### 5.3.4 Comparison with Linear Search

Linear search is a straightforward method that always produces correct results. However, it has the disadvantage of being slow, making it unsuitable for applications requiring faster query times. Figure 5.2b illustrates the performance of linear search across different-sized datasets, showing a linear relationship as expected. The corresponding query times are also presented in the following table.

| Linear Search Query Times | | | | | | | |
|---|---|---|---|---|---|---|---|
| Pointset Size | 1,000 | 3,000 | 6,000 | 10,000 | 20,000 | 30,000 | 60,000 |
| Time ($\mu s$) | 127.183 | 381.731 | 763.587 | 1272.08 | 2543.72 | 3817.48 | 7706.68 |

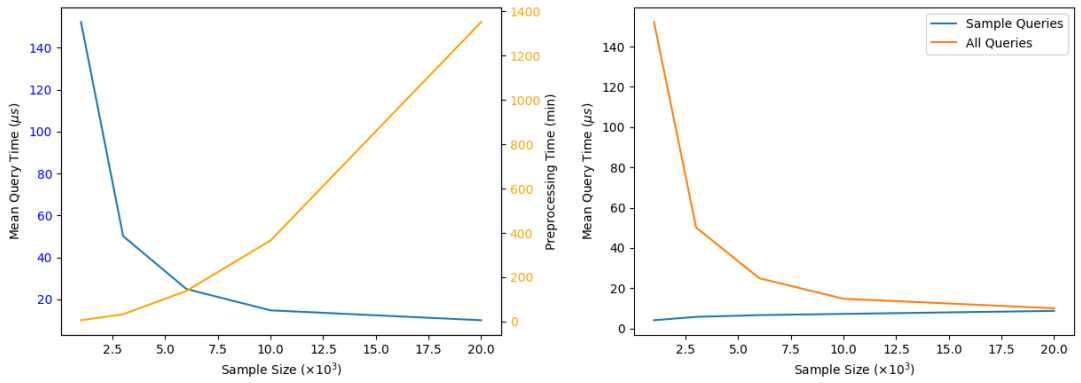
We observe that even the tree built with a sample size of 1,000 significantly outperforms the linear search. The mean query time for the tree is approximately 152 microseconds, while the linear search has a mean query time of around 7,707 microseconds - nearly two orders of magnitude slower.

### 5.3.5 Final Remarks

In this subsection, we discuss two key observations from our results. Firstly, we identify a specific sample size that serves as a sweet spot. Beyond this size, the overhead in preprocessing time becomes excessively large, while the query time does not improve significantly. In our case, this optimal sample size appears to be 10,000. Doubling the sample size results in a mean query time reduction of only about 4 microseconds, while the preprocessing time increases by more than 16 hours. This is illustrated in Figure 5.3a.

The second observation is the convergence of query times. As the sample size increases, the mean query time for the full query set converges from above to a specific value. Simultaneously, the mean query time for the sample queries converges from below to the same value. This is intuitive, as the sample size approaches the actual dataset size, making the query times increasingly similar. Figure 5.3b illustrates beautifully this observation.

As it seems from the results, the mean query time that the structure would offer if it was built on all the point set should be between $8.7\mu s$ and $10.0\mu s$ (the mean query times of sample queries and all queries respectively when sample size is 10,000). Therefore, the tree built on 10,000 points (which gives query search time around $14.8\mu s$) is sufficient for the whole point set.




(a) Query Vs Preprocessing Times (b) Convergence of Query Times

Figure 5.3: Evaluation of sample size and convergence of query times.

# Chapter 6

# Conclusion and Further Research

In this thesis, we explored the NNS problem and the challenges of designing a data-driven data structure that also provides theoretical guarantees. In this chapter, we summarize our theoretical and practical results. We then discuss our work to provide a comprehensive overview of the challenges, key concepts, and a critical assessment. Finally, we outline potential directions for future research.

## 6.1 Contributions

Our approach to bridging the gap between a data-driven structure and theoretical guarantees involves applying our data-driven method as long as the query distribution $D_Q$ remains sparse relative to the point set $P$. However, when the distribution is no longer sparse and guarantees cannot be ensured, we switch to the bucketing method, a well-known technique that provides guarantees. This strategy allows our structure to stay data-driven while maintaining theoretical guarantees.

Since our structure is fundamentally a binary tree, we needed to determine the method for dividing the point set at each vertex. We introduced $P-balanced-D_Q-sparse$ separators, which are essentially ring separators designed to uphold the theoretical guarantees. Specifically, we prove that for $(\alpha, r)$-sparse query distributions (meaning that for every ball of radius $r$, the accumulated probability is at most $\alpha$), there exists a ring separator (of width $2r$) that splits the point set $P$ into two subsets, each containing at least a $1/(c+1)$ of the original, where $c$ is the doubling constant of the space our point set resides in. Additionally, the accumulated probability from the query distribution on the separator is at most $2\alpha^{O(1/d)}$.

Moreover, as we strive for a data-driven approach, we aim to find the optimal separator from among all those that meet the aforementioned guarantees. As this is an optimization problem and not any known efficient algorithmic paradigms seem applicable, we propose a heuristic algorithm inspired by local search which runs in $O(n^2 d \log n)$ time, where $n$ is the size of the point set $P$ and $d$ is the number of dimensions.

By incorporating these separators on our structure, we are able to provide guarantees

that are competitive to other solutions for low-dimensional NNS instances. Specifically, we show an $O(n^{1-1/d}c\log n)$ expected query time, $O(n)$ space complexity, and $O(n^2dc\log^2 n)$ preprocessing time. These results pertain to the purely data-driven model when the query distribution satisfies the sparsity property. The hybrid structure, which also incorporates bucketing, requires slightly more space.

In our experiments with the MNIST dataset, we applied preprocessing steps and built multiple tree instances with varying sample sizes to evaluate the model's adaptability. Our findings revealed that preprocessing time increased faster than linearly with sample size, due to its inherent complexity, while query time showed sublinear growth relative to the point set size. By integrating the remaining points into the trees constructed with different sample sizes, we assessed their performance on the entire dataset. The results indicated that query times decreased rapidly with increasing sample sizes, demonstrating efficient tree adaptation.

We identified a sample size of around 10,000 as the optimal balance. Beyond this, preprocessing time became excessively large with only marginal improvements in query time. For instance, doubling the sample size from 10,000 resulted in a mean query time reduction of just 4 microseconds, while preprocessing time increased by over 16 hours. As sample size increased, the mean query time for the full query set and the sample queries converged to the same value. Our results suggest that a tree built on 10,000 points is sufficient to achieve efficient query performance for the entire dataset, with the mean query time likely falling between 8.7 and 10.0 microseconds.

## 6.2 Discussion

In this section, we discuss key aspects of our work that are important for understanding our model. These points highlight the model's strengths and vulnerabilities, helping the reader determine its suitability for specific applications.

To start, we address that the model is best suited for low-dimensional spaces. The expected query time, $O(n^{1-1/d}c\log n)$, is sublinear; however, as the number of dimensions increases, the factor $n^{1-1/d}$ approaches $n$. Additionally, an exponential factor is hidden in $c$, the doubling constant, which is $c \leq 2^{O(d)}$. These are worst-case guarantees, and the model will be faster in practice, particularly if the balance is good (directly affecting the tree height $h = c\log n$, and thus the query time $O(n^{1-1/d} \cdot h)$). Nonetheless, this is still insufficient to ensure the model's efficiency in high-dimensional spaces.

Additionally, to achieve this query time, the query distribution $D_Q$ is considered to be $(1/n, r)$-sparse. To visualize this, imagine that the point set and the queries follow the same distribution, making the point set a good sample of the distribution. $(1/n, r)$-sparsity means that in every ball of radius $r$, there is at most one point, and therefore all points are at least $2r$ apart. This is a very strong assumption. However, our heuristic algorithm does not directly rely on the sparsity of the distribution and can still find an efficient separator even if the distribution is not $(1/n, r)$-sparse.

Regarding the heuristic algorithm, it has a time complexity of $O(n^2 d \log n)$, which, while polynomial, is still impractical for large datasets. For example, a point set with a million points in a 10-dimensional space would result in approximately $2 \cdot 10^{14}$ operations in the worst case, requiring even powerful computers several days to complete - and this is only for the root vertex. However, as the size of the subsets decreases, the running time will also decrease significantly. To mitigate this, we take a sample from the point set and find the separator on that sample. According to theory, this approach will yield a good approximation of an optimal separator, improving as the sample size increases.

On the other hand, our model exemplifies a hybrid structure by integrating two distinct methods: the data-driven approach we designed and bucketing. The key advantage is the ability to switch between these techniques, allowing us to harness the strengths of both without incurring any penalties. This flexibility provides numerous options and enables the creation of various variations by modifying one or both techniques.

Another significant aspect of our data-driven structure, setting it apart from other existing solutions, is its guaranteed query success rate of 1. This means that if a query has a near neighbor (or an approximate near neighbor), our search will definitely find it. However, this comes at the cost of increased query time. Additionally, our data-driven structure (the binary tree without bucketing) is constructed independently of the approximation factor $(1 + \epsilon)$. This allows us to dynamically perform exact near neighbor searches or approximation queries with different $\epsilon$ values. Despite this flexibility, the running time remains the same, which may lead some users to prefer sticking to exact near neighbor queries.

Regarding the experiments, we must consider whether our test conditions represented an ideal point set and queries. For more challenging point sets and varying search radii, the results could differ significantly. Despite this, our findings demonstrate that our model offers a competitive solution, performing exceptionally well on appropriate point sets.

Additionally, we demonstrated that for this dataset, consisting of 60,000 points, it was sufficient to build a tree using just one sixth of the data. For larger datasets, ranging from hundreds of thousands to millions of points, the appropriate sample size required to accurately capture the dataset's structure and maintain practical preprocessing times remains uncertain. For instance, with our current dataset, preprocessing a sample size of 10,000 took approximately 6 hours, while increasing the sample size to 20,000 and 30,000 extended preprocessing times to around 22 hours and 43 hours, respectively.

Lastly, we reflect on the fairness of comparing our model's results with those of linear search, as the latter is inherently slower. It might be more appropriate to compare our preprocessing and query times with probabilistic models, even though we are likely to be slower. Our goal would be to achieve comparable times without significantly lagging behind. Despite this consideration, the substantial performance difference still reassures us that our model is both effective and competitive.

## 6.3 Further Research

There are many opportunities for improvement and further exploration. First, efficiently extending the model to higher dimensions remains an open challenge. One approach could be to retain the tree structure but modify the separators, such as using hyperplanes. Additionally, research could explore adapting Andoni's method [1] from Hamming space to Euclidean space. Andoni suggests that this adaptation is feasible if the number of hash functions (or separators, in our case) is limited to $poly(d)$. An interesting direction for future work would be to identify the most suitable separators, given that our discretization method provides a significantly larger number of potential separators

Another area for consideration is developing a different method to address the non-sparse regions of distributions. This technique could leverage the density of the distribution to achieve better results. By replacing the bucketing method in our model with this new technique, we could create a fully data-driven model that still maintains theoretical guarantees.

Finally, exploring alternative methods for discretizing the separator space or devising a completely different approach for finding the optimal separator could be valuable. This might involve developing a better heuristic or even finding an exact solution. If these options are not feasible, proving the hardness of the problem would also be a significant contribution.

# Bibliography


[1] A. Andoni and D. Beaglehole, «Learning to hash robustly, guaranteed», in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 599–618. [Online]. Available: https://proceedings.mlr.press/v162/andoni22a.html.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, «An optimal algorithm for approximate nearest neighbor searching fixed dimensions», *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998, ISSN: 0004-5411. DOI: 10.1145/293347.293348. [Online]. Available: http://doi.acm.org/10.1145/293347.293348.

[3] S. Har-Peled and M. Mendel, «Fast construction of nets in low dimensional metrics, and their applications», in *Proc. 21st Annual Symp. Computational Geometry*, ser. SCG'05, Pisa, Italy, 2005, pp. 150–158, ISBN: 1-58113-991-8. DOI: 10.1145/1064092.1064117. [Online]. Available: http://doi.acm.org/10.1145/1064092.1064117.

[4] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, «Efficient search for approximate nearest neighbor in high dimensional spaces», *SIAM J. Comput.*, vol. 30, no. 2, pp. 457–474, 2000. DOI: 10.1137/S0097539798347177. [Online]. Available: http://dx.doi.org/10.1137/S0097539798347177.

[5] R. Panigrahy, «Entropy based nearest neighbor search in high dimensions», in *Proc. 17th Annual ACM-SIAM Symp. Discrete Algorithms*, ser. SODA'06, Miami, USA, 2006, pp. 1186–1195, ISBN: 0-89871-605-5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1109557.1109688.

[6] A. Andoni and P. Indyk, «Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions», *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008, ISSN: 0001-0782. DOI: 10.1145/1327452.1327494. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327494.

[7] D. R. Karger and M. Ruhl, «Finding nearest neighbors in growth-restricted metrics», in *Proc. 34th Annual ACM Symp. Theory of Computing*, ser. STOC'02, Montreal, Canada, 2002, pp. 741–750, ISBN: 1-58113-495-9. DOI: 10.1145/509907.510013. [Online]. Available: http://doi.acm.org/10.1145/509907.510013.

[8] H. Jegou, M. Douze, and C. Schmid, «Product quantization for nearest neighbor search», *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2010.57.

[9] P. Indyk and A. Naor, «Nearest-neighbor-preserving embeddings», *ACM Trans. Algorithms*, vol. 3, no. 3, 2007, ISSN: 1549-6325. DOI: 10.1145/1273340.1273347. [Online]. Available: http://doi.acm.org/10.1145/1273340.1273347.

[10] S. Meiser, «Point location in arrangements of hyperplanes», *Inf. Comput.*, vol. 106, no. 2, pp. 286–303, 1993, ISSN: 0890-5401. DOI: 10.1006/inco.1993.1057. [Online]. Available: http://dx.doi.org/10.1006/inco.1993.1057.

[11] A. Beygelzimer, S. Kakade, and J. Langford, «Cover trees for nearest neighbor», in *Proc. 23rd Intern. Conf. Machine Learning*, ser. ICML'06, Pittsburgh, USA, 2006, pp. 97–104, ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143857. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143857.

[12] P. Indyk and R. Motwani, «Approximate nearest neighbors: Towards removing the curse of dimensionality», in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, J. S. Vitter, Ed., ACM, 1998, pp. 604–613. DOI: 10.1145/276698.276876. [Online]. Available: https://doi.org/10.1145/276698.276876.

[13] S. Arya, T. Malamatos, and D. M. Mount, «Space-time tradeoffs for approximate nearest neighbor searching», *J. ACM*, vol. 57, no. 1, 1:1–1:54, 2009, ISSN: 0004-5411. DOI: 10.1145/1613676.1613677. [Online]. Available: http://doi.acm.org/10.1145/1613676.1613677.

[14] A. Andoni, T. Laarhoven, I. Razenshteyn, and E. Waingarten, «Optimal hashing-based time-space trade-offs for approximate near neighbors», in *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA)*, Also as arxiv.org/abs/1608.03580, 2017.

[15] M. Aumüller and M. Ceccarello, «Recent approaches and trends in approximate nearest neighbor search, with remarks on benchmarking», *IEEE Data Eng. Bull.*, vol. 46, no. 3, pp. 89–105, 2023. [Online]. Available: http://sites.computer.org/debull/A23sept/p89.pdf.

[16] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, «A survey on locality sensitive hashing algorithms and their applications», *CoRR*, vol. abs/2102.08942, 2021. arXiv: 2102.08942. [Online]. Available: https://arxiv.org/abs/2102.08942.

[17] S. Har-Peled, P. Indyk, and R. Motwani, «Approximate nearest neighbor: Towards removing the curse of dimensionality», *Theory Comput.*, vol. 8, no. 1, pp. 321–350, 2012. DOI: 10.4086/TOC.2012.V008A014. [Online]. Available: https://doi.org/10.4086/toc.2012.v008a014.

[18] A. Z. Broder, «On the resemblance and containment of documents», in *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, B. Carpentieri, A. D. Santis, U. Vaccaro, and J. A. Storer, Eds., IEEE, 1997, pp. 21–29. DOI: 10.1109/SEQUEN.1997.666900. [Online]. Available: https://doi.org/10.1109/SEQUEN.1997.666900.

[19] S. Har-Peled, «A replacement for voronoi diagrams of near linear size», in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, IEEE Computer Society, 2001, pp. 94–103. DOI: 10.1109/SFCS.2001.959884. [Online]. Available: https://doi.org/10.1109/SFCS.2001.959884.

[20] J. Wang, W. Liu, S. Kumar, and S. Chang, «Learning to hash for indexing big data - A survey», *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, 2016. DOI: 10.1109/JPROC.2015.2487976. [Online]. Available: https://doi.org/10.1109/JPROC.2015.2487976.

[21] M. Wang, X. Xu, Q. Yue, and Y. Wang, «A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search», *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 1964–1978, 2021. DOI: 10.14778/3476249.3476255. [Online]. Available: http://www.vldb.org/pvldb/vol14/p1964-wang.pdf.

[22] Y. A. Malkov and D. A. Yashunin, «Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, 2020. DOI: 10.1109/TPAMI.2018.2889473. [Online]. Available: https://doi.org/10.1109/TPAMI.2018.2889473.

[23] S. J. Subramanya, Devvrit, H. V. Simhadri, R. Krishnaswamy, and R. Kadekodi, «Rand-nsg: Fast accurate billion-point nearest neighbor search on a single node», in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 13 748–13 758. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Abstract.html.

[24] I. Azizi, K. Echihabi, and T. Palpanas, «Elpis: Graph-based similarity search for scalable data science», *Proc. VLDB Endow.*, vol. 16, no. 6, pp. 1548–1559, 2023. DOI: 10.14778/3583140.3583166. [Online]. Available: https://www.vldb.org/pvldb/vol16/p1548-azizi.pdf.

[25] M. Dobson, Z. Shen, G. E. Blelloch, *et al.*, «Scaling graph-based ANNS algorithms to billion-size datasets: A comparative analysis», *CoRR*, vol. abs/2305.04359, 2023. DOI: 10.48550/ARXIV.2305.04359. arXiv: 2305.04359. [Online]. Available: https://doi.org/10.48550/arXiv.2305.04359.

[26] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin, ≪Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0)≫, *CoRR*, vol. abs/1610.02455, 2016. arXiv: 1610.02455. [Online]. Available: http://arxiv.org/abs/1610.02455.

[27] S. Har-Peled, *Geometric Approximation Algorithms* (Mathematical surveys and monographs). American Mathematical Society, 2011, ISBN: 9780821849118. [Online]. Available: https://books.google.gr/books?id=EySCAwAAQBAJ.

[28] A. G. Sanjoy Dasgupta, ≪An elementary proof of a theorem of Johnson and Lindenstrauss≫, *Random Struct. Algorithms*, vol. 22, no. 1, pp. 60–65, 2003, ISSN: 1042-9832. DOI: 10.1002/rsa.10073. [Online]. Available: http://dx.doi.org/10.1002/rsa.10073.

[29] J. Matoušek, ≪On variants of the Johnson-Lindenstrauss lemma≫, *Random Struct. Algorithms*, vol. 33, no. 2, pp. 142–156, 2008, ISSN: 1042-9832. DOI: 10.1002/rsa.v33:2. [Online]. Available: http://dx.doi.org/10.1002/rsa.v33:2.

[30] D. Achlioptas, ≪Database-friendly random projections: Johnson-lindenstrauss with binary coins≫, *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 671–687, 2003, Special Issue on PODS 2001, ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(03)00025-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000003000254.

[31] N. Ailon and B. Chazelle, ≪The fast johnson–lindenstrauss transform and approximate nearest neighbors≫, *SIAM Journal on Computing*, vol. 39, no. 1, pp. 302–322, 2009. DOI: 10.1137/060673096. eprint: https://doi.org/10.1137/060673096. [Online]. Available: https://doi.org/10.1137/060673096.

[32] W. Johnson and J. Lindenstrauss, ≪Extensions of lipschitz maps into a hilbert space≫, *Contemporary Mathematics*, vol. 26, pp. 189–206, Jan. 1984. DOI: 10.1090/conm/026/737400.

[33] U. Feige and M. Mahdian, ≪Finding small balanced separators≫, in *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '06, Seattle, WA, USA: Association for Computing Machinery, 2006, pp. 375–384, ISBN: 1595931341. DOI: 10.1145/1132516.1132573. [Online]. Available: https://doi.org/10.1145/1132516.1132573.

[34] M. Anthony and P. L. Bartlett, *Neural Network Learning - Theoretical Foundations*. Cambridge University Press, 2002, ISBN: 978-0-521-57353-5. [Online]. Available: http://www.cambridge.org/gb/knowledge/isbn/item1154061/?site%5C_locale=en%5C_GB.

[35] A. Andoni and I. Razenshteyn, ≪Tight lower bounds for data-dependent locality-sensitive hashing≫, in *Proc. Intern. Symp. on Computational Geometry (SoCG)*, 2016, 9:1–9:11. DOI: 10.4230/LIPIcs.SoCG.2016.9. [Online]. Available: http://dx.doi.org/10.4230/LIPIcs.SoCG.2016.9.

[36] R. Motwani, A. Naor, and R. Panigrahi, ≪Lower bounds on locality sensitive hashing≫, *SIAM J. Discrete Math.*, vol. 21, no. 4, pp. 930–935, 2007. DOI: 10.1137/050646858. [Online]. Available: http://dx.doi.org/10.1137/050646858.

[37] R. O'Donnell, Y. Wu, and Y. Zhou, ≪Optimal lower bounds for locality-sensitive hashing (except when $Q$ is tiny)≫, *ACM Trans. Comput. Theory*, vol. 6, no. 1, 5:1–5:13, Mar. 2014, ISSN: 1942-3454. DOI: 10.1145/2578221. [Online]. Available: http://doi.acm.org/10.1145/2578221.

[38] A. Andoni and I. Razenshteyn, ≪Optimal data-dependent hashing for approximate near neighbors≫, in *the Proc. 47th ACM Symp. Theory of Computing*, ser. STOC'15, 2015.

[39] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, ≪Practical and optimal LSH for angular distance≫, in *Proc. Advances in Neural Information Processing Systems 28: Annual Conf. Neural Information Processing Systems*, 2015, pp. 1225–1233. [Online]. Available: http://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance.

[40] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, ≪Locality-sensitive hashing scheme based on p-stable distributions≫, in *Proc. 20th ACM Symp. on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, 2004, pp. 253–262. DOI: 10.1145/997817.997857. [Online]. Available: https://doi.org/10.1145/997817.997857.

[41] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, ≪Beyond locality–sensitive hashing≫, in *Proc. Symp. Discrete Algorithms (SODA)*, 2014.

[42] M. Aumüller, T. Christiani, R. Pagh, and M. Vesterli, ≪PUFFINN: parameterless and universally fast finding of nearest neighbors≫, in *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, Munich/Garching, Germany*, M. A. Bender, O. Svensson, and G. Herman, Eds., ser. LIPIcs, vol. 144, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 10:1–10:16. DOI: 10.4230/LIPICS.ESA.2019.10. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2019.10.

[43] M. Bawa, T. Condie, and P. Ganesan, ≪LSH forest: Self-tuning indexes for similarity search≫, in *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, A. Ellis and T. Hagino, Eds., ACM, 2005, pp. 651–660. DOI: 10.1145/1060745.1060840. [Online]. Available: https://doi.org/10.1145/1060745.1060840.

[44] B. Zheng, X. Zhao, L. Weng, Q. V. H. Nguyen, H. Liu, and C. S. Jensen, ≪PM-LSH: a fast and accurate in-memory framework for high-dimensional approximate NN and closest pair search≫, *VLDB J.*, vol. 31, no. 6, pp. 1339–1363, 2022. DOI: 10.1007/S00778-021-00680-7. [Online]. Available: https://doi.org/10.1007/s00778-021-00680-7.

[45] T. Skopal, J. Pokorný, and V. Snásel, ≪Nearest neighbours search using the pm-tree≫, in *Database Systems for Advanced Applications, 10th International Conference, DASFAA 2005, Beijing, China, April 17-20, 2005, Proceedings*, L. Zhou, B. C. Ooi, and X. Meng, Eds., ser. Lecture Notes in Computer Science, vol. 3453, Springer, 2005, pp. 803–815. DOI: 10.1007/11408079_73. [Online]. Available: https://doi.org/10.1007/11408079%5C_73.

[46] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, ≪SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index≫, *Proc. VLDB Endow.*, vol. 8, no. 1, pp. 1–12, 2014. DOI: 10.14778/2735461.2735462. [Online]. Available: http://www.vldb.org/pvldb/vol8/p1-sun.pdf.

[47] X. Zhao, B. Zheng, X. Yi, *et al.*, ≪FARGO: fast maximum inner product search via global multi-probing≫, *Proc. VLDB Endow.*, vol. 16, no. 5, pp. 1100–1112, 2023. DOI: 10.14778/3579075.3579084. [Online]. Available: https://www.vldb.org/pvldb/vol16/p1100-zheng.pdf.

[48] Y. Lei, Q. Huang, M. S. Kankanhalli, and A. K. H. Tung, ≪Locality-sensitive hashing scheme based on longest circular co-substring≫, in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., ACM, 2020, pp. 2589–2599. DOI: 10.1145/3318464.3389778. [Online]. Available: https://doi.org/10.1145/3318464.3389778.

[49] X. Zhao, Y. Tian, K. Huang, B. Zheng, and X. Zhou, ≪Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces≫, *Proc. VLDB Endow.*, vol. 16, no. 8, pp. 1979–1991, 2023. DOI: 10.14778/3594512.3594527. [Online]. Available: https://www.vldb.org/pvldb/vol16/p1979-zhao.pdf.

[50] Y. Tian, X. Zhao, and X. Zhou, ≪DB-LSH: locality-sensitive hashing with query-based dynamic bucketing≫, in *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, IEEE, 2022, pp. 2250–2262. DOI: 10.1109/ICDE53745.2022.00214. [Online]. Available: https://doi.org/10.1109/ICDE53745.2022.00214.

[51] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, ≪Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces≫, *Proc. VLDB Endow.*, vol. 11, no. 8, pp. 906–919, 2018. DOI: 10.14778/3204028.3204034. [Online]. Available: http://www.vldb.org/pvldb/vol11/p906-arora.pdf.

[52] G. Guennebaud and B. Jacob, *Eigen v3*, 2001. [Online]. Available: https://eigen.tuxfamily.org/index.php?title=Main_Page.

# Appendix

## A.1 Insights on Sparse Distributions

In our analysis of separators, we utilize $(\alpha, r)$-sparse query distributions. A key property derived from these distributions is expressed in Equation 4.1. Here, we present a more careful analysis on how this property is derived and provide insights into the underlying intuition. We start with the inequality:

$$\alpha^{O\left(\frac{1}{\log c}\right)} \leq c^{\Theta\left(\log\left(\frac{1}{r'}\right)\right)}\alpha$$

By the definition of $O$ and $\Theta$, we get:

$$\exists\, \lambda, \mu \in \mathbb{R}^+ :\ \alpha^{\frac{\lambda}{\log c}} \leq c^{\mu \log\left(\frac{1}{r'}\right)}\alpha = \alpha r'^{-\mu \log c}$$

Then, by doing some manipulation we end up with:

$$r' \leq \alpha^{\frac{1}{\mu \log c}\left(1-\frac{\lambda}{\log c}\right)}$$

The only thing left to get the desirable result is to show that $\left(1 - \frac{\lambda}{\log c}\right) = O(1)$, which is equivalent to $\frac{\lambda}{\log c} < O(1)$. But since $\frac{\lambda}{\log c} = O\left(\frac{1}{\log c}\right)$, we are done. Thus, we proved that for $(\alpha, r')$-sparse distributions:

$$r' \leq \alpha^{\Theta(1/\log c)}$$

In conclusion, this type of distribution sparsity directly provides a bound on the search radius. Furthermore, if the bound for $r'$ is not applicable, we can confidently assert that the distribution is not sparse:

$$r' > \alpha^{\Theta(1/\log c)} \implies D_Q \text{ is NOT } (\alpha, r')\text{-sparse}$$

## A.2 Finding Separators Defined by Points

In this section we show how to compute a separator $\varrho(o, r_s)$ in $\mathbb{R}^d$ defined by $d+1$ points, distributed on 3 radii: $r_s$, $r_s + r$ and $r_s + 2r$. Let $p_1, p_2, \ldots, p_{d+1}$ be the points that define the separator $\varrho(o, r_s)$, and for each point $p_i$ its coordinates $p_i^1, p_i^2, \ldots, p_i^d$. In

addition, we define the variables $c_i$, one for each $p_i$, and set them according to which radii their corresponding point is on: 0 if it is on $r_s$, 1 if it is on $r_s + r$, 2 if it is on $r_s + 2r$. By the definition of the sphere:

$$\forall p_i,\ \sum_{j=1}^{d}(p_i^j - o^j)^2 = (r_s + c_i r)^2$$

W.l.o.g. we assume that $c_1 = 0$. If not, then we find a point $p_i$ that has $c_i = 0$, and we swap them with the first one. In case there is no point on the inner radius $r_s$, we find a point $p_i$ that is on the smallest radius, we swap it with the first, set the new $r_s' = r_s + c_i r$, and update all $c_j$ accordingly. Then by expanding the quadratic terms in each equation, subtracting the equation of the first point from the rest, and doing a little manipulation, we are lead to the following system of linear equations for the centre $o$:

$$M \cdot o = \Pi_1 + \Pi_2 \cdot r_s$$

where

$$M = \begin{bmatrix} (p_1^1 - p_2^1) & (p_1^2 - p_2^2) & \dots & (p_1^d - p_2^d) \\ (p_1^1 - p_3^1) & (p_1^2 - p_3^2) & \dots & (p_1^d - p_3^d) \\ \vdots & \vdots & \ddots & \vdots \\ (p_1^1 - p_{d+1}^1) & (p_1^2 - p_{d+1}^2) & \dots & (p_1^d - p_{d+1}^d) \end{bmatrix} \qquad o = \begin{bmatrix} o^1 \\ o^2 \\ \vdots \\ o^d \end{bmatrix}$$

$$\Pi_1 = \frac{1}{2}\begin{bmatrix} c_2^2 r^2 + \sum_{i=1}^{d}\left((p_1^i)^2 - (p_2^i)^2\right) \\ c_3^2 r^2 + \sum_{i=1}^{d}\left((p_1^i)^2 - (p_3^i)^2\right) \\ \vdots \\ c_{d+1}^2 r^2 + \sum_{i=1}^{d}\left((x_1^i)^2 (x_{d+1}^i)^2\right) \end{bmatrix} \qquad \Pi_2 = \begin{bmatrix} c_2 r \\ c_3 r \\ \vdots \\ c_{d+1} r \end{bmatrix}$$

This system has a unique solution if the determinant $|M|$ is not equal to 0. By solving the systems $M \cdot \alpha = \Pi_1$ and $M \cdot \beta = \Pi_2$, we can get that $o = \alpha + \beta \cdot r_s$. We can use again the equation of the first point to get:

$$\sum_{i=1}^{d}\left(p_1^i - \alpha^i - \beta^i \cdot r_s\right)^2 = r_s^2$$

which is actually just a quadratic equation:

$$d_1 r_s^2 + d_2 r_s + d_3 = 0$$

where

$$d_1 = \left[\sum_{i=1}^{d}(\beta^i)^2 - 1\right], \qquad d_2 = -2\left[\sum_{i=1}^{d}(x_1^i - \alpha^i)\beta^i\right], \qquad d_3 = \left[\sum_{i=1}^{d}\left(p_1^i - \alpha^i\right)^2\right]$$

The inner radius $r_s$ of the separator will be the bigger out of the two solutions. By substituting $r_s$ back to the equation of the center $o = \alpha + \beta \cdot r_s$, we compute the center. Now the separator $\varrho(o, r_s)$ is fully defined.

## A.3 Implementation Details

Our code is available on GitHub (https://github.com/ntua-el19709/ring_tree). The experiments were implemented in C++ and compiled using g++. For efficient matrix and vector computations, we used the Eigen library for C++ ([52]). The experiments were conducted on an AMD Ryzen Threadripper 3960X 24-Core Processor CPU @ 4.5 GHz, running without parallelization. Detailed running times can be found in the results section of the experiment.