

All-Near-Neighbor Search Among
High-Dimensional Data
via Hierarchical Sparse Graph Code Filtering

by

Alexandros-Stavros Iliopoulos

Department of Computer Science
Duke University

Date: _____

Approved:

Xiaobai Sun, Advisor

Pankaj K. Agarwal

Nikos Pitsianis

Carlo Tomasi

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2020

ABSTRACT

All-Near-Neighbor Search Among High-Dimensional Data
via Hierarchical Sparse Graph Code Filtering

by

Alexandros-Stavros Iliopoulos

Department of Computer Science
Duke University

Date: _____

Approved:

Xiaobai Sun, Advisor

Pankaj K. Agarwal

Nikos Pitsianis

Carlo Tomasi

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2020

Copyright © 2020 by Alexandros-Stavros Iliopoulos
All rights reserved except the rights granted by the Creative Commons
Attribution-Noncommercial Licence

Abstract

This thesis addresses the problem of all-to-all near-neighbor (all-NN) search among a large dataset of discrete points in a high-dimensional feature space endowed with a distance metric. Near-neighbor search is fundamental to numerous computational tasks arising in modern data analysis, modeling, and knowledge discovery. The accuracy and efficiency of near-neighbor search depends on the underlying structure of the dataset. In emerging data-driven analysis applications, the dataset is not necessarily stationary, the structure is not necessarily built once for all queries, and search is not necessarily limited to a few query points. To facilitate accurate and efficient near-neighbor search in a stationary or dynamic dataset, we make a systematic investigation of all-NN search at multiple resolution levels, with attention to several standing or previously overlooked issues associated with high-dimensional feature spaces.

The key contributions are outlined as follows. (i) We proclaim the inherent sparse distribution of a finite discrete dataset in a high-dimensional space. We demonstrate that exploiting this sparsity is pivotal to dispelling the so-called curse of dimensionality in theoretical analysis and practical algorithm design. (ii) We introduce the adjacency graph hierarchy (AGH) as an analysis apparatus as well as base infrastructure for efficient all-NN search algorithm design. (iii) We present an efficient AGH construction algorithm by directly and deterministically pinpointing adjacent nodes at each resolution level via sparse adjacency code filtering, in lieu of exhaustive

search within local node neighborhoods. We provide at the same time a statistical estimate of the significant gap in algorithm complexity between direct location of adjacent nodes and local search. (iv) We introduce an adjacency-sensitive, graph-based but decentralized approach for hierarchical partition of a discrete dataset, with potential advantage for accurate and efficient all-NN search in comparison to existing non-interacting partition strategies.

To my loved ones.

Table of Contents

Abstract	iv
List of Tables	xi
List of Figures	xii
List of Algorithms	xiv
List of Abbreviations	xv
Acknowledgements	xvi
1 Introduction	1
1.1 All-NN search: problem description	1
1.1.1 Neighborhood types	2
1.1.2 All-NN structure: a descriptive data model	3
1.1.3 Comprehensiveness and parsimony	5
1.2 All-NN graphs & matrices	5
1.2.1 Notation and conventions	6
1.2.2 Elementary properties	6
1.2.3 Inter-graph relationships & critical values	7
1.2.4 Supernodes & supernode neighbor graphs	8
1.3 Long-standing challenges	9
1.4 Dissertation organization	11

2	Overview of Related Work	13
2.1	A glimpse of research activity	14
2.2	All-near-neighbors search	15
2.2.1	Spatial partitions	18
2.2.2	Multiple partition trees	23
2.2.3	Locality sensitive hashing	25
2.2.4	Proximity graph traversal	26
2.2.5	Discussion	28
3	Spatial Partitions & Adjacency Graph Hierarchy	30
3.1	Sparsity-adaptive spatial partition	31
3.1.1	Recursive mid-point partition (RMP) tree	31
3.1.2	Inherent sparsity & adaptation to sparsity	32
3.1.3	“Qode”: spatial quantization code	34
3.2	Spatial adjacency of partition boxes	37
3.2.1	Level-wise adjacency lists	37
3.2.2	Level-wise far-neighbor interactions & compression	39
3.3	Adjacency lineage & qode relationships	40
3.3.1	Single dimension	41
3.3.2	Multiple dimensions	43
3.4	Adjacency graph hierarchy (AGH)	45
3.4.1	AGH components and properties	46
3.4.2	Measures for evaluating AGH construction	51
3.4.3	AGH construction algorithm with minimal cost	53
3.4.4	Two-generation adjacency filtering with local qode operations	57

4	Adjacency-Sensitive Partitions	61
4.1	Partially partitioned boxes	62
4.1.1	Mass-deficient boxes in uniform partitions	62
4.1.2	Partial mid-point partition	64
4.1.3	Partial partition codes	65
4.1.4	Adjacency relations between partially partitioned boxes	65
4.2	Adjacency-sensitive partition model	66
4.2.1	Global model	67
4.2.2	Local model	67
4.3	Partition and AGH construction algorithm	69
4.3.1	Local partition of a box	69
4.3.2	Level-wise partition and AGH construction	72
5	All-NN Search in Multi-Dimensional Space	74
5.1	Coordinated search of all range- r neighborhoods	75
5.1.1	Single-level box adjacency in range- r near neighbors (r NN) search	75
5.1.2	Multi-level refinement of search regions	77
5.2	Coordinated search of all k -nearest neighbors	79
5.2.1	Multiple RMP resolution levels	79
5.2.2	Algorithm for all- k NN search	81
6	All-NN Search in High-Dimensional Space	83
6.1	Dimension reduction	84
6.1.1	Random subspaces	85
6.1.2	Principal axes	86
6.1.3	Dimension partition	87
6.2	Coordinated all- k NN search across subspaces	90

7	Conclusions	93
A	Algorithm Pseudocodes	95
B	Collection of Scholarly Publications on Nearest-Neighbor Search	99
B.1	Collection method	99
B.2	Measures of growth and influence	102
	Bibliography	105

List of Tables

2.1	Summary of selected all- k -nearest neighbors (k NN) search algorithms	16
3.1	Comparison of adjacency graph hierarchy (AGH) construction complexity: adjacency code filtering and conventional dual-tree traversal	56
B.1	Basic measures of the three citation graphs S2-ORC, ALGO and APPL: number of articles, number of citation links, and time span.	101

List of Figures

1.1	Illustration of neighborhood types for near-neighbor (NN) search with a 2D dataset	3
1.2	All-near-neighbors (all-NN) search complexity phase diagram with respect to dataset size and dimensionality	10
2.1	Number of yearly publications related to NN search algorithms and applications.	14
3.1	Sparsity-adaptive recursive mid-point partition (RMP) tree over an example 1D dataset	36
3.2	Sparsity-adaptive RMP tree over an example 2D dataset	36
3.3	Illustration of adjacency lineage phases in 1D box codes	42
3.4	Illustration of adjacency graphs and edge propagation between two successive levels of a 2D adjacency graph hierarchy (AGH)	47
3.5	Multi-scale interaction sub-graphs of the AGH in the 2D fast multipole method (FMM)	47
3.6	Illustration of code-directed descent in 1D binary tree towards boundary-adjacent box pairs.	56
3.7	Example data structures for storing and filtering children boxes: binary trie and array of local codes	59
4.1	Axial slice of the thoracic CT image used in Example 4.1	63
4.2	Spatial scale and point mass distributions of leaf boxes in a uniform RMP tree over a dataset of 3×3 patches in a thoracic CT image	64
4.3	Illustration of adjacency-sensitive partition of a 2D box	71
5.1	Filtering of descendant sub-boxes during all-range- r near neighbors (r NN) search	78

5.2	Adjacent boxes and sub-box filtering for all- r NN search in a sparsity-adaptive 2D RMP tree	79
6.1	Volume of a Euclidean D -ball of fixed radius as dimension increases	84
B.1	Annual publication and growth rate of NN-search articles	103
B.2	Mutual influence of ALGO and APPL articles	104
B.3	Influence/impact measure and ranking by intra- and inter-citation counts	104

List of Algorithms

A.1	Pseudocode for recursive construction of the adjacency graph hierarchy (AGH) over a recursive mid-point partition (RMP) tree	96
A.2	Pseudocode for all- k -nearest neighbors (k NN) search using the adjacency graph hierarchy	97
A.3	Pseudocode for construction of an adjacency-sensitive partition graph	98

List of Abbreviations

- AGH** adjacency graph hierarchy.
- all-NN** all-near-neighbors.
- BBD** balanced box decomposition.
- FFT** fast Fourier transform.
- FIFO** first in, first out.
- FMM** fast multipole method.
- HSS** hierarchically semi-separable.
- JL** Johnson-Lindenstrauss.
- k NN** k -nearest neighbors.
- LSH** locality sensitive hashing.
- NN** near-neighbor.
- PCA** principal component analysis.
- r NN** range- r near neighbors.
- RDP** recursive dyadic partition.
- RMP** recursive mid-point partition.
- SVD** singular value decomposition.

Acknowledgements

I am eternally grateful to my advisor, Xiaobai, for her guidance throughout my academic journey at Duke. She introduced me to a variety of interesting problems, helped me chart ways for seeking solutions, and, most importantly, instilled in me a mindset to never stop questioning my own understanding of concepts both hard and simple. I must also thank Nikos, who was largely responsible for sparking my interest in efficient computation models and algorithms while I was still an undergraduate student in Greece. Nikos introduced me to Xiaobai and has remained important throughout my evolution as a researcher to this day. I thank Carlo and Pankaj for all of their input and support, not only as part of my committee but also outside of it. I am thankful to Abhishek Dubey, Dimitris Floros, Jun Hu, and Tiancheng Liu for their collaboration with me on several projects, past and ongoing.

I have no doubt in mind that I would not have made it through this in (almost) one piece if not for all the people I had the good luck to come across, without whom life would be painted in drab colors: in no particular order, Ios, Ben, Songchun, Bryan, Matthias, Rodrigo, Alex, Jub, Anlo, Cassi, Fotis, Mitsos, Alex, Nikos, Rizos, Foteini, Savvas, Hatzi, Alekaki—and many others that my brain is currently too addled to conjure. Most of all, I thank my parents, Christina and Vasilis, and my brother Kostas, for their undying love.

Introduction

All-near-neighbors (all-NN) search in a discrete data set has a fundamental role in the process of information extraction, learning, understanding, modeling and advancing our scientific knowledge, and ultimately making good use of our knowledge. We introduce in this chapter the all-near-neighbors (all-NN) search problem, its fundamental role in data analysis apparatus and applications, the theoretical and empirical components of all-NN search, and long-standing challenges to efficient and accurate solutions to it.

1.1 All-NN search: problem description

We describe in this section the all-NN search problem in three constituent parts. First, we specify and identify the input and output attributes of an all-NN search; second, we look at the structure or model of the data in terms of the all-NN output; and third, we comment on the comprehensiveness and parsimony of the search process itself.

1.1.1 Neighborhood types

The all-NN search problem may be specified as follows. We are provided with (\mathcal{X}, d) , where \mathcal{X} is a set of points in a feature space of D dimensions, equipped with a metric d . The dataset size $|\mathcal{X}| = N$ is large. The relative positions of the data points to each other are irregular. Such datasets are commonly referred to as point clouds. The all-NN search problem is to find for each point, $\mathbf{x} \in \mathcal{X}$, its neighbors according to a neighborhood specification.

Two types of neighborhoods are prevalent. A neighborhood of the first type is specified by a fixed radius $r > 0$ and contains all points which lie within distance r from each $\mathbf{x} \in \mathcal{X}$:

$$\mathcal{N}_r(\mathbf{x}) = \{\mathbf{x}' \in \mathcal{X} \mid d(\mathbf{x}, \mathbf{x}') \leq r\}. \quad (1.1)$$

The neighborhood of the second type is specified by a fixed integer number $k > 0$ and contains the k points that are nearest to each $\mathbf{x} \in \mathcal{X}$:

$$\begin{aligned} \mathcal{N}_k(\mathbf{x}) &= \{\mathbf{x}\} \cup \{\mathbf{x}'_j\}_{j=1}^k, \\ \text{s.t. } \max_{\mathbf{x}' \in \mathcal{N}_k(\mathbf{x})} d(\mathbf{x}, \mathbf{x}') &\leq \min_{\mathbf{x}'' \in \mathcal{X} \setminus \mathcal{N}_k(\mathbf{x})} d(\mathbf{x}, \mathbf{x}''). \end{aligned} \quad (1.2)$$

By definition, the host or reference point \mathbf{x} is included in its neighborhood. When we refer to point neighborhoods, the host point may or may not be included, depending on the context. By (1.1) and (1.2), we can specify a all-NN search as either a range- r near neighbors (r NN) or a k -nearest neighbors (k NN) search. The two types of neighborhoods are illustrated in Figure 1.1 with an example 2D dataset.

We assume in the rest of the document that the axial weights in the distance function are equalized by axis-wise scaling. For an L_p distance function $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$, we use the following form of the p -norm:

$$\|\mathbf{x}\|_p = \left(\frac{1}{D} \sum_{i=1}^D |x_i|^p \right)^{1/p}, \quad p \geq 1; \quad \text{and} \quad \|\mathbf{x}\|_\infty = \max_{1 \leq i \leq D} |x_i|. \quad (1.3)$$

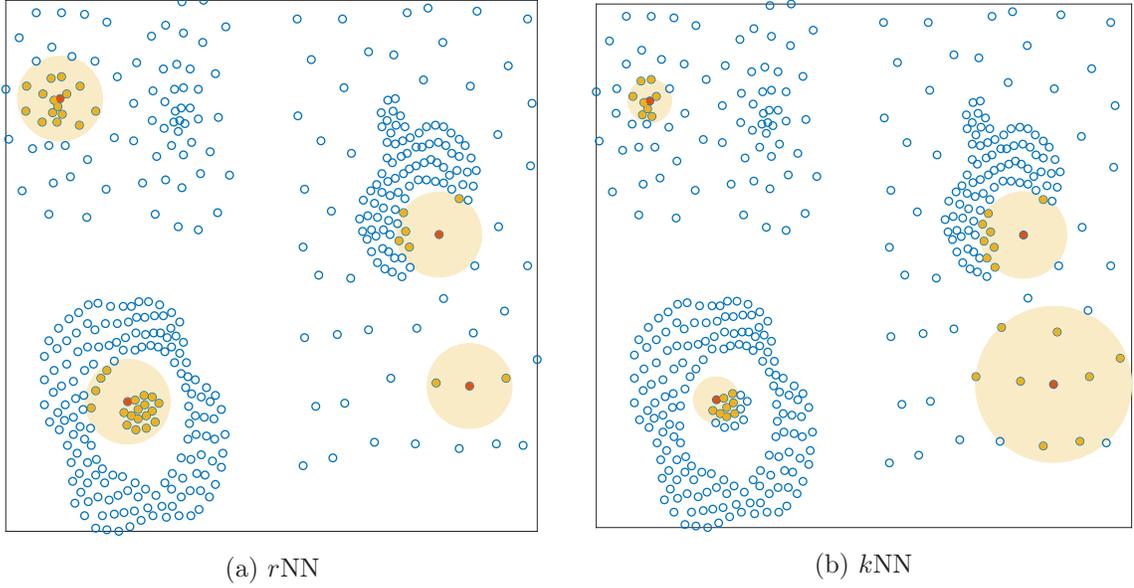


FIGURE 1.1: Illustration of neighborhood types for near-neighbor (NN) search with a 2D dataset: r NN (see (1.1)) and k NN (see (1.2)), with respect to Euclidean distance. The neighborhoods of 4 host points are shown for either case. Host points are marked in red and their respective neighbors are marked in yellow; the “NN ball” is also drawn around each host point. In the case of r NN search, the NN ball radius is constant and equal to r . In the case of k NN search, the NN ball radius around each host point is equal to the distance from the host point to its k -th neighbor. (Data source: <https://github.com/deric/clustering-benchmark>, “compound” dataset.)

That is, the p -norm of a point or vector is the generalized mean (or power mean) of its dimension-wise elements, with equal weights $1/D$. In particular, $\|\mathbf{x}\|_1$ is the arithmetic mean of element-wise magnitudes $|x_i|$. By the generalized mean inequality [Bul03],

$$\|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_p \leq \|\mathbf{x}\|_{p+1} \leq \|\mathbf{x}\|_\infty. \quad (1.4)$$

We make frequent use of the extremal norms, namely, $\|\cdot\|_1$ and $\|\cdot\|_\infty$, in r NN search.

1.1.2 All-NN structure: a descriptive data model

The output of an all-NN search is occasionally described as field-valued in order to be differentiated from scalar-valued output. Beyond the category of data types, we look into the structure with which the all-NN search output is organized and rendered. The structure shall be regarded, in our opinion, as a descriptive model

of the point data, with their local and global inter-relationships described in terms of neighbors at multiple resolution scales. The (model) structure and properties are crucial to effective and extensive use of the all-NN results, including the support and validation of predictive and prescriptive models.

This gravity of the all-NN structure is evident by the traditional example of the connection and distinction between all-NN search and the search of near-neighbors for (arbitrary) query points. We term the latter search as query near-neighbor search, or simply query search, to qualify the otherwise ambivalent term “search” often used in the literature. In a query search, given a corpus or a training data set $\mathcal{X} \subset \mathbb{R}^D$ in some feature space, we are also presented with a query point $\mathbf{q} \in \mathbb{R}^D$ in the same feature space, or a set \mathcal{Q} of multiple query points. The query search task is to find for each $\mathbf{q} \in \mathcal{Q}$ its neighbors within the dataset \mathcal{X} . The query search complexity depends solely on how \mathcal{X} is structured in the so-called pre-processing, indexing, or learning stage. Increasingly, an all-NN search in the data cloud \mathcal{X} is used at the core of the pre-processing stage [HAS⁺11; JZH⁺14; MPL⁺14; FC16; FWC18; MY18].

Emerging applications of all-NN search impose the practical and pressing demand that the pre-processing stage itself be fast, besides and beyond fast query search. All-NN search and query search are intimately related. In all-NN search we locate for every data point $\mathbf{x} \in \mathcal{X}$ its neighbors among the rest of the dataset, $\mathcal{X} \setminus \{\mathbf{x}\}$. This one-to-the-rest search is essentially a query search. A fundamental difference, however, is in the presence or absence of an index structure over \mathcal{X} that supports efficient neighbor search, and on the fact that such a structure need not support neighbor search for arbitrary points not in \mathcal{X} . In other words, with the new demand we face a chicken-and-egg situation between neighbor search and search structure, and hence more challenges. On the other hand, an index structure or algorithm for all-NN search need not support neighbor search for arbitrary query points not in \mathcal{X} , which presents an additional opportunity towards efficient solutions to all-NN

search.

1.1.3 *Comprehensiveness and parsimony*

We are interested in all-NN search processes that are comprehensive and parsimonious. By comprehensive, we mean that the search process is reliable and adaptive to changes in size, dimension and distribution of data, and to changes in the type (r NN or k NN) of neighborhoods, not ad hoc to a particular search on a particular dataset. By parsimonious, we mean that redundancy in data structure representation and search computations is kept to a minimum. We hold the parsimony principle in high regard for all-NN search in data within a probable size scope. We comment on the scope of probable data size in Section 1.3, with respect to physical, biological and social data, or in short, real-world data, in digital form, at present and in a perceivable future. We find that the probable scope of data size is not well aware of, or well respected, as it should be, in existing algorithms for all-NN search in high-dimensional data. In other words, many existing algorithms fail the parsimony criterion when compared with the baseline algorithm on datasets within probable size range.

The baseline, or brute-force, algorithm is a naive approach to all-NN search. It locates the near neighbors of each data point independently via an exhaustive search over the entire dataset. In total, it makes $O(N^2)$ pairwise distance evaluations, i.e., $O(DN^2)$ arithmetic operations in distance calculations. By exploring and utilizing structural relationships among the data points, a parsimonious search process is expected to reduce redundant operations and minimize the search cost. We will describe in Section 1.3 the main barriers in reducing the all-NN search complexity.

1.2 All-NN graphs & matrices

We describe and analyze all-NN search algorithms and data structures in terms of neighbor graphs and their respective adjacency matrices. The (1.1) or (1.2) specifi-

cation dictates the local structure of one-to-many neighborhood relationships, while the neighbor graph or corresponding adjacency matrix captures the global structure of all-to-all relationships or interactions.

1.2.1 Notation and conventions

Let $\mathcal{X} \subset \mathbb{R}^D$ be a set of feature points. The result of an all-NN search can be described by a weighted neighbor graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where the vertex set \mathcal{V} identifies with the point set \mathcal{X} and the edge set describes the near-neighbor relationships. The weight, defined on the edges, is a function of the distance. Often, the weight decays as the distance increases. In other words, a lower weight on an edge between two feature points indicates dissimilarity, while a higher weight indicates similarity. Sometimes, the weight function is the characteristic function of the edge set, i.e., the graph is essentially unweighted. By the neighborhood type, we specify the neighbor graph as either an r NN graph $G_r = (\mathcal{V}, \mathcal{E}_r)$ or a k NN graph $G_k = (\mathcal{V}, \mathcal{E}_k)$. A graph is identified with its adjacency matrix. In particular, we denote by \mathbf{A}_r and \mathbf{A}_k the adjacency matrices of graphs G_r and G_k , respectively.

In the extreme case, the neighborhood is all-inclusive and trivial, i.e., $r = 1$ or $k = N - 1$. The r NN and k NN graphs then become cliques, and they are one and the same if the same weight function is employed. The corresponding adjacency matrix is symmetric and full, with elements defined by the weight function over pairwise distances. We are concerned with neighbor graphs of non-trivial neighborhoods.

1.2.2 Elementary properties

We note here certain elementary properties of non-trivial neighbor graphs. We introduce first our convention about the neighbor graph edges and their corresponding elements in corresponding adjacency matrices. We consider a graph $G = (\mathcal{V}, \mathcal{E})$ to be *sparse* if $|\mathcal{E}| = O(N)$ or $|\mathcal{E}| = O(N \log N)$, where $N = |\mathcal{V}|$.

For a k NN graph G_k , the outgoing edges from vertex $\mathbf{x}_i \in \mathcal{V}$ reach the k -nearest neighbors of \mathbf{x}_i , and the incoming edges to \mathbf{x}_i are from the vertices that rank \mathbf{x}_i within their k -nearest neighbors. In matrix \mathbf{A}_k , the nonzero elements in the i -th column correspond to the outgoing edges of \mathbf{x}_i , and the nonzero elements in the i -th row correspond to its incoming edges. The out-degree of G_k is regular with constant k , and the total number of edges is pre-determined, $|\mathcal{E}_k| = kN$. When $k \ll N$, matrix \mathbf{A}_k is sparse. The in-degree distribution is in general irregular. Matrix \mathbf{A}_k is not necessarily symmetric. The neighborhood range, i.e., the distance from a point to its k -th neighbor, varies among points.

For an r NN graph G_r , the dynamic range $[w_{\min}, w_{\max}]$ of edge weights, i.e., the numerical range of the nonzero elements of \mathbf{A}_r , is predetermined by r and the weight function. Matrix \mathbf{A}_r is symmetric, i.e., graph G_r can be seen as undirected, and generally irregular in degree. The r NN graph is not necessarily sparse even when r is relatively small. The neighborhood population varies among points. Some vertices may have empty neighborhoods (not counting the host vertex itself); that is, G_r may have isolated vertices and the \mathbf{A}_r may have zero rows/columns.

1.2.3 Inter-graph relationships & critical values

We introduce a concept of critical values in neighbor connectivity of the neighborhood graphs. For a particular set of feature points, all r NN and k NN graphs share the same vertex set, $\mathcal{V} = \mathcal{X}$, while they differ in their edge sets. For any non-trivial neighborhood specification, the neighbor graph may be disconnected, strongly connected, or weakly connected (the latter applies only to k NN graphs). If a graph, G_r or G_k , is connected, all of its super-graphs, $G_{r'}$ ($r' > r$) or $G_{k'}$ ($k' > k$), are connected.

With k NN graphs, if $k < k'$, then G_k is a proper sub-graph of $G_{k'}$. In other words, there is a nesting chain relations among k NN graphs. We say k_c is a critical value in

neighbor graph connectivity if G_{k_c} is weakly connected but G_k is disconnected with any $k < k_c$.

Proposition 1.1. The set of k NN graphs $\{G_k\}$, $G_k = (\mathcal{V}, \mathcal{E}_k)$ over a discrete set \mathcal{V} of points is an ordered set by the binary relationship $\mathcal{E}_{k'} \subset \mathcal{E}_k$ if $k' < k$. The critical value k_c in neighbor graph connectivity is unique.

With r NN graphs, if $r < r'$, then G_r is a sub-graph of $G_{r'}$. A radius value r_c is critical in connectivity if G_{r_c} is connected but G_r is disconnected with any $r < r_c$.

Proposition 1.2. The set of r NN graphs $\{G_r\}$, $G_r = (\mathcal{V}, \mathcal{E}_r)$ over a discrete set \mathcal{V} of points is a partially ordered set by the binary relationship $\mathcal{E}_{r'} \subseteq \mathcal{E}_r$. The critical value r_c in neighbor graph connectivity is unique.

Denote by r_k the maximal neighborhood radius in a k NN graph G_k , and by k_r the minimal neighborhood population in an r NN graph G_r .

Lemma 1.1. For graph G_r to be a super-graph of G_k , the following conditions are necessary:

$$r \geq r_k, \quad k \leq k_r. \quad (1.5)$$

Conversely, for graph G_k to be a super-graph of G_r , the following conditions are necessary:

$$r \leq r_k, \quad k \geq k_r. \quad (1.6)$$

A necessary condition for graph G_r to be connected is that $r \geq r_1$.

1.2.4 Supernodes & supernode neighbor graphs

The above apply straightforwardly to sub-graphs of a neighbor graph induced by any subset of the graph vertices. For clarity, we associate a supernode with a subset of the feature points. It is feasible and desirable to stitch and integrate, via neighbor

graphs defined on supernodes, neighbor graphs on vertex subsets for the purpose of constructing all-NN graphs on the entire dataset.

The vast majority of existing solutions to all-NN search, especially those that adhere to the divide-and-conquer paradigm, can be characterized as a particular way to define supernodes, construct neighbor graphs over supernodes, and connect intra-supernode neighborhoods to inter-supernode neighborhoods. This unifying perspective draws upon our literature study on all-NN search, a brief review of which can be found in Chapter 2. From this perspective, we investigate further in the rest of the dissertation the integration conditions and constructions for efficient, accurate, and robust all-NN search.

1.3 Long-standing challenges

As mentioned earlier, a simple solution to all-NN search is to compute all pairwise distances between points in a dataset and then scan them to keep, for each point, its neighbors according to a r NN or k NN specification. The cost of this brute-force approach is $O(DN^2)$, which is prohibitive for large datasets.

Early attempts towards efficient algorithms have focused on *exact* solutions (meaning that their output is the same as that of the brute-force method), using the geometric structure of points in a dataset and reduce the number of pairwise distances that must be evaluated to locate point neighborhoods [Ben80; Cla83; Vai89]. These algorithms have been effective in the low-dimensional case, but their cost is at least $O(2^D N)$ —and often higher than that—rendering them intractable in high-dimensional spaces. This is commonly referred to as an aspect of the “curse of dimensionality” [Bel61] in data analysis.

Indeed, as dimensionality increases, such algorithms eventually become more expensive than the brute-force method. This scaling relationship in data size and dimensionality with respect to the computational cost of exact all-NN search algo-

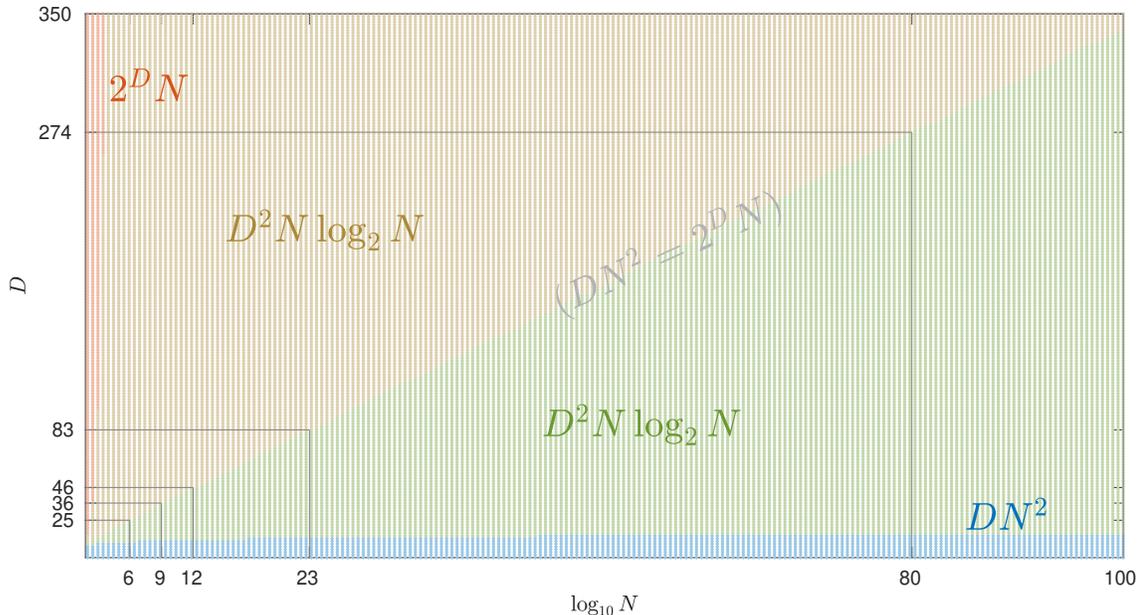


FIGURE 1.2: All-NN search complexity phase diagram with respect to dataset size and dimensionality. Three complexity factors are compared: $2^D N$ (exponential growth in exact search algorithms), DN^2 (exhaustive search), and $D^2 N \log_2 N$ (as representative of efficient algorithms for approximate search). The D - N parameter space where each complexity factor is more efficient than the other is annotated on the plot. Even with moderately high dimensionality, the dataset size must be exceedingly large for algorithms that scale exponentially with D to be preferable to brute-force search. Some specific sizes and corresponding dimensionality boundaries are highlighted: 1 million, billion, and trillion points (for reference, one of the largest databases today¹ is reported to contain around $6 \cdot 10^{15}$ bytes of data); while 10^{23} and 10^{80} corresponds to estimates for the total number of stars and atoms, respectively, in the known universe. Algorithms whose complexity scales poly-logarithmically with N and polynomially with D are more efficient in practically all cases of interest.

Figure 1.2, which plots the complexity boundary between the brute-force method and algorithms whose cost are exponential in dimensionality. Clearly, the latter are desirable only with data of relatively modest dimensionality. Based on the complexity boundary plot in Figure 1.2, we distinguish between *multi-dimensional* (exponential growth with D is preferable to quadratic growth with N) and *high-dimensional* (quadratic growth with N is preferable to exponential growth with D) spaces.

To address this challenge, research on NN search in more recent years has shifted largely towards *approximate* solutions, where the returned neighborhoods may con-

tain non-neighboring points. Ideally, these non-neighbors of a point are either still close to the exact NN-ball around it, or can be discarded by a post-processing step. Approximate algorithms can reduce the complexity all-NN with respect to D and N are clearly beneficial with any dataset of practical interest, as shown in Figure 1.2. Indeed, research in this direction has been crucial in mitigating the curse of dimensionality. Algorithms for approximate search may offer geometric or probabilistic guarantees (or both) regarding the quality of the results [AMN⁺98; AC09; WTF09; AIR18], or they may be heuristic [SH08; DML11; JDS11; MY18]. In practice, heuristic approaches are often shown empirically to outperform algorithms with sounder theoretical grounding [ABF17].

Among several remaining challenges in all-NN search are: (i) the development of robust and efficient methods that can adapt to different data distributions and accuracy objectives; (ii) a better understanding of the theoretical guarantees (or lack thereof) of commonly employed heuristics, and the cases where they are applicable or not; and (iii) a tight analysis of the computational cost of different algorithms, ideally in relation to data-dependent quantities that can be measured and verified in advance at low cost.

1.4 Dissertation organization

The rest of the dissertation is organized as follows. In Chapter 2, we give a brief overview of related work on all-NN search. In Chapter 3, we describe a class of regular partition trees to group data points into box supernodes and introduce the adjacency graph hierarchy (AGH), a set of inter-connected graphs that describe geometric relationships among boxes at multiple scales. We show how to utilize a binary coding scheme of partition boxes to obtain an optimal algorithm for constructing an

¹ This refers to the entire volume of the database of the World Data Centre for Climate (WDCC); see <https://www.comparebusinessproducts.com/fyi/10-largest-databases-in-the-world> (accessed on March 16, 2020).

AGH. In Chapter 4, we generalize the partition scheme to admit irregular boxes. We use the box adjacency structure and corresponding binary code operations to propose an adjacency-sensitive partition tree to promote sparsity in the AGH. In Chapter 5, we address all-NN search in multi-dimensional spaces and show how the AGH can serve as a skeleton for efficient all-NN search. In Chapter 6, we discuss several strategies for adapting our algorithms to all-NN search in high-dimensional spaces by considering feature subspace partitions. Finally, in Chapter 7, we offer concluding remarks and discuss remaining challenges and opportunities.

2

Overview of Related Work

The general concept of identifying near neighbors as a means to classification can be traced to nearly a millennium ago [Pel14]. A systematic and theoretical analysis of nearest neighbor methods for non-parametric classification [HP49; FH51; CH67] and density estimation [Ros56; Par62; LQ65; Cac66] started being established around the 1950s, laying the foundation for an array of statistical estimation and prediction techniques that continues to be useful today [CS18]. Over time, range- r near neighbors (r NN) and k -nearest neighbors (k NN) searches has become a core component in a diverse set of algorithms, employed in a wide variety of fields.

With the ever-increasing availability and ease of collection of data, computational tractability and efficiency became important considerations for near-neighbor search, which is often the bottleneck in processing pipelines that depend on it. Today, near-neighbor search is itself an active area of research, producing theoretical results and algorithms which impact the wider scientific community.

In this chapter, we review certain advances in near-neighbor search, focusing mainly on the development of algorithmic solutions and their properties. We will address aspects that are more closely related to the techniques presented in this

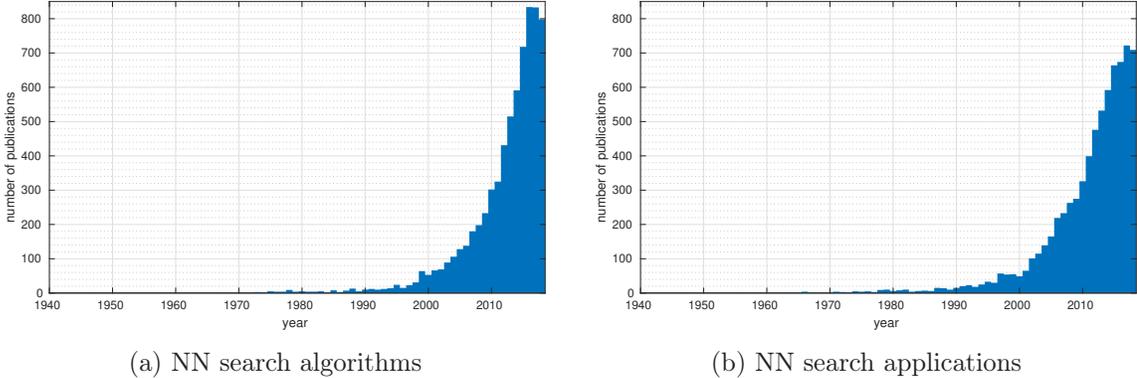


FIGURE 2.1: Number of yearly publications related to near-neighbor (NN) search algorithms and applications. The total number of publications in each plot is 7432 (algorithms) and 7660 (applications). Results were collected from the Semantic Scholar Open Research Corpus [AGB⁺18] (data snapshot as of November 1, 2019); see Appendix B for more details. (Plots courtesy of D. Floros and N. Pitsianis.)

thesis, and will also try to outline the scope of applications that directly benefit from near-neighbor search solutions.

2.1 A glimpse of research activity

Figure 2.1 shows a rough estimate of the number of published articles that are related to k NN search and applications over several decades. Relevant publications were collected from the Semantic Scholar Open Research Corpus [AGB⁺18], starting from a set of influential publications; details are found in Appendix B.

We observe a marked increase of research activity involving NN algorithms and applications starting around 2000. This appears to be fueled in part by the introduction of algorithms and analyses which address NN search in high-dimensional feature spaces. Previously existing solutions deteriorate fast with increasing dimensionality, eventually becoming inferior to a brute-force scan of the entire dataset—an aspect of the so-called “curse of dimensionality” [Bel61; Che09]. These advances have in turn enabled other algorithms and applications that were previously impractical due to the high computational cost of scanning a potentially very large dataset.

2.2 All-near-neighbors search

Any algorithm for query NN search can be used for the all-NN search problem: simply apply the algorithm N times, using every point in the dataset as a query point; the indexing structure that is used in the NN search algorithm is constructed once and reused for all queries. This approach, while certainly viable, does not take advantage of the fact that all-NN search is essentially a special case of query NN search, since there is no need to support efficient search for *any* query point in the feature space. Moreover, nearby data points are expected to share a number of neighbors.

A tabulated summary of the algorithms discussed in this section is shown in Table 2.1.

Table 2.1: Summary of selected all- k NN search algorithms.

Reference(s)	Structure	Accuracy*	Complexity	Summary notes
[Ben76; Ben80]	k -d tree	exact	$O(N \log^{d-1} N)$	multidimensional divide and conquer
[Cla83]	RMP tree	exact	$O(N(d \log N + c^d))$ [$k = 1$]	search adjacent leaf boxes
[Vai89]	RMP tree	exact	$O(k(cd)^d N \log N)$	like [Cla83], but each tree node spans the bounding box of contained points
[CK95]	fair-split tree	exact	$O(N(d \log N + k(cd)^{d/2}))$	find coarsest set of well-separated box pairs
[GM01; GM03; Cur15]	k -d/ball tree	exact	$O(N \log^{d-1} N)$	dual-tree traversal (group both query and reference points)
[BKL05; RLM ⁺ 09; CLM ⁺ 15]	cover tree	exact	$O(c_x^6 N(\log N + kc_x^3))$ [c_x : expansion const. [KR02]]	dual-tree traversal
[SSV07]	RMP tree	exact	$O(N(d \log N + c^d k \log k))$	find subsets of boxes which must contain neighboring points
[CK10]	Z-curve	exact	$O(N(d \log N + c^d k \log k))$	restrict search in contiguous subarray of points (in Z-curve order)
[CFS09]	PCA tree	approximate	$O(dN^\alpha k^2)$ [$1 < \alpha < 2$] [α depends on overlap extent]	divide and conquer with overlapping partitions; merge partition-wise neighborhoods of points in overlap region
[JOR11]	rand. RDP trees	approximate	$O(N(d \log d + k(\log N + k)(d + \log k)))$	fast random rotations and axis-wise nested partitions; search within boxes with Hamming distance ≤ 1

... Table 2.1, continued...

Reference(s)	Structure	Accuracy*	Complexity	Summary notes
[WWZ ⁺ 12]	rand. PCA trees	approximate	$O(dN(\log N + k \log k))$	divide and conquer; overlap across multiple randomized tree boxes
[FC16]	rand.	approximate	$O(dN(\log N + kp \log p))$ [p : #neighbor checks during refinement traversal]	like [WWZ ⁺ 12]; overlap achieved across multiple trees as well as by grouping adjacent boxes
[ZHG ⁺ 13]	LSH	approximate	$O(hN(\log N + d(m + b) + k) + dNk^2)$ [h : #hash tables] [m : hash code length] [b : block size]	divide and conquer with multiple hash tables; group points into equisized blocks by projecting and sorting hash codes on random axis; merge block-wise k NN sub-graphs across tables
[DML11]	proximity graph	approximate	$O(n_I N(dk^2 + k \log k))$ [n_I : #iterations]	initialize with random k -degree graph; iteratively update graph by traversing point-wise neighborhoods
[HD16]	proximity graph	approximate	$O(n_I dN \delta k' \log k')^\dagger$ [n_I : #iterations] [δ : diameter] [k' : max degree]	add edges to connect sampled point-pairs if corresponding paths are not found by local greedy search; edge-wise “occlusion rule” to prune edges while guaranteeing local greedy search converges (ideal case)

* Most exact algorithms with spatial partition tree search structures can be adapted to ϵ -approximate search.

† Simplified complexity. Diameter and max degree refer to the proximity graph while it is being constructed.

2.2.1 Spatial partitions

An early efficient solution to the all-near-neighbors (all-NN) search problem in low-dimensional spaces was given by Bentley [Ben76; Ben80]. He introduced k -d trees as a fundamental partition structure in an algorithmic framework he termed “multidimensional divide-and-conquer”. In brief, the latter states that, given a problem involving N points in d dimensions, one reduces it to two problems involving roughly $N/2$ points in d dimensions (divide) and a problem involving $N' < N$ points in $(d-1)$ dimensions (conquer), and solves them recursively. Bentley provided an algorithm for all- r NN search in d -dimensional space with complexity $O(dN \log N)$, with the assumption that every neighborhood contains $O(1)$ points, by showing that one can always find good axis-parallel partition hyperplanes in a k -d tree, in the following sense: the resulting partitions contain no more than $N(1 - 1/4d)$ points each and the “conquer” subset contains $N' = O(dN^{1-1/d})$ points. By a similar approach, he also provided an algorithm for all-NN search ($k = 1$) in $O(3^d N \log^{d-1} N)$ time. Due to the varying neighborhood range around each point, a partition property like for all- r NN search could not be guaranteed, leading to the exponential logarithmic factor. The 3^d factor reflects that each point may have at most as many equidistant nearest neighbors and is often much smaller in practice.

Structures such as k -d trees are population-balanced: the number of points contained in each box decreases exponentially with the tree level. This property ensures that the size of each divided sub-problem in a divide-and-conquer setting is controlled. A related but somewhat complementary approach is to consider partition hierarchies that are volume-balanced: the geometric size of each box decreases exponentially with the tree level. In such partitions, the relative placement of any two boxes serves to bound the pairwise distances of points contained therein, controlling the size of the conquer step in a divide-and-conquer setting. Neighbor-finding

problems then are largely reduced to finding geometrically adjacent boxes.

Clarkson presented an algorithm for the all-NN ($k = 1$) problem using a recursive mid-point partition (RMP) tree (or “hyper-octree”) [Cla83]. Given a d -dimensional dataset $\mathcal{X} \in [0, 1]^d$ (without loss of generality), the RMP tree recursively partitions the unit hypercube by axis-parallel hyperplanes that pass through its center, until each non-empty partition contains a single point. During this process, a list of nearby boxes and corresponding distance bounds are maintained for each box. This is done based on the principle that near boxes will either have the same parent box, or their respective parent boxes will be near each other; children of nearby parent boxes are scanned to discard the ones are not near each other. Clarkson proposed a randomized algorithm for compressing the tree structure, such that linear-path subtrees (where each box contains only one non-empty child box) are reduced to a single box, and showed how to adapt the above algorithm to the compressed tree structure. The resulting algorithm has an expected complexity of $O(c^d N \log N)$, for some constant c .

A similar algorithm for all- k NN with worst-time complexity $O(k(cd)^d N \log N)$ was developed by Vaidya [Vai89]. The partition tree of Vaidya is different from that of Clarkson in that every box containing a set of points \mathcal{P} is shrunk to the smallest cubic bounding box of \mathcal{P} before it is partitioned. The construction of such compressed trees was shown by Aluru and Sevilgen [AS99] to be possible in $O(dN \log N)$ time. This partition strategy guarantees that every internal tree node has at least two children, but means that box size is now variable. Every box B then maintains a list of boxes which may contain near neighbors of points in B , as well as a list of boxes whose points may have near neighbors in B .

A common feature of the above solutions is their exponential dependence on d , which severely hampers their applicability as feature dimensionality increases. Often, this exponential dependence in the worst case is because of the number of *potentially*

adjacent boxes to any box in the partition structure. We note, however, that such cases are impossible with high-dimensional data where $2^d \gg N$. A challenge in these algorithms is whether irrelevant box pairs can be discarded with as little work as possible to keep the *effective* dimension-dependent complexity factor within acceptable limits.

Callahan and Kosaraju [CK95] introduced fair-split trees, a partition hierarchy that bears similarities to both the k -d trees and RMP trees, and applied them to all- k NN search and N -body computations [CK95]. The data domain box is split along one dimension at a time, along the corresponding interval mid-point (or close to it). They show that the tree can be built in $O(dN \log N)$ time even if the splits are unbalanced in population mass, by a partial depth-first partition recursion into the partition that contains the most points after each split. Inspired by the fast multipole method (FMM) [Gre87; GR87], they parse the partition hierarchy to produce a “well-separated realization” of the tree. Roughly, this is a set of box pairs which covers all pairs of points in the dataset such that each point-pair is covered exactly once and the distance between every pair of boxes in the realization set is larger than the diameter of the point subset covered by each box, by some specified factor. Using the geometric relationship of boxes in the tree, they construct the realization in $O((cd)^{d/2}N)$ time. The all- k NN search algorithm, which entails an additional $O(k)$ factor, is akin to that of Vaidya [Vai89], using box adjacency and the well-separated realization to restrict the search to relevant box pairs.

Gray and Moore proposed a class of algorithms to extend single-query algorithms on partition tree hierarchies to grouped-query algorithms; they term these “dual-tree” algorithms [GM01]. The traversal proceeds with pairs of nodes such that, at every step, the search region may be contracted for all query points in a node using node-node distance bounds, rather than point-node ones. This was once again motivated by the FMM [Gre87; GR87], and applied to a number of statistical computation

problems, including and related to all- r NN and all- k NN search. Dual-tree algorithms initially employed a k -d tree partition structure. An abstraction was later proposed by Curtin et al. [CMR⁺13] to generalize the dual-tree approach to a wide range of partition trees and algorithms.

In recent years, dual-tree algorithms for all-NN search have primarily employed the cover tree structure [BKL05; BKL06]. The cover tree is a partition hierarchy with all data points as leaves, and a subset of points as nodes at every tree level such that each point is near its parent point and well-separated from other points at the same level; it can be constructed in $O(c_x^6 N \log N)$ time, where c_x is the expansion constant [KR02] of the dataset.¹ Ram et al. showed that all-NN search via dual-tree traversal on a cover tree takes $O(c_x^{16} N)$ [RLM⁺09] time. This bound was later tightened by Curtin et al. [CLM⁺15] to $O(c_x^9 N)$, under favorable conditions—specifically, as long as a measure of cover tree imbalance introduced by the authors grows linearly with N (in degenerate cases, imbalance can be proportional to N^2).

Focusing on the low-dimensional spaces (mainly 3D for graphics applications), Sankaranarayanan et al. [SSV07], and Connor and Kumar [CK10] proposed two algorithms for all- k NN search using octrees.

Sankaranarayanan et al. used an explicit tree data structure and separated the search into two distinct stages [SSV07]. First, they find a set of tree boxes which are guaranteed to contain the nearest neighbors of points in some query box; they term this set the “locality” of the query box. Then, the true neighbors are found by searching within the locality. Once a the neighbors of a point \mathbf{x} in the query box have been located, they attempt to reduce redundant computations for the next query point \mathbf{y} by setting $\mathcal{N}_k(\mathbf{x})$ as an initial estimate for $\mathcal{N}_k(\mathbf{y})$ and refining the search region for neighbors of \mathbf{y} using the triangle inequality.

¹ The expansion constant is a measure of intrinsic dimensionality, defined as the lowest ratio of the mass (number of contained points) of a ball of radius $2r$ to that of a ball of radius r centered at the same point, for any center point and radius [KR02; BKL06].

The notion of the locality of a box bears some similarity to the box adjacency lists and adjacency graph hierarchy (AGH) we use in this work (see Sections 3.2 and 3.4). Indeed, we may use the latter to identify a set of boxes which must contain the nearest neighbors of all points in some query box (e.g., Proposition 5.2). Two differences pertain to the way that the locality of a box is built and utilized in the algorithm of Sankaranarayanan et al. First, the locality is built anew for each leaf box, entailing redundant tree traversal steps, since the localities of adjacent boxes will in general have substantial overlap. Second, the requirement that the entire locality of some query box is found before any neighboring points are sought is unnecessarily strict, since the search region for any query point or box can be restricted on-the-fly as candidate neighbors are found. In this sense, the locality represents the worst-case scenario for the search region of a box. Moreover, the AGH can be used to support a variety of strategies for all- k NN search, as we show in Chapter 5.

Connor and Kumar used octrees implicitly by restricting k NN search to a contiguous subset of points in Z-curve order [CK10]. For each point with Z-order index i , an initial estimate of its k -nearest neighbors is found by scanning the points with indices ranging in $[i - O(k), i + O(k)]$. This gives an upper bound for the local k NN ball radius. Then, they find the smallest contiguous index interval which contains all points lying within the k NN ball estimate. The true neighbors are located by scanning the points in that interval.

Moving away from algorithms which support exact all- k NN search, Chen et al. proposed two variants of a method for approximate search among high-dimensional data using recursive dyadic partitions along spectral axes [CFS09]. At every step, a partition hyperplane is placed such that half the points lie on either side of it. The partition axis is selected to maximize the total sum of pairwise distances among all points in the current box, and can be computed easily by a truncated singular value decomposition (SVD). This strategy is expected to reduce the number of points

with neighbors on both sides of a partition, and was shown to be effective in quickly reducing the volume of each partition box as one descends in the tree [VKD09]. The partitioned boxes are not disjoint but overlapping. The domain of each box is extended beyond the partition hyperplane, such that a pre-fixed proportion of the points in the parent box are contained in the overlap region. The algorithm proceeds in divide-and-conquer fashion, conceptually similar to the k -d tree algorithm of Bentley [Ben80], but with a key difference in how the overlap regions are used in the “conquer” steps: first, a local k NN graph is constructed within each leaf box by brute-force search; points in overlap regions then have two sets of neighbor candidates, which are merge-sorted to keep the k NNs, thereby bridging the local graphs. This is done in all overlap regions all the way to the root of the tree, resulting in an approximate k NN graph. The algorithm complexity is shown to be $O(dN^\alpha k^2)$, where $\alpha \in (1, 2)$ depends on the overlap extent. Larger overlap generally improves the quality of the results, but also increases the computational cost of the algorithm.

2.2.2 Multiple partition trees

As with single-query NN search, a common technique for dealing with all-NN search in high-dimensional spaces involves the use of multiple partition trees over the dataset, typically with randomization in tree parameters or transformations of the dataset prior to tree construction.

Wang et al. [WWZ⁺12] employed this approach in a divide-and-conquer algorithm that shares many similarities with that of Chen et al. [CFS09]. The key difference between the two approaches lies in how neighborhood overlap across partitions is achieved. Wang et al. also use randomized local truncated SVDs to find a partition axis which approximately maximizes pairwise distances among points in a partition box; specifically, each SVD is computed with a randomly sampled subset of the points

in a box. A local k NN graph is built by brute force within each leaf. Neighborhood overlap then results among trees, since each leaf box is expected to contain a different subset of points. The k -nearest neighbors of each point are sought in the union of the point’s neighborhoods across all trees. Wang et al. found empirically that increasing the number of trees yields diminishing returns after a certain point. They used a post-processing step to refine the resulting k NN graph: starting from each point, the graph is traversed in best-first manner to find nearer neighbors than the ones that were initially identified. The total complexity of their algorithm is $O(dN(\log N + k))$, as long as the number of trees and refinement traversal steps are constant.

Roughly the same approach is used by Fu and Cai [FC16], but the points are partitioned via multiple randomized k -d trees. Fu and Cai bridge the partition-wise local k NN sub-graphs in two ways: by merging neighborhoods across trees, and by performing additional local searches in adjacent leaf nodes of each tree. The latter is done bottom-up, with a pre-fixed limit on the tree levels that are ascended for each leaf box. At each level, the search for neighbors of a point in the box at hand is restricted to a sub-region of its sibling box by finding the nearest leaf box in the sibling sub-tree. This process is carried out for each point. After an approximate k NN graph has been constructed, it is then traversed to refine the point neighborhoods. If the number of trees and the number of adjacent leaf boxes search for each point in each tree are constant, and the number of potential neighbors for each point that are checked during the refinement traversal is p , then the algorithm takes approximately $O(dN(\log N + kp \log p))$ time.

Jones et al. proposed an algorithm using multiple random rotations and a specialized type of recursive dyadic partition (RDP) trees [JOR11]. In their partition hierarchy, the splitting dimension is selected in round-robin fashion, i.e., the data are partitioned along the median in first dimension at level 1, the second dimension at level 2, etc. If the height of the tree is $\ell_{\max} < d$, this is tantamount to a partitioning

where boxes are vertices on a hypercube in an ℓ_{\max} -dimensional subspace (not all vertices are “occupied”, though). By the Johnson-Lindenstrauss (JL) lemma [JL82] pairwise distances among the rotated points are expected to be approximately preserved in each subspace. Rotations are computed via an approximate numerical factorization which leverages the fast Fourier transform (FFT). Within each subspace, the k -nearest neighbors of points in a leaf box are sought in all leaf boxes at Hamming distance at most 1 on the partition hypercube. After the subspace-wise k NN graphs have been built and merged, the resulting graph is refined by a depth-1 breadth-first traversal of point-wise neighborhoods. If the number of trees/subspaces is constant, the total complexity of the algorithm is $O(N(d \log d + k(k + \log N)(d + \log k)))$.

2.2.3 Locality sensitive hashing

Locality sensitive hashing (LSH) [IM98; Ind00; DII⁺04] provides strong probabilistic guarantees for near-neighbor search in high-dimensional spaces and has long been used for query k NN search. Zhang et al. proposed an algorithm for all- k NN search based on LSH [ZHG⁺13]. Their algorithm leverages a divide-and-conquer paradigm similar to the algorithm of Wang et al. [WWZ⁺12] discussed above, except that points are grouped via hashing rather than spatial partition trees.

In the algorithm of Wang et al. [WWZ⁺12], the high-dimensional data are first hashed onto length- m binary codewords. This can be done using any appropriate hashing method [IM98; DII⁺04; AI06; KG09; WTF09; WSS⁺14; AR15]. Typically, with LSH methods, the search would be restricted to points that fall in the same or nearby hash bins. It has been observed, however, that the bin population distribution is often highly irregular, in part due to the irregular distribution of pairwise distances among data points. To control the computational cost of the algorithm, the point-wise hash codes are projected onto a random line, sorted along the projection axis, and grouped into equisized contiguous blocks. A local k NN sub-graph

is then constructed among the points within each block by brute force, resulting in an approximate k NN graph with multiple disconnected components. This process is repeated a number of times, yielding different disconnected neighborhood graphs due to randomization in the hashing and projection functions. The k -nearest neighbors of each point are then sought in the union of its neighborhoods. The k NN graph is then refined by a depth-1 breadth-first traversal of each point’s neighbors; that is, checking the neighbors of its neighbors. The total cost of the algorithm is $O(hN(\log N + d(m + b) + k) + dNk^2)$, where h is the number of hash tables (i.e., separate neighborhood graphs), m is the code length in each hash table, and b is the block size (i.e., local k NN sub-graph size).

It should be noted that the probabilistic guarantees of LSH generally hold for points whose distance is within a certain range. In practice, it is common to get numerous hash tables corresponding to different distance ranges. Moreover, it is unclear how much the random projection of the hash codes in the above algorithm distorts the probabilistic guarantees of LSH. While points with the same hash code will obviously have contiguous sorted indices on the projection line, the preset-size blocking may separate nearby points while grouping far-away ones.

2.2.4 Proximity graph traversal

A feature of many of the algorithms discussed is that once an approximate k NN graph has been constructed, it is then traversed in order to refine the point-wise neighborhoods. The underlying argument is sometimes summarized as “the neighbors of my neighbors are likely to be my neighbors, too”. Traversing a graph whose edges capture spatial proximity among point-vertices has also long been employed towards query k NN search, where the query point need not belong in the vertex-set of the

proximity graph [Kle97; HAS⁺11; MY18].² While the algorithms we have seen so far only use this approach as a post-processing step after constructing an approximate graph based on some spatial partition hierarchy, some algorithms have been proposed which eschew data partitions and the divide-and-conquer paradigm altogether.

Dong et al. proposed an algorithm in which point neighborhoods are refined by iteratively traversing an approximate k NN graph, until no neighborhoods are updated (or a preset number of iterations is reached) [DML11]. The initial graph has the data points as vertices and random edges such the out-degree of each point is k . Both outgoing (neighbors) and incoming (reverse neighbors) are traversed during the iteration. Under certain conditions, the radius of the k NN neighborhood around each point is expected to be halved in a constant number of iteration steps, eventually converging to the true k NN neighborhood radius. The authors also discuss some practical considerations to reduce the computational cost of the algorithm, including an equivalent reformulation of the neighborhood update step, traversing a sampled subset of the edges incident on each point, and early termination when only a small fraction of the point-wise neighborhoods are updated. The complexity of the algorithm is $O(n_I N(dk^2 + k \log k))$, assuming an L_p distance metric and a total of n_I iteration steps.

Harwood and Drummond [HD16] observed that certain edges in the proximity graph can be removed while still guaranteeing that an iterative process like that of Dong et al. [DML11] will converge to the true neighborhoods (under ideal conditions). Thus, the cost of each iteration step can be reduced, possibly at the expense of an increased number of steps. Specifically, they proposed an “occlusion rule”, whereby an edge from point \mathbf{u} to point \mathbf{v} is occluded and can be removed if there exists

² We use “proximity graph” as an umbrella term for graphs that are used as the search index structure in k NN search algorithms, to differentiate them from “neighborhood graphs” (solutions to all-NN search). Such proximity graphs include approximate k NN graphs over all points in the dataset at hand, but there are also variants, such as small-world graphs, where edges may connect points that lie far from one another.

a point \mathbf{p} with an incoming edge from \mathbf{u} and an outgoing edge to \mathbf{v} such that $d(\mathbf{u}, \mathbf{p}), d(\mathbf{p}, \mathbf{v}) < d(\mathbf{u}, \mathbf{v})$. The rationale is that one can then get from \mathbf{u} to \mathbf{v} via \mathbf{p} by local greedy search, i.e., following the edge to the nearest current neighbor. The proximity graph is initialized with an empty edge-set and updated as follows. Point-pairs are sampled randomly and a path is sought between them (both ways) on the current graph by local greedy search. If no path is found, then an edge is added between the last point on the traversed path and the destination point. Whenever an outgoing edge is added to some point \mathbf{u} , it is checked whether it occludes any of the existing outgoing edges of \mathbf{u} ; if it does, those edges are removed and a path is sought between the endpoints of the removed edges, repeating the process.

Due to the multi-nested nature of the algorithm of Harwood and Drummond [HD16], as well as various heuristics employed by the authors, it is difficult to obtain a tight bound on its complexity. We suggest a loose approximation of it as $O(n_I d N \delta k' \log k')$, where n_I is the number of iteration steps, δ is the maximum diameter of the proximity graph during the iteration, and k' is the maximum outgoing degree of the graph during the iteration. (The δ factor corresponds to the maximum number of traversal steps starting from each reference node, and the $k' \log k'$ factor is due to maintaining distance-sorted lists of neighbors for each visited node.)

2.2.5 Discussion

The majority of algorithms for all- k NN search utilize a divide-and-conquer paradigm, constructing local k NN sub-graphs among small subsets of points before proceeding to bridge and refine them. From an operational viewpoint, the main differences between algorithms lie in the way the dataset is partitioned to smaller subsets and in the strategy that is used to find neighbor connections between points in disjoint subsets. These choices in turn shape the theoretical properties of the algorithm, in terms of accuracy, applicability to different feature spaces and distance measures,

and computational complexity.

Starting around 2000, and especially in the last decade, the prevalence of high-dimensional data and applications that depend on efficient all-NN search has fueled the development of numerous heuristics for approximate search. These are sometimes in contrast to approximate algorithms, which offer geometric or probabilistic guarantees regarding the quality of the resulting k NN graphs, while sometimes they are used in conjunction with them. Despite the lack of strong theoretical guarantees, however, such heuristics remain popular and are often found empirically to perform better than their more analytically-grounded counterparts [ABF17]. Further research is necessary to improve our understanding of these methods and the conditions under which they are efficient as well as dependable.

Spatial Partitions & Adjacency Graph Hierarchy

In this chapter, we introduce the central analysis apparatus, theory, and algorithms of the dissertation in support of efficient near-neighbor (NN) search, built upon a sparse adjacency graph hierarchy (AGH) structure.

All-near-neighbors (all-NN) search among points in an irregularly distributed dataset is typically aided by a point partition induced by a partition of the spatial domain. Numerous partition structures have been proposed to leverage particular dataset properties or specified data distributions. Here, we investigate (i) a coding scheme for spatial partition, in its basic and simplest form, and its properties with respect to adjacency among partitions at multiple resolution scales in a multi- or high-dimensional feature space; and (ii) the full exploitation of adjacency properties towards navigation, construction, and analysis of the AGH, and efficient all-NN search, in adaptation to data sparsity.

Our results with the basic partition structure used in this chapter are extensible to related partition schemes—e.g., the balanced box decomposition (BBD) tree [AMN⁺98]—that are regulated by additional or alternative conditions. We do not address such schemes in this chapter, but describe a more generalized structure

and an expanded strategy for spatial partitions based on adjacency interactions in the next chapter.

3.1 Sparsity-adaptive spatial partition

Denote by $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$ a set of feature vectors in a d -dimensional metric space. We refer to the feature vectors as points or particles in \mathbb{R}^d . We define a hyper-box, or simply box, B in \mathbb{R}^d as the Kronecker product of d axial closed intervals,

$$B = \bigotimes_{i=1}^d [a_i, b_i], \quad a_i < b_i, \quad i \in \{1, \dots, d\}. \quad (3.1)$$

The axial interval $[a_i, b_i]$ is the projection of B onto the i -th dimension axis. We denote it by $B(i)$; thus, $B = \bigotimes_{i=1}^d B(i)$.

A box B is regular if all side-intervals have the same length, i.e., $b_i - a_i = c$, $i \in \{1, \dots, d\}$, for some constant $c > 0$. Box B is a unit box if $c = 1$. Let B_0 be the smallest regular box containing \mathcal{X} , $\mathcal{X} \subset B_0 \subset \mathbb{R}^d$; we call B_0 the bounding box of \mathcal{X} . Without loss of generality, we may assume that the bounding box B_0 is a unit box, $B_0 = \bigotimes_{i=1}^d [0, 1]$.

3.1.1 Recursive mid-point partition (RMP) tree

We partition B_0 at its mid-point along each axial interval, $B(i)$, yielding 2^d children boxes. Each child box is regular, with a side length of $1/2$. The partitioning rule is recursively applied to each child box. At the recursive partition level ℓ , there are up to $2^{d\ell}$ regular boxes of side length $1/2^\ell$. The volume of any box B at level ℓ is $\text{vol}(B) = 2^{-d\ell}$. Recursive mid-point partitions (RMPs) are a special case of recursive dyadic partitions (RDPs).

The spatial partition terminates when certain conditions are met. For example, one may specify the maximal partition level L , limiting the spatial resolution of the partition, such that the side-length of a box is at most 2^{-L} along each dimension. One

may also specify a different resolution level threshold, $L(j)$, along each dimension, $i \in \{1, \dots, d\}$. We assume an isotropic resolution level threshold in keeping with the uniform partition strategy described above, for simplicity. (In practice, anisotropic resolution and non-uniform partitions are often preferable; we will consider those in the next chapter.)

The recursive partition of B_0 induces a recursive partition of the points in \mathcal{X} ; the latter are associated with the boxes that contain them, at every level of the partition hierarchy. If the i -th coordinate of a point $\mathbf{x} \in \mathcal{X}$ lies exactly on the boundary between two box intervals, then \mathbf{x} is assigned to the right interval on the i -th axis, $i \in \{1, \dots, d\}$. Every point is contained by exactly one box at each level. We denote the number of points contained by a box B by $\text{mass}(B)$.

The RMP can be represented by a rooted tree, $T(B_0)$. The root node of the RMP tree is associated with the bounding box B_0 . The tree nodes at level 1 are associated with non-empty children boxes of B_0 , and tree nodes at level ℓ are associated with non-empty boxes by the recursive partition at level ℓ . We denote the non-empty children boxes of a box B by $\mathcal{C}(B)$, and the set of all nodes (i.e., non-empty boxes) of a tree $T(B_0)$ by $\mathcal{V}(T)$. Tree edges represent the inclusion relationship of children boxes by their parental box. The RMP tree is a natural extension of the quadtree ($d = 2$) and octree ($d = 3$) [Sam06] to higher dimensions, $d > 3$.¹

3.1.2 *Inherent sparsity & adaptation to sparsity*

An essential issue with high-dimensional data is that of the inherent sparsity of the data distribution in space and the corresponding sparsity in the box partition structure. We address their connection and relation to increasing feature dimensionality and finer spatial partition resolution.

First, assume that points in \mathcal{X} are uniformly distributed in B_0 . The RMP tree is

¹ Such trees have also been referred to as “hyperoctrees” [Cla83; YS83; AS99].

then full up to level $\ell_{\min} = \lceil \log_2(N)/d \rceil$, where $N = |\mathcal{X}|$ is the number of points. At any finer level $\ell > \ell_{\min}$, the number of non-empty boxes is bounded from above by N . We are concerned with the case where points in \mathcal{X} are distributed non-uniformly. The connection between the two cases can be expressed via the following unifying view.

The sparsity ratio of a RMP tree $T(B_0)$ over a finite, discrete dataset $\mathcal{X} \subset \mathbb{R}^d$, defined as the ratio of the number of non-empty boxes in $T(B_0)$ to the number of boxes in a complete tree ($2^{d\ell}$), at some level ℓ , grows exponentially smaller with higher dimensionality and finer partition resolution.

The tree partition rules in adaptation to sparsity in the point distribution are described to some extent by the partition termination conditions. In addition to the tree level threshold, L (equivalently, spatial resolution threshold $h_{\max} = 2^{-L}$), we subject the partition to a point population or mass threshold, p . A box B in the RMP tree is a leaf if contains no more than p points, or it is at level L (i.e., its side length is no larger than h_{\max}).

Proposition 3.1 (Mixed-resolution partition of \mathcal{X}). Assume that the points of \mathcal{X} are non-uniformly distributed within a unit bounding box $B_0 \subset \mathbb{R}^d$. Let $T(B_0)$ be the sparsity-adaptive RMP tree over \mathcal{X} , with spatial resolution level threshold L and mass resolution threshold p , and let $|x| = N$. The data set \mathcal{X} , $|\mathcal{X}| = N$, is covered and partitioned by the leaf boxes in $T(B_0)$. Every point resides in exactly one leaf box. A leaf box at level ℓ indicates a sparser point distribution in comparison to any non-leaf box at the same level. The tree has the following properties.

- (i) The height of $T(B_0)$ is bounded between $\lceil d^{-1} \log_2(N/p) \rceil$ and L , if $\log_2(N/p) < dL$.

- (ii) The leaf boxes in $T(B_0)$ cover and partition the points in \mathcal{X} . Every point resides in exactly one leaf box, which generally lie at different resolution levels. A leaf box at some level ℓ is sparser than any non-leaf box at the same level. The number of leaf boxes in $T(B_0)$ is at most N .

By Proposition 3.1, the boxes at any level ℓ of $T(B_0)$, together with all leaf boxes at all levels $\ell' < \ell$, form a set of boxes which cover and partition the points in \mathcal{X} . We may refer to such a set as the level- ℓ *canopy* of $T(B_0)$.

3.1.3 “Qode”: spatial quantization code

Every box node on the RMP tree is associated with a unique, finite codeword which specifies its spatial location in quantized form. The root node represents the bounding box unambiguously, and we associate it with the empty sequence (zero-length codeword). For every other box, its codeword gives its placement within the root box. We refer to a box quantization code as its “qode” (pronounced the same as “code”), for brevity.

Definition 3.1 (Qode). Let $T(B_0)$ be an RMP tree rooted at B_0 . Every box node B in $T(B_0)$ is associated with a *qode* $q(B)$ in bit-array format. For the root node, $q(B_0) \triangleq []$ (empty array). For any box node B at level $\ell > 0$, let A be its parent box ($B \in \mathcal{C}(A)$). Then, $q(B)$ is an extension of $q(A)$ with an additional d -bit segment,

$$q(B) \triangleq [q(A) \ c_\ell], \quad c_\ell \in \{0, 1\}^{d \times 1}, \quad (3.2)$$

where the new bit column c_ℓ specifies the position of B relative to its parent box A : for each $i \in \{1, \dots, d\}$, $c_\ell(i) = 0$ indicates that B resides in a left sub-box (lower half-interval) of A along the i -th axis; otherwise, $c_\ell(i) = 1$. We refer to c_ℓ as the *local qode* of B .

The explicit form of the qode of a box is a 2D bit-array, $q(B) = [c_1 \ \dots \ c_\ell] \in \{0, 1\}^{d \times \ell}$. The j -th column of $q(B)$ contains the local qode of the j -th box on the path

from B_0 (exclusive) to B . The i -th row of $q(B)$ contains the quantized coordinate value along the i -th dimension of all points in B (with respect to the spatial resolution of B). The quantization representation of the boxes, and by extension the points contained in them, is specific to the partition structure.

With regular RMP tree boxes in particular, the rows of a qode array are binary representations of uniformly quantized coordinates (scaled onto a regular grid of 2^ℓ nodes along each axis). When read in column-major order as a codeword of length $d\ell$, the qode of an box is the same as its so-called locational code [Gar82; Sch92; Sam06]. Sorting the locational codes of boxes in a multi-dimensional RMP tree results in a level-wise ordering of the boxes along a Z-curve.

Two example RMP trees and the corresponding leaf box qodes are shown in Figures 3.1 and 3.2, with a 1D and 2D sample dataset, respectively.

The qodes of Definition 3.1 allow us to readily answer certain queries regarding the spatial relationship between boxes in $T(B_0)$. For example, sibling boxes at level ℓ share the same qode prefix of $\ell - 1$ columns. Similarly, the finest-scale common ancestor of any two boxes is identified by the largest common column-prefix of their qodes. In the sections that follow, we will decipher more spatial relationships encoded in box qodes, and show how to exploit them for efficient processing of the partition structure.

Example 3.1 (1D RMP tree and box qodes). Figure 3.1 shows a sparsity-adaptive RMP tree structure over a sample 1D dataset, alongside a table with all point-wise qodes.

Example 3.2 (2D RMP tree and box qodes). The boxes of a sparsity-adaptive RMP tree over a synthetic 2D dataset is shown in Figure 3.2, alongside a table listing the qodes of all non-empty boxes at each level of the tree.

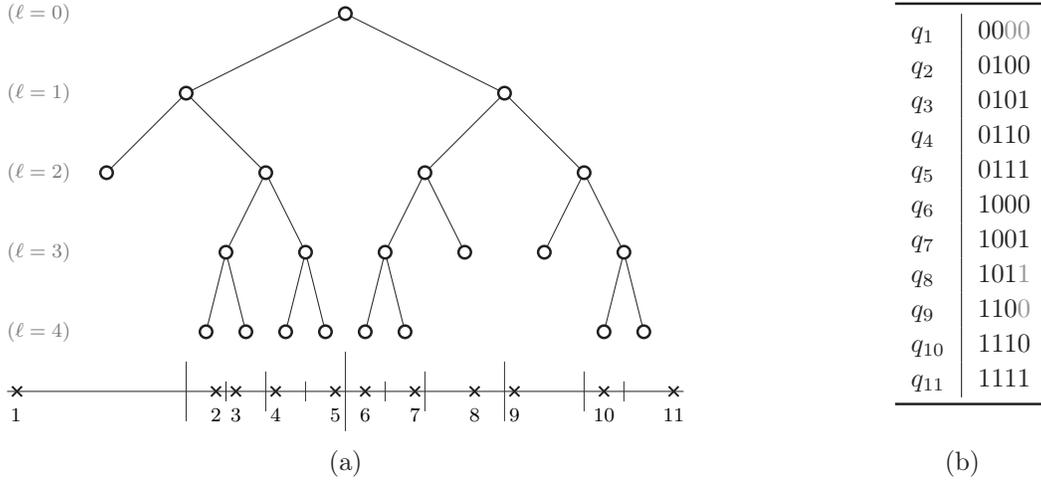


FIGURE 3.1: Sparsity-adaptive RMP tree over an example 1D dataset. (a) Dataset and combinatorial tree structure. Points are marked \times on the 1D domain (line segment below the tree). The spatial and mass resolution parameters of the tree are $L = 4$ and $p = 1$, respectively. (b) Point-wise qodes at the finest resolution level. Gray bits indicate levels where the corresponding tree node is not partitioned due to the mass resolution threshold.

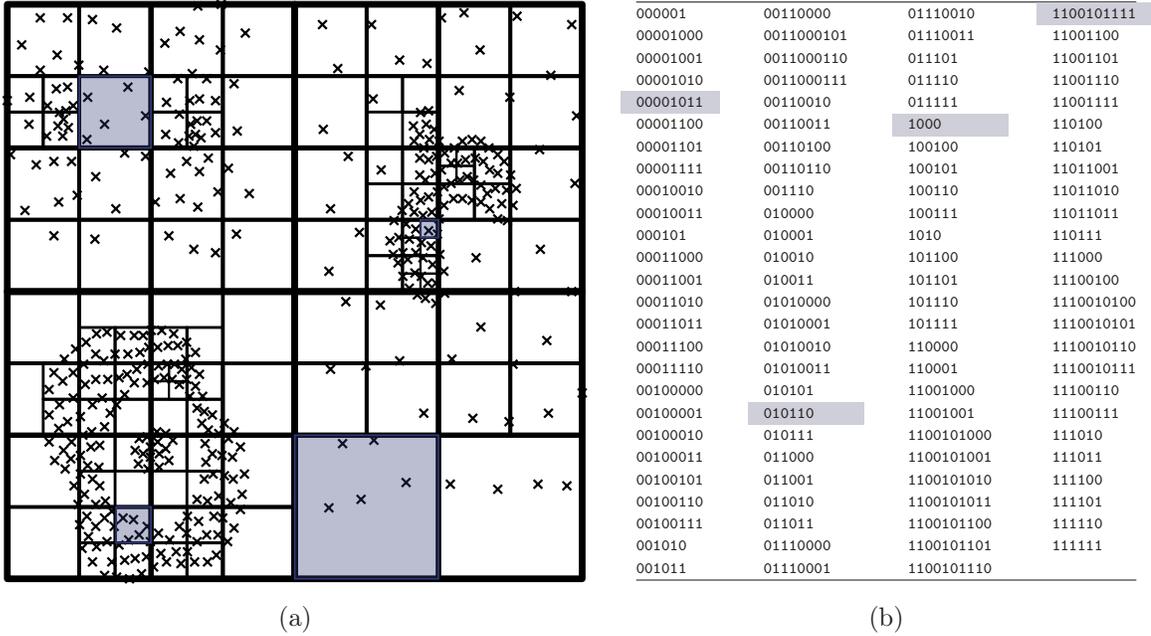


FIGURE 3.2: Sparsity-adaptive RMP tree over an example 2D dataset. (a) Dataset and spatial partitions. Points are marked \times in the 2D domain. The spatial and mass resolution thresholds of the tree are $\ell_{\max} = 5$ and $p = 8$, respectively. (b) Leaf box qodes (bits in column-major order per (3.2) definition), truncated by tree level. Four leaf boxes at different spatial resolution levels and their respective qodes are highlighted.

3.2 Spatial adjacency of partition boxes

Definition 3.2 (Box adjacency). Two boxes B and B' in a partition tree $T(B_0)$ are *adjacent* if they share a common boundary, $B \cap B' \neq \emptyset$. We denote adjacency by $B \rightarrowleftarrow B'$, or equivalently $q \rightarrowleftarrow q'$, where $q = q(B)$ and $q' = q(B')$ are the unique qodes of B and B' .

The intersection $B \cap B'$ is the largest common face between B and B' . We refer to it as the adjacency face of B and B' .

Proposition 3.2 (Product of adjacency face axial intervals). The adjacency face of two d -dimensional boxes, $B \rightarrowleftarrow B'$, in a d -dimensional partition tree is itself a f -dimensional box, $0 \leq f \leq d$,

$$B \cap B' = \bigotimes_{i=1}^d B(i) \cap B'(i), \quad (3.3)$$

where $\dim(B \cap B') = f$ is the adjacency dimension between B and B' .

If B and B' are at the same level of an RMP tree, the adjacency dimension is equal to the number of identical axial projections between B and B' :

$$\dim(B \cap B') = |\{i \mid B(i) = B'(i), i = 1, \dots, d\}|. \quad (3.4)$$

The adjacency dimension is 0 when B and B' only share a boundary point or vertex, 1 when they share a linear segment, etc. Adjacency includes the case where the two boxes are identical, i.e., $B = B'$ and $\dim(B \cap B') = d$.

For the time being, we only consider spatial adjacency between boxes at the same resolution level. We will extend our notion of adjacency in Section 3.4.

3.2.1 Level-wise adjacency lists

Definition 3.3 (Same-level adjacency list). Let $T(B_0)$ be an RMP tree, and let B be a box node at level $\ell(B) > 0$ on the tree. The *adjacency list* of B is the set of its

adjacent boxes at level $\ell(B)$:

$$\mathcal{A}(B) \triangleq \{B' \in \mathcal{V}(T) \mid B' \leftrightarrow B, \ell(B') = \ell(B)\}. \quad (3.5)$$

The adjacency list $\mathcal{A}(B)$ includes the host box, B . In our analysis and in practice, we often separate the cases where two adjacent boxes B and B' may or may not be identical. We will make this distinction explicit, unless it is clear from the context.

The size of the adjacency list of a d -dimensional box B is bounded from above by

$$|\mathcal{A}(B)| \leq \min \left\{ N, \sum_{f=0}^d \binom{d}{f} 2^{d-f} \right\}, \quad (3.6)$$

where $\binom{d}{f}$ are binomial coefficients, and each term in the sum is the number of f -dimensional faces on a d -dimensional box ($0 \leq f \leq d$). Due to the sparse and irregular distribution of points and non-empty boxes in high-dimensional spaces (see Section 3.1.2), the bound in (3.6) becomes very loose as d increases. We will show later how we can calculate a probabilistic estimate of the expected value of $|\mathcal{A}(B)|$ for any specific dataset, given empirical measurements of adjacency and number of non-empty children boxes at the parent level in the tree.

There is a definitive spatial gap between a box B at some level ℓ and any other box at the same level that is not adjacent to B . Specifically, we define

$$\text{gap}_p(B) \triangleq \min \{ \|\mathbf{u} - \mathbf{v}\|_p \mid \mathbf{u} \in B, \mathbf{v} \in \mathcal{A}^c(B) \}, \quad (3.7)$$

where $\mathcal{A}^c(B)$ is the set of non-adjacent, same-level boxes to B (i.e., the complement of $\mathcal{A}(B)$ in the set of boxes at the ℓ -th level of the RMP tree), and $p \in \mathbb{N}_+ \cup \{\infty\}$.

(We slightly abuse notation here and later when we write, for example, $\mathbf{v} \in \mathcal{A}^c(B)$, since $\mathcal{A}^c(B)$ is a set of boxes, not points. Whenever we make such statements addressing the inclusion of a point in a set of boxes, the latter is to be understood as the union of boxes in the set.)

The spatial adjacency gap of a box B is useful to restricting the local search region for points in $B \cap \mathcal{X}$ in NN search, and to approximate computation in multi-scale interactions among points [GR87; CK95; AMN⁺98; GM01; SP01]. In particular, for a box B at level ℓ , we have $\text{gap}_\infty(B) = 2^{-\ell}$. This is a lower bound of the distance between points $\mathbf{x} \in B \cap \mathcal{X}$ and $\mathbf{x}'' \in \mathcal{A}_\ell^c(B) \cap \mathcal{X}$: $\text{gap}_\infty(B) \leq \text{gap}_p(B) \leq d(\mathbf{x}, \mathbf{x}'')$. If the NN search radius (either fixed in r NN or current-estimate in k NN) is smaller than the gap, the search can be restricted to the spatial adjacency lists at the relevant resolution level.

Adjacency lists further define a partial ordering among boxes in $T(B_0)$ and points in \mathcal{X} , across resolution levels in the RMP tree. Let $C \in \mathcal{C}(B)$ be a child box of B . By the partition structure, $\mathcal{A}(C) \subset \mathcal{A}(B)$; accordingly, we get $(\mathcal{A}(C) \cap \mathcal{X}) \subseteq (\mathcal{A}(B) \cap \mathcal{X})$.

3.2.2 Level-wise far-neighbor interactions & compression

The level-wise adjacency concepts (geometric and sparsity-adaptive) discussed above have a direct connection to several fast algorithms that are based on multi-scale partition and compression.

Specifically, points in the adjacency set difference across successive resolution levels, $(\mathcal{A}(B) \setminus \mathcal{A}(C)) \cap \mathcal{X}$, are referred to as the *far neighbors* of $C \cap \mathcal{X}$, at the level where C resides, in the fast multipole method (FMM) [GR87]. The same notion is applicable to the Barnes-Hut algorithm [BH86] and other fast transformations with hierarchically semi-separable (HSS) interaction kernels or matrices [XCG⁺10; HG12]. In these algorithms, far-neighbor interactions can be approximated via compressive matrix representations at every resolution level. In the FMM in particular, these level-wise compressive representations are further connected by inter-level translation, thereby yielding the remarkable linearly-scaling complexity of interaction computations with the FMM [SP01].

These approximate compression techniques were initially applied to molecular and astrophysical dynamics simulations [BH86; Gre87], which are also known as N -body simulations. Later applications include the simulation and design of electromagnetic systems [Che14; PHS⁺16], all-NN search [CK95; AMN⁺98; LMG04], pairwise statistical estimation problems [GS91; GM01; RLM⁺09; MXY⁺16], and embeddings for data analysis and visualization [vdMaa14].

3.3 Adjacency lineage & qode relationships

In Section 3.1.3, we discussed box qodes for quantizing and indexing box locations along dimension axes and partition levels, as well as encoding the combinatorial structure of the RMP tree in terms of parent-child and sibling relationships. In this section, we introduce the use of qode patterns in recognizing and locating spatially adjacent boxes.

There are two prevalent, existing ways of identifying and locating adjacent boxes. One involves qode arithmetic and global search or backtracking in the RMP tree. Given a box qode and an adjacency direction, one first calculates the adjacent qode the box in the specified direction. That qode identifies a candidate adjacent box. The tree is then traversed to determine whether the candidate box exists in the tree, i.e., whether or not it is non-empty [Sam89; Sch92; Sam06]. The other is based on the fact that if two boxes $B \in \mathcal{C}(A)$ and $B' \in \mathcal{C}(A')$ are adjacent, $B \rightarrow\leftarrow B'$, then their parent boxes must also be adjacent, $A \rightarrow\leftarrow A'$ (the converse is not true). This is used to narrow down the search region for adjacent box pair candidates; one then checks all candidate pairs to identify whether they are adjacent or not. This approach is framed as dual-tree traversal [GM01; Cur15].

Both approaches are passive, in the sense that they traverse the tree towards boxes that may or may not be adjacent. They suffer in computational efficiency due to following spurious paths in the tree, or “over-visiting” boxes. The degree of

uncertainty and over-visiting becomes higher with increased dimensionality.

We depart from these approaches and introduce an approach for active, directed pinpointing and traversal of adjacent box pairs. Our approach is enabled by decoding and utilizing the deterministic, geometric nature in box adjacency lineage and the corresponding qode relations.

3.3.1 Single dimension

Lemma 3.1 (Qode patterns for box adjacency: single dimension). *Let $T(B_0)$ be an RMP tree over a dataset in a one-dimensional metric space. Let B and B' be two non-identical boxes at level ℓ , with qodes q and q' , respectively, $q \neq q'$. The two boxes are adjacent, $B \rightarrow\leftarrow B'$, or equivalently $q \rightarrow\leftarrow q'$, if and only if for some $j \in \{1, \dots, \ell\}$,*

$$\begin{aligned} q(1 : j - 1) &= q'(1 : j - 1), && \text{(common ancestor path)} \\ q(j) &= \overline{q'(j)} = \beta, && \text{(sibling split)} \\ q(j + 1 : \ell) &= \overline{q'(j + 1 : \ell)} = \overline{\beta}, && \text{(flip toward sibling adjacency face)} \end{aligned} \tag{3.8}$$

where $\beta \in \{0, 1\}$ is the j -th bit of the qode $q(B) = q$; \bar{q} is the bit-wise complement of q ; and $j : j'$ denotes that the relation must hold at all levels from j through j' .

The two adjacent boxes have common ancestors from the root up to the $(j - 1)$ -th level. The paths to B and B' first split at level j . If $j = \ell$ (the qode segment at levels $j + 1 : \ell$ is empty), the two boxes are siblings. Otherwise, $j < \ell$, and the two paths are constrained to keep close to the adjacency face between their sibling ancestor pair at all subsequent generations. That is, the local qode in the paths to B and B' is flipped *once* from level j to $j + 1$, and remains constant thereafter. In brief, we may distinguish four phases in the adjacency lineage qode relationships: (i) common ancestry, (ii) sibling split, (iii) conjugate flip, and (iv) constant replication until termination. The adjacency lineage phases and corresponding qode relationships are illustrated in Figure 3.3.

- (i) *Sibling adjacency.* If $B = B'$, the children boxes in $\mathcal{C}(B)$ are mutually adjacent.
- (ii) *Non-sibling adjacency inheritance.* If $B \neq B'$, adjacency is inherited by a single pair of children boxes, $C \in \mathcal{C}(B)$ and $C' \in \mathcal{C}(B')$, such that $C \cap (B \cap B') \neq \emptyset$ and $C' \cap (B \cap B') \neq \emptyset$. If either C or C' do not exist in the tree, then there are no adjacent descendant-pairs of B and B' .

In the one-dimensional case, by Lemma 3.1 and Corollary 3.1, a new adjacency pair is spawned by the children of any box A .² Once a sibling adjacency, $B \rightarrow\leftarrow B'$, has been spawned, it is carried onto future generations by a single pair of eligible descendants, one in the B side and one in the B' side, until at least one of the eligible descendant boxes is empty.

3.3.2 Multiple dimensions

By Lemma 3.1 and Proposition 3.2, adjacency between boxes in a multi-dimensional space can be expressed via dimension-wise qodes relationships.

Proposition 3.3 (Qode patterns for box adjacency: multiple dimensions). Let $T(B_0)$ be an RMP tree over a dataset in a d -dimensional metric space. Let B and B' be two non-identical boxes at level ℓ , with qodes q and q' , respectively, $q \neq q'$. The two boxes are adjacent, $B \rightarrow\leftarrow B'$, or equivalently $q \rightarrow\leftarrow q'$, if and only if

$$q(i, 1 : \ell) \rightarrow\leftarrow q'(i, 1 : \ell), \quad \forall i \in \{1, \dots, d\}. \quad (3.10)$$

Let j_i be the level at which the paths to B and B' split along the i -th axis, i.e.,

$$q(i, 1 : j_i - 1) = q'(i, 1 : j_i - 1), \quad q(i, j_i) \neq q'(i, j_i), \quad (3.11)$$

and let $j_{\min} = \min_{i \in \{1, \dots, d\}} j_i$. The nearest common ancestor of B and B' is at level $j_{\min} - 1$. At any level $j \leq \ell$, the paths to B and B' are split in $d - f_j$ dimensions,

² Unless A only has one child. In practice, however, single-descendant paths in the tree are typically contracted into a single edge.

where $f_j = |\{j \mid j \leq j_i, i = 1, \dots, d\}|$; by level ℓ where B and B' reside, their paths have split in $d - \dim(B \cap B')$ dimensions.

The adjacency lineage and corresponding qode relationships in multi-dimensional spaces are generalized accordingly, as follows.

Proposition 3.4 (Adjacency lineage: multiple dimensions). Let $B \rightarrow\leftarrow B'$ be two adjacent boxes at level ℓ of a partition tree in a d -dimensional space.

- (i) *Sibling adjacency.* If $B = B'$, all children boxes in $\mathcal{C}(B)$ are mutually adjacent to one another.
- (ii) *Non-sibling adjacency inheritance.* If $B \neq B'$, adjacency is inherited by all pairs of eligible boxes among the children of B and B' , by the following conditions. Let $C \in \mathcal{C}(B)$ and $C' \in \mathcal{C}(B')$, with qodes $q(C) = [q, c_{\ell+1}]$ and $q(C') = [q', c'_{\ell+1}]$, where $c_{\ell+1}, c'_{\ell+1} \in \{0, 1\}^d$ are the local qodes of C and C' . The children boxes C and C' are adjacent, $C \rightarrow\leftarrow C'$, if and only if, along each dimension $i \in \{1, \dots, d\}$, it holds that $c_{\ell+1} \neq c'_{\ell+1}$ and either

$$c_{\ell+1}(i) = \overline{q(i, \ell)}, \quad \text{if } \begin{cases} q(i, 1 : \ell - 1) = q'(i, 1 : \ell - 1), \\ q(i, \ell) \neq q'(i, \ell); \end{cases} \quad (3.12a)$$

or

$$c_{\ell+1}(i) = q(i, \ell), \quad \text{if } \begin{cases} q(i, 1 : j_i - 1) = q'(i, 1 : j_i - 1), \\ q(i, j_i : \ell) \neq q'(i, j_i : \ell), \\ j_i < \ell. \end{cases} \quad (3.12b)$$

Conditions (3.12a) and (3.12b) correspond to the conjugate flip ($j_i = \ell$) and constant replication ($j_i < \ell$) phases, respectively, of the adjacency lineage along the i -th dimension.

3.4 Adjacency graph hierarchy (AGH)

Box adjacency defines a spatial proximity relation between tree nodes and the underlying dataset, in addition to the spatial inclusion (parent-child) relation captured by the tree edges. The adjacency structure is made explicit via the introduction of level-wise adjacency graphs.

Definition 3.4 (Adjacency graph hierarchy). Let $T(B_0)$ be an RMP tree with height ℓ_{\max} . At every level $\ell \in \{1, \dots, \ell_{\max}\}$, the *box adjacency graph* is a symmetric graph, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$, whose vertex set \mathcal{V}_ℓ is the set of non-empty boxes at the ℓ -th tree level, and whose edge set \mathcal{E}_ℓ represents pairwise adjacency among non-identical boxes in \mathcal{V}_ℓ :

$$\mathcal{E}_\ell \triangleq \{(B, B') \mid B, B' \in \mathcal{V}_\ell, B \rightarrow\leftarrow B'\}. \quad (3.13)$$

The level-wise graphs $\{G_\ell\}_{\ell=0}^{\ell_{\max}}$ are interconnected by the parent-child edges in $T(B_0)$, yielding an *adjacency graph hierarchy (AGH)*.

In practice, it is often desirable to extend the vertex set at level ℓ to include also all leaf boxes at coarser resolution levels, i.e., $\mathcal{V}_\ell = \{B \in \mathcal{V}(T) \mid \ell(B) = \ell\} \cup \{B \in \mathcal{V}(T) \mid \ell(B) < \ell, \mathcal{C}(B) = \emptyset\}$. (Equivalently, \mathcal{V}_ℓ would be the set of leaf boxes if the tree was truncated at level ℓ .) Adjacency edges are defined as before. With this definition, the AGH covers the entire dataset, \mathcal{X} , at every level. In the interest of simplicity, we will forego this extended definition of the AGH vertex sets. Nonetheless, all our analysis and algorithms on describing, locating, and traversing adjacent box sets can be amended to encompass it fairly straightforwardly.³

³ For instance, the 1D qode adjacency pattern (3.9) holds as-is for qodes q and q' at different resolution levels. In general, adjacency lineage relations remain the same, except that if one of two adjacent boxes in the lineage is a leaf, then the relations carry on between the leaf box and the descendants of the other box; if both boxes are leaves, the lineage is terminated.

3.4.1 AGH components and properties

We make use of the AGH as an analysis apparatus as well as algorithmic infrastructure for all-NN search. We describe in the following Theorem the macroscopic relationship between level-wise adjacency graphs, and in particular the patterns of adjacency edge formation and propagation, based on the adjacency lineage rules we characterized in Section 3.3.

Theorem 3.1 (Adjacency propagation between successive AGH levels). *Let $\{G_\ell\}_{\ell=0}^{\ell_{\max}}$, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$ be the AGH over an RMP tree, as per Definition 3.4. At the root level, $G_0 = (\{B_0\}, \emptyset)$. For any $\ell \geq 0$, the adjacency graph at level $\ell + 1$ is derived from the adjacency graph at the previous level as*

$$\mathcal{V}_{\ell+1} = \bigcup_{B \in \mathcal{V}_\ell} \mathcal{C}(B), \quad \mathcal{E}_{\ell+1} = \mathcal{E}_{\ell+1}^{[v]} \cup \mathcal{E}_{\ell+1}^{[e]}, \quad (3.14a)$$

where

$$\mathcal{E}_{\ell+1}^{[v]} = \bigcup_{B \in \mathcal{V}_\ell} \mathcal{K}(\mathcal{C}(B)), \quad \mathcal{E}_{\ell+1}^{[e]} = \bigcup_{(B, B') \in \mathcal{E}_\ell} \mathcal{K}_2(\mathcal{C}_f(B, B'), \mathcal{C}_f(B', B)); \quad (3.14b)$$

$\mathcal{K}(\mathcal{V}) = \mathcal{V} \times \mathcal{V}$ is a clique over the vertex set \mathcal{V} ; $\mathcal{K}_2(\mathcal{V}, \mathcal{V}') = \mathcal{V} \times \mathcal{V}'$ is a complete bipartite graph over two disjoint vertex sets \mathcal{V} and \mathcal{V}' ; and $\mathcal{C}_f(B, B') \triangleq \{C \in \mathcal{C}(B) \mid C \cap (B \cap B') \neq \emptyset\}$ are the children of B that are incident on the adjacency face $B \cap B'$.

Proof. The AGH propagation patterns follow from the RMP tree structure and the adjacency lineage relationships of Proposition 3.4. \square

Figure 3.4 shows the level-wise adjacency graphs over two successive levels of a sample 2D RMP tree, and highlights the adjacency edge components described in Theorem 3.1. Figure 3.5 shows the level-wise interaction sub-graphs around boxes in a path across three resolution levels in a 2D FMM. The AGH serves as the interaction “skeleton” the multi-scale interaction sub-graphs.

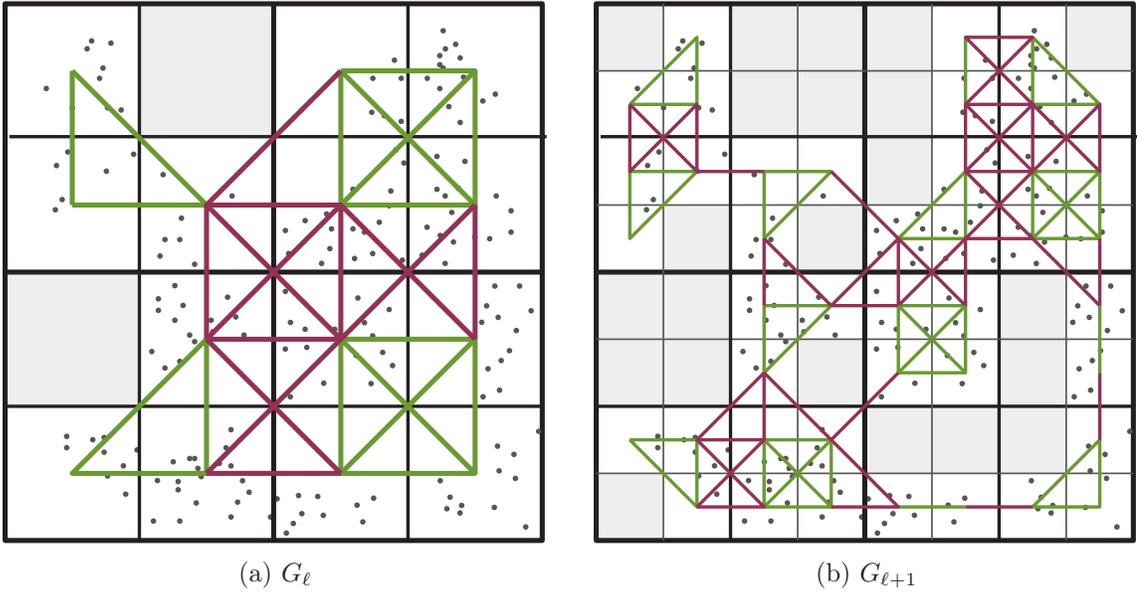


FIGURE 3.4: Illustration of level-wise adjacency graphs and adjacency edge propagation between two successive levels of the AGH over a 2D RMP tree. Adjacency edges in sibling cliques and face-adjacent complete bipartite graphs (see Theorem 3.1) are colored green and red, respectively. Empty boxes are shaded light gray.

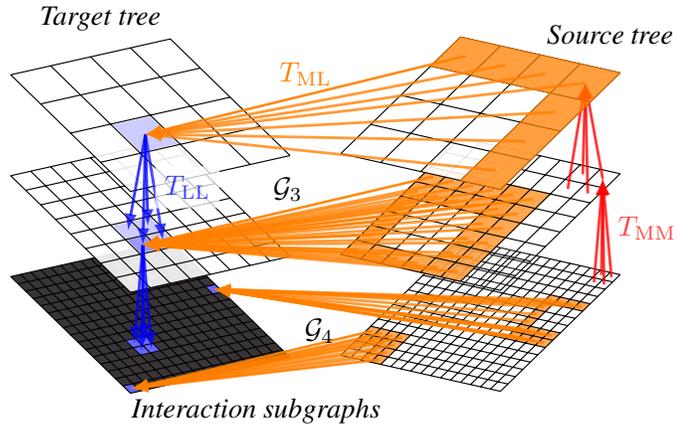


FIGURE 3.5: Multi-scale interaction sub-graphs of the AGH in the 2D FMM over a regular partition. The interaction sub-graph domain around a box B at any level ℓ covers the “far neighbors” of B , defined as $\mathcal{A}(A) \setminus \mathcal{A}(B)$, where A is the parent box of B , $B \in \mathcal{C}(A)$; see also Section 3.2.2. (Plot taken from [ZHP⁺11], courtesy of the authors.)

A few remarks are in order, regarding the relationship between the AGH structure and the inherent sparsity of the underlying dataset, \mathcal{X} , in a high-dimensional feature space. As usual, let $\mathcal{X} \subset \mathbb{R}^d$, and $|\mathcal{X}| = N$.

- (i) At any level, $|\mathcal{V}_\ell| \leq N$, since the total number of leaf boxes in $T(B_0)$ is bounded by N .
- (ii) Although the overall structure of G_ℓ is irregular, following the spatial distribution of \mathcal{X} , it is composed of regular elements, in particular clique and complete bipartite sub-graphs. With a sparse dataset \mathcal{X} , the adjacency graphs in finer resolution levels are locally dense, but globally sparse.
- (iii) The clique sub-graphs vary in size. The number of sibling edges that are spawned by B is $|\mathcal{C}(B)|(|\mathcal{C}(B)| - 1)/2$. No box B can have close to 2^d children when $|B \cap \mathcal{X}| \ll 2^d$.
- (iv) The complete bipartite sub-graphs vary in size. The number of adjacency edges that are spawned by any adjacent pair, (B, B') , may seem to be increasing exponentially with $\dim(B \cap B')$, the adjacency face dimensionality, which can be as high as $d - 1$. The actual number of spawned edges, however, is $|\mathcal{C}_f(B, B')||\mathcal{C}_f(B', B)| \leq |\mathcal{C}(B)||\mathcal{C}(B')|$. In fact, given some empirical measurements of the tree and AGH at some level ℓ , we can pre-calculate the exact number of spawned edges or a probabilistic estimate of its expected value under mild assumptions. Roughly speaking, the sparser the local point subset $B \cap \mathcal{X}$, the lower the expected value of $|\mathcal{C}_f(B, B')|$.
- (v) In the special case where points in \mathcal{X} are uniformly distributed within B_0 , the AGH represents regular, equispaced grids at multiple resolution levels. The RMP tree is complete and shallow, with height $\ell_{\max} = \lceil d^{-1} \log_2(N/p) \rceil$. The adjacency list of a box B (not on the boundary of B_0) at level $\ell > 1$ is 3^d , if

$N/p \geq 3^d$. The adjacency edges \mathcal{E}_ℓ are amenable to a closed-form representation (we omit this detail). Notwithstanding, it is impossible to have a complete tree, even with height $\ell_{\max} = 1$, when the dimensionality is high enough that $N \ll 2^d$. We are interested in the multi-dimensional irregularly distributed case.

The AGH is endowed with additional attributes over the vertex and edge sets. For the level- ℓ adjacency graph, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$, each box $B \in \mathcal{V}_\ell$ is associated with a mass function which counts the number of points contained in it, $\text{mass}(B) = |B \cap \mathcal{X}|$. Each edge $(B, B') \in \mathcal{E}_\ell$ is associated with a face descriptor, $f(B, B') \in \{0, 1\}^d$, where $f(i) = 0$ if $q(i, 1 : \ell) = q'(i, 1 : \ell)$ and $f(i) = 1$ otherwise, with $q(B) = q$, $q(B') = q'$. The face descriptor indicates which dimensions are perpendicular to the $B \cap B'$ adjacency face; it follows that $\dim(B \cap B') = d - \sum_{i=1}^d f(i)$. We may further associate each edge with a pair of integers, $(|\mathcal{C}_f(B, B')|, |\mathcal{C}_f(B', B)|)$. This allows one to readily calculate $|\mathcal{E}_{\ell+1}|$ given G_ℓ —or, conversely, to make informed decisions when partitioning \mathcal{V}_ℓ into the next resolution level (see Chapter 4). The edge attributes above can be efficiently computed on the fly, using only local code information, as we will show in Section 3.4.4.

Rather than calculating the exact values of $|\mathcal{C}_f(B, B')|$ and $|\mathcal{C}_f(B', B)|$, we may instead leverage the adjacency graph structure of Theorem 3.1 to calculate a probabilistic estimate of their expected values. In turn, this allows us to estimate the growth in adjacency edges from one level of the AGH to the next, before actually constructing the next level.

Theorem 3.2 (Probabilistic estimate of adjacency growth between successive AGH levels). *Let $\{G_\ell\}_{\ell=1}^{\ell_{\max}}$, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$ be the AGH over a RMP tree in a d -dimensional space, and let $B \in \mathcal{V}_\ell$ be a box at level ℓ . Assume that the point subset $B \cap \mathcal{X}$ is sparsely distributed within B , such that $|\mathcal{C}(B)| = c \ll 2^d$. The value of c is known*

empirically by the tree partition, but the spatial placement of the children within B is unknown; we assume any placement is equally likely.

For any adjacency face $B \cap B'$ of high enough dimensionality, $c < 2^{\dim(B \cap B')}$, the number of children that are incident on $B \cap B'$ can be approximated by a Bernoulli distribution:

$$\Pr [|\mathcal{C}_f(B, B')| = a] \approx p^a (1 - p)^{c-a}, \quad (3.15)$$

where $p = 2^{\dim(B \cap B') - d}$. Using (3.15) and empirical measurements of G_ℓ , we may estimate the expected number of adjacency edges at level $\ell + 1$ by

$$\mathbb{E} [|\mathcal{E}_{\ell+1}|] = \sum_{(B, B') \in \mathcal{E}_\ell} \sum_a \sum_{a'} a a' \Pr [|\mathcal{C}_f(B, B')| = a] \Pr [|\mathcal{C}_f(B', B)| = a'], \quad (3.16)$$

where the summation index a ranges from 1 to $\min\{|\mathcal{C}(B)|, 2^{\dim(B \cap B')}\}$ (and similarly for a').

Proof. The random event that a child of a d -dimensional box is incident on an adjacency face of dimensionality $\dim(B \cap B') = f < d$ is equivalent to the event that a random d -bit string has its first f bits set (see also Section 3.4.4). Since children boxes are disjoint, the event that a children are incident on the adjacency face amounts to a repeated sampling of d -bit strings *without* replacement. The exact form of the distribution of $|\mathcal{C}_f(B, B')|$ becomes a little complicated, thus we opt for an approximation for the sake of simplicity: we assume a repeated sampled of d -bit strings *with* replacement, which directly leads to the Bernoulli distribution in (3.15). The approximation becomes better as d increases, since the total number of potential d -bit strings (equivalently, children box locations) grows exponentially with d .

The adjacency graph size estimate in (3.16) follows from the definition of the expected value with a discrete distribution and the adjacency propagation structure in Theorem 3.1. □

Remark 3.2. Theorem 3.2 allows us to estimate the size of the adjacency graph at level $\ell + 1$ given the adjacency graph at level ℓ and the size of box partitions at level ℓ . It can serve as an alternative to calculating the sizes of complete bipartite sub-graph components of $\mathcal{E}_{\ell+1}$ that is computationally cheaper than finding $\mathcal{C}_f(B, B')$ for every adjacency edge $(B, B') \in \mathcal{E}_\ell$; this may be useful in practice with the adjacency-sensitive partition of Chapter 4, for example.

Moreover, it provides a way for estimating the expected size of the AGH without having to actually construct it—expect possibly at a few coarse levels to “seed” the propagated adjacency size estimates. This can be useful in all-NN search, where it is common practice to set a hard upper limit to the number of adjacency edges that are explored when searching for neighbors of points in a box. Such strategies are (implicitly) constructing and exploring a sub-graph of the AGH at every level. Theorem 3.2 allows us to estimate the number of edges that were not explored around each host box, which can inform our confidence in the accuracy of the resulting near-neighborhoods.

3.4.2 Measures for evaluating AGH construction

An algorithm for constructing the AGH over a partition tree, $T(B_0)$ creates the adjacency list, $\mathcal{A}(B)$, for every box $B \in \mathcal{V}(T)$. At level $\ell = 1$, the adjacency list of a box comprises all its sibling boxes. Algorithms for AGH construction generally differ in how they process subsequent levels.

We evaluate the complexity of AGH construction algorithms by the following three measures.

- (i) *Traversal length.* The total number of tree edges traversed by the algorithm while examining and locating adjacent box pairs.
- (ii) *Number of adjacency test operations.* The total number of logical operations that are performed for checking and differentiating adjacency patterns. The

unit is a bit-wise logical operation between d -bit words.⁴

- (iii) *Amount of working memory space.* The total amount of memory space used while constructing the AGH, in addition to the memory needed to store the AGH itself. The unit is a d -bit word.

With respect to traversal length, when traversing the tree with a recursive algorithm, we assume that the cost of terminating a recursion path is zero; that is, we do not need to backtrack on the tree to reach the box where that recursion path was initiated. For example, the traversal length of a simple depth-first enumeration of all tree nodes is $|\mathcal{E}(T)| = |\mathcal{V}(T)| - 1$.

Consider two example algorithms, which were also mentioned briefly in Section 3.3. One is naive, but sometimes used in low-dimensional spaces [Sam89; Sch92; Sam06; Yok13]. For every box in the tree, the algorithm seeks to find all boxes at the same level that fall within the adjacency “window”. Conceptually, this resembles a convolution-like operation over the level-wise adjacency graphs. When the dataset, and thereby the tree, is sparse and irregular, this is done by traversing the tree, up and down, from each host box to its adjacency candidates; or, similarly, searching for the relevant nodes in a level-wise box code array [Gar82; DI18]. As the tree becomes sparser, a greater number of such candidates are empty, and this approach becomes highly inefficient with respect to traversal length.

The other algorithm, which follows the dual-tree traversal paradigm, is the one that is conventionally used [GM01; Yok13; Cur15]. The tree is traversed in box-pairs. Starting from the root, all pairs of children boxes are visited recursively and tested for adjacency; the recursion terminates whenever a non-adjacent pair is found. This approach does not suffer from inefficiency due to empty boxes like the previous one, since the tree is used to guide the search, which is further constrained by parental

⁴ If the adjacency tests are calculated using, say, box midpoint coordinates in the domain, we consider those coordinates as ℓ_{\max} words of d -bit length, where ℓ_{\max} is the height of the tree.

adjacency relations at any level. Nevertheless, the algorithm does visit and test a number non-adjacent box pairs before they can be discarded. The degree of over-visiting is due to traversing children pairs in $(\mathcal{C}(B) \setminus \mathcal{C}_f(B, B')) \times (\mathcal{C}(B') \setminus \mathcal{C}_f(B', B))$ for every pair of adjacent boxes, $B \rightarrow \leftarrow B'$ (see Theorem 3.1); over-visiting generally increases with dimensionality.

Proposition 3.5 (Lower bound on traversal length for AGH construction). The traversal length for constructing the level-wise AGH is at least $\sum_{\ell=1}^{\ell_{\max}} (|\mathcal{V}_\ell| + |\mathcal{E}_\ell|)$.

Proof. First, observe that the lower bound above is equal to the number of edges in the tree plus the total number of adjacency edges in the AGH. A traversal cost of 1 edge per box is required to reach every box in the tree, starting from the root. Assume that, for any box B in $T(B_0)$, one can find all its adjacent boxes with the bare minimum cost of traversing exactly 1 tree edge per adjacent box. The total traversal length then becomes $\sum_{B \in \mathcal{V}(T)} (1 + |\mathcal{A}(B)|) = \sum_{\ell=1}^{\ell_{\max}} (|\mathcal{V}_\ell| + |\mathcal{E}_\ell|)$. \square

To our knowledge, no previously existing algorithm is able to construct the AGH without considering spurious pairwise box connections that do not correspond to edges in the AGH. We introduce such an algorithm in the next section.

3.4.3 AGH construction algorithm with minimal cost

By Proposition 3.4 and Theorem 3.1, we can exploit the consistency and node patterns along adjacency lineages to efficiently construct the adjacency graphs across all levels. Specifically, we can eliminate redundancy in traversal and adjacency tests while constructing the AGH, thereby minimizing its cost.

We present a recursive algorithm to that effect. We describe the algorithm in terms of visiting pairs of adjacent boxes while traversing the tree, as each such pair is an edge in the AGH. Starting at the root, we visit all children pairs, whose clique is the level-1 adjacency graph. For each pair, we initiate a descent towards all adjacent

descendant pairs (i.e., complete bipartite sub-graphs) which are in the lineage of the sibling pair adjacency face. This is illustrated on a 1D binary tree in Figure 3.6a, and contrasted to the passive dual-tree traversal. By Theorem 3.1, all adjacent pairs are propagated from the adjacency face of some ancestral sibling pair, therefore this procedure yields all edges in the AGH.

The box codes carry all information necessary to direct the descent to only those children boxes which lie along the relevant boundary face of a parent box, as described in Lemma 3.1 and Proposition 3.4. In fact, two-level code information is sufficient to distinguish between the two cases above and direct the traversal into adjacent descendants. Let us consider the 1D case, since, by Proposition 3.3, in the multi-dimensional case the following simply apply to each dimension separately. Let q, q' be the codes of two adjacent boxes at level ℓ . Since $q \rightarrow\leftarrow q'$, we have $q(\ell) \neq q'(\ell)$ —unless the boxes are identical ($q = q'$), which never occurs by the tree traversal description above. The boxes are siblings if $q(\ell) \neq q(\ell - 1)$ and non-siblings if $q(\ell) = q(\ell - 1)$; the same holds for q' . Consequently, each box in the tree need only be associated with its local orientation code, $q(\ell) = c_\ell$, meaning that no extra information is stored in the tree data structure. We show how to filter boundary-adjacent children boxes with local code operations in Section 3.4.4.

A pseudocode listing of the recursive algorithm for AGH construction can be found in Appendix A as Algorithm A.1.

The following theorem addresses the complexity of the above code-directed algorithm for AGH construction, and states that it is optimal with respect to tree traversal length (cf. Proposition 3.5).

Theorem 3.3 (AGH construction algorithm complexity). *Let $T(B_0)$ be an RMP tree with height ℓ_{\max} . The code-directed algorithm described in this section constructs the adjacency graph hierarchy, $\{G_\ell\}_{\ell=1}^{\ell_{\max}}$, with a minimal traversal length of $\sum_{\ell=1}^{\ell_{\max}} (|\mathcal{V}_\ell| +$*

$|\mathcal{E}_\ell|$) edges crossed in $T(B_0)$. The total cost of adjacency tests during the traversal is at most $\sum_{\ell=1}^{\ell_{\max}-1} \sum_{(B,B') \in \mathcal{E}_\ell} (6 + |\mathcal{C}(B)| + |\mathcal{C}(B')|)$ logical operations between d -bit words. The amount of working memory space required is $O(1) + \sum_{\ell=1}^{\ell_{\max}} |\mathcal{V}_\ell|$ d -bit words, excluding the space for storing the data points and the AGH.

Proof. The minimal traversal length follows from observing that: (i) no box-pair is visited twice; (ii) all visited box-pairs are guaranteed to be adjacent by Proposition 3.4 and Theorem 3.1; and (iii) when descending from each pair of boxes, we cross 1 tree edge to a reference child box and as many edges as there are adjacent boxes along the parental boundary face.

(The pseudocode in Algorithm A.1 actually has a traversal length of $\sum_{\ell=1}^{\ell_{\max}} (|\mathcal{V}_\ell| + 2|\mathcal{E}_\ell|)$, since, for every adjacent children-pair, $C \rightarrow C'$ it recurses into both (C, C') and (C', C) , but it is straightforward to amend it such no duplicate pairs are visited.)

By Proposition 3.6 and Corollary 3.2, calculating the local qode patterns for the face-adjacent descendants of a pair of adjacent boxes, $B \rightarrow B'$ requires 6 logical operations. Applying these patterns to filter the relevant children of B and B' entails at most 1 logical operation per child—less if a data structure such as a compressed prefix trie is used to index the children by their local qodes.

The working memory space amount follows from the fact that the algorithm only requires the local qode of each child box to be stored in the tree data structure. The additional constant term simply accounts for the memory working set while performing the adjacency computations. \square

Figure 3.6 illustrates the difference in traversal length between the minimal-length algorithm for AGH construction described above and the conventional dual-tree traversal approach, by showing the how the two algorithms locate adjacent descendant pairs in a 1D tree. The three complexity measures for both algorithm are compared in Table 3.1.

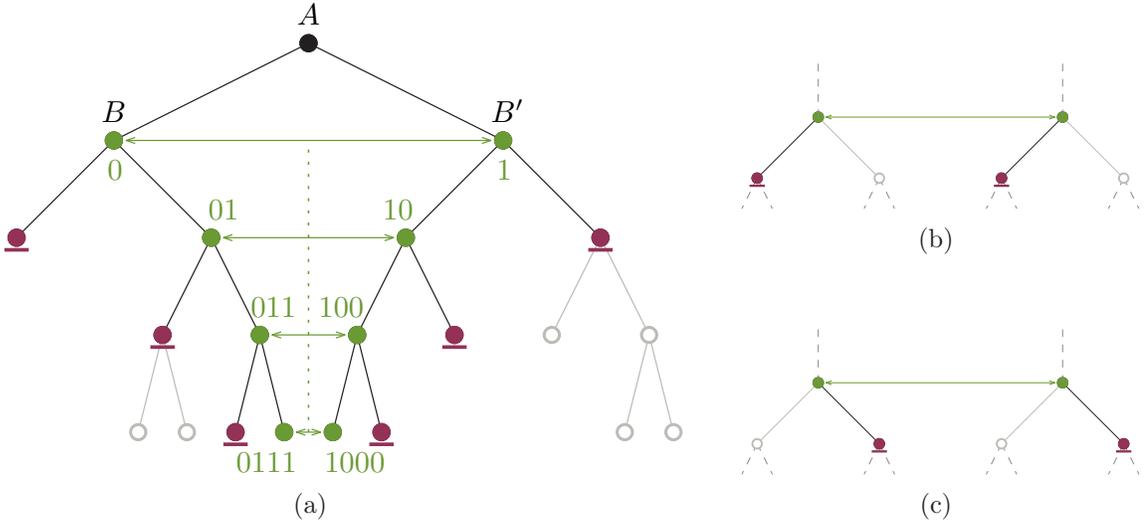


FIGURE 3.6: Illustration of AGH construction descent in 1D binary tree. **Minimal-length traversal:** Whenever a sibling pair (e.g., boxes $B \leftrightarrow B'$ with common ancestor A) is encountered, the AGH algorithm proceeds towards all descendant pairs which lie along the boundary face $B \cap B'$. The algorithm is directed towards the face-adjacent children of each box by the constant-complement code pattern of Lemma 3.1. Only the boxes marked green in (a) are visited during the (B, B') -initiated descent. **Conventional dual-tree traversal:** By comparison, a dual-tree traversal would prune paths beneath non-adjacent boxes without making use of the code adjacency pattern. Consequently, the (B, B') -initiated descent would find the relevant adjacent boxes (green), but would also visit the non-adjacent box pairs marked red in (a), as well as those in (b) and (c) (at all levels of the descent shown in (a)).

Table 3.1: Comparison of AGH construction complexity between direct adjacency code filtering (“AQF”) and conventional dual-tree traversal (“DT”). Comparison in three measures: traversal length, adjacency test operations, and working memory space.

Alg.	Traversal length	Adjacency operations	Working memory
AQF	$\sum_{\ell=1}^{\ell_{\max}} (\mathcal{V}_{\ell} + \mathcal{E}_{\ell})$	$\sum_{\ell=1}^{\ell_{\max}-1} \sum_{(B, B') \in \mathcal{E}_{\ell}} (6 + \mathcal{C}(B) + \mathcal{C}(B'))^{\dagger}$	$O(1) + \sum_{\ell=1}^{\ell_{\max}} \mathcal{V}_{\ell} $
DT	$\sum_{\ell=0}^{\ell_{\max}-1} \sum_{(B, B') \in \mathcal{E}_{\ell}} \mathcal{C}(B) \mathcal{C}(B') $	$\sum_{\ell=0}^{\ell_{\max}-1} \sum_{(B, B') \in \mathcal{E}_{\ell}} (\ell_{\max} \mathcal{C}(B) \mathcal{C}(B'))$	$O(\ell_{\max}) + \sum_{\ell=1}^{\ell_{\max}} \mathcal{V}_{\ell} $

\dagger upper bound

3.4.4 Two-generation adjacency filtering with local qode operations

Here, we show how to filter children boxes to separate adjacent and non-adjacent children pairs, per the adjacency lineage relations, using only local qode information.

The local qodes of d -dimensional boxes are binary words of length d . We may view them as elements of the Galois field \mathbb{F}_2^d . In the proposition below, the addition and multiplication operations between local qodes correspond to dimension-wise addition modulo 2 (bit-wise exclusive disjunction, XOR) and dimension-wise multiplication (bit-wise conjunction, AND), respectively.

Proposition 3.6 (Two-generation adjacency filtering with local qode operations).

Let B and B' be two adjacent boxes in an RMP tree, with qodes $q = [c_1 \cdots c_\ell]$ and $q' = [c'_1 \cdots c'_\ell]$ such that $q \rightarrow\leftarrow q'$. Let $f_\ell = c_\ell + c'_\ell$ be the local descriptor of their adjacency face, $B \cap B'$; similarly, $f_{\ell-1}$ denotes the local descriptor of the adjacency face between the parent boxes of B and B' . Children of B and B' are mutually adjacent if and only if their local qodes, $c_{\ell+1}$ and $c'_{\ell+1}$, satisfy the equations

$$\begin{aligned} c_{\ell+1}f_\ell &= c_\ell f_\ell + f_\ell + f_{\ell-1}, & \text{and} \\ c'_{\ell+1}f_\ell &= c'_\ell f_\ell + f_\ell + f_{\ell-1}. \end{aligned} \tag{3.17}$$

Proof. The f descriptors indicate the dimensions along which the local qodes of two adjacent boxes differ. By Proposition 3.4 and Theorem 3.1, two-level information is sufficient for determining the current qode pattern along each dimension; that is, the local qode bit relationship between face-adjacent children boxes and their parent boxes.

This relationship is encoded by the adjacency face descriptors, f_ℓ and $f_{\ell-1}$. We show how this is done for $c_{\ell+1}$; the same hold for $c'_{\ell+1}$. For each $i \in \{1, \dots, d\}$, if $f_\ell(i) = f_{\ell-1}(i) = 1$, the path to B and B' along the i -th is in the ‘‘constant replication’’ phase (3.12b), and it must hold that $c_{\ell+1} = c_\ell$. If $f_\ell(i) = 1$ and $f_{\ell-1}(i) = 0$, then B

and B' are siblings along the i -th axis and the corresponding axial projection of the path to their adjacent descendants enters the “conjugate flip” phase (3.12a), hence it must hold that $c_{\ell+1} = \overline{c_\ell}$. Last, if $f_\ell(i) = f_{\ell-1}(i) = 0$, the paths to B and B' along the i -th axis are identical, and $c_{\ell+1}$ can be either 0 or 1. The case where $f_\ell(i) = 0$ and $f_{\ell-1}(i) = 1$ is impossible because the adjacency face at level ℓ is a subset of that at level $\ell - 1$.

We can express all this with simple logical operations between elements of \mathbb{F}_2^d . The operation $a + b$ flips the bits of a at the positions where b is set. Hence, $c_\ell + f_\ell + f_{\ell-1}$ flips the bits of c_ℓ at all positions where f_ℓ is set but $f_{\ell-1}$ is unset, realizing the adjacency lineage patterns in Proposition 3.4. The bit positions where $c_{\ell+1}$ can be either 0 or 1 are masked off by multiplication with f_ℓ (these are the dimensions that are perpendicular to the $B \cap B'$ boundary face). This gives us the equation $c_{\ell+1} f_\ell = (c_\ell + f_\ell + f_{\ell-1}) f_\ell$. The right-hand side is simplified because $a^2 = a$ for any $a \in \mathbb{F}_2$; and $f_{\ell-1} f_\ell = f_{\ell-1}$ since $f_{\ell-1}(i) = 1 \implies f_\ell(i) = 1$. \square

The way in which Proposition 3.6 is applied to filter face-adjacent descendants depends on how the children of a box are stored in the tree. If the tree is stored in an explicit, pointer-based data structure, two convenient ways for organizing children boxes are by a binary trie or Patricia tree [Knu98] over their local qodes, or by an array or hash table indexed by their local qodes. A simple example of using a binary trie or array to store and filter children is shown in Figure 3.7. The binary trie and Patricia tree structures have the benefit that only the relevant boxes are actually reached (spurious paths are pruned before reaching the leaves) and that common local-qode prefixes are processed once. However, they may require up to d logical operations for each filtered child box, whereas if d is below the machine word length a d -bit operation can be carried out in a single cycle. The array structure has the benefit that it is very simple to implement and allows for coalesced vector operations

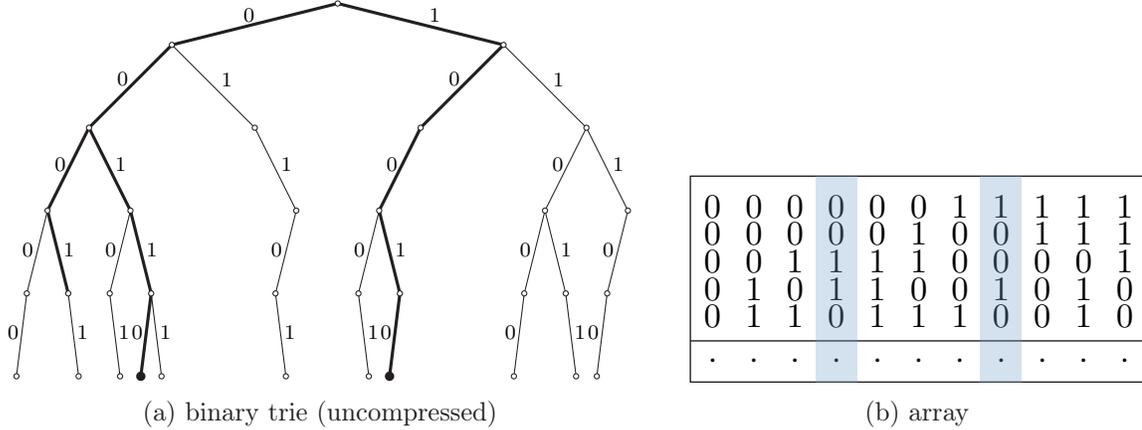


FIGURE 3.7: Example data structures for storing and filtering children boxes: binary trie and array of local codes. In the binary trie, paths from the root to the leaves correspond to the local codes of the children boxes; the latter are stored at the leaves of the trie. In the children code array, the children boxes are stored (via a pointer or index) together with their respective local code. Adjacency filtering amounts to identifying those children whose local codes satisfy the filtering equation (3.17). This is illustrated here with $f_p = 01001$, $f = 01011$, and $c_\ell = 00100$, which selects children boxes whose local code is of the form $c_{\ell+1} = *0*10$, where $*$ means that the bit in the corresponding dimension may take either value.

when filtering children nodes, which are very efficient with modern computer architectures. However, it requires that all children nodes are processed when applying the filtering equation.

If the tree is stored implicitly, e.g., by appropriately reordering the points in \mathcal{X} in a linear array indexed by global codes [Gar82; DI18], the process of filtering is essentially the same as with the local array structure described above. The difference is that rather than seeking single array cells, we now seek relevant sub-array ranges. The filtering equation (3.17) may afford an additional benefit in some contexts such as multi-scale interaction computations [qzhangDynamicPrioritization2011; Gre87; SP01; Yok13; vdMaa14; PIF⁺19]. One may use the filter pattern to resort the sub-array that corresponds to the parent box such that all relevant children “slabs” or sub-ranges are contiguous in the resorted order. The resorting does incur an extra computational cost, but that may be well paid off by the increased memory locality in subsequent arithmetic operations.

Corollary 3.2. Given a pair of adjacent boxes in a d -dimensional RMP tree, the local qode pattern of their children boxes that are adjacent to each other (i.e., the right-hand sides in (3.17)) can be computed with 6 logical operations⁵ between d -bit words, using the two-level recurrence equation (3.17).

Using that pattern to filter out non-adjacent children takes no more than 1 logical d -bit operation per child box.

Remark 3.3. The qode adjacency pattern of Lemma 3.1 has been observed, to some extent, in previous work [Sam82; Sam89; Sch92; Vör00], but attempts to leverage it for locating adjacent boxes have invariably taken the route of keeping look-up tables with qode operations for each possible neighbor direction and either backtracking or searching among the entire point-set.

⁵ The sum $f_\ell + f_{\ell+1}$ need only be computed once.

Adjacency-Sensitive Partitions

The construction of the recursive mid-point partition (RMP) hierarchy that was described in Section 3.1 is adaptive to sparsity in the point distribution in space, but otherwise oblivious to the structure of the corresponding adjacency graph hierarchy (AGH). Indeed, this is true of virtually all partition hierarchies which satisfy level-wise coverage of the underlying point cloud, including k -d trees [Ben80; SH08; ML14; RS19], k -means trees [ML14], ball trees [GM03], and PCA trees [CFS09; VKD09]: the partitioning of a box depends solely on the local distribution of points within it, irrespective of the local distributions within its adjacent boxes.

In this chapter, we amend the RMP tree to yield a partition hierarchy that is conducive to sparsity in the corresponding AGH. Rather than separate the partition and adjacency graph construction into two distinct stages, as is done conventionally, we construct both in tandem. The combined partitioning and adjacency filtering process is carried out level-by-level, in order to enable a box-interactive partition strategy whereby adjacent boxes communicate and consult with one another while determining their local partition structure. At every level, the AGH informs the partitioning of each box such that the total number of box adjacency edges in the

next level is reduced.

In the context of all-near-neighbors (all-NN) search, where box adjacency is used to direct the search onto relevant point subsets (see Chapter 5), a sparser AGH reduces the total cost of traversing the hierarchy index to locate nearby point subsets. If a limit is placed on the number of adjacent boxes to be checked during the search, then a sparser AGH leads to higher expected accuracy in the resulting k -nearest neighbors (k NN) graph.

4.1 Partially partitioned boxes

We seek a partition hierarchy that retains the geometric properties of the RMP tree, while being sensitive to inter-box adjacency relationships at every tree level. The partition structure should still be adaptive to point distribution sparsity, and support the parsimonious representation and efficient processing of boxes through local qodes, but at the same time we want to avoid partitions that introduce unnecessary edges in the AGH.

We shall clarify the adjacency-sensitivity condition shortly, in Section 4.2. First, we briefly discuss a potential issue with uniform RMP trees in multi-dimensional spaces, and describe a variant based on partial box partitions to remedy that issue.

4.1.1 Mass-deficient boxes in uniform partitions

The uniform partition of each box into 2^d subdomains (including both empty and non-empty boxes) may lead to many boxes that are mass-deficient, i.e., boxes that contain only a small number of points, far below the mass threshold, p . We refer to the resulting tree as a uniform RMP tree. We will use a “u” superscript to denote the corresponding tree partitions and AGH; e.g., $G_\ell^u = (\mathcal{V}_\ell^u, \mathcal{E}_\ell^u)$ is the level- ℓ box adjacency graph, where boxes in \mathcal{V}_ℓ^u are obtained by a uniform mid-point partition of (non-leaf) boxes in $\mathcal{V}_{\ell-1}$.



FIGURE 4.1: Axial slice of the thoracic CT image used in Example 4.1. Image patches are extracted and used as multi-dimensional features in all-NN search for subsequent processing with algorithms such as non-local means filtering [BCM05; MCM⁺10].

Example 4.1 (Mass-deficient leaf boxes in uniform RMP tree). Let $\mathcal{X} \subset \mathbb{R}^9$ be the set of all 3×3 patches in an axial slice of a thoracic CT image, shown in Figure 4.1. The image size is 512×512 and there are $|\mathcal{X}| = 207,782$ unique patches. The features/coordinates of each patch are its pixel values. The CT image is taken from the collection of the Deformable Image Registration Laboratory (DIR-Lab) at the University of Texas Medical Branch. Specifically, Figure 4.1 shows the central axial slice of the end-of-expiration CT image of the patient labeled “COPD4” in the DIR-Lab collection [CCF⁺13].

Figure 4.2 shows the spatial scale (tree level) and point mass distribution of leaf boxes in a uniform RMP tree over the set of image patches, with height $\ell_{\max} = 9$ and mass threshold $p = 64$. Roughly half of all leaf boxes contain only a single point.

Mass-deficient boxes increase the size of the partition tree and greatly increase the number of edges in the AGH (see also Theorem 3.1). Yet, such boxes offer little or no improvement with respect to efficiency in interaction or neighbor-search computations: the mass threshold p reflects a small enough size such that it is preferable to process all points in a box B with $\text{mass}(B) \leq p$ at once rather than partition

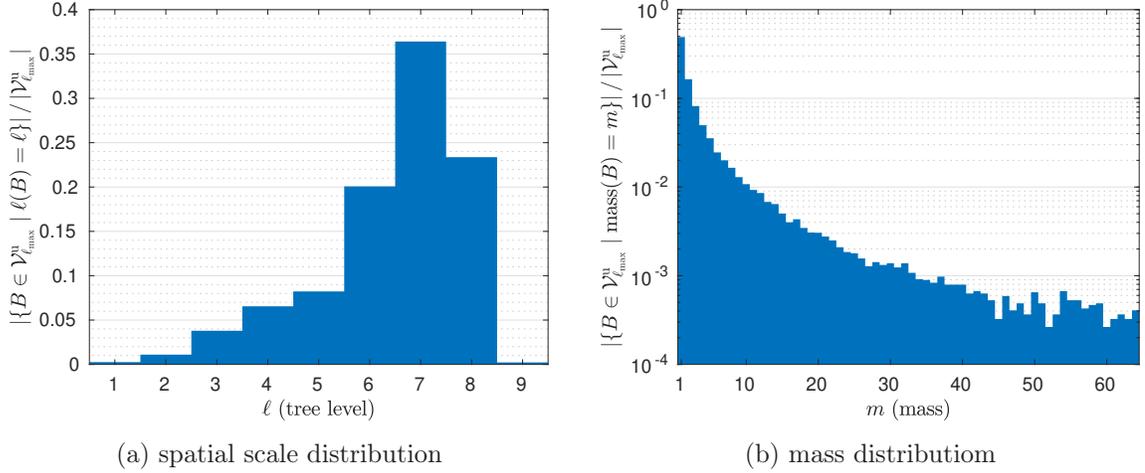


FIGURE 4.2: Histograms of spatial scale (tree level) and point mass distributions of leaf boxes in a uniform RMP tree over 3×3 patches in the thoracic CT image of Figure 4.1. The height of the tree is $\ell_{\max} = 9$ and the leaf mass threshold is $p = 64$. Both histograms are normalized by the total number of leaf boxes in the tree. The mass distribution histogram is shown in log-scale.

them further.¹

4.1.2 Partial mid-point partition

We consider *partial* partitions as a means to reducing the number of mass-deficient children when partitioning a box. That is, given a box B , some of the axis-wise intervals of its children boxes may be the same as those of B .

Definition 4.1 (Partially partitioned child box). Let $B = \bigotimes_{i=1}^d [a_i, b_i]$ be a box in an RMP tree, with $\ell(B) < L$ and $\text{mass}(B) > p$. The spatial domain of a partially partitioned child box, $C \in \mathcal{C}(B)$, is

$$C = \left(\bigotimes_{i \in \mathcal{D}_L} [a_i, b_i/2] \right) \left(\bigotimes_{i \in \mathcal{D}_R} [a_i/2, b_i] \right) \left(\bigotimes_{i \in \mathcal{D} \setminus (\mathcal{D}_L \cup \mathcal{D}_R)} [a_i, b_i] \right), \quad (4.1)$$

where $\mathcal{D} = \{1, \dots, d\}$; $\mathcal{D}_L, \mathcal{D}_R \subseteq \mathcal{D}$ indicate the dimensions along which C occupies the left or right half-interval of B , respectively; and $\mathcal{D} \setminus (\mathcal{D}_L \cup \mathcal{D}_R) \neq \emptyset$ indicates

¹ We also note that further partitioning may reduce data locality in computations, as it is difficult to guarantee that nearby points in separate boxes will be processed together, especially on a parallel architecture.

non-partitioned dimensions.

Partially partitioned children boxes are irregular, i.e., their side-intervals need not all have the same length. If B is a regular box at level ℓ and $C \in \mathcal{C}(B)$ was partitioned along $d_p = |\mathcal{D}_L \cup \mathcal{D}_R|$ dimensions, then the volume of the child box is $\text{vol}(C) = 2^{-(d\ell+d_p)}$.

We may refer to a partition tree where each box is recursively partitioned into children boxes as per Definition 4.1 as a non-uniform RMP tree. The uniform partition of Section 3.1.1 is simply a special case of the non-uniform partition we consider here, where $\mathcal{D}_L \cup \mathcal{D}_R = \mathcal{D}$ in (4.1).

4.1.3 Partial partition codes

The binary codes in Definition 3.1 are not sufficient for encoding the placement of boxes in a non-uniform RMP tree. By (4.1), we can describe a partially partitioned box with a ternary codeword instead; that is, for any box B_ℓ at level ℓ with parent box A , the partial-partition or non-uniform RMP code of B is

$$q(B_\ell) \triangleq [q(A) \ c_\ell], \quad c_\ell \in \{0, 1, *\}^{d \times 1}, \quad (4.2)$$

where

$$c_\ell(i) = \begin{cases} 0, & i \in \mathcal{D}_L, \\ 1, & i \in \mathcal{D}_R, \\ *, & i \in \mathcal{D} \setminus (\mathcal{D}_L \cup \mathcal{D}_R), \end{cases} \quad (4.3)$$

and $q(A) \in \{0, 1, *\}^{d \times (\ell-1)}$ is the parent box code. The ternary code definition for partially partitioned boxes extends the binary code for uniformly partitioned ones, with the $*$ symbol indicating a non-partitioned axial interval.

4.1.4 Adjacency relations between partially partitioned boxes

The code patterns for box adjacency, as shown in Section 3.3 remain practically the same, with a slight reinterpretation of the relations therein. In essence, we treat $*$

as a wildcard symbol, meaning that in the context of whether or not two qodes are adjacent ($q \rightarrow \leftarrow q'$), we accept “ $* = 0$ ” and “ $* = 1$ ” as true; accordingly, $\bar{*} = *$. In the lineage phases of (3.8) and (3.12), $\beta \in \{0, 1\}$ is still a binary value, since $q(j+1)$ and $\overline{q'(j+1)}$ must differ for the paths to q and q' to diverge.

Ternary qodes are impractical in program implementations. Hence, we may represent the qode of a box B_ℓ , at level ℓ of a non-uniform RMP tree, with a $d \times 2\ell$ binary codeword:

$$\tilde{q}(B_\ell) \triangleq [\tilde{q}(A) \ c_\ell \ m_\ell], \quad c_\ell \in \{0, 1\}^{d \times 1}, \quad m_\ell \in \{0, 1\}^{d \times 1}, \quad (4.4)$$

where $\tilde{q}(A) \in \{0, 1\}^{d \times 2(\ell-1)}$ is the parent-box qode, and m_ℓ is a dimension-wise partition mask with $m_\ell(i) = 1$ if $i \in \mathcal{D}_L \cup \mathcal{D}_R$ and $m_\ell(i) = 0$ otherwise. The local qode descriptor c_ℓ is as before, except it can take any value along non-partitioned dimensions. (In practice, it can be useful for the value of c_ℓ to reflect the half-interval that accounts for the majority of the box mass along non-partitioned dimensions.)

The local qode equations of Proposition 3.6 for filtering adjacent children boxes are easily amended to cover the case of partially partitioned boxes. The only change that needs to be made in Proposition 3.6 is in the definition of the shared boundary face descriptor, f_ℓ , between two adjacent boxes with qodes $\tilde{q} \rightarrow \leftarrow \tilde{q}'$. We now let the face descriptor be $f_\ell = (c_\ell + c'_\ell) m_\ell m'_\ell$, and the qode adjacency filtering equations in (3.17) stay the same. That is, m_ℓ and m'_ℓ are used to mask out non-partitioned dimension bits when filtering children boxes by their local qodes.

4.2 Adjacency-sensitive partition model

We formulate the problem of finding an adjacency-sensitive RMP hierarchy via a level-wise global minimization. We then relax the minimization problem to a local one, reducing the size of the solution space and allowing for each box to partitioned separately.

4.2.1 Global model

Ideally, given a partition tree and corresponding AGH up to level ℓ , we seek a partition that minimizes the number of adjacency edges, $|\mathcal{E}_{\ell+1}|$, at the next level, without compromising adaptivity to sparsity in the point distribution. Since $|\mathcal{E}_\ell| = \sum_{B \in \mathcal{V}_\ell} |\mathcal{A}(B)|/2$ at any level ℓ , we may express this as

$$\begin{aligned} \mathcal{V}_{\ell+1}^* = \arg \min_{\mathcal{V}_{\ell+1} \in \mathcal{P}(\mathcal{V}_{\ell+1})} \sum_{C \in \mathcal{V}_{\ell+1}} |\mathcal{A}(C)|, \\ \text{s.t. } \mathcal{V}_{\ell+1}^* \supseteq \{C \in \mathcal{V}_{\ell+1}^u \mid \text{mass}(C) \geq p\}. \end{aligned} \quad (4.5)$$

In the above, $\mathcal{P}(\mathcal{V}_{\ell+1})$ denotes the set of all possible combinations of partially partitioned children boxes at level $\ell + 1$. The constraint ensures that any children boxes per a uniform mid-point partition that would contain at least p points must be included in the result. Boxes in \mathcal{V}_ℓ which contain fewer than p points are not partitioned, since that would introduce additional adjacency edges.

We refer to (4.5) as a global model for adjacency-sensitive box partitions, due to the fact that the partitioning of a box $B \in \mathcal{V}_\ell$ depends on the how its adjacent boxes, $B' \in \mathcal{A}(B)$, are partitioned, and vice versa.

This coupling or inter-dependence between the partitions of adjacent boxes, together with the extremely large size of the solution space, precludes us from finding a solution to (4.5) efficiently. We thus relax it to a local model, where we consider the contribution to $|\mathcal{E}_{\ell+1}|$ due to the partition of each box $B \in \mathcal{V}_\ell$ separately.

4.2.2 Local model

We group the summands in (4.5) by their parent box in \mathcal{V}_ℓ , as

$$\sum_{C \in \mathcal{V}_{\ell+1}} |\mathcal{A}(C)| = \sum_{B \in \mathcal{V}_\ell} \sum_{C \in \mathcal{C}(B)} |\mathcal{A}(C)|. \quad (4.6)$$

We further separate the sum of adjacency edges that are incident on the children of a box B into two parts: edges between children pairs, and edges between a child of B and children of boxes adjacent to B , corresponding to the adjacency sub-graph components in Theorem 3.1. In particular, we have

$$\sum_{C \in \mathcal{C}(B)} |\mathcal{A}(C)| = \underbrace{|\mathcal{C}(B)|^2}_{\substack{\text{sibling clique} \\ \text{(internal edges)}}} + \sum_{B' \in \mathcal{A}(B)} \underbrace{|\mathcal{C}(B) \cap (B \cap B')| |\mathcal{C}(B') \cap (B \cap B')|}_{\substack{\text{boundary-face children bipartite graph} \\ \text{(external edges)}}}. \quad (4.7)$$

Thus, we have separated the adjacency edges that are introduced when partitioning a box B into *internal* and *external* edges, relative to the domain of B .

By (4.7), the number of edges due to the partition of B cannot be calculated locally, since the external-edges terms depend on the partitions of boxes that are adjacent to B . We may, however, calculate an upper bound for the number of external edges by considering the number of potential children of an adjacent box, $B' \in \mathcal{A}(B)$, that would result from a uniform mid-point partition of B' . Define

$$e(B', B) \triangleq |\mathcal{C}^u(B') \cap (B \cap B')|, \quad (4.8)$$

where $\mathcal{C}^u(B')$ denotes the children boxes of B' due to a uniform mid-point partition. Clearly, $|\mathcal{C}(B') \cap (B \cap B')| \leq e(B', B)$, since, if partial partitions are allowed, $|\mathcal{C}(B)| \leq |\mathcal{C}^u(B)|$ for any box B .

We may now attempt to minimize each term of the outer sum in (4.6) separately. Given a set of boxes at level ℓ and their adjacency graph, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$, we seek a partition

$$\mathcal{V}_{\ell+1}^* = \left(\bigcup_{\substack{B \in \mathcal{V}_\ell \\ \text{mass}(B) > p}} \mathcal{C}^*(B) \right) \cup \{B \in \mathcal{V}_\ell \mid \text{mass}(B) \leq p\}, \quad (4.9)$$

where

$$\mathcal{C}^*(B) = \underset{\mathcal{C}(B) \in \mathcal{P}(\mathcal{C}(B))}{\operatorname{argmin}} \left\{ |\mathcal{C}(B)|^2 + \sum_{B' \in \mathcal{A}(B)} e(B', B) |\mathcal{C}(B) \cap (B \cap B')| \right\}, \quad (4.10)$$

s.t. $\mathcal{C}^*(B) \supseteq \{C \in \mathcal{C}^u(B) \mid \operatorname{mass}(C) \geq p\}$.

Similarly to (4.5), $\mathcal{P}(\mathcal{C}(B))$ denotes the set of all possible combinations of partially partitioned children boxes of B .

We refer to (4.10) as the local model for adjacency-sensitive box partitions, and to the minimization argument as the *adjacency cost* associated with a partition of a box $B \in \mathcal{V}_\ell$. Calculating the adjacency cost for some partitioning of B requires some information from its adjacent boxes, namely $e(B', B)$, but does not depend on how these adjacent boxes will be partitioned.

4.3 Partition and AGH construction algorithm

In this section, we present an algorithm for constructing an adjacency-sensitive partition and the corresponding AGH. We first consider an algorithm for the local partition of a single box as per (4.10). We then design an algorithm for level-wise construction of the entire partition and adjacency graph hierarchy, making use of the local partition algorithm.

4.3.1 Local partition of a box

The local model for adjacency-sensitive partitions greatly simplifies the problem compared to the global model, but the size of the local solution space still poses a challenge. That is, $|\mathcal{P}(\mathcal{C}(B))|$ grows at least factorially with the number of leaf children boxes per a uniform partition of B . We therefore consider a greedy algorithm where we recursively select a dimension along which to partition a box, progressively refining it into irregular slabs in the sense of (4.1), such that each axial partition

minimizes the number of introduced adjacency edges.

Before we describe the partition algorithm, let us briefly discuss how the partition of a box B affects the resulting adjacency cost. The adjacency cost comprises internal and external adjacency edges. The internal-edges cost depends only on the number of children boxes in the partition of B . Minimizing it is tantamount to minimizing the number of mass-deficient children boxes. We introduce an additional mass threshold parameter, p_{low} , to define mass-deficient children boxes; that is, a (potential) child box C is mass-deficient if $\text{mass}(C) \leq p_{\text{low}}$. We typically set $p_{\text{low}} = p/2$. The external-edges cost depends on the spatial distribution of adjacent boxes around B and the spatial distribution of points within them. Roughly speaking, partition hyperplanes that are perpendicular to boundary faces which are incident to fewer (potential) children of adjacent boxes incur a lower adjacency cost.

We provide an illustration of the adjacency-sensitive partition of a box in the following example.

Example 4.2. Figure 4.3 shows two sample 2D point distributions in a host box B and its adjacent boxes, $B' \in \mathcal{A}(B)$. We discuss how the host box is partitioned per the local adjacency-sensitive partition model of (4.10). We use a mass threshold of $p = 8$.

Sub-example 4.2.a (Unequal external adjacency costs per dimension-wise partition). In Figure 4.3a, each potential child box in $\mathcal{C}^u(B)$ contains 4 points, hence only a single partition line is necessary. The external adjacency cost of partitioning B along the x -axis is $1 + 4 + 1 + 0 + 0 + 0 + 1 + 1 = 8$ (where the summands correspond to the face-wise costs in clockwise order, starting from the northwestern face). The external adjacency cost of partitioning B along the y -axis is $1 + 2 + 1 + 0 + 0 + 0 + 1 + 2 = 7$. The internal adjacency cost for either partition is 2. Hence, we partition B along the y -axis, yielding a total adjacency cost of 9.

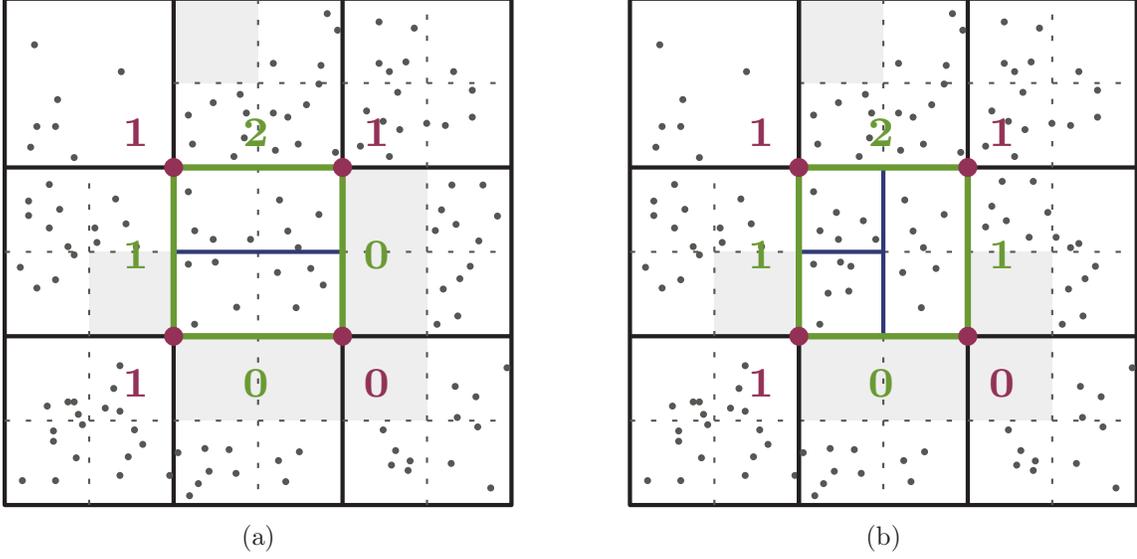


FIGURE 4.3: Illustration of adjacency-sensitive partition of a 2D box with two example point distributions. The box $B \in \mathcal{V}_\ell$ to be partitioned is the one in the middle, shown together with its adjacent boxes. The mass threshold for the partition is $p = 8$. Boxes in the uniform partition, $\mathcal{V}_{\ell+1}^u$, are drawn with dotted lines (the northwestern box is not partitioned as it contains only 7 points). Empty boxes per the uniform partition are shaded gray. The shared boundary faces between B and each of its adjacent boxes, $B' \in \mathcal{A}(B)$, are highlighted along with the corresponding bound for adjacent children, $e(B', B)$; 1D faces are shown in green, 0D faces are shown in red. See Example 4.2 for a description of how B is partitioned per (4.10) in cases (a) and (b).

Sub-example 4.2.b (Equal external adjacency costs per dimension-wise partition). In Figure 4.3b, no children boxes in $\mathcal{C}^u(B)$ contain more than $p = 8$ points. The external adjacency cost of partitioning B along the x -axis is $1+4+1+1+0+0+1+1 = 9$. The corresponding cost of partitioning B along the y -axis is $1+2+1+2+0+0+1+2 = 9$. If we were to partition B along the y -axis, both of the resulting rectangular boxes would contain more than p points and would therefore have to be partitioned along the x -axis as well, yielding an internal adjacency cost of $4^2 = 16$. Hence, we partition B along the x -axis and then only need to further partition the western rectangular child, yielding an internal adjacency cost of $3^2 = 9$. The total adjacency cost is 19.

We now describe an algorithm for realizing an adjacency-sensitive partition of a box as illustrated in Example 4.2. Assume that, for each box $B \in \mathcal{V}_\ell$, we know $\mathcal{A}(B)$ and the local codes of all children, $\mathcal{C}^u(B)$, per a uniform partition of B ; we will

show how this is achieved in Section 4.3.2. For each dimension $i \in \{1, \dots, d\}$, we measure the external adjacency cost, $\sum_{B' \in \mathcal{A}(B)} e(B', B) |\mathcal{C}(B) \cap (B \cap B')|$, that would be incurred if B were partitioned along its i -th axial interval mid-point. We partition B along the dimension with the lowest adjacency cost, among those dimensions that would not yield mass-deficient children boxes.² If there is more than one dimension that satisfies this, we pick the one that leads to a more balanced partition with respect to the children box populations. Once we partition B along a dimension, we remove this dimension from consideration and repeat the process recursively for each of the partially partitioned children boxes of B .

The partition algorithm finds a local minimum of (4.10) by recursively selecting partition hyperplanes such that each partial partition incurs the lowest external adjacency cost. The total number of children boxes is minimized, since mass-deficient boxes per a uniform partition are effectively merged into larger, irregular boxes whenever possible. The constraint of (4.10) is satisfied because any partially partitioned child box C' with $\text{mass}(C') \geq p$ that is created in the process of partitioning B is partitioned further until the mass threshold is reached C' is regular (i.e., its axial intervals have been partitioned in all dimensions).

4.3.2 Level-wise partition and AGH construction

By the partition model in Section 4.2.2 and the direct adjacency location operations in Section 3.4, the adjacency-sensitive partition algorithm exhibits an interdependence between constructing the partition tree and the AGH. Namely, in order to compute $\mathcal{V}_{\ell+1}$, we need $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$; and, as shown in Section 3.4 in order to compute $\mathcal{E}_{\ell+1}$, we need $\mathcal{V}_{\ell+1}$ and G_ℓ . This suggests that the partition tree must be constructed in tandem with the AGH, construction must be done level-wise.

² If every axial partition would yield at least one mass-deficient child box, then we simply pick the dimension with the lowest adjacency cost.

The level-wise construction requirement is met by altering the order in which boxes or box-pairs are processed from depth-first to breadth-first. For example, in the AGH construction algorithm of Section 3.4.3 (listed in pseudocode as Algorithm A.1), instead of recurring into children box-pairs, we may insert them into a first in, first out (FIFO) queue; box pairs are processed by popping the queue head while it is nonempty.

To construct the partition in tandem with the corresponding adjacency graphs, it suffices to process \mathcal{V}_ℓ before \mathcal{E}_ℓ at each level ℓ . Specifically, we first process each $B \in \mathcal{V}_\ell$ to find its children boxes, $\mathcal{C}(B) \subset \mathcal{V}_{\ell+1}$, and the corresponding adjacency clique, $\mathcal{C}(B) \times \mathcal{C}(B) \subset \mathcal{E}_{\ell+1}$; and then process each $(B, B') \in \mathcal{E}_\ell$ to add the corresponding descendant edges (face-adjacent children bipartite graph) per the partition, $\mathcal{C}_f(B, B') \times \mathcal{C}_f(B', B) \subset \mathcal{E}_{\ell+1}$, where $\mathcal{C}_f(B, B') = \{C \in \mathcal{C}(B) \mid C \cap (B \cap B') \neq \emptyset\}$. The algorithm is initialized with the degenerate root graph, $G_0 = (B_0, \emptyset)$. A pseudocode of the algorithm is listed in Appendix A as Algorithm A.3.

When computing the partition of a box, we require not only its adjacent boxes but also the (potential) children box distribution for each adjacent box, as discussed in Section 4.3.1. To that end, whenever we partition a box $B \in \mathcal{V}_\ell$ into its children boxes, $\mathcal{C}(B)$, we also initialize its grandchildren as $\mathcal{C}^u(C)$, for each $C \in \mathcal{C}(B)$. This amounts to computing the local qode distribution among all points in $\mathcal{X} \cap C$, at resolution level $\ell + 2$. By this initialization, when we proceed to partition boxes in \mathcal{V}_ℓ , we may calculate adjacency costs without having to recompute the local children qodes for each adjacent box. As soon as we compute the partition of a box $C \in \mathcal{V}_{\ell+1}$, $\mathcal{C}(C)$ is updated to contain its actual children boxes. The above strategy also has the effect that the face-adjacent children bound, $e(B', B)$ is only used before B' is partitioned; afterwards, we calculate the actual number of face-adjacent children of B' , for any of its boundary faces.

All-NN Search in Multi-Dimensional Space

In this chapter, we discuss how to leverage the adjacency graph hierarchy (AGH) towards efficient all-near-neighbors (all-NN) search among points in a multi-dimensional metric space. We first address the case of range- r near neighbors (r NN) search in Section 5.1. Efficient solutions to this problem have been known for a long time, for example with the help of a k -d tree [Ben76]. Nevertheless, the AGH is intimately connected to r NN graphs and directly leads to an effective algorithm for building the latter. We then proceed to the more difficult problem of all- k -nearest neighbors (k NN) search in Section 5.2, building upon our solution for r NN search.

In our presentation of all-NN search with the AGH, we will generally assume that the latter corresponds to a uniform recursive mid-point partition (RMP) tree, for simplicity. Nonetheless, it is straightforward to extend our results to AGHs over non-uniform RMP trees, as per the discussion in Section 4.1.

Throughout this chapter, our notion of a “multi-dimensional” space is as illustrated in Figure 1.2. We will relax that condition and discuss all-NN search in high-dimensional spaces in Chapter 6.

5.1 Coordinated search of all range- r neighborhoods

5.1.1 Single-level box adjacency in r NN search

The level-wise box adjacency graphs of Section 3.4 can be thought of as range neighborhood graphs among box supernodes at specific spatial resolution scales and ranges. As such, the AGH leads directly to a partial solution for all- r NN search. By the spatial gap (3.7) between a box and its non-adjacent boxes, all neighborhood edges in the r NN graph G_r are between points in adjacent boxes at a level ℓ_r such that $-\log_2 r \leq \ell_r < -\log_2 r + 1$. That is, r NN search need only take place among point subsets whose container boxes are adjacent in G_{ℓ_r} .

Proposition 5.1. Let $T(B_0)$ be a sparsity-adaptive RMP tree over a dataset $\mathcal{X} \subset [0, 1]^d$. For every point $\mathbf{x} \in B_0$, its range- r near neighbors, with respect to some L_p metric, lie within boxes in $\mathcal{A}(B)$, where B is the box at level $\ell_r = \lfloor -\log_2 r \rfloor$ which contains \mathbf{x} :

$$\mathcal{N}_r(\mathbf{x}) \subset \mathcal{A}(B), \quad \mathbf{x} \in B \in \mathcal{V}_{\ell_r}, \quad \ell_r = \lfloor -\log_2 r \rfloor. \quad (5.1)$$

Proof. By Definition 3.3, for any box B at level $\ell_r = \lfloor -\log_2 r \rfloor$, points $\mathbf{x} \in B$ and $\mathbf{x}' \in \mathcal{A}_{\ell}^c(B)$, and L_p distance metric d , we have $d(\mathbf{x}, \mathbf{x}') \geq \text{gap}_p(B) \geq \text{gap}_{\infty}(B) = 2^{-\ell_r} \geq r$. Therefore, no range- r near neighbors of \mathbf{x} can lie within $\mathcal{A}_{\ell}^c(B)$. \square

The level- ℓ_r adjacency graph allows us to leverage the RMP tree structure to restrict the r NN search to point subsets within adjacent boxes. It remains to filter out any non-neighboring point pairs within these subsets.

A simple way to do so is to calculate all pairwise distances between points in adjacent boxes at level ℓ_r and add to G_r all pairs with distance at most r . As long as the mass of boxes in \mathcal{V}_{ℓ_r} is reasonably small, this leads to an efficient algorithm for all- r NN search.

Lemma 5.1 (All- r NN search). *Let $T(B_0)$ be a sparsity-adaptive RMP tree over a dataset $\mathcal{X} \subset [0, 1]^d$ in a d -dimensional metric space, and $\{G_\ell\}$ be the AGH over $T(B_0)$. Assume that $|\mathcal{A}(B)| \leq c_a$ for every tree box B , and $\text{mass}(B_{\ell_r}) \leq c_m$ for every box B_{ℓ_r} at level $\ell_r = \lfloor -\log_2 r \rfloor$. Then, computing the all- r NN graph G_r among points in \mathcal{X} by calculating all pairwise distances among points in adjacent boxes in G_{ℓ_r} entails a traversal length of $O(\ell_r c_a N)$ tree edges and $O(c_m c_a N)$ distance evaluations.*

Proof. The level- ℓ_r canopy contains $|\mathcal{V}_{\ell_r}| = O(N)$ non-empty boxes. Using the algorithm of Section 3.4.3 for constructing the AGH, we can visit all box-pairs in \mathcal{E}_{ℓ_r} without descending any “false” paths (i.e., paths to non-adjacent boxes) in the tree. The total number of adjacent box-pairs is $|\mathcal{E}_{\ell_r}| = O(c_a N)$. Since we only visit boxes up to level ℓ_r , the total traversal cost is at most $O(\ell_r c_a N)$; and in general less than that, since there will be shared-prefix paths among boxes in \mathcal{V}_{ℓ_r} .

The adjacency list of each box contains at most $c_m(c_a + 1)$ points, which leads to the $O(c_a c_m N)$ bound for the number of distance evaluations. \square

In short, the algorithm is characterized by single-level box adjacency, followed by point neighborhood filtering. Box adjacency at levels $\ell' < \ell_r$ facilitates the construction of G_{ℓ_r} , as detailed in Section 3.4.3. The AGH may be stored explicitly as part of the tree data structure if the latter is to serve as a preprocessed index to support future r NN queries; or it may be used only implicitly as part of calculating the r NN graph G_r on the fly.

By Lemma 5.1, the efficiency of the algorithm depends on the adjacency expansion constant, c_a , and the box mass bound, c_m , at level ℓ_r . We briefly comment on these two constants here.

A desirable box mass bound is $c_m = O(\log N)$, leading to $O(c_a N \log N)$ distances being evaluated during all- r NN search; or at most $c_m = O(\sqrt{N})$. The RMP tree does not guarantee any such bounds, but we will show shortly how to eliminate

certain unnecessary distance evaluations by using finer-scale boxes in the RMP tree. Moreover, the same approach can be applied with a balanced box decomposition (BBD) tree structure [AMN⁺98], which guarantees that the point mass of a box decreases exponentially with tree level.¹

The adjacency expansion constant, c_a , depends on the dataset and the partition tree structure, and is not known until the AGH is constructed. In the worst case, it is 3^d , but that would imply that data points are distributed uniformly in the domain. In practice, c_a reflects the sparsity of the data distribution—and the extent to which the AGH conforms to it. By this token, we may consider $\log_3 c_a$ as a measure of the effective dimensionality of the AGH; this is conceptually somewhat similar to the bounded expansion constant of Karger and Ruhl [KR02; BKL06], but restricted to partition boxes.

5.1.2 Multi-level refinement of search regions

The number of distance evaluations can be reduced if we consider the spatial relationships of boxes at finer resolution levels than ℓ_r . This comes at the cost of a higher, but almost asymptotically unchanged, traversal length.

The idea is that certain regions of adjacent boxes will be too far apart from each other to contain neighboring points. For example, consider a simple case where boxes are 1D intervals and $\ell_r = -\log_2 r$, i.e., the length of boxes at level ℓ_r is exactly r . For any pair of adjacent boxes (B, B') , their respective children which lie away from the $B \cap B'$ boundary are separated by a gap equal to r . Similarly, we may find, and exclude, well-separated grandchildren pairs at level $\ell_r + 2$, and so on.

In general, if $2^{-\ell_r-1} + 2^{-\ell_r-j-1} < r \leq 2^{-\ell_r}$, such descendant pairs can be found

¹ The only change to the AGH construction algorithm on a BBD tree is that the code of each box is extended with d bits that denote the dimensions along which a shrunken box is “sticky”, in the terminology of Arya et al. [AMN⁺98].

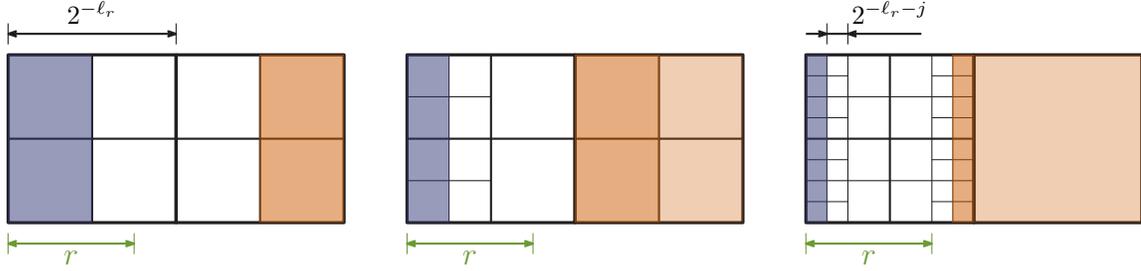


FIGURE 5.1: Descendant sub-box filtering at resolution levels finer than $\ell_r = \lfloor -\log_2 r \rfloor$ during all- r NN search. Points in the blue-shaded boxes cannot have neighbors in the orange-shaded boxes. If $2^{-\ell_r-1}(1+2^j) \leq r \leq 2^{-\ell_r}$, then such children box slabs can be found up to level $\ell_r + j$. This illustration focuses on the leftmost descendants of the left box in the level- ℓ_r box-pair; a similar situation occurs in any descent direction.

up to level $\ell_r + j$.² Descendant separation between adjacent box-pairs is illustrated in Figure 5.1. In multi-dimensional spaces, a pair of boxes are well-separated if and only if they are well-separated along *any* dimensional axis.

By filtering out well-separated descendants as described above, the number of distance evaluations is no higher than with the single-level algorithm of Lemma 5.1. In the worst case, no pair of well-separated descendants comprises non-empty boxes on both sides of the level- ℓ_r boundary, and both algorithms entails the same number of distance evaluations. The traversal length is at worst $O(\ell_{\max}cN)$, since the total number of nodes is $O(N)$.

Figure 5.2 offers a graphical illustration of the effect of sub-box filtering in all- r NN search by highlighting the level- ℓ_r adjacency lists and relevant sub-box domains of three sample leaf boxes in the 2D RMP tree of Example 3.2.

The box qodes allow us to efficiently identify and filter out well-separated children pairs, similarly to how they enable identifying adjacent children pairs. Essentially, the constant complement pattern of Lemma 3.1 becomes “flipped”, to lead the traversal

² Leaf boxes encountered during this process can be partitioned on the fly, but in practice this should be unnecessary. If a pair of boxes are leaves due to the point mass threshold, p , the number of pairwise distance evaluations they entail should be acceptable; if they are leaves due to the spatial resolution level threshold, L , the distance difference at the next level should be negligible. Otherwise, one should revisit the tree resolution thresholds, p and L .

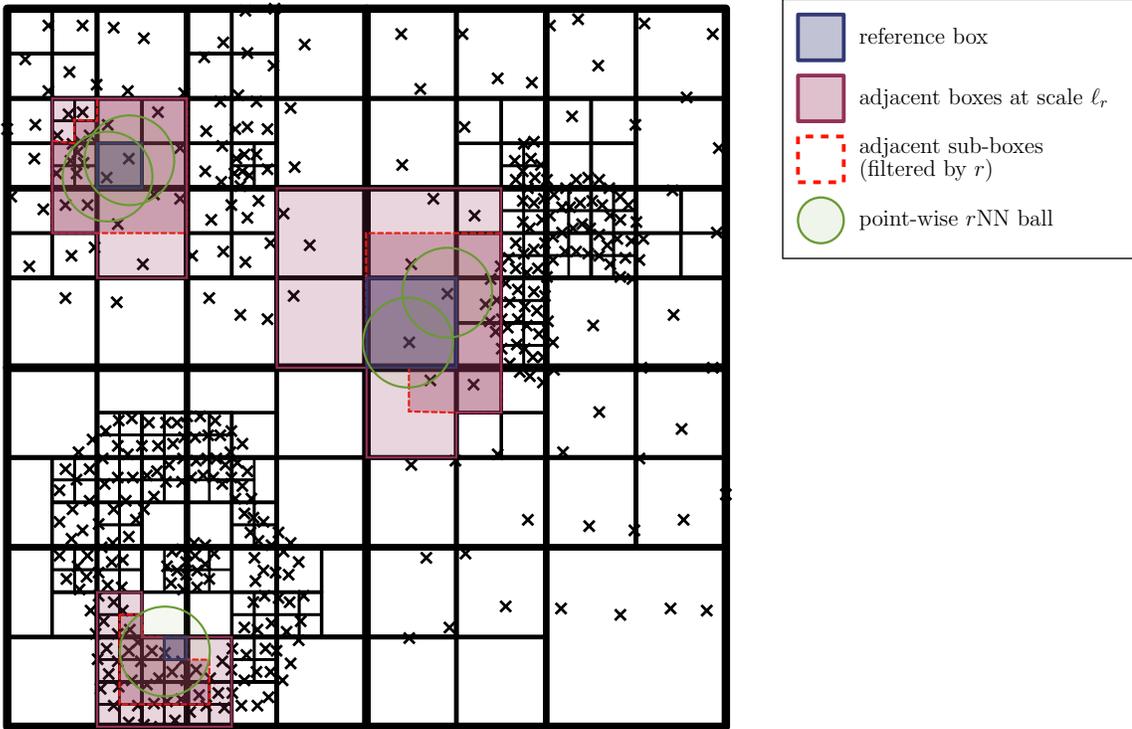


FIGURE 5.2: Adjacent boxes and sub-box filtering for all- r NN search in a sparsity-adaptive 2D RMP tree. The spatial and mass resolution threshold parameters of the tree are $\ell_{\max} = 5$ and $p = 4$, respectively. Three sample reference boxes are highlighted and the r NN balls of the points contained within them are drawn. Also highlighted are the corresponding box adjacency lists at resolution level $\ell_r = \lfloor -\log_2 r \rfloor$, as well as their relevant sub-boxes after finer-scale box filtering (cf. Figure 5.1). The r NN search is only carried out within the filtered sub-box domains.

away, rather than towards, the common boundary; and instead of traversing the boxes that adhere to the new code pattern, we traverse all boxes that do not. The two-level code equation of Section 3.4.4 for identifying adjacent children pairs can be adapted accordingly with ease.

5.2 Coordinated search of all k -nearest neighbors

5.2.1 Multiple RMP resolution levels

By the non-adjacent box gap of Definition 3.3, we may bound the spatial resolution level at which all k -nearest neighbors of points in a box are found in adjacent boxes. Specifically, if a box B contains at least $k + 1$ points, then the k NN of any $\mathbf{x} \in B$,

with respect to the L_∞ distance, must lie within the adjacency list of B . This means that the k NN graph, G_k , among points in a dataset \mathcal{X} is contained within a “hybrid-scale” box adjacency graph whose vertices comprise the finest-resolution tree boxes with a mass of at least $k + 1$ and which cover the dataset, found at various resolution levels of the AGH.

The above bound can be fairly loose for boxes in a tree over an irregularly distributed dataset. A sparsely populated leaf box B at some coarser level $\ell < \ell_{\max}$ may have much denser boxes adjacent to it at the same level. In such cases, a majority of points in adjacent boxes will not be neighbors of points in B ; rather, we expect that the neighbors we seek will be contained in adjacent sub-boxes. We may get a tighter bound on the box-wise range for k NN neighborhoods by considering a two-level condition.

Proposition 5.2. Let $T(B_0)$ be a sparsity-adaptive RMP tree over a dataset \mathcal{X} . If the adjacent boxes of a box B contain a total of at least $k + 1$ points, then all k -nearest neighbors of any point $\mathbf{x} \in B$, with respect to the L_∞ distance, must lie within the adjacency list of the parent box of B (including the parent box itself):

$$\mathcal{N}_k(\mathbf{x}) \subset \mathcal{A}(A) \cap \mathcal{X}, \quad \mathbf{x} \in B, \quad \text{mass}(\mathcal{A}(B)) \geq k + 1, \quad B \in \mathcal{C}(A). \quad (5.2)$$

Proof. If $\text{mass}(\mathcal{A}(B)) \geq k + 1$ for some box B at level ℓ , then for every point $\mathbf{x} \in B \cap \mathcal{X}$ there are at least k other points $\mathbf{x}' \in \mathcal{A}(B) \cap \mathcal{X} \setminus \{\mathbf{x}\}$ for which $d_\infty(\mathbf{x}, \mathbf{x}') < 2^{-\ell+1}$. Denote by A the parent box of B . By the non-adjacency gap of (3.7), and since B is contained within A , the distance from any point $\mathbf{x} \in B$ to any point in a non-adjacent box of A , $\mathbf{x}'' \in \mathcal{A}_\ell^c(A)$, is $d_\infty(\mathbf{x}, \mathbf{x}'') \geq \text{gap}(A) = 2^{-\ell+1}$, which proves (5.2). \square

5.2.2 Algorithm for all- k NN search

Proposition 5.2 suggests a simple two-level adjacency search algorithm for finding the L_∞ -distance k NN of points in any leaf box B of an RMP tree using the AGH. In the tree path to B , find the finest resolution level ℓ_k at which a box B' (or the set of its adjacent boxes) contains no fewer than $k + 1$ points; then search for the neighbors of all points in $B' \cap \mathcal{X}$ among boxes that are adjacent to B' (or its parent box). Applying this strategy while traversing the AGH, as described in Section 3.4.3, produces the all- k NN graph, G_k .

This algorithm leaves a little to be desired: Searching within level-wise box adjacency lists may be wasteful with sparse, irregularly distributed datasets. Consider, for example, a sparsely populated leaf box B at some coarse level $\ell < \ell_{\max}$ with a (non-leaf) densely populated adjacent box B' at the same level. We can expect that most of the points in B' will not be neighbors of points in B . In such cases, a preferable strategy would be to start the search in adjacent sub-boxes at finer resolution levels.

These considerations lead us to the following algorithm. We descend the tree and traverse the AGH to search for neighbors within finest-level adjacent boxes. For each box, once we have found k potential neighbors of points in it, we have an upper bound on the distance of the true neighbors of these points. This upper bound defines a *range* within which we may constrain subsequent searches. We keep visiting adjacent boxes, updating the k NN graph edges, refining the effective search range, and ascending to coarser resolution levels as necessary, until all remaining boxes are outside the current search range. That is, the all- k NN search is cast as a number of adaptive r NN searches. While searching for neighbors in adjacent boxes, point subsets in both the reference box and its adjacent boxes are filtered in the same way as in Section 5.1, to eliminate unnecessary distance evaluations.

The AGH serves as the backbone of the algorithm, directing the search to spatially local subdomains around each box, at multiple resolution scales. We may conceptualize the process as building a coarse-to-fine sequence of neighborhood graphs to find an approximate k NN graph, followed by refinement/correction steps to get the exact one. The coarse-scale point-wise adjacency graphs are not built explicitly, but are implied by the traversal of the AGH.

Lemma 5.2 (All- k NN search). *Let $T(B_0)$ be a sparsity-adaptive RMP tree over a dataset $\mathcal{X} \subset [0, 1]^d$ in a d -dimensional metric space, and $\{G_\ell\}$ be the AGH over $T(B_0)$. Assume that $|\mathcal{A}(B)| \leq c_a$ for every tree box B , and $\text{mass}(\mathcal{A}(B_k)) \leq c_m$, $c_m \geq k + 1$, for every box B_k in some set of boxes which covers \mathcal{X} . Then, the AGH-directed all- k NN search algorithm described above returns G_k after traversing $O(\ell_{\max} c_a N)$ edges and calculating $O(c_a c_m N)$ point-wise distances.*

Proof. Since every box B_k contains at least $k + 1$ points, the k -nearest neighbors of each point in B_k must lie in $\mathcal{A}(B_k)$. The rest of the proof is similar to that of Lemma 5.1. \square

The algorithm described here can be amended to produce a $(1 + \epsilon)$ -approximate k NN graph. As mentioned in Section 5.1, the non-adjacency gap can be used to provide a lower bound on the distance to potential neighbors in boxes that are yet to be visited while traversing the AGH. Any point \mathbf{x} whose k -th neighbor estimate lies at a distance $d_k(\mathbf{x}) \leq (1 + \epsilon) \text{gap}_p(B) \leq (1 + \epsilon) \text{gap}_\infty(B)$, where B is the box containing \mathbf{x} at the current stage of the search, may then be excluded from future k NN search computations. In the interest of brevity, we do not consider approximate k NN searches in multi-dimensional spaces any further.

A pseudocode of the algorithm for all- k NN search is given in Appendix A.

All-NN Search in High-Dimensional Space

We now consider all-near-neighbors (all-NN) search in high-dimensional geometric spaces. We say that a feature space is of high dimensionality when $2^D \gg N$ (cf. Figure 1.2). A consequence of this is that the data distribution in space is highly sparse and irregular.

As mentioned in Section 1.3, near-neighbor search algorithms in such cases suffer from the curse of dimensionality. This is particularly true of exact search algorithms; as a result, solutions invariably focus on finding *approximate* near neighbors. Numerous approaches have been studied to adapt and extend NN search algorithms from the multi-dimensional to the high-dimensional regime, with varying notions of what it means to seek approximate solutions, as discussed in Chapter 2.

In this chapter, we limit our attention specifically to k -nearest neighbors (k NN) search, and do not discuss range- r near neighbors (r NN) search. A peculiar geometric phenomenon in high-dimensional spaces is that the volume of a ball with fixed radius r vanishes as dimensionality increases, regardless of r ; see Figure 6.1. Combined with the highly sparse and irregular distribution of data points, this means that point neighborhoods tend to be either empty or contain the entire dataset, depending on

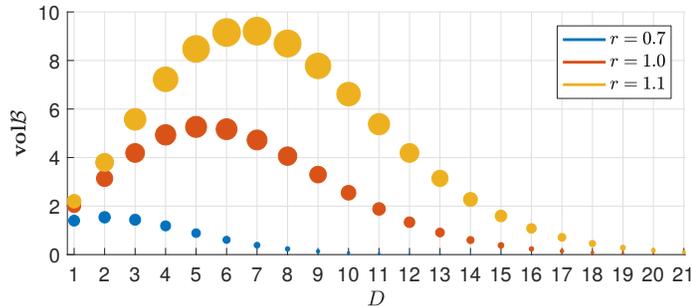


FIGURE 6.1: Volume of a fixed-radius Euclidean D -ball as a function of dimension D . As dimension increases, the volume, shown on the y -axis as well as by disc marker size, becomes infinitesimally small. (Plot taken from the presentation of [FLP⁺18], courtesy of Floros et al.)

r . In other words, the critical range, r_1 , for connectivity is typically large enough to render r NN graphs unsuitable to high-dimensional datasets [BN03].

We discuss several strategies for using the adjacency graph hierarchy (AGH) over a partition tree towards all-NN search in high-dimensional space. In doing so, we will attempt to relate our approach to existing algorithms which have employed these strategies, or similar ones. We note that further research is needed to properly assess the relative merits and shortcomings of the methodologies herein, and we do not claim any particular one as superior. Rather, we intend to outline the versatility and potential of the AGH over partition trees towards efficient all-NN search in high-dimensional spaces.

6.1 Dimension reduction

A common approach to dealing with data in high-dimensional spaces is to apply a transformation such that the dimensionality of the transformed data is substantially lower than that in the original feature space. Collectively, these are known as dimension reduction methods. One may then use a multi-dimensional k NN search algorithm in the reduced space. A key principle in choosing an appropriate transformation is that nearness among the feature vectors is maintained in the reduced space.

6.1.1 Random subspaces

One result which serves as the foundation of many high-dimensional k NN algorithms is the Johnson-Lindenstrauss (JL) lemma [JL82]. Roughly, the lemma states that pairwise Euclidean distances among N points in a D -dimensional space are expected to be distorted by a factor of $O(1 + \epsilon)$ after projection to a random subspace of dimension $O(\log N/\epsilon^2)$. Since its original publication, the JL lemma has been refined and simplified [DG03], and computationally efficient transformations with similar properties have been designed [LHC06; AC10].

These transformations often take the form of random rotations, rather than projections, combined with a partition hierarchy structure which operates on a subspace of the rotated high-dimensional features, such as a k -d tree [LMG⁺05; SH08; JOR11; ML14]. It has been shown [RS19], further, that this approach yields similar probabilistic guarantees to random projection trees and principal component analysis (PCA) trees [CFS09; VKD09; WWZ⁺12; DS15]. (PCA trees are constructed by recursively bisecting the data along the direction which maximizes variance.)

The basic approach of using randomized transformations in k NN search is summarized as follows. The dataset features are projected onto several random, multi-dimensional subspaces. A k NN algorithm is then applied within each subspace, producing a set of candidate neighbors for each query point. Typically, a limit is placed on the number of points or partition boxes that may be checked during the search process. The candidate sets are combined, during or after the subspace-wise searches, and filtered by distance in the original feature space to yield the output neighbors.

We may adopt this strategy in conjunction with the k NN search algorithm of Section 5.2. We obtain m recursive mid-point partitions (RMPs) trees and corresponding AGHs by applying m random rotations to the dataset. Each tree is built

on a subspace of the rotated data. Neighbor candidates are sought based on box adjacency in each subspace. If we assume that the subspace-wise trees are independent from one another, and that the probability that the i -th true neighbor of some point $\mathbf{x} \in \mathcal{X}$ is not within the neighborhood of \mathbf{x} in any one subspace is $p_i(\mathbf{x})$, then the i -th true neighbor is found with probability $1 - p_i(\mathbf{x})^m$.

This simplified analysis parallels that for randomized k -d trees [SH08; RS19]. We note, however, that the simplifying assumption that, for any query point, neighboring points in the k -d tree are visited by order of subspace distance to the query point [SH08] is better-suited to the AGH-directed search of Section 5.2. Box adjacency in the latter bounds the geometric distance between contained points, whereas the distance between points in adjacent boxes of a k -d tree may be arbitrarily large as there are no guarantees on box size.

6.1.2 Principal axes

Random projections and rotations are instances of data-oblivious dimension reduction, as the transformations are determined independently of the dataset at hand. Alternatively, one may use data-adaptive transformations. Possibly the simplest one is that of a truncated PCA.

The cumulative distortion in pairwise Euclidean distances by a truncated PCA can be controlled via the singular values of the data matrix. Let \mathbf{X} be the $D \times N$ matrix whose j -th column contains the features/coordinates of data point $\mathbf{x}_j \in \mathcal{X}$, and let $\mathbf{D} = [d_{ij}^2]$, $d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$, be the $N \times N$ matrix of pairwise squared Euclidean distances. The distance matrix can be expressed as $\mathbf{D} = \mathbf{e}\mathbf{e}^\top(\mathbf{X} \odot \mathbf{X}) + (\mathbf{X} \odot \mathbf{X})^\top \mathbf{e}\mathbf{e}^\top - 2\mathbf{X}^\top \mathbf{X}$, where \odot is the Hadamard (element-wise) product and \mathbf{e} is the constant vector (of appropriate dimension). Denote by σ_i the i -th singular value in the k -truncated singular value decomposition (SVD) $\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$. We may quantify the total distortion of distances via the ratio of the “energy” of $\mathbf{X}^\top \mathbf{X}$ that

is preserved in the k -truncated subspace: $\|\mathbf{X}_k^\top \mathbf{X}_k\|_F / \|\mathbf{X}^\top \mathbf{X}\|_F = \sum_{i=1}^k \sigma_i^2 / \|\mathbf{X}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm. The rank-1 terms of the distance matrix simply contain the diagonal elements of $\mathbf{X}^\top \mathbf{X}$.

In practice, PCA is used on its own [vdMaa14; ZTB⁺17] or as a pre-processing step to decorrelate features and filter out low-variance subspaces before applying other dimension reduction methods [SH08].

6.1.3 Dimension partition

We now consider partitions of the dimensions in a feature space. Given a dataset in a high-dimensional feature space, we may split the feature axes into groups and get a number of lower-dimensional datasets, each containing the data coordinates along axes in a group. In other words, we project the dataset onto a number of independent subspaces, whose Cartesian product spans the original feature space.

We denote the subspace-specific datasets as follows. Assume, for simplicity, that each D -dimensional feature vector is partitioned into m sub-vectors of dimensionality $d = D/m$ each. That is, a dataset $\mathcal{X} \in \mathbb{R}^D$ is split into $\{\mathcal{X}_s\}$, such that $\mathcal{X}_s \subset \mathbb{R}^d$ and $\mathbf{x}_i = [\mathbf{x}_{i,1}^\top \cdots \mathbf{x}_{i,m}^\top]^\top$, for each $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{x}_{i,s} \in \mathcal{X}_s$, $s = 1, \dots, m$.

Dimension partitions allow us to reduce the dimensionality of the space we operate in, which can be critical to controlling the depth and sparsity of the AGH, while retaining properties of the data distribution in the higher-dimensional feature space. We may use dimension partition together with other dimension reduction techniques. For example, with random subspace approaches, dimension partition can be applied within each subspace to yield sub-subspaces, particularly if the random subspaces are relatively high-dimensional (say, as a consequence of the JL lemma).

Computing distances between points in a dataset \mathcal{X}_s amounts to a partial distance computation between the corresponding points in \mathcal{X} , since with an L_p distance metric

d, it holds that

$$d^p(\mathbf{x}_i, \mathbf{x}_j) = \sum_{s=1}^m d^p(\mathbf{x}_{i,s}, \mathbf{x}_{j,s}). \quad (6.1)$$

We may use that to progressively refine distance estimates and bounds while searching for neighbors across subspaces. Moreover, (6.1) makes it possible to extend Proposition 5.2 to adjacency across subspace-wise AGHs. Before we do so, however, it is convenient to consider the extension of box adjacency edges in the AGH onto the data points contained in the boxes.

Definition 6.1 (Point-expanded box adjacency graph). Let $T(B_0)$ be an RMP tree with height ℓ_{\max} over a dataset \mathcal{X} , and $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$, $\ell \in \{1, \dots, \ell_{\max}\}$ be the corresponding level-wise box adjacency graphs. At each level ℓ , the *point-expanded box adjacency* graph, $\tilde{G}_\ell = (\mathcal{X}, \tilde{\mathcal{E}}_\ell)$, connects all points contained within adjacent boxes:

$$\tilde{\mathcal{E}}_\ell \triangleq \{(\mathbf{x}, \mathbf{x}') \mid \mathbf{x} \in B \cap \mathcal{X}, \mathbf{x}' \in B' \cap \mathcal{X}, (B, B') \in \mathcal{E}_\ell\}. \quad (6.2)$$

In (6.2), we take the box adjacency edges, \mathcal{E}_ℓ , to include self-loops, (B, B) ; i.e., $\tilde{\mathcal{E}}_\ell$ includes edges between (distinct) points in the same box. By extension, the level- ℓ adjacency list of a point $\mathbf{x} \in \mathcal{X}$ is defined as the set of points incident to \mathbf{x} in \tilde{G}_ℓ : $\tilde{\mathcal{A}}(\mathbf{x}) \triangleq \{\mathbf{x}' \mid (\mathbf{x}, \mathbf{x}') \in \tilde{\mathcal{E}}_\ell\}$.

We note that such point-expanded adjacency graphs are not to actually be constructed in practice, as they provide no additional information from the box adjacency graphs. We only use them to define point-wise adjacency across subspace products.

Definition 6.2 (Product-adjacency neighborhood). Let $\mathcal{X} \subset [0, 1]^D$ be a dataset in D -dimensional space, and let $\{\mathcal{X}_s\}$ be a subspace partition of the dataset such that $\mathbf{x}_i = [\mathbf{x}_{i,1}^\top \cdots \mathbf{x}_{i,m}^\top]^\top$, for each $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{x}_{i,s} \in \mathcal{X}_s$, $s = 1, \dots, m$. Let $\{G_{\ell,s}\}$ be the

AGH over the partition tree for \mathcal{X}_s . The level- ℓ *product-adjacency neighborhood* of a point $\mathbf{x} \in \mathcal{X}$ is defined as the set of points in \mathbf{x} which are contained in the adjacency list, at level ℓ , of the box which contains \mathbf{x} in every subspace:

$$\tilde{\mathcal{N}}_\ell(\mathbf{x}) \triangleq \bigcap_{s=1}^m \tilde{\mathcal{A}}_{\ell,s}(\mathbf{x}). \quad (6.3)$$

Accordingly, we define the product-adjacency gap at level ℓ as the minimum distance from a point \mathbf{x} to any point outside its product-adjacency neighborhood,

$$\text{gap}_\ell(\mathbf{x}) \triangleq \min\{\|\mathbf{x} - \mathbf{x}''\|_p \mid \mathbf{x}'' \in \tilde{\mathcal{N}}_\ell^c(\mathbf{x})\} \geq 2^{-\ell}. \quad (6.4)$$

This is the equivalent of (3.7) in a single AGH.

Proposition 5.2, together with (6.1), leads to the following corollary, which extends Proposition 5.2 to adjacency across subspace-wise AGHs.

Corollary 6.1. Let $\{\mathcal{X}_s\}_{s=1}^m$ describe a dimension partition of a dataset \mathcal{X} . Denote by $\tilde{\mathcal{N}}_\ell(\mathbf{x})$ the product-adjacency neighborhood at level ℓ of a point $\mathbf{x} \in \mathcal{X}$. If $\tilde{\mathcal{N}}_\ell(\mathbf{x})$ contains at least k points, then the k NN of \mathbf{x} must be within its product-adjacency neighborhood at level $\ell - 1$:

$$|\tilde{\mathcal{N}}_\ell(\mathbf{x})| \geq k \implies \mathcal{N}_k(\mathbf{x}) \subseteq \tilde{\mathcal{N}}_{\ell-1}(\mathbf{x}). \quad (6.5)$$

Corollary 6.1 suggests an algorithm for exact all- k NN search, similar to that of Section 5.2. One would perform a level-wise coordinate traversal of the AGHs across subspaces, keeping track of product-adjacency neighborhoods at each level. The corresponding gap value (6.4) gives a lower bound for the distance to non-neighboring points.

This approach, however, is impractical for two reasons. First, the requirement that neighbor candidates must be in adjacent boxes in *every* subspace may be too strong, effectively resulting in a quadratic-complexity scan over all point-pairs. Second, keeping track of all product-adjacency neighborhoods implies a quadratic-size

data structure, such as an $N \times N$ array of counters which are incremented for each pair of points in adjacent boxes in any subspace [Yun76]. Hashing techniques could mitigate this issue, if not for the almost-quadratic scan challenge mentioned above.

We discuss an algorithm for approximate all- k NN search using AGHs across subspaces in the next section.

6.2 Coordinated all- k NN search across subspaces

Let $\mathcal{X}_s \in [0, 1]^d$ be the data coordinates in the s -th subspace after a dimension partition of a dataset, $\mathcal{X} \subset [0, 1]^D$, $s = 1, \dots, m$. We assume for simplicity that m is an integer divisor of D and the dimensionality of each subspace is $d = D/m$. The dataset \mathcal{X} may be in the original feature space, or after some transformation (see Sections 6.1.1 and 6.1.2). We construct a partition tree, $T_s(B_{0,s})$, and corresponding AGH, $\{G_{\ell,s}\}$, over each \mathcal{X}_s . The search then proceeds as follows.

We traverse the adjacency graphs and compute the local k NNs within each leaf box, and between each pair of adjacent boxes (updating the initial, box-wise k NNs). That is, if $G_{L,s} = (\mathcal{V}_{L,s}, \mathcal{E}_{L,s})$ is the leaf box adjacency graph in the s -th subspace, then for each $B_s \in \mathcal{V}_{L,s}$ and $B' \in \mathcal{A}_{\ell,s}(B_s)$, we compute the pairwise distances $d(\mathbf{x}, \mathbf{x}')$, $\mathbf{x} \in B_s$, $\mathbf{x}' \in B'$, and update the point-wise neighborhoods, $\mathcal{N}_k(\mathbf{x})$, and their subspace-specific adjacency gap, $\text{gap}_{p,s}(B_s)$. Distances are computed using the D -dimensional feature vector of each point. Let $d_k(\mathbf{x})$ be the distance from x to its current k -th nearest neighbor at any point during the subspace-wise search. If $d_k(\mathbf{x}) \geq \sum_s \text{gap}_{p,s}(\mathbf{x})$, then $\mathcal{N}_k(\mathbf{x})$ contains the true neighbors of \mathbf{x} and \mathbf{x} . While there are points for which the above condition does not hold, we ascend the AGHs and repeat the process.

While searching for neighbors in coarser-level adjacent boxes, we adaptively descent into children sub-boxes such that the population mass—equivalently, the number of distance evaluations in each local k NN search—is controlled. This is done

using the approach detailed in Section 5.1.

As mentioned before, the condition $d_k(\mathbf{x}) \leq \text{gap}_{p,s}(\mathbf{x})$ for verifying that we have found the exact k NN of \mathbf{x} may be too strict with high-dimensional datasets; true or approximate neighbors are often found much before they can be verified as such [SH08; ML14]. An effective heuristic in practice is to terminate the search after a preset number of points or boxes have been tested. We employ this heuristic by limiting the number of adjacent boxes that are visited for each leaf box in each subspace to some constant c .

Lemma 6.1. *Let $|\mathcal{X}_s|$ be subspace-wise datasets per a dimension partition of \mathbf{x} , and let $G_{\ell,s} = (\mathcal{V}_{\ell,s}, \mathcal{E}_{\ell,s})$ be the level- ℓ box adjacency graph of the AGH in the s -th subspace, $s = 1, \dots, m$. The total arithmetic cost of distance computations in the all- k NN search algorithm described above is $O(mNDk^2)$, if each visited box B satisfies $\text{mass}(B) = O(k)$ and no more than $c = O(1)$ boxes are searched for each leaf box in $\mathcal{V}_{L,s}$.*

Proof. The cost of a local brute-force k NN search between points in a pair of boxes, whose population mass is proportional to k , is $O(Dk^2)$. Performing a local k NN search for every pair of adjacent boxes at the leaf level of the AGH in each subspace entails a total cost of $\sum_{s=1}^m \sum_{B \in \mathcal{V}_{L,s}} \sum_{B' \in \mathcal{A}(B)} O(Dk^2) = \sum_{s=1}^m \sum_{B \in \mathcal{V}_{L,s}} Dk^2 |\mathcal{A}(B)|$. If only a constant number of boxes are visited for each leaf box in each subspace AGH, and since the number of leaf boxes in each partition tree is $|\mathcal{V}_{\ell,s}(B)| = O(N)$, then the total cost of distance computations becomes $\sum_{s=1}^m O(NDk^2) = O(mNDk^2)$. \square

Remark 6.1. If the space partition in each subspace is approximately balanced in the sense that $\text{mass}(B) = \Theta(k)$ for each leaf box, $B \in \mathcal{V}_{L,s}$, then there are only $O(N/k)$ leaf boxes in each subspace and the cost in Lemma 6.1 becomes $O(mNDk)$. The balanced leaf mass condition can be satisfied by partition structures such as the balanced box decomposition (BBD) tree [AMN⁺98].

Remark 6.2. The cost of distance computations may be further reduced if we calculate distances by the d -dimensional subspace coordinates of each point, rather than its D -dimensional feature vector. If $d = D/m$, then the cost of Lemma 6.1 becomes $O(NDk^2)$ —or, together with Remark 6.1, $O(NDk)$.

This comes at the expense of lower search accuracy, since we may now only bound pointwise distances. Specifically, consider a point \mathbf{x} , and another point \mathbf{x}' which has been encountered only in the first (without loss of generality) m' subspaces, $0 \leq m' \leq m$, during the local k NN search around the boxes containing \mathbf{x} ; the distance between them is bounded from below by $\sum_{s=1}^{m'} d_s(\mathbf{x}, \mathbf{x}') + \sum_{s=m'+1}^m \text{gap}_{p,s}(\mathbf{x}) \leq d(\mathbf{x}, \mathbf{x}')$, where $d_s(\mathbf{x}, \mathbf{x}')$ and is their distance in the s -th subspace, and $\text{gap}_{p,s}(\mathbf{x})$ is the current gap value from \mathbf{x} to points in non-adjacent boxes in the s -th subspace. In this context, we select neighboring points based on their distance bounds instead of exact distances.

Conclusions

Discovering and exploiting multi-scale adjacency relations between points in a multi- or high-dimensional feature space is useful to many problems in data analysis and pairwise interaction computations. In this work, we have focused on leveraging such relations towards efficient all-near-neighbors (all-NN) search, but our methods and analysis are transferable to applications in potential evaluation [GR87; GS98], statistical kernel summations [GM01; MXY⁺16], skeletonization in structured matrix computations [HG12], and neighborhood-preserving embeddings [vdMaa14].

Exploring and exploiting the inherent sparsity in high-dimensional datasets is crucial to designing efficient data structures and algorithms for all-NN search. We have discussed certain key characteristics of the sparse and irregular distribution of high-dimensional, discrete datasets. We have introduced the adjacency graph hierarchy (AGH) as an analysis apparatus and algorithmic infrastructure, expounding the structure of multi-scale neighborhoods among points in the dataset. The AGH is used, explicitly or implicitly, to describe and support a class of multi-scale interaction algorithms, including all-NN search.

Efficient processing of the AGH is enabled by a comprehensive analysis of a

quantized coding scheme (the “qode” system) for the underlying partition structure. Using the adjacency lineage patterns in the AGH and a novel qode filtering approach, we have designed an efficient algorithm for constructing and navigating the AGH. The algorithm is optimal with respect to partition traversal length, and entails only local binary operations. To our knowledge, this is the first AGH algorithm that does not visit non-adjacent partition boxes. With dense and uniformly distributed low-dimensional datasets, this yields a reduction of $O(2^d)$ in traversal length; with irregularly distributed and higher-dimensional datasets, it leads to better adaptation to sparsity in the underlying dataset.

Furthermore, we have shown how to leverage the AGH to obtain an adjacency-sensitive partition structure. We use adjacency relations within and across resolution levels, such that the partitioning of each box takes into consideration its geometric interaction with adjacent boxes. The resulting partition yields a sparser, more parsimonious AGH. Qode filtering operations are again key to enabling efficient communication between boxes during this process. To our knowledge, no other disjoint partition hierarchies that satisfy level-wise coverage of the dataset are adaptive to interaction or adjacency among partition boxes.

The AGH is flexible enough to support a variety feature transformation and partition strategies for all-NN search in high-dimensional feature spaces, but further study is needed to assess their comparative benefits.

Appendix A

Algorithm Pseudocodes

A pseudocode for construction of the adjacency graph hierarchy (AGH) with the minimal-length algorithm described in Section 3.4 is given in Algorithm A.1. In it, we do not make explicit reference to box-wise adjacency lists or similar data structures in order to abstract out implementation-specific details of how the AGH is represented. For simplicity and clarity, we also restrict this pseudocode to finding only same-level box adjacency edges (the change to support cross-level adjacency can be seen in Algorithm A.2).

A pseudocode for all- k -nearest neighbors (k NN) search in a multi-dimensional feature space, using the AGH to direct and restrict the local search domains, is shown in Algorithm A.2.

A pseudocode for constructing an adjacency-sensitive partition of a discrete dataset in \mathbb{R}^d , as described in Section 4.3.2, is shown in Algorithm A.3. The AGH in Algorithm A.3 is augmented with a function over the edges, $f_p : \bigcup_{\ell} \mathcal{E}_{\ell} \rightarrow \{0, 1\}^d$; for every adjacency edge (B, B') , the value of $f_p(B', B)$ is the corresponding parental face descriptor code (see Sections 3.4.4 and 4.1.3).

Algorithm A.1 Pseudocode for recursive construction of the adjacency graph hierarchy (AGH) over a recursive mid-point partition (RMP) tree.

Input: boxes $B, B' \in T(B_0)$; parental face descriptor f_p

Output: box adjacency graph hierarchy edges $\{\mathcal{E}_\ell\}$

Initialization: $B = B' = B_0$; $f_p = 0$; $\ell = 0$; $\mathcal{E}_\ell = \emptyset, \forall \ell$

Invariance: $B \rightarrow \leftarrow B'$

```

1 function  $\{\mathcal{E}_\ell\} = \text{AGH\_CONSTRUCTION\_MINCOST}(B, B', f_p, \ell; \{\mathcal{E}_\ell\})$ 

2    $f \leftarrow c_\ell(B) + c_\ell(B')$   $\triangleright B \cap B'$  face descriptor
3    $\mathcal{C}_f \leftarrow \{C \in \mathcal{C}(B) \mid C \cap (B \cap B') \neq \emptyset\}$   $\triangleright$  see Proposition 3.6
4    $\mathcal{C}'_f \leftarrow \{C' \in \mathcal{C}(B') \mid C' \cap (B \cap B') \neq \emptyset\}$   $\triangleright$   $\gg$ 

5    $\mathcal{E}_{\ell+1} \leftarrow \mathcal{E}_{\ell+1} \cup (\mathcal{C}_f \times \mathcal{C}'_f)$   $\triangleright$  add children edges to  $G_{\ell+1}$ 

6   for all  $(C, C') \in \mathcal{C}_f \times \mathcal{C}'_f$   $\triangleright$  descent into adjacent children pairs
7      $\{\mathcal{E}_\ell\} \leftarrow \text{AGH\_CONSTRUCTION\_MINCOST}(C, C', f, \ell + 1; \{\mathcal{E}_\ell\})$ 

 $\triangleright$  termination is implicit whenever  $\mathcal{C}_f = \emptyset$  or  $\mathcal{C}'_f = \emptyset \triangleleft$ 

```

Algorithm A.2 Pseudocode for all- k NN search using the adjacency graph hierarchy (AGH) on a recursive mid-point partition (RMP) tree over a multi-dimensional dataset.

Input: dataset \mathcal{X} ; #neighbors k ; boxes $B, B' \in T(B_0)$; face descriptor f_p

Output: k NN graph $G_k = (\mathcal{X}, \mathcal{E}_k, \mathcal{D}_k)$

Initialization: $B = B' = B_0$; $f_p = 0$; $\mathcal{E}_k = \mathcal{D}_k = \emptyset$

Invariance: $B \rightarrow \leftarrow B'$

1 **function** $(\mathcal{E}_k, \mathcal{D}_k) = \text{ALLKNNSEARCH_AGH}(\mathcal{X}, k, B, B', f_p, \mathcal{E}_k, \mathcal{D}_k)$

\triangleright AGH traversal (cf. Algorithm A.1) \triangleleft

2 $f \leftarrow c_\ell(B) + c_\ell(B')$

3 **if** $\mathcal{C}(B) \neq \emptyset$

4 $\mathcal{C}_f \leftarrow \{C \in \mathcal{C}(B) \mid C \cap (B \cap B') \neq \emptyset\}$

5 **else**

6 $\mathcal{C}_f \leftarrow B$

7 (similarly for B' and its children)

8 **if** $\mathcal{C}_f = B$ and $\mathcal{C}'_f = B'$ **return**

9 **for all** $(C, C') \in \mathcal{C}_f \times \mathcal{C}'_f$

10 $(\mathcal{E}_k, \mathcal{D}_k) \leftarrow \text{ALLKNNSEARCH_AGH}(\mathcal{X}, k, C, C', f, \mathcal{E}_k, \mathcal{D}_k)$

\triangleright Local k NN search/refinement \triangleleft

11 $(\mathcal{E}_k, \mathcal{D}_k) \leftarrow \text{UPDATEKNNGRAPH}(B \cap \mathcal{X}, B' \cap \mathcal{X}, f, f_p, \mathcal{E}_k, \mathcal{D}_k)$

12 **function** $(\mathcal{E}_k, \mathcal{D}_k) = \text{UPDATEKNNGRAPH}(\mathcal{T}, \mathcal{S}, f, f_p, \mathcal{E}_k, \mathcal{D}_k)$

13 $\mathcal{T}' \leftarrow \left\{ \mathbf{x} \in \mathcal{T} \mid \min_{\mathbf{x}' \in \mathcal{S}} d(\mathbf{x}, \mathbf{x}') < d_k(\mathbf{x}) \right\}$ \triangleright see Section 5.1

14 $\mathcal{S}' \leftarrow \left\{ \mathbf{x}' \in \mathcal{S} \mid \min_{\mathbf{x} \in \mathcal{T}} d(\mathbf{x}, \mathbf{x}') < \max_{\mathbf{x} \in \mathcal{T}} d_k(\mathbf{x}) \right\}$ \triangleright \gg

15 Compute pairwise distances in $\mathcal{T}' \times \mathcal{S}'$ and update $(\mathcal{E}_k, \mathcal{D}_k)$
over points in \mathcal{T}' (e.g., by k -truncated merges)

Algorithm A.3 Pseudocode for construction of an adjacency-sensitive partition graph.

Input: discrete dataset $\mathcal{X} \subset \mathbb{R}^d$; bounding box B_0

Parameters: spatial resolution level threshold L ; mass resolution threshold p

Output: partition and AGH, $\{G_\ell\}_{\ell=1}^L$, $G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell, f_p(\mathcal{E}_\ell))$

Initialization: $G_\ell = (\emptyset, \emptyset, \emptyset)$, $\forall \ell \in \{1, \dots, L\}$

```

1 function  $\{G_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell, f_p(\mathcal{E}_\ell))\} = \text{PARTITION\_AGH}(\mathcal{X}, B_0; L, p)$ 
2    $G_0 \leftarrow (\{B_0\}, \emptyset, \{0\})$ 
3   for  $\ell \in \{0, \dots, L\}$ 
4      $\triangleright$  box partitions and sibling cliques  $\triangleleft$ 
5     for  $B \in \mathcal{V}_\ell$ 
6        $\mathcal{C}(B) \leftarrow \text{PARTITION}(B, \mathcal{A}(B), \mathcal{X}, p)$   $\triangleright \mathcal{A}(B) = \{B' \mid (B, B') \in \mathcal{E}_\ell\}$ 
7        $\mathcal{V}_{\ell+1} \leftarrow \mathcal{V}_{\ell+1} \cup \mathcal{C}(B)$ 
8        $\mathcal{E}_{\ell+1} \leftarrow \mathcal{E}_{\ell+1} \cup (\mathcal{C}(B) \times \mathcal{C}(B))$ 
9        $f_p(\mathcal{C}(B) \times \mathcal{C}(B)) \leftarrow 0$   $\triangleright$  batch assignment
10     $\triangleright$  face-adjacent children bipartite graphs  $\triangleleft$ 
11    for  $(B, B') \in \mathcal{E}_\ell$ 
12       $f \leftarrow (c_\ell(B) + c_\ell(B')) (m_\ell(B) m_\ell(B'))$ 
13       $\mathcal{C}_f \leftarrow \{C \in \mathcal{C}(B) \mid C \cap (B \cap B') \neq \emptyset\}$   $\triangleright$  see Proposition 3.6
14       $\mathcal{C}'_f \leftarrow \{C' \in \mathcal{C}(B') \mid C' \cap (B \cap B') \neq \emptyset\}$   $\triangleright$   $\gg$ 
15       $\mathcal{E}_{\ell+1} \leftarrow \mathcal{E}_{\ell+1} \cup (\mathcal{C}_f \times \mathcal{C}'_f)$ 
16       $f_p(\mathcal{C}_f \times \mathcal{C}'_f) \leftarrow f$   $\triangleright$  batch assignment

```

Appendix B

Collection of Scholarly Publications on Nearest-Neighbor Search¹

We describe a literature collection of scholarly publications relevant to algorithms for near-neighbor (NN) search and/or their applications. We present some characteristic measures of the data. The annual growth in the number of article publications, for example, reflects growing interests and applications of NN search, as well as persistent efforts in advancing NN search algorithms. We show also the significant impact of articles on NN search, by their citation counts, on broader studies and research fields. [AGB⁺18]

B.1 Collection method

We give an account how the literature collection is gathered and the key resource we used. We collect articles relevant to NN search from the Semantic Scholar Open Research Corpus (S2-ORC) [AGB⁺18], as released on November 1st, 2019. The corpus is a literature database and citation graph. The graph nodes are the articles in

¹ This appendix is provided by Dimitris Floros, Nikos Pitsianis and Xiaobai Sun.

the corpus; the graph edges represent the citation relations (outCitations are the references, inCitations are induced by the outCitations). We extract two subgraphs from the big **S2-ORC**. One contains nodes of articles mostly on NN search applications. We may refer to this subgraph as the NN-search application graph and denote it as **APPL**. The other subgraph, denoted by **ALGO**, contains articles mostly on NN search algorithm design and development.

Each of the subgraphs is extracted by the following systematic way. A small set of seed papers is chosen and provided by the user, with keyword search or/and specific knowledge. We make a forward span of the seed papers by locating the papers that cited the seed papers. We make a backward span by tracing the reference list in each of the papers in the union of seed list and forward span.

In this particular case of study and use of our theme-specific literature graph extraction method, the seed papers are seminal papers chosen and specified by Alexandros Iliopoulos, for the purpose of providing facts and evidence about the persisting and growing interests and efforts in NN search. Specifically:

- There are 13 seed papers for the generation of graph **ALGO**. They are [Ben80; CK95; Kle97; AMN⁺98; IM98; SK98; BKL06; WTF09; DML11; HAS⁺11; JDS11; ML14; MY18].
- There are historic 6 seed papers for the generation of graph **APPL**. They are [Ros56; LQ65; Lev66; CH67; FH89; CS18].

The graph data are archived as supplementary material together with the Ph.D. dissertation of Alexandros-Stavros Iliopoulos.

The earliest articles in **S2-ORC** date back to the year of 1936. We found that the earliest article in **APPL** is [dFin37] in 1937, the earliest in **ALGO** is [Kar47] in 1947.

We give in Table B.1 the specific number of articles, the number of citation links and the time span in each of the three graphs.

The intersection of the two sets **ALGO** and **APPL** has only 220 articles, less than

Table B.1: Basic measures of the three citation graphs S2-ORC, ALGO and APPL: number of articles, number of citation links, and time span.

graph	# articles	# citations	avg. cit.	max cit.	time span
S2-ORC	178,320,053	627,875,793	3.5	67,118	1936–2018
ALGO	7432	63,413	8.5	2644	1947–2018
APPL	7660	20,137	2.6	4896	1937–2018

1% in each of the subgraphs.

Two remarks are in order about the link-to-node ratios, i.e., the average citation numbers as listed in the table. First, the link-to-node ratio of S2-ORC is 3.5, which is lower than expected from our common sense about the length of reference list length per article. Among other possible explanations, the main reason is perhaps due to the limited range and the closeness of the graph. The latter means that only the links within the same corpus are recognized, recorded and accounted for in the machine-learning based process [AGB⁺18]. This is a common drawback in existing citation graphs that are closed.

Secondly, the link-to-node ratio in the ALGO subgraph is 8.5, which is much higher than that with S2-ORC, but the ratio in the APPL subgraph is lower, only 2.6. This gap may disappear in future collection and subgraph extraction. We expect the collection of S2-ORC to expand continuously to include more articles from diverse study fields, the initial core of S2-ORC was of computer science articles. The gap also depends on the art of the subgraph extraction. Extracting application articles via keyword search and/or graph spans from seed papers is more challenging than extracting algorithm design articles. In particular, the extraction of APPL is more sensitive to the selection of keywords and seed papers, given the great diversity in application fields and domain-specific languages. The graph extraction of ALGO is relatively stable, the articles are mostly from the same and closely connected community of

computer science.

B.2 Measures of growth and influence

We provide a few additional characteristic measures of the two literature subgraphs ALGO and APPL of scholarly articles relevant to NN search.

The first two measures are about dynamic growth in publication. We present in Figure B.1 the annual growth and growth rate. The growth in NN-search publications in ALGO and APPL remarkably outpace the rest in S2-ORC. The growth rate is shown from 1975 when NN search started to get recognized by and large as a basic research problem. The publication growth rate R_t at year t is defined as follows,

$$R_t = \frac{n_t - n_{t-1}}{n_{t-1}},$$

where n_t is the number of articles published in year t with respect to a particular literature graph.

Next we measure and rank, by intra-citation and inter-citation counts, the intra stimulation and influence of the articles within each subgraph, mutual influence between the two subgraphs, as well as the external impact of each subgraph on the rest of the base graph S2-ORC.

In Figure B.2 we show the mutual influence of ALGO and APPL articles. In Figure B.3 we show the significant impact of the articles in each subgraph onto the rest of the articles in the base graph S2-ORC. We provide below the list of top 10 articles by each of the rank-size distributions shown in Figure B.3.

- Within ALGO, the top 10 articles are [AMN⁺98; IM98; GIM99; Cha02; DII⁺04; Low04; AI06; WTF09; JDS11; ML14]. Five of them are in the set of seed papers for the subgraph span. The number of citation links to these articles is 18% of the total number of citations within the graph.

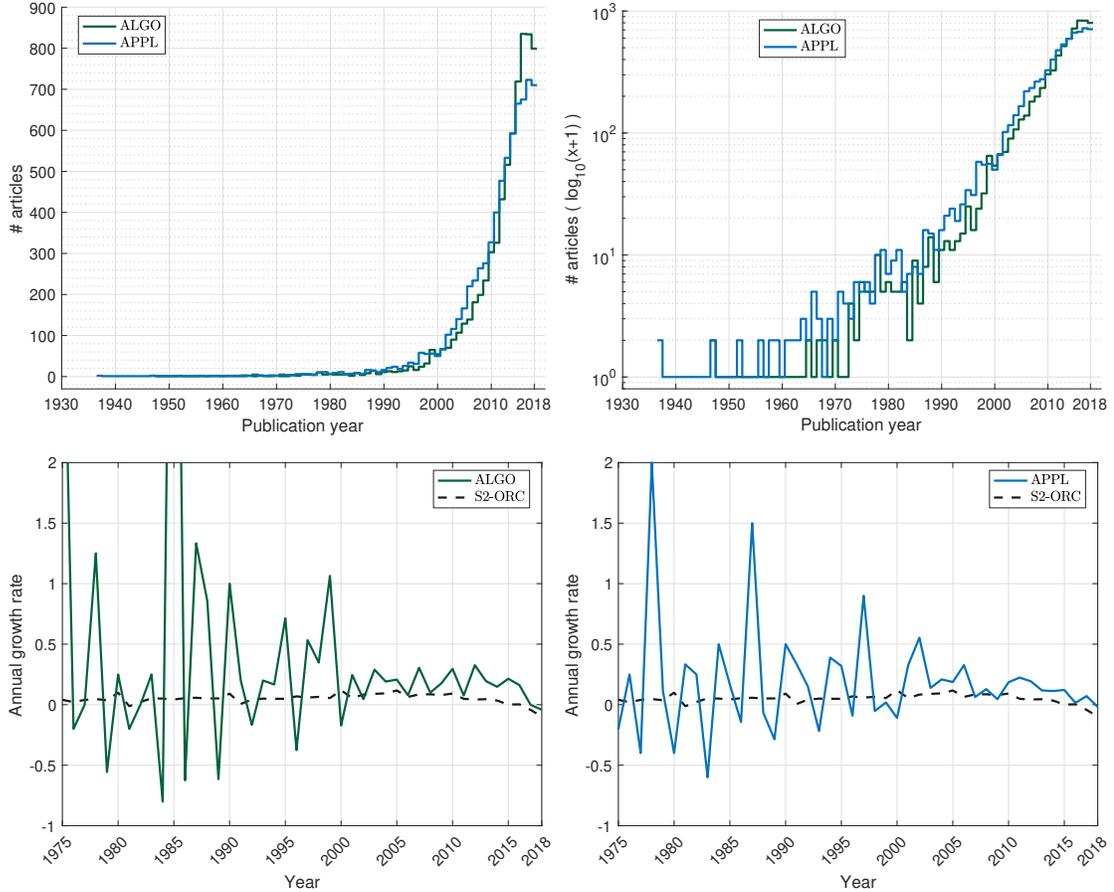


FIGURE B.1: Annual publication and growth rate of NN-search articles. Top row: annual publication from 1937 to 2018, in the numbers of articles on the left plot, which is shown in log scale on the right; In each plot, the ALGO curve is in green, APPL in blue. The annual publication associated with each subgraph grows steadily and accelerated in recent years. Bottom row: growth rate in publication from 1975 to 2018 for each subgraph in contrast to that by the S2-ORC graph. The left plot shows the contrast between ALGO in solid line and S2-ORC in dashed line; the right plot, the contrast between APPL and S2-ORC. The publication growth rate for each subgraph, despite the oscillation, is significantly higher than the growth rate for the base graph.

- Within APPL, the top 10 articles are [Ros56; Par62; LQ65; CH67; BFS+84; FH89; AKA91; FS95; Vap00; Bre01]. Three of them are in the set of seed papers for the subgraph span. The number of citation links to these articles is 54% of the total in the subgraph.

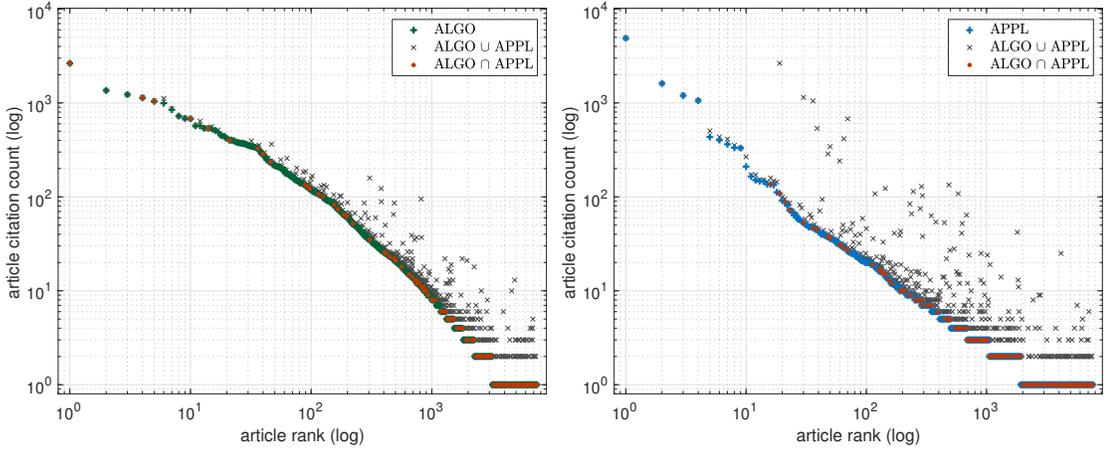


FIGURE B.2: Mutual influence of ALGO and APPL articles. **Left:** the ranking by inCitation count internal to ALGO articles, in green + markers. The additional citation counts in x markers (see the legend) are placed above the ranking curve as displacements. The red markers are not counts, they indicate the 220 articles in the intersection of the two subgraphs. **Right:** the ranking by inCitation count internal to APPL articles, in blue + markers. The top 10 article lists by the respective rankings can be found in the document text.

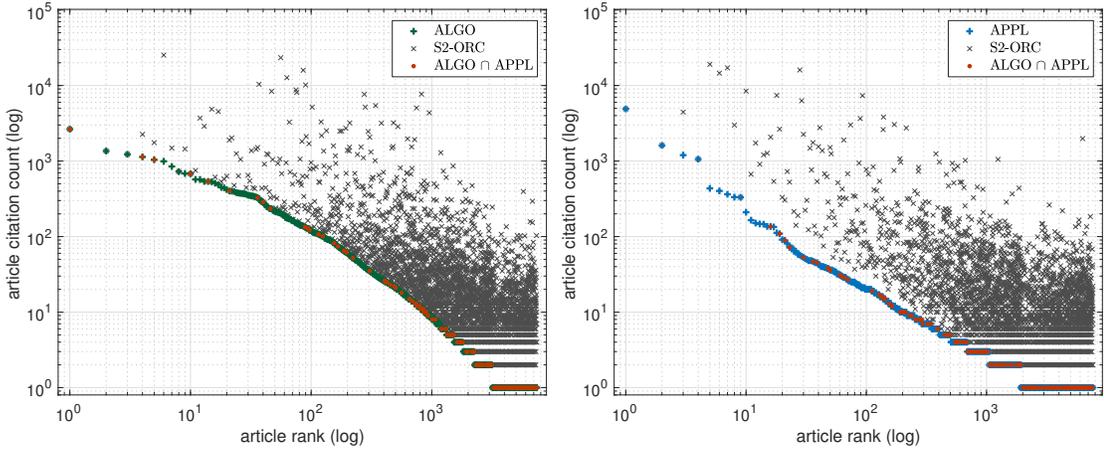


FIGURE B.3: Influence/impact measure and ranking by intra-citation counts and inter-citation counts. On the left, the ranking is by inCitation counts within graph ALGO in green + markers, and inCitation counts across the entire graph S2-ORC in black x markers. The red markers indicate the 220 articles in the intersection of ALGO and APPL. On the right, the ranking is by inCitation counts within graph APPL. The total number of external citations to each subgraph is about $7\times$ greater than the intra-citations. This phenomenon indicates the strong impact or influence of NN search on broad studies.

Bibliography

- [AKA91] D. W. Aha, D. Kibler, and M. K. Albert. “Instance-based learning algorithms”. *Machine Learning*, 6(1):37–66, 1991.
- [AC09] N. Ailon and B. Chazelle. “The fast Johnson–Lindenstrauss transform and approximate nearest neighbors”. *SIAM Journal on Computing*, 39(1):302–322, 2009.
- [AC10] N. Ailon and B. Chazelle. “Faster dimension reduction”. *Communications of the ACM*, 53(2):97–104, 2010.
- [AS99] S. Aluru and F. E. Sevilgen. “Dynamic compressed hyperoctrees with application to the N-body problem”. *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 21–33, Madras, India, 1999.
- [AGB⁺18] W. Ammar, D. Groeneveld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. Ha, R. Kinney, S. Kohlmeier, K. Lo, T. Murray, H.-H. Ooi, M. Peters, J. Power, S. Skjonsberg, L. Wang, C. Wilhelm, Z. Yuan, M. van Zuylen, and O. Etzioni. “Construction of the literature graph in Semantic Scholar”. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 3 (Industry Papers), pages 84–91, New Orleans, LA, USA, 2018.
- [AI06] A. Andoni and P. Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, Berkeley, CA, USA, 2006.
- [AIR18] A. Andoni, P. Indyk, and I. Razenshteyn. *Approximate nearest neighbor search in high dimensions*. 2018. arXiv: 1806.09823 [cs.DS].

- [AR15] A. Andoni and I. Razenshteyn. *Optimal data-dependent hashing for approximate near neighbors*. 2015. arXiv: 1501.01062 [cs.DS].
- [AMN⁺98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions”. *Journal of the ACM*, 45(6):891–923, 1998.
- [ABF17] M. Aumüller, E. Bernhardsson, and A. Faithfull. “ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms”. *Similarity Search and Applications*, pages 34–49, Munich, Germany, 2017.
- [BH86] J. Barnes and P. Hut. “A hierarchical $O(N \log N)$ force-calculation algorithm”. *Nature*, 324(6096):446–449, 1986.
- [BN03] M. Belkin and P. Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. *Neural Computation*, 15(6):1373–1396, 2003.
- [Bel61] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, USA, 1961.
- [Ben80] J. L. Bentley. “Multidimensional divide-and-conquer”. *Communications of the ACM*, 23(4):214–229, 1980.
- [Ben76] J. L. Bentley. *Divide and Conquer Algorithms for Closest Point Problems in Multidimensional Space*. Ph.D. dissertation, Department of Computer Science, University of North Carolina, Chapel Hill, NC, USA, 1976. UNC Report TR-76-103.
- [BKL05] A. Beygelzimer, S. Kakade, and J. Langford. *Cover trees for nearest neighbor*. Extended version of ICML 2006 paper of the same title by the same authors., 2005.
- [BKL06] A. Beygelzimer, S. Kakade, and J. Langford. “Cover trees for nearest neighbor”. *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, Pittsburgh, PA, USA, 2006.
- [Bre01] L. Breiman. “Random Forests”. *Machine Learning*, 45(1):5–32, 2001.
- [BFS⁺84] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, Boca Raton, FL, USA, 1st edition, 1984.

- [BCM05] A. Buades, B. Coll, and J.-M. Morel. “A non-local algorithm for image denoising”. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 60–65, San Diego, CA, USA, 2005.
- [Bul03] P. S. Bullen. *Handbook of Means and Their Inequalities*, volume 560 of *Mathematics and Its Applications*. Springer Netherlands, 2nd edition, 2003.
- [Cac66] T. Cacoullos. “Estimation of a multivariate density”. *Annals of the Institute of Statistical Mathematics*, 18(1):179–189, 1966.
- [CK95] P. B. Callahan and S. R. Kosaraju. “A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields”. *Journal of the ACM*, 42(1):67–90, 1995.
- [CCF⁺13] R. Castillo, E. Castillo, D. Fuentes, M. Ahmad, A. M. Wood, M. S. Ludwig, and T. Guerrero. “A reference dataset for deformable image registration spatial accuracy evaluation using the COPDgene study archive”. *Physics in Medicine and Biology*, 58(9):2861–2877, 2013.
- [Cha02] M. S. Charikar. “Similarity estimation techniques from rounding algorithms”. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, Montréal, QC, Canada, 2002.
- [CS18] G. H. Chen and D. Shah. “Explaining the success of nearest neighbor methods in prediction”. *Foundations and Trends® in Machine Learning*, 10(5-6):337–588, 2018.
- [CFS09] J. Chen, H.-r. Fang, and Y. Saad. “Fast approximate k NN graph construction for high dimensional data via recursive Lanczos bisection”. *Journal of Machine Learning Research*, 10(Sep):1989–2012, 2009.
- [Che09] L. Chen. “Curse of dimensionality”. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 545–546. Springer, Boston, MA, USA, 2009.
- [Che14] W. C. Chew. “Vector potential electromagnetics with generalized gauge for inhomogeneous media: formulation”. *Progress In Electromagnetics Research*, 149:69–84, 2014.

- [Cla83] K. L. Clarkson. “Fast algorithms for the all nearest neighbors problem”. *24th Annual Symposium on Foundations of Computer Science*, pages 226–232, Tucson, AZ, USA, 1983.
- [CK10] M. Connor and P. Kumar. “Fast construction of k -nearest neighbor graphs for point clouds”. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010.
- [CH67] T. M. Cover and P. E. Hart. “Nearest neighbor pattern classification”. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- [Cur15] R. R. Curtin. “Faster dual-tree traversal for nearest neighbor search”. *Similarity Search and Applications*, pages 77–89, Glasgow, Scotland, UK, 2015.
- [CLM⁺15] R. R. Curtin, D. Lee, W. B. March, and P. Ram. “Plug-and-play dual-tree algorithm runtime analysis”. *Journal of Machine Learning Research*, 16(101):3269–3297, 2015.
- [CMR⁺13] R. R. Curtin, W. B. March, P. Ram, D. V. Anderson, A. G. Gray, and C. L. Isbell Jr. “Tree-independent dual-tree algorithms”. *Proceedings of the 30th International Conference on Machine Learning*, volume 28(3), pages 1435–1443, Atlanta, GA, USA, 2013.
- [DG03] S. Dasgupta and A. Gupta. “An elementary proof of a theorem of Johnson and Lindenstrauss”. *Random Structures & Algorithms*, 22(1):60–65, 2003.
- [DS15] S. Dasgupta and K. Sinha. “Randomized partition trees for nearest neighbor search”. *Algorithmica*, 72(1):237–263, 2015.
- [DII⁺04] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. “Locality-sensitive hashing scheme based on p -stable distributions”. *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 253–262, Brooklyn, NY, USA, 2004.
- [dFin37] B. de Finetti. “La prévision: ses lois logiques, ses sources subjectives”. *Annales de l’Institut Henri Poincaré*, 17(1):1–68, 1937.
- [DML11] W. Dong, C. Moses, and K. Li. “Efficient k -nearest neighbor graph construction for generic similarity measures”. *Proceedings of the 20th International Conference on World Wide Web*, page 577, Hyderabad, India, 2011.

- [DI18] B. H. Drost and S. Ilic. “Almost constant-time 3D nearest-neighbor lookup using implicit octrees”. *Machine Vision and Applications*, 29(2):299–311, 2018.
- [FH51] E. Fix and J. L. Hodges Jr. *Discriminatory Analysis – Nonparametric Discrimination: Consistency Properties*. Report No. 4, Project No. 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX, USA, 1951.
- [FH89] E. Fix and J. L. Hodges Jr. “Discriminatory analysis – Nonparametric discrimination: consistency properties”. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989. Reprint.
- [FLP⁺18] D. Floros, T. Liu, N. Pitsianis, and X. Sun. “Sparse dual of the density peaks algorithm for cluster analysis of high-dimensional data”. *2018 IEEE High Performance Extreme Computing Conference*, page 14, Waltham, MA, USA, 2018.
- [FS95] Y. Freund and R. E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. *European Conference on Computational Learning Theory*, pages 23–37, Bertinoro, Italy, 1995.
- [FC16] C. Fu and D. Cai. *EFANNA: An extremely fast approximate nearest neighbor search algorithm based on kNN graph*. 2016. arXiv: 1609 . 07228 [cs.CV].
- [FWC18] C. Fu, C. Wang, and D. Cai. *Fast approximate nearest neighbor search with the navigating spreading-out graph*. 2018. arXiv: 1707.00143 [cs.LG].
- [Gar82] I. Gargantini. “Linear octrees for fast processing of three-dimensional objects”. *Computer Graphics and Image Processing*, 20(4):365–374, 1982.
- [GIM99] A. Gionis, P. Indyk, and R. Motwani. “Similarity search in high dimensions via hashing”. *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, Edinburgh, Scotland, UK, 1999.
- [GM01] A. G. Gray and A. W. Moore. “‘N-body’ problems in statistical learning”. *Advances in Neural Information Processing Systems*, volume 13, pages 521–527, Denver, CO, USA, 2001.
- [GM03] A. G. Gray and A. W. Moore. “Nonparametric density estimation: toward computational tractability”. *Proceedings of the 2003 SIAM In-*

- ternational Conference on Data Mining*, pages 203–211, San Francisco, CA, USA, 2003.
- [Gre87] L. F. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. Ph.D. dissertation, Yale University, New Haven, CT, USA, 1987.
- [GR87] L. Greengard and V. Rokhlin. “A fast algorithm for particle simulations”. *Journal of Computational Physics*, 73(2):325–348, 1987.
- [GS91] L. Greengard and J. Strain. “The fast Gauss transform”. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.
- [GS98] L. Greengard and X. Sun. “A new version of the fast Gauss transform”. *Documenta Mathematica*, Extra Volume ICM(III):575–584, 1998.
- [HAS⁺11] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. “Fast approximate nearest-neighbor search with k -nearest neighbor graph”. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, volume 2, pages 1312–1317, Barcelona, Spain, 2011.
- [HD16] B. Harwood and T. Drummond. “FANNG: Fast approximate nearest neighbour graphs”. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5713–5722, Las Vegas, NV, USA, 2016.
- [HG12] K. L. Ho and L. Greengard. “A fast direct solver for structured linear systems by recursive skeletonization”. *SIAM Journal on Scientific Computing*, 34(5):A2507–A2532, 2012.
- [HP49] P. G. Hoel and R. P. Peterson. “A solution to the problem of optimum classification”. *The Annals of Mathematical Statistics*, 20(3):433–438, 1949.
- [Ind00] P. Indyk. *High-Dimensional Computational Geometry*. Ph.D. dissertation, Stanford University, Palo Alto, CA, USA, 2000.
- [IM98] P. Indyk and R. Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality”. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, Dallas, TX, USA, 1998.

- [JDS11] H. Jégou, M. Douze, and C. Schmid. “Product quantization for nearest neighbor search”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.
- [JZH⁺14] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He. “Fast and accurate hashing via iterative nearest neighbors expansion”. *IEEE Transactions on Cybernetics*, 44(11):2167–2177, 2014.
- [JL82] W. B. Johnson and J. Lindenstrauss. “Extensions of Lipschitz mappings into a Hilbert space”. In, *Conference in Modern Analysis and Probability*. Volume 26, Contemporary Mathematics, pages 189–206. New Haven, CT, USA, 1982.
- [JOR11] P. W. Jones, A. Osipov, and V. Rokhlin. “Randomized approximate nearest neighbors algorithm”. *Proceedings of the National Academy of Sciences*, 108(38):15679–15686, 2011.
- [KR02] D. R. Karger and M. Ruhl. “Finding nearest neighbors in growth-restricted metrics”. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 741–750, Montreal, QC, Canada, 2002.
- [Kar47] K. Karhunen. “Über lineare Methoden in der Wahrscheinlichkeitsrechnung”. *Annales Academiae Scientiarum Fennicae*, Ser. A 1(37):1–79, 1947.
- [Kle97] J. M. Kleinberg. “Two algorithms for nearest-neighbor search in high dimensions”. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 599–608, El Paso, TX, USA, 1997.
- [Knu98] D. E. Knuth. *The Art of Computer Programming: Volume 3 (Sorting and Searching)*. Addison-Wesley Professional, Reading, MA, USA, 2nd edition, 1998.
- [KG09] B. Kulis and K. Grauman. “Kernelized locality-sensitive hashing for scalable image search”. *IEEE 12th International Conference on Computer Vision*, pages 2130–2137, Kyoto, Japan, 2009.
- [Lev66] C. Levinthal. “Molecular model-building by computer”. *Scientific American*, 214(6):42–52, 1966.
- [LHC06] P. Li, T. J. Hastie, and K. W. Church. “Very sparse random projections”. *Proceedings of the 12th ACM SIGKDD International Conference*

on *Knowledge Discovery and Data Mining*, pages 287–296, Philadelphia, PA, USA, 2006.

- [LMG04] T. Liu, A. W. Moore, and A. Gray. “Efficient exact k-NN and non-parametric classification in high dimensions”. *Advances in Neural Information Processing Systems*, volume 16, pages 265–272, Vancouver, BC, Canada, 2004.
- [LMG⁺05] T. Liu, A. W. Moore, A. Gray, and K. Yang. “An investigation of practical approximate nearest neighbor algorithms”. *Advances in Neural Information Processing Systems*, volume 17, pages 825–832, Vancouver, BC, Canada, 2005.
- [LQ65] D. O. Loftsgaarden and C. P. Quesenberry. “A nonparametric estimate of a multivariate density function”. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965.
- [Low04] D. G. Lowe. “Distinctive image features from scale-invariant keypoints”. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [MY18] Y. A. Malkov and D. A. Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. (Early access.)
- [MPL⁺14] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. “Approximate nearest neighbor algorithm based on navigable small world graphs”. *Information Systems*, 45:61–68, 2014.
- [MCM⁺10] J. V. Manjón, P. Coupé, L. Martí-Bonmatí, D. L. Collins, and M. Robles. “Adaptive non-local means denoising of MR images with spatially varying noise levels”. *Journal of Magnetic Resonance Imaging*, 31(1):192–203, 2010.
- [MXY⁺16] W. B. March, B. Xiao, C. D. Yu, and G. Biros. “ASKIT: An efficient, parallel library for high-dimensional kernel summations”. *SIAM Journal on Scientific Computing*, 38(5):S720–S749, 2016.
- [ML14] M. Muja and D. G. Lowe. “Scalable nearest neighbor algorithms for high dimensional data”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

- [Par62] E. Parzen. “On estimation of a probability density function and mode”. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.
- [Pel14] M. Pelillo. “Alhazen and the nearest neighbor rule”. *Pattern Recognition Letters*, 38:34–37, 2014.
- [PHS⁺16] Z. Peng, R. Hiptmair, Y. Shao, and B. MacKie-Mason. “Domain decomposition preconditioning for surface integral equations in solving challenging electromagnetic scattering problems”. *IEEE Transactions on Antennas and Propagation*, 64(1):210–223, 2016.
- [PIF⁺19] N. Pitsianis, A.-S. Iliopoulos, D. Floros, and X. Sun. “Spaceland embedding of sparse stochastic graphs”. *IEEE High Performance Extreme Computing Conference*, Waltham, MA, USA, 2019.
- [RLM⁺09] P. Ram, D. Lee, W. March, and A. G. Gray. “Linear-time algorithms for pairwise statistical problems”. *Advances in Neural Information Processing Systems*, volume 22, pages 1527–1535, Vancouver, BC, Canada, 2009.
- [RS19] P. Ram and K. Sinha. “Revisiting *kd*-tree for nearest neighbor search”. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1388, Anchorage, AK, USA, 2019.
- [Ros56] M. Rosenblatt. “Remarks on some nonparametric estimates of a density function”. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.
- [Sam82] H. Samet. “Neighbor finding techniques for images represented by quad-trees”. *Computer Graphics and Image Processing*, 18(1):37–57, 1982.
- [Sam89] H. Samet. “Neighbor finding in images represented by octrees”. *Computer Vision, Graphics, and Image Processing*, 46(3):367–386, 1989.
- [Sam06] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco, CA, USA, 2006.
- [SSV07] J. Sankaranarayanan, H. Samet, and A. Varshney. “A fast all nearest neighbor algorithm for applications involving large point-clouds”. *Computers & Graphics*, 31(2):157–174, 2007.

- [Sch92] G. Schrack. “Finding neighbors of equal size in linear quadtrees and octrees in constant time”. *CVGIP: Image Understanding*, 55(3):221–230, 1992.
- [SK98] T. Seidl and H.-P. Kriegel. “Optimal multi-step k -nearest neighbor search”. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 154–165, Seattle, WA, USA, 1998.
- [SH08] C. Silpa-Anan and R. Hartley. “Optimised KD-trees for fast image descriptor matching”. *IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA, 2008.
- [SP01] X. Sun and N. P. Pitsianis. “A matrix version of the fast multipole method”. *SIAM Review*, 43(2):289–300, 2001.
- [Vai89] P. M. Vaidya. “An $O(n \log n)$ algorithm for the all-nearest-neighbors problem”. *Discrete & Computational Geometry*, 4(2):101–115, 1989.
- [vdMaa14] L. van der Maaten. “Accelerating t-SNE using tree-based algorithms”. *Journal of Machine Learning Research*, 15(Oct):3221–3245, 2014.
- [Vap00] V. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer-Verlag, New York, NY, USA, 2nd edition, 2000.
- [VKD09] N. Verma, S. Kpotufe, and S. Dasgupta. “Which spatial partition trees are adaptive to intrinsic dimension?” *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 565–574, Montréal, QC, Canada, 2009.
- [Vör00] J. Vörös. “A strategy for repetitive neighbor finding in octree representations”. *Image and Vision Computing*, 18(14):1085–1091, 2000.
- [WWZ⁺12] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li. “Scalable k -NN graph construction for visual descriptors”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, Providence, RI, USA, 2012.
- [WSS⁺14] J. Wang, H. T. Shen, J. Song, and J. Ji. *Hashing for similarity search: a survey*. 2014. arXiv: 1408.2927 [cs.DS].

- [WTF09] Y. Weiss, A. Torralba, and R. Fergus. “Spectral hashing”. *Advances in Neural Information Processing Systems*, volume 21, pages 1753–1760, 2009.
- [XCG⁺10] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. “Fast algorithms for hierarchically semiseparable matrices”. *Numerical Linear Algebra with Applications*, 17(6):953–967, 2010.
- [YS83] M.-M. Yau and S. N. Srihari. “A hierarchical data structure for multi-dimensional digital images”. *Communications of the ACM*, 26(7):504–515, 1983.
- [Yok13] R. Yokota. “An FMM based on dual tree traversal for many-core architectures”. *Journal of Algorithms & Computational Technology*, 7(3):301–324, 2013.
- [Yun76] T. P. Yunck. “A technique to identify nearest neighbors”. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(10):678–683, 1976.
- [ZHP⁺11] B. Zhang, J. Huang, N. Pitsianis, and X. Sun. “Dynamic prioritization for parallel traversal of irregularly structured spatio-temporal graphs”. *3rd USENIX Workshop on Hot Topics in Parallelism*, Berkeley, CA, USA, 2011.
- [ZHG⁺13] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu. “Fast k NN graph construction with locality sensitive hashing”. *Machine Learning and Knowledge Discovery in Databases*, pages 660–674, Prague, Czech Republic, 2013.
- [ZTB⁺17] G. X. Y. Zheng, J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, T. D. Wheeler, G. P. McDermott, J. Zhu, M. T. Gregory, J. Shuga, L. Montesclaros, J. G. Underwood, D. A. Masquelier, S. Y. Nishimura, M. Schnall-Levin, P. W. Wyatt, C. M. Hindson, R. Bharadwaj, A. Wong, K. D. Ness, L. W. Beppu, H. J. Deeg, C. McFarland, K. R. Loeb, W. J. Valente, N. G. Ericson, E. A. Stevens, J. P. Radich, T. S. Mikkelsen, B. J. Hindson, and J. H. Bielas. “Massively parallel digital transcriptional profiling of single cells”. *Nature Communications*, 8(1):14049, 2017.