ABSTRACT

Title of Dissertation: EFFICIENT GEOMETRIC ALGORITHMS FOR STOCHASTIC AND CATEGORICAL DATA

Aditya Acharya
Doctor of Philosophy, 2026

Dissertation Directed by: Professor David M. Mount
Department of Computer Science

Modern algorithmic frameworks must contend with data that is both massive and inherently uncertain. This dissertation explores efficient geometric algorithms for handling such data across two regimes: temporal evolution and spatial classification.

In the first part, we address *temporal uncertainty* by developing algorithms to track evolving data. We introduce a framework for tracking labels on trees and moving points in the Euclidean plane, establishing the necessary speedup factors required to maintain an accurate hypothesis of the ground truth. We extend this to track evolving probability distributions under a local-motion model, using the Kullback-Leibler (KL) divergence to bound the information loss over time.

In the second part, we address *spatial uncertainty* by developing classifiers for data constrained to bounded domains, such as the probability simplex. We investigate Support Vector Machines (SVMs) under a non-Euclidean geometry called the Hilbert metric. This is a projective geometry that like the KL divergence is sensitive to domain boundaries. We prove that the Hilbert SVM problem is of LP-type, allowing for exact solutions, and we present a polynomial-time approximation algorithm that overcomes the curse of dimensionality inherent in exact approaches. Together, these results provide a robust algorithmic foundation for managing uncertainty in dynamic and bounded geometric systems.

EFFICIENT GEOMETRIC ALGORITHMS FOR STOCHASTIC AND CATEGORICAL DATA

by

Aditya Acharya

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:
Professor David M. Mount, Chair/Advisor
Associate Professor Michael W. Otte
Professor John Yiannis Aloimonos
Professor Ramani Duraiswami
Professor William Gasarch

Table of Contents

List of Figures

List of Algorithms

Chapter 1: Introduction

1.1 Motivation: Beyond Static and Euclidean

Modern computational applications are increasingly defined by two core challenges: *massive scale* and *stochastic variation*. Traditional algorithmic models, which typically assume a static single-input/single-output paradigm within a standard Euclidean space, are frequently inadequate for modern systems. In these systems, the underlying structure changes continuously and stochastically, invisible to the algorithm, or the data resides in complex, non-Euclidean domains where standard notions of distance fail to capture semantic similarity.

This dissertation explores efficient geometric algorithms designed to address these challenges primarily over the intersection of the following data types:

- *Stochastic*: Data which is evolving, or is uncertain, either temporally or spatially.
- *Categorical*: Data that requires maintaining and assigning labels, via tracking or classification.

The first motivation for this work is the problem of *Tracking Evolving Data*. In many real-world scenarios, such as tracking a swarm of unmanned aerial vehicles (UAVs) via GPS, monitoring traffic congestion in real-time, or tracking the spread of disease hot-spots, an algorithm cannot assume the data is fixed. The data does not wait for the algorithm to process it; instead, a stochastic "evolver" continuously modifies the ground truth behind the scenes. The algorithm must judiciously probe the

system to maintain a "hypothesis" that remains close to this moving target. The fundamental challenge is to determine whether an algorithm can optimally keep pace with this shifting reality, and whether it can achieve this while operating at a constant speedup relative to the evolver.

The second motivation stems from *Geometric Classification* in high-dimensional machine learning. Standard Euclidean metrics often fail to capture the intrinsic sensitivity of stochastic data, particularly when the data is constrained within a bounded domain like a probability simplex (e.g., histograms or discrete probability distributions). In such spaces, the "distance" between points should reflect their proximity to the boundary, as a small change in a probability near zero is often more significant than a similar change near 0.5. This necessitates the use of non-Euclidean geometries, specifically the *Hilbert metric*, which generalizes hyperbolic geometry to arbitrary convex bodies. Using the Hilbert metric provides a robust framework for categorical classification tasks where the domain boundaries impose strict constraints on the data.

1.2 Thematic Bridge: The Geometry of Uncertainty

While Part I (Chapters 2, 3, and 4) and Part II (Chapters 5 and 6) address different algorithmic problems (tracking categorical labels versus classification) they are strongly linked by a common thread: *the geometric representation of stochastic uncertainty*.

In **Part I**, we deal with *Temporal Uncertainty*. We develop algorithms to track categorical labels and objects whose locations are constantly perturbed by a stochastic evolver. By Chapter 4, we explicitly model these moving objects not as precise coordinates, but as *spatial stochastic probability distributions*. To measure the error between our hypothesis and the ground truth, we employ the *Kullback-Leibler (KL) divergence* (relative entropy). This metric quantifies the "information loss"

incurred when approximating the true stochastic distribution with our hypothesis.

In **Part II**, we deal with *Spatial Uncertainty*. We transition from tracking evolving stochastic distributions to the categorical classification of points that reside within fixed, bounded domains (such as the probability simplex). Here, the *Hilbert metric* becomes the natural geometric successor to the KL divergence.

The link is mathematically profound and is characterized by a curious inversion of domains:

- *Cross-Pollination of Metrics:* There is a beautiful symmetry in how we apply our distance measures. In Part I, we utilize the KL divergence, which is traditionally an information-theoretic measure used to compare probability distributions, to maintain proximity to a geometric evolver in physical space. Conversely, in Part II, we employ the Hilbert metric, a purely geometric distance defined by chords and convex boundaries, to solve classification tasks within a probability space (the simplex).

- *Boundary Sensitivity:* Both the KL divergence used in Chapter 4 and the Hilbert metric used in Chapters 5 and 6 diverge as points approach the boundary of the domain. This sensitivity is crucial for handling uncertain data, where edge cases like probabilities near 0 or 1 carry high information content.

- *Information Geometry:* On the probability simplex, the Hilbert metric behaves similarly to information-theoretic distances. It has been shown to be information monotone and performs competitively with KL divergence in clustering tasks.

Thus, this thesis tells a cohesive story: We begin by *tracking* dynamic stochastic uncertainty in Part I, culminating in a distributional model. We then freeze time in Part II to develop rigorous

categorical classification tools for those same probabilistic distributional spaces using the Hilbert geometry. This provides a complete toolkit for dynamic data: when data moves significantly, we *track* its categorical states (Part I); when data jitters or is uncertain within a local region, we rely on *geometric robustness* (Part II) to maintain valid categorical classification. We next present the organization of the dissertation in two parts.

1.3 Algorithms for Evolving Stochastic Data (Part I)

This part focuses on maintaining geometric structures (trees, point sets, and distributions) under continuous, unseen stochastic changes.

Chapter 2: Optimally Tracking Categorical Labels on an Evolving Tree

The Problem: We consider the fundamental problem of tracking evolving data on a labeled tree topology, where a stochastic "evolver" continuously swaps adjacent categorical labels. This models hierarchical systems (like network routing tables or organizational charts) undergoing constant local reorganization. The core problem we address is whether an algorithm can optimally track this changing ground truth while operating at a constant speedup.

Contribution: We introduce a novel geometric framework for evolving data. Unlike standard models, our framework adapts to spatial structures by decoupling the tracking hypothesis from the strict physical capacity constraints of the real-world structure. Within this framework, we present a simple, deterministic algorithm utilizing a directional oracle that, given a constant speedup factor $c \geq 2$ over the evolver, maintains the categorical labels within the asymptotically optimal average distance of $O(1)$ from their true locations.

Publication: *"Optimally Tracking Labels on an Evolving Tree"* Aditya Acharya, David Mount, 34th Canadian Conference on Computational Geometry (CCCG 2022) [3].

Connection: We will directly apply and expand the decoupled geometric framework, introduced here, in subsequent chapters to navigate continuous Euclidean domains using cone-based oracles (Chapter 3) and to track full stochastic probability distributions under local motion (Chapter 4).

Chapter 3: Tracking Evolving Categorical Labels using Cone-Based Oracles

Problem: We move from discrete trees to the continuous Euclidean plane. Here, updates require physical movement, and exact position queries may be expensive or impossible. This is motivated by robotics and sensor networks where agents (like drones) have limited directional sensing capabilities.

Contribution: We introduce a randomized algorithm using a *cone-based oracle*, which returns a directional "cone" containing the target rather than exact coordinates. By routing categorical labels via a sparse geometric spanner (Theta graph), we show that a speedup factor greater than 3 suffices to maintain an expected distance of $O(n)$.

Publication: *"Tracking Evolving Labels using Cone-Based Oracles"* Aditya Acharya, David Mount. Young Researchers Forum at The 39th International Symposium on Computational Geometry (SoCG 2023) [4].

Connection: This introduces geometric approximation (cones) into the tracking framework, paving the way for the full stochastic models in Chapter 4.

Chapter 4: Evolving Stochastic Distributions Under Local Motion

Problem: We generalize the framework to "imprecise points" modeled as stochastic probabil-

ity distributions. We introduce the *local-motion model*, where objects move a fraction of the distance to their nearest neighbor. This reflects real-world stochastic crowd dynamics where congestion limits speed.

Contribution: We use the *KL divergence* to measure the error of our hypothesis. We present an algorithm, `TRACKBYZOOM`, which dynamically adjusts the scale of "hypothesis balls" (Cauchy distributions) to maintain an $O(n)$ total divergence from the truth.

Publication: *"Evolving Distributions Under Local Motion"* Aditya Acharya, David Mount. 19th International Symposium on Algorithms and Data Structures (WADS 2025) [5].

Connection: This chapter is the pivot of the thesis. It explicitly introduces distributional distances (KL divergence) to handle uncertainty, motivating the study of geometry-aware categorical classification metrics in Part II.

1.4 Categorical Classification in Non-Euclidean Metrics (Part II)

This part explores static categorical classification problems using the geometric insights developed for bounded domains.

Chapter 5: Support Vector Machines in Hilbert Metric

Problem: Standard SVMs assume Euclidean geometry, which is ill-suited for stochastic data confined to a polytope (e.g., histograms in computer vision or biological profiles). We formulate the SVM problem for categorical classification in the *Hilbert metric*, which respects the domain boundaries.

Contribution: We prove that the Hilbert SVM problem is of *LP-type* (Linear Programming type),

meaning it can be solved efficiently by considering a small, fixed number of constraints (a basis). For bounding polytopes of constant complexity, we provide exact algorithms which are linear in the number of points.

Publication: *"Support Vector Machines in the Hilbert Geometry"* Aditya Acharya, Auguste H. Gezalyan, Julian Vanecek, David M. Mount, and Sunil Arya. 19th International Symposium on Algorithms and Data Structures (WADS 2025) [2].

Connection: This chapter establishes the theoretical foundation for linear categorical classification in Hilbert geometry but highlights the computational cost of high dimensions, motivating the approximation algorithms in Chapter 6.

Chapter 6: Improved Classifiers in High Dimensional Hilbert Metrics

Problem: The exact algorithms in Chapter 5 scale poorly with the complexity of the defining polytope, which is often exponential in high dimensions (d). Real-world stochastic feature vectors often live in high-dimensional spaces where exact computation is infeasible.

Contribution: We present a polynomial-time approximation algorithm for the Hilbert SVM. By formulating the categorical classification problem as a linear program (LP) with a specific error tolerance ϵ, we avoid the exponential dependency on dimension. We further extend the framework to include *soft-margin classifiers* (for non-linearly separable data) and *nearest-neighbor classifiers* in the embedding space.

Publication: *"Classifiers in High-Dimensional Hilbert Metrics"*. Aditya Acharya, Auguste H. Gezalyan, and David Mount. 20th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2026) [1].

Conclusion: This final chapter bridges the theoretical elegance of Hilbert geometry with the practical efficiency required for large-scale categorical classification tasks in machine learning.

Chapter 7: Conclusion and Future Directions

In this chapter we present a few promising avenues for future research, particularly at the intersection of temporal tracking, geometric classification, and machine learning.

Part I

Temporal Uncertainty: Tracking Evolving Data

Chapter 2: Optimally Tracking Labels on an Evolving Tree

2.1 Chapter Overview

As established in the Introduction, the first major challenge in modern algorithms is handling *temporal uncertainty*, where the ground truth changes continuously behind the scenes. Before addressing complex continuous domains or probability distributions (the focus of Chapters 3-4), we must first understand the fundamental limits of tracking in a discrete structural setting. In this chapter, we focus entirely on the core mechanics of maintaining proximity to a changing ground truth. We study the evolving data framework on a fixed tree topology, where labels continuously change positions due to the action of an agent called the "evolver". To tackle this effectively, we introduce a new geometric framework for evolving data. This novel approach adapts standard models to better suit spatial structures by decoupling our hypothesis from strict physical capacity constraints and utilizing directional oracles. An algorithm, which can only track these changes by explicitly probing individual vertices, is tasked with maintaining an approximate sketch of the underlying tree.

Our core investigation centers on a fundamental problem. We ask whether an algorithm can optimally keep pace with a shifting reality, and whether it can achieve this while operating at a constant speedup relative to the evolver. We present an algorithm that allows for both randomized and adversarial evolution of the data. We prove that a constant speedup factor of $c \geq 2$ is necessary and sufficient to maintain the labels within an average distance of $O(1)$ of their actual locations. This establishes

the theoretical baseline for the "work" required to track dynamic data, a concept we will expand to Euclidean space in Chapter 3.

2.2 Introduction

While Chapter 1 introduced the broad motivation of massive, dynamic datasets, this chapter focuses on the specific algorithmic mechanics required to track them. Standard models for dynamic structures (e.g., [35]) often assume we are notified of changes (e.g., an edge insertion). However, in the *evolving data framework* proposed by Anagnostopoulos *et al.* [7], changes occur invisibly.

In this chapter, we consider the problem of maintaining a tree with n distinct labeled nodes. We assume the tree topology is fixed, but the "evolver" changes label locations by swapping the labels of two adjacent vertices. This models hierarchical systems such as organizational charts, file systems, or network routing trees, where local updates occur frequently and without global notification.

We consider the problem in two modes:

- *Random Evolver:* Swaps are chosen uniformly at random.
- *Adversarial Evolver:* Swaps are chosen arbitrarily to maximize error.

To probe the structure's current state, we utilize an *oracle* (defined formally in Section 2.5). Unlike a sorting oracle which compares values, our oracle provides a *directional pointer* indicating which edge to traverse to find a specific label. This anticipates the "Cone-Based Oracle" we will develop for continuous space in Chapter 3.

Our update algorithm is extremely simple: query a label to find if it is at its hypothesized location; if not move the hypothesis one step closer to the truth. Because the evolver is also moving labels,

we define a *speedup factor* $c \geq 1$. This factor represents the relative speed of the algorithm compared to the rate of change in the environment.

2.2.1 Contributions:

We establish four main results that define the limits of this framework:

Lower Bound (Random): Even with a uniform random evolver, any algorithm with constant speedup on a bounded degree tree yields a steady-state error of $\Omega(n)$ (Theorem 2.1).

Upper Bound ($c = 2$): With a speedup factor of $c = 2$ against a random evolver, our simple algorithm achieves a steady-state distance of $O(n)$ (Theorem 2.2).

Upper Bound ($c > 2$): With a speedup factor of $c > 2$, the same algorithm succeeds against *any* evolver (including adversarial), achieving $O(n)$ distance (Theorem 2.3).

Lower Bound ($c < 2$): If the speedup is $c < 2$, there exists an adversarial strategy that forces the error to grow to $\Omega(n^2)$ (Theorem 2.4).

2.3 Related Work

2.3.1 Label Swapping

The problem we consider here falls under the general category of pebble motion problems. Given a graph $G = \{V, E\}$, a set of labels $L = \{l_1, l_2, \ldots, l_k\}$, a labeling configuration is defined as a mapping $M : L \to V$, such that $M(l_i) \neq M(l_j)$, for $l_i \neq l_j$. A single move of a label l, where $M(l) = v$, can be defined as updating the mapping to $M(l) = u$, where u is a neighbor of v.

Given two such label assignments M_1 and M_2 on a common graph, the problem of deciding whether there is a sequence of moves to transform M_1 to M_2 was first referred to as the *pebble motion problem* by Kornhauser *et al.* [56]. Under the restriction that a label can only be moved to an unmapped neighboring vertex, Goraly and Hassin *et al.* [42] show that the feasibility problem can be decided in linear time. Ratner *et al.* [77] proved that the associated problem of finding the optimal sequence of moves is NP-hard.

A variant of pebble motion that is more closely related to this dissertation is the problem of *token swapping*. Again we have a graph with n vertices, and there are n distinct labels. A single move involves swapping the labels of two neighboring vertices. It is easy to see that on a simple path, transforming one configuration to another is akin to sorting the path, and therefore such a sequence of swaps can be generated by a variant of bubble sort. Yamanaka *et al.* [95] showed that there exists a polynomial time 2-approximation when the graph is tree. Miltzow *et al.* [67] generalized this to a polynomial time 4-approximation on general graphs. Graf considered a very similar problem of moving objects along a tree by a robot and presents an excellent collection of similar problems [43, Section 6].

2.3.2 Evolving Data Framework

Another related line of work involves algorithms for evolving data sets, which was first introduced by Anagnostopoulos *et al.* [7]. In their framework, the input data set is constantly changing through the actions of a random evolving agent, or *evolver*, and an algorithm is tasked with maintaining an output that is close to the one corresponding to the current data. The algorithm can only access the data set through a series of probes, each of which returns some relevant local information. They

looked at the problem of maintaining a sorted order of points, where the true ranking of points evolves over time. They give an algorithm which maintains an approximate ordering with $O(n \log \log n)$ inversions.

One of the most fascinating and counterintuitive aspects of the evolving data framework is that algorithmic performance often defies conventional static-world wisdom. In static settings, comparison-based algorithms like quicksort or merge sort are optimal with $O(n \log n)$ time complexity, while quadratic algorithms like insertion sort or bubble sort are considered highly suboptimal. However, in an evolving environment where data continuously shifts, optimal static algorithms often fail to maintain a close approximation of the ground truth. This occurs because these algorithms make structural, long-range decisions early in their execution, which quickly become obsolete and incorrect as the underlying data evolves. In contrast, simpler quadratic algorithms typically make purely local, incremental updates. This continuous local repair mechanism makes them remarkably robust to ongoing perturbations.

Vial *et al.* [93] first observed this phenomenon empirically, noting that quadratic sorting algorithms seemed to perform better than optimal sorting algorithms in an evolving scenario. Besa *et al.* [15] later proved that a repeated run of an $O(n^2)$ time sorting algorithm like the *insertion sort* is good enough to maintain an approximate ordering with only $O(n)$ inversions. In fact they show that such an algorithm is optimal by showing a lower bound on the number of inversions. This counterintuitive robustness was further generalized by Disser *et al.* [41], where the authors analyzed an extremely naive sorting algorithm that simply samples a random pair of adjacent items in each step and swaps them if they are out of order. They proved that this simple, locally acting algorithm achieves and maintains, with high probability, an optimal total deviation of $O(n)$ and a maximum deviation of $O(\log n)$.

Researchers have considered other problems in the evolving context, including path connectivity, minimum spanning trees [9], shortest paths [99], and page rank [11], among others. A common theme across these papers is the evolution of the list of edges of the graph, either through introducing a new edge and deleting an existing one, or by changing the ranking of the edge weights. In all of the previous works in an evolving data scenario, the evolver is a randomly uniform process, which only effects a small local change in any particular time step.

In the field of computational geometry, maintaining a data structure or its approximate sketch has been viewed under the lens of either dynamic data [26] or in a stricter streaming model [37]. In both of these cases the changes are available to the maintaining algorithm either at once or in the form of a stream, and the goal is to output an exact or an approximate structure. Many of those algorithms only allow for a $O(\log n)$ time update per constant number of changes to the data set. In an evolving data scenario, however, a $\omega(1)$ time update might be too slow in keeping up with the evolver, not to mention the actions of the evolver are completely hidden from the algorithm.

2.4 A New Geometric Framework for Evolving Data

The primary motivation behind our proposed framework is to bridge the gap between simple linear sorting and complex geometric data sets. In a standard evolutionary sorting problem, the total ordering of data points provides an obvious, one-dimensional metric for evaluation. However, in geometric domains, such as trees or multidimensional Euclidean spaces, the relative ranking or spatial distribution of points is not strictly totally ordered. Our framework adapts the evolving data paradigm to make it amenable to these higher-dimensional and structural settings.

Our framework differs from the standard evolving data framework in a few significant aspects:

- The first difference involves the behavior of the evolver. An important characteristic of the evolving model introduced in [7] is that the evolver acts randomly, and algorithms in this model exploit the fact that the evolver will occasionally improve matters. In this chapter, we consider both uniformly random evolvers as well as evolvers that are non-uniform, possibly deterministic, which may act in an adversarial manner.

- The second, and perhaps most crucial, difference is that our hypothesis structure is fundamentally decoupled from the strict physical constraints of the real data structure. In standard models, the mapping of labels to vertices is strictly one-to-one. We think of the structure that the evolver acts on as a "real world" object, which possesses strict capacity constraints on the number of labels each location or vertex can hold. In contrast, our hypothesized labeled point set is a theoretical tracking model that is not constrained by these physical limitations. For instance, our algorithm might temporarily assign multiple labels to a single vertex. We argue that maintaining structural isomorphism between the hypothesis and reality is unnecessary. As long as we possess a well-defined distance metric to quantify the divergence between our hypothesis and the actual data structure, we can effectively measure and minimize the tracking error. This relaxation is highly advantageous for geometric data sets, where forcing the algorithm to resolve physical capacity constraints at every incremental step would be computationally prohibitive and unnecessary for approximate tracking.

- We also provide our algorithm with a constant speed-up factor to handle cases when each step of the evolver effects a bigger change than that of the algorithm. In compensation for this asymmetry, our algorithms and analyses are much simpler.

- The final difference is the nature of the Oracle. In contrast to standard comparison oracles, our

oracle directly compares the actual and hypothesized locations for a particular label and returns relevant geometric information to aid the algorithm in fixing the hypothesis. We can view our problem as a generalization of evolutionary sorting, but where the domain is a tree structure rather than a linear list. In sorting, the oracle determines whether two objects are out of order, but this binary concept is not strictly meaningful in our non-linear, tree-based setting. Instead, our oracle provides a directional pointer to the current location of the label, which natively respects the geometry of the domain.

2.5 Problem Formulation

In this section we provide the specifics of our evolving token/label swapping problem. We are given a fixed undirected tree $T = (V, E)$ with n vertices and maximum degree k. Each vertex of the tree is assigned a unique label from the set of labels $L = \{l_1, \ldots, l_n\}$, that is, there is a bijective mapping $M_T : L \to V$. At any time, let $\mathcal{T} = \{T, M_T\}$ denote the current "true" labeled tree (see Figure 2.1(a)).

The *evolver*, denoted $\mathcal{E}$, introduces changes to the labelings. Each time it runs it selects a pair of adjacent vertices in T and swaps their labels. The evolver may either be *random* or *adversarial*. In the former case the pair to be swapped is chosen uniformly at random, and in the latter the adjacent pair can be chosen arbitrarily, deterministically or adversarially. In Figure 2.1(a) and (b), the evolver swaps labels X and G.

Our algorithm maintains a model of current labeled tree in the form of a structure we call a *hypothesis tree*, denoted $\mathcal{H} = \{T, M_H\}$, where T is the same tree, and $M_H : L \to V$ is a (not necessarily bijective) mapping from labels to vertices. Note that M_H may assign multiple labels to a

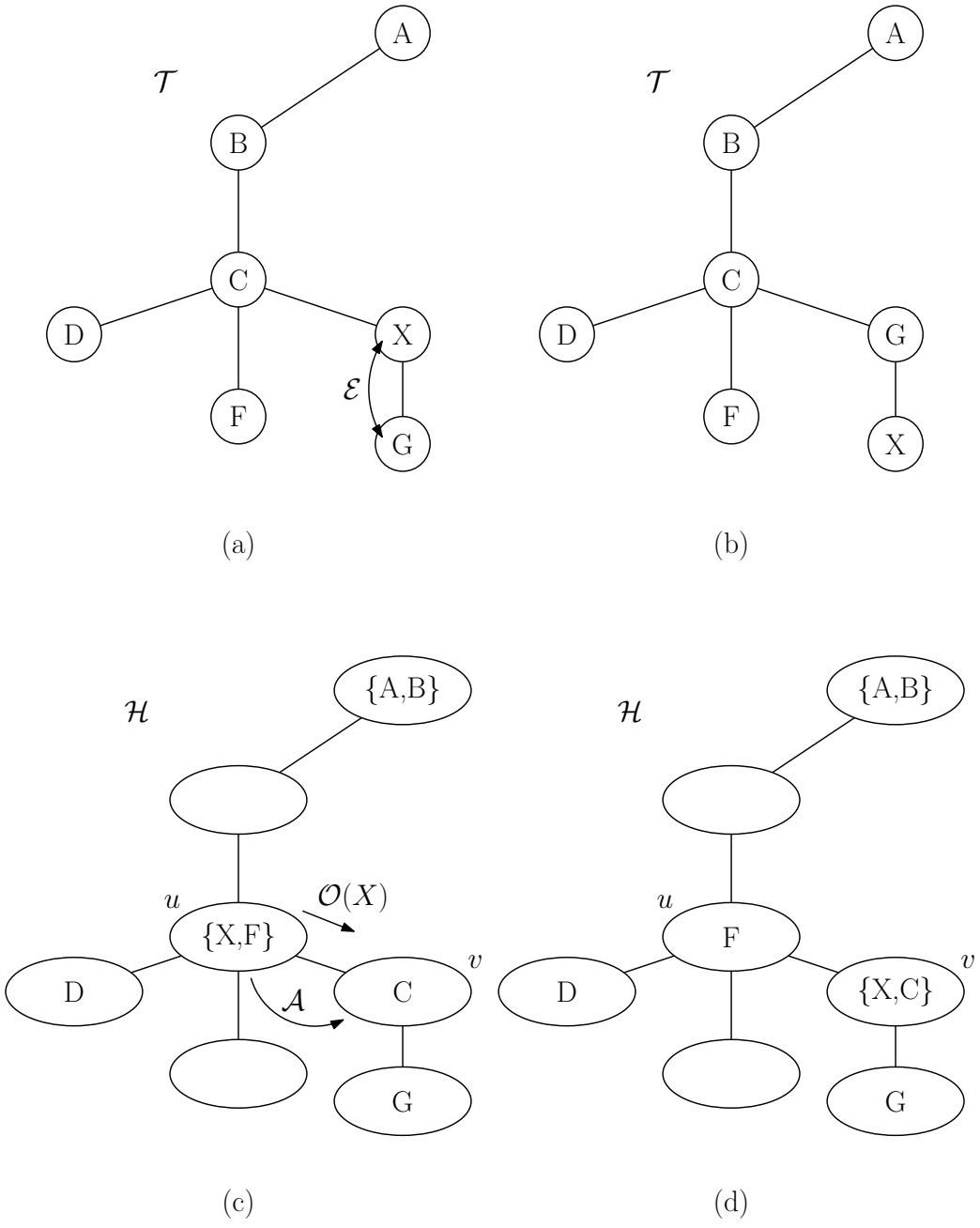


Figure 2.1: The action of the algorithm on a labeled tree $\mathcal{T}$, evolver $\mathcal{E}$, a labeled hypothesis tree $\mathcal{H}$, and oracle $\mathcal{O}$. (a): The current state of the underlying labeled tree $\mathcal{T}$. (b): The state of $\mathcal{T}$ after the evolver swapped labels across a pair of adjacent nodes. (c): A single step in our algorithm $\mathcal{A}$ on $\mathcal{H}$—Query label X, find that the oracle is pointing us to the location of X on $\mathcal{T}$, and then move the label X to the adjacent node in the returned direction. (d): The final state of our hypothesis tree $\mathcal{H}$ after a single step of $\mathcal{A}$.

vertex of T (see Figure 2.1(c)).

In order to probe the current actual state, we assume the existence of *oracle*, denoted $\mathcal{O}$. Each query to the oracle is presented in the form of a pair (l_i, u), where l_i is a label and u is a vertex. If l_i is currently located at u, the oracle returns a special value *null*. Otherwise, it returns the edge incident to u that lies on the shortest path from u to $M_T(l_i)$, the vertex that contains l_i in the true labeling. (In Figure 2.1(c), the query $\mathcal{O}(\mathrm{X}, u)$ returns the edge (u, v) because in the actual tree, the path to the node w containing X contains this edge.)

Each single step of algorithm $\mathcal{A}$ involves the following actions: $\mathcal{A}$ selects a label l and a vertex u. It then queries the oracle to find $\mathcal{O}(l, u)$ and then is free to move the label l from $M_H(l)$ to any adjoining node in the tree. A step of one such algorithm is illustrated in Figure 2.1(c) and (d), where the algorithm is applied to label X. The query $\mathcal{O}(\mathrm{X}, u)$ returns (u, v), and the algorithm moves label X to v. We define $\mathcal{C}$ as the class of such algorithms, and throughout this chapter we only consider algorithms from this class.

To measure how close our hypothesized labeling is to the true labeling we introduce a natural *distance function*. Given two vertices u and v in T, define their distance $d(u, v) = d_T(u, v)$ to be the tree distance, i.e., the length (number of edges) of the path between them. Given the true labeling $\mathcal{T}$ and the hypothesized labeling $\mathcal{H}$ and any label l_i, let $D_i = d(M_T(l_i), M_H(l_i))$ denote the distance between the assigned label positions. Define the overall distance to be $D(\mathcal{T}, \mathcal{H}) = \sum_{l_i \in L} D_i$.

Remark: $\mathcal{D}(\mathcal{P}, \mathcal{H})$ is a metric since it is the sum of tree distances, which are themselves metrics for a particular label. Observe that with each step the evolver can affect the overall distance by at most 2, moving each of the labels being swapped one node farther from our current hypothesis. Since we have n vertices and the maximum distance between two nodes on the tree is $n - 1$, we have $D(\mathcal{T}, \mathcal{H}) \in O(n^2)$. It is easy to see that there exists a tree T and a sequence of swaps by the evolver,

which results in $D(\mathcal{T}, \mathcal{H}) \in \Omega(n^2)$. Specifically, consider the case where T is a path and the labels are swapped in a sequence to result in a labeled path with the labels sorted in the opposite order. On the other hand, our algorithm clearly satisfies the following invariant: Every step of an algorithm from class $\mathcal{C}$ reduces the overall distance $\mathcal{D}(\mathcal{P}, \mathcal{H})$ by at most 1.

Given the disparity between the evolver's and our algorithm's effect on $\mathcal{D}(\mathcal{P}, \mathcal{H})$, we will allow our algorithm a modest *speedup factor*. We denote this by a constant $c \geq 1$. This means that the time taken by a single step of the evolver is c times as that of the algorithm. Or in other words, over a large enough time interval if the algorithm takes m steps, the evolver takes m/c steps.

We consider the following problem for a given speedup factor c and any arbitrary starting configuration of $\mathcal{H}$: Does there exist an algorithm with this speedup factor such that, in the steady state, after arbitrarily long execution sequences, $D(\mathcal{T}, \mathcal{H}) = o(n^2)$? We will in fact show that (depending on the nature of the evolver) that there exists a deterministic algorithm and an associated speedup factor such that $D(\mathcal{T}, \mathcal{H}) = O(n)$ from the underlying labeled tree, after some sufficiently large time and with high probability.

2.6 Probabilistic Tools

In this section we mention the probabilistic tools we use through out the paper. First, as a concentration bound, we use a weak version of Chernoff's inequality (Theorem 4.5 in [69]).

Lemma 2.1 (Chernoff Bound)**.** *Let $X_1, X_2, \ldots, X_n$ be independent random indicator variables, let $X = \sum_i X_i$, and let $\mu = E[X]$. Then,* $\Pr\left[X \leq \frac{\mu}{2}\right] \leq \exp(-\frac{\mu}{8})$.

Next we use a concept called Poisson approximation. Suppose $X_1, X_2, \ldots, X_n$ are the random variables indicating the number of balls in the i^{th} bin, when m balls are thrown into n bins uniformly

at random. We call this the exact case.

Let $Y_1, Y_2, \ldots, Y_n$ be independent Poisson random variables with $\Pr[Y_i = k] = e^{-\lambda}\frac{\lambda^k}{k!}$, where $\lambda = m/n$. In other words, Y_i represents the load in a bin, when the number of balls in each of them is a Poisson distribution with parameter λ. We note the following on any event that is a function of the loads of each bin. (Corollary 5.9 [69].)

Lemma 2.2 (Poisson Approximation)**.** *Any event that takes place with probability p in the Poisson case takes place with probability at most $p\ e\ \sqrt{m}$ in the exact case.*

2.7 Lower Bounds on the Distance

We first prove a lower bound on $\mathcal{D}(\mathcal{P}, \mathcal{H})$, when the maintaining algorithm is in the class $C(\mathcal{A})$ as defined in Section 2.5 and for any constant speedup factor c. Our proof follows the same structure as a similar proof by Anagnostopoulos *et al.* [7]. We prove the following for $D(\mathcal{T}, \mathcal{H})_{(t)}$, for a sufficiently large t, where $D(\mathcal{T}, \mathcal{H})_{(t)}$ denotes $\mathcal{D}(\mathcal{P}, \mathcal{H})$ at time t.

Theorem 2.1. *For any speedup factor $c \geq 1$ and for all sufficiently large t, irrespective of the algorithm $\mathcal{A}$, $D(\mathcal{T}, \mathcal{H})_{(t)} = \Omega(n)$ with high probability, even in the case of a random evolver.*

Proof. For ease of analysis we let our algorithm $\mathcal{A}$ run a single step every time unit, and the evolver, which runs c times slower perform a swap every c time units. Consider the time interval $[t - n/w, t]$, where w is a large constant. The algorithm and the evolver can reduce $\mathcal{D}(\mathcal{P}, \mathcal{H})$ by at most n/w and $2n/cw$ during this time interval, respectively. So if $D(\mathcal{T}, \mathcal{H})_{(t-n/w)}$ was at least $n/w + 2n/cw + \Omega(n)$, then $D(\mathcal{T}, \mathcal{H})_{(t)}$ remains $\Omega(n)$.

Next, let us assume $D(\mathcal{T}, \mathcal{H})_{(t-n/w)}$ is at most $n/w + 2n/cw + o(n)$. That implies there are at most $n/w + 2n/cw + o(n)$ labels displaced from their true location at time $t - n/w$. Let L' denote the

set of displaced labels, that is, $L' = \{l_i \mid D_i > 0\}$. We define $V' = \{M_T(l_i) \mid l_i \in L'\}$, as the set of corresponding vertices on T. And then the set of incident edges as $E' = \{(u, v) \mid u \in V' \vee v \in V'\}$. Since the degree of the T is k, we have $|E'| \leq k(n/w + 2n/cw + o(n))$.

In the same time frame, the algorithm $\mathcal{A}$ can act on at most n/w labels. Call that set of labels $L_{\mathcal{A}}$. Define $V_{\mathcal{A}} = \{M_T(l_i) \mid l_i \in L_{\mathcal{A}}\}$, as the set of corresponding vertices on T. And then the set of incident edges as $E_{\mathcal{A}} = \{(u, v) \mid u \in V_{\mathcal{A}} \vee v \in V_{\mathcal{A}}\}$. Now, $|E_{\mathcal{A}}| \leq kn/w$.

Next we look at the set of edges that were unaltered at time, $t - n/w$, and were not affected by the algorithm throughout the time interval. Call it $E^* = E \setminus (E' \cup E_{\mathcal{A}})$. Now, $|E^*| \geq n - 2kn/w - 2nk/cw - k \cdot o(n) \geq n\gamma$, for some sufficiently large w, and $\gamma = (1 - 2k/w - 2k/cw - k/w)$. The evolver picking any edge from E^* exactly once, guarantees that the labels stay swapped at the end of the time interval. Let X_e be the indicator variable, representing the fact that e is picked by the evolver exactly once.

We use the Poisson approximation scheme from Lemma 2.2. The evolver chooses n/cw edges at random from the n available ones. Therefore $\lambda = (n/cw)/n = 1/cw$, which is a constant. Hence, $\Pr[Y_e = 1] = \lambda e^{-\lambda} = s$, a constant. That implies, $\mathrm{E}[\sum_{e \in E^*} Y_e] \geq s\gamma n$. Using a Chernoff bound (Lemma 2.1), we have

$$\Pr\left[\sum_{e \in E^*} Y_e \leq s\gamma n/2\right] \leq e^{-\Omega(n)}.$$

Using Lemma 2.2 again, we have

$$\begin{aligned}
&\Pr\left[\sum_{e\in E^*} X_e \le s\gamma n/2\right] \\
&\le\ e\sqrt{\frac{n}{cw}}\Pr\left[\sum_{e\in E^*} Y_e \le s\gamma n/2\right] \\
&\le\ e\sqrt{\frac{n}{cw}}e^{-\Omega(n)}\ \le\ e^{-\Omega(n)}
\end{aligned}$$

Therefore with exponentially high probability, the evolver picks at least $s\gamma n/2$ edges from E^*, ensuring that those edges stay swapped at the end of the interval. Therefore, $D(\mathcal{T},\mathcal{H})_{(t)} \ge s\gamma n \in \Omega(n)$, as desired. □

2.8 Algorithm

Here, we describe a simple algorithm to track the labels. We use the same algorithm in both the cases of a random and an adversarial evolver. Recall the set of labels $L = \{l_1, \ldots, l_n\}$, and the definition of the oracle from Section 2.5. Intuitively, the algorithm works as follows. For each $l_i \in L$, we query the oracle on $(l_i, M_H(l_i))$ and update its location by moving it one step in the direction returned by the oracle. We keep doing this until the oracle returns *null*, that is, when l_i is in its true location. We then move on to the next label, repeating the process indefinitely. A single pass over all the labels is called an *iteration* of the algorithm. See Algorithm TRACKTREELABELS for the details.

2.9 Analysis

Again for ease of analysis, we let our algorithm $\mathcal{A}$ run a single step every time unit, and the evolver, which runs c times slower, perform a swap every c time units.

▷ Continuously run the algorithm
for $j \leftarrow 1, 2, \ldots, \infty$ **do**

  ▷ For every label in order
  for $i \leftarrow 1$ to n **do**

    ▷ Until the label is in its true location
    while $(\mathcal{O}(l_i, M_H(l_i)) \neq \textit{null})$ **do**

      ▷ Query the oracle to find the direction

      $(u, v) \leftarrow \mathcal{O}(l_i, M_H(l_i))$

      ▷ Update the location of the label
      $M_H(l_i) \leftarrow v$

    end while
  end for
end for

Let t_0 be the time when the algorithm starts. Let t_j be the time when the jth iteration of the algorithm ends. Let $\mathcal{D}(\mathcal{P}, \mathcal{H})$ at the start of the jth iteration be $D(\mathcal{T}, \mathcal{H})_j$. And for a specific label l_i we denote the distance at the start of the jth iteration to be $D_{i,j}$. We set the total number of moves effected on l_i, by the algorithm in the j^{th} iteration as $\mathcal{A}_{i,j}$. Therefore the total decrease in D_i, the distance with respect to label l_i, in the j^{th} iteration is $\mathcal{A}_{i,j}$. We define the total decrease in $\mathcal{D}(\mathcal{P}, \mathcal{H})$ due to the algorithm, in the j^{th} iteration as $\mathcal{A}_j$, $\mathcal{A}_j = \sum_{l_i \in L} \mathcal{A}_{i,j}$. We note the following about Δt_j, the time taken by the j^{th} iteration.

Lemma 2.3. $\Delta t_j = t_j - t_{j-1} = n + \mathcal{A}_j$.

Proof. Every step of the algorithm either moves a label in the direction of its true location, or fixes it, i.e., finds the label is in its true location. Since there are n labels, and $\mathcal{A}_j$ is the total moves effected by the algorithm, we have the result □

Next we show a lower bound for the time taken by the j^{th} iteration.

Lemma 2.4. $\Delta t_j \geq \frac{c}{2+c}(D(\mathcal{T},\mathcal{H})_j + n)$.

Proof. For a specific label l_i, our algorithm reduces its distance by $\mathcal{A}_{i,j}$, then finds that the label is at its true location, and then moves on to the next label. This implies that for some subset of steps taken by the evolver, the distance associated with l_i was reduced by $D_{i,j} - \mathcal{A}_{i,j}$. Otherwise, the algorithm would not have moved on to the next label. This further implies that in the j^{th} iteration for some subset of its steps, the evolver reduced the overall distance by at least $\sum_i (D_{i,j} - \mathcal{A}_{i,j}) = D(\mathcal{T},\mathcal{H})_j - \mathcal{A}_j$. That takes the evolver at least $(D(\mathcal{T},\mathcal{H})_j - \mathcal{A}_j)/2$ steps, or at least $(c/2)(D(\mathcal{T},\mathcal{H})_j - \mathcal{A}_j)$ time. Therefore we have $\Delta t_j \geq (c/2)(D(\mathcal{T},\mathcal{H})_j - \mathcal{A}_j)$. Using Lemma 2.3, we have $\Delta t_j \geq \frac{c}{2}(D(\mathcal{T},\mathcal{H})_j - \Delta t_j + n)$. Simplifying the inequality gives us the desired result. □

2.9.1 Random Evolver and Speedup of 2

In this section we prove the following: In the case of a random evolver, where the evolver $\mathcal{E}$ picks an edge at random and swaps its labels, an algorithm that runs at least twice as fast as the evolver maintains an optimal distance. Or in other words, we show that for $c \geq 2$, our algorithm ensures $D(\mathcal{T},\mathcal{H}) \in O(n)$ with high probability. Using Theorem 2.1, we can conclude that our algorithm is optimal for $c \geq 2$ and a random evolver.

As in [15], we first prove an interesting result about the random evolver. We show that a constant fraction of the steps taken by the random evolver do not increase the overall distance $\mathcal{D}(\mathcal{P},\mathcal{H})$.

Lemma 2.5. *For $c = 2$ and degree k, there exists a constant ϵ, $0 < \epsilon < 1$, such that for all j, the random evolver does not increase the overall distance in at least $\epsilon\Delta t_j$ steps in the j^{th} iteration, with high probability.*

Proof. From Lemma 2.3, we know Δt_j is at least n. We look at the first $n/10k$ steps of this particular iteration. The algorithm can process at most $n/10k$ nodes in this time. The number of edges incident on these nodes is at most $n/10$. Let E' denote the set of edges left unaltered by the algorithm in this time interval. Then $|E'| \geq 9n/10$. In the same time period, the evolver picks edges at random from the edge set E, $n/20k$ times with replacement. For every edge e in E, we set $X_e = 1$, if $e \in E'$, and the evolver picks e, at least twice in the time-frame, but picks none of the edges incident on e.

We use the Poisson approximation scheme from Lemma 2.2. The evolver chooses $n/20k$ edges from the n available ones. Therefore $\lambda = (n/20k)/n = 1/20k$, which is a constant. Now let Y_e be the independent Poisson approximations of X_e, with $\lambda = 1/20k$.

Next we find $\Pr[Y_e = 1]$. That represents the event when e is picked from E', and e is picked twice but none of the edges incident on e are picked. In the Poisson approximation scenario, each edge is picked j times with a probability $e^{-\lambda}\lambda^j/j!$. Therefore, the probability that an edge is picked at least twice is $(1 - e^{-\lambda} - \lambda e^{-\lambda})$, and the probability that it is not picked whatsoever is $e^{-\lambda}$. Since at most $2k$ edges can be incident on e, we have

$$\Pr[Y_e = 1] \;\geq\; \frac{9}{10}\left(1 - e^{-\lambda} - \lambda e^{-\lambda}\right) e^{-2k\lambda}.$$

Since the right hand side is a constant, there exists $s = O(1)$ such that $\Pr[Y_e = 1] \geq s$. Therefore, $\mathrm{E}[\sum_{e\in E} Y_e] \geq sn$. Using a Chernoff bound (Lemma 2.1), we have

$$\Pr\left[\sum_{e\in E} Y_e \leq sn/2\right] \;\leq\; e^{-\Omega(n)}.$$

Using Lemma 2.2 again, we have

$$\Pr\left[\sum_{e\in E} X_e \leq sn/2\right] \leq e\sqrt{\frac{n}{20k}}\Pr\left[\sum_{e\in E} Y_e \leq sn/2\right]$$
$$\leq e\sqrt{\frac{n}{20k}}e^{-\Omega(n)} \leq e^{-\Omega(n)}.$$

We note that if $X_e = 1$, then e is left unaltered by the algorithm, but it is altered at least twice by the evolver. That further means, one of those steps by the evolver either decreases the overall distance $\mathcal{D}(\mathcal{P}, \mathcal{H})$ or leaves it unchanged. And since the number of such edges e, with $X_e = 1$, is at least $sn/2$ with exponentially high probability, we conclude that in at least $sn/2$ of the evolver steps, in the first $n/10k$ steps of the iteration, $\mathcal{D}(\mathcal{P}, \mathcal{H})$ does not increase. Dividing the iteration into chunks of $n/10k$ steps, we obtain the desired result. □

Finally we prove one of the main theorems of this paper, that for a long enough passage of time, $\mathcal{D}(\mathcal{P}, \mathcal{H})$ converges to $O(n)$, in the case of $c = 2$, and a random evolver.

Theorem 2.2. *Given a tree of size n and a constant degree, and a random evolver, there exists z (a function of n) such that for all $j > z$, Algorithm* TRACKTREELABELS *achieves $D(\mathcal{T}, \mathcal{H})_j \in O(n)$, with a speed-up factor $c = 2$.*

Proof. Consider the j^{th} iteration. From Lemma 2.5, the evolver increases $\mathcal{D}(\mathcal{P}, \mathcal{H})$ by at most $(1 -$

$\epsilon)\Delta t_j$. In the same iteration the algorithm reduces $\mathcal{D}(\mathcal{P},\mathcal{H})$ by $\mathcal{A}_j$. Therefore, with high probability:

$$\begin{aligned}
D(\mathcal{T},\mathcal{H})_{j+1} &\leq D(\mathcal{T},\mathcal{H})_j + (1-\epsilon)\Delta t_j - \mathcal{A}_j \\
&\leq D(\mathcal{T},\mathcal{H})_j + n - \epsilon\Delta t_j && \text{[Lemma 2.3]} \\
&\leq \left(1-\frac{\epsilon}{2}\right) D(\mathcal{T},\mathcal{H})_j + \left(1-\frac{\epsilon}{2}\right) n && \text{[}c=2\text{ in Lemma 2.4]} \\
&= \left(1-\frac{\epsilon}{2}\right)^j D(\mathcal{T},\mathcal{H})_0 + \sum_{s=1}^{j}\left(1-\frac{\epsilon}{2}\right)^j n \\
&\leq \left(1-\frac{\epsilon}{2}\right)^j n^2 + O(n). && \text{[since } D(\mathcal{T},\mathcal{H}) \leq n^2\text{]}
\end{aligned}$$

By choosing $z = \log_{1/(1-\epsilon/2)} n$, we have $D(\mathcal{T},\mathcal{H})_{z+1} \in O(n)$. □

Remark: We showed that for large enough j, $D(\mathcal{T},\mathcal{H})_j \in O(n)$. Can we conclude the same about $\mathcal{D}(\mathcal{P},\mathcal{H})$ throughout the j^{th} iteration as well? In particular we look at $D(\mathcal{T},\mathcal{H})_{i,j}$. We note that in our Algorithm TRACKTREELABELS we could have started with processing the label l_i first (instead of l_1), l_{i+1} next, and so on. Therefore for a large enough j, $D(\mathcal{T},\mathcal{H})_{i,j} \in O(n)$ as well. Since $D(\mathcal{T},\mathcal{H})_{i+1,j} \leq D(\mathcal{T},\mathcal{H})_{i,j} + O(n)$, we conclude that for a large enough passage of time $\mathcal{D}(\mathcal{P},\mathcal{H})$ converges to $O(n)$.

In our labeled hypothesis tree $\mathcal{H}$ multiple labels could reside at a particular node. We show a simple result on the maximum number of labels that could be mapped to single vertex in T.

Corollary 2.1. *Let $L_{\mathcal{H},v}$ be the set of labels residing at a node v in $\mathcal{H}$, after a long enough passage of time. Then, $|L_{\mathcal{H},v}| \in O(\sqrt{n})$*

Proof. Let $|L_{\mathcal{H},v}| = w$. For l_i's, $l_i \in L_{\mathcal{H},v}$, we consider the corresponding distances D_i's. Consider that set as D_v, $D_v = \{D_i | l_i \in L_{H,v}\}$. Since the tree has degree k, there can be at most k 1's in D_v,

similarly k number of 2's, and so on. At most one member of D_v can be zero. Therefore

$$D(\mathcal{T},\mathcal{H}) \;\geq\; \sum_{x\in D_v} x \;\geq\; k(1+2+\cdots+(w-1)/k) \in \Omega(w^2).$$

Since $D(\mathcal{T},\mathcal{H}) \in O(n)$ after a long enough time from Theorem 2.2, we conclude $w \in O(\sqrt{n})$. □

2.9.2 Adversarial Evolver and Speedup ¿ 2

We conclude with the case when the evolver is adversarial. That means we cannot rely on a result similar to Lemma 2.5. We show that for a speedup factor of $c > 2$, or in other words, if there exists $\delta \in \mathbb{R}$, such that $c = 2 + \delta$, we can still maintain an optimal distance.

Theorem 2.3. *Given a tree of size n, an adversarial evolver, there exists z (a function of n) such that for all $j > z$, Algorithm* TRACKTREELABELS *achieves $D(\mathcal{T},\mathcal{H})_j \in O(n)$, with any speed-up factor $c > 2$.*

Proof. Consider the j^{th} iteration. The evolver increases $\mathcal{D}(\mathcal{P},\mathcal{H})$ by at most $\frac{2\Delta t_j}{c}$. Therefore

$$\begin{aligned} D(\mathcal{T},\mathcal{H})_{j+1} &\leq D(\mathcal{T},\mathcal{H})_j + \frac{2\Delta t_j}{c} - \mathcal{A}_j \\ &= D(\mathcal{T},\mathcal{H})_j + \frac{2\Delta t_j}{c} + n - \Delta t_j && \text{[Lemma 2.3]} \\ &= D(\mathcal{T},\mathcal{H})_j + n - \left(1 - \frac{2}{c}\right)\Delta t_j. \end{aligned}$$

By applying Lemma 2.4, we have

$$
\begin{aligned}
D(\mathcal{T},\mathcal{H})_{j+1} &\leq D(\mathcal{T},\mathcal{H})_j + n - \frac{c-2}{2+c}\left(D(\mathcal{T},\mathcal{H})_j + n\right) && \text{[Lemma 2.4]}\\
&= \frac{4}{2+c} D(\mathcal{T},\mathcal{H})_j + \frac{4}{2+c} n\\
&\leq \left(\frac{4}{2+c}\right)^j D(\mathcal{T},\mathcal{H})_0 + \sum_{s=1}^{j} \left(\frac{4}{2+c}\right)^j n\\
&\leq \left(\frac{4}{2+c}\right)^j n^2 + O(n). && [c > 2,\ D(\mathcal{T},\mathcal{H}) \leq n^2]
\end{aligned}
$$

For $c > 2$, and by choosing $j > \log_{\frac{c+2}{4}} n$, we have $D(\mathcal{T},\mathcal{H})_{j+1} = O(n)$. □

2.9.3 Adversarial Evolver and Speedup ¡ 2

We adapt a construction from Biniaz *et al.* [16] to prove a lower bound on the required speed-up to ensure $D(\mathcal{T},\mathcal{H})_{(t)} \in O(n)$. Construct two configurations of a labeled tree $\mathcal{T}_0$, and $\mathcal{T}_1$ as in Figure 2.2. On such a tree: $D(\mathcal{T}_1,\mathcal{T}_0) \sim 2 \cdot OPT$, where *OPT* is the number of optimum swaps required to go from one configuration to the other. Intuitively, an algorithm running at a speed-up factor less than 2, will fail to catch up with an adversarial evolver that takes *OPT* swaps to modify $\mathcal{T}_0$, to $\mathcal{T}_1$. We can show that any algorithm from class $\mathcal{C}$ running with speed-up $2-\delta$, where δ is a small positive constant, cannot achieve $D(\mathcal{T},\mathcal{H}) \in O(n)$. In fact we can prove something stronger:

Theorem 2.4 (Lower bounds on speed-up)**.** *Given any time instant t_0, there exists a tree T, an adversarial evolver $\mathcal{E}$, and a time instant $t > t_0$ s.t. $D(\mathcal{T},\mathcal{H})_{(t)} \in \Omega(n^2)$, for any algorithm from class $\mathcal{C}$, which runs with a speedup $2-\delta$, where δ is a positive constant.*

Proof. Suppose we have access to an algorithm $\mathcal{A}$ from the class $\mathcal{C}$, as defined in Section 2.5, with a speed-up factor of $c = 2-\delta$, δ is a positive real constant. We show the existence of a tree, and

an adversarial evolver, where such a speed-up is not sufficient for $D(\mathcal{T}, \mathcal{H}) \in O(n)$. We adapt a construction from Biniaz *et al.* [16]. See Figure 2.2. Let $\mathcal{T}_0$ be a uniquely labeled tree, with β wings, α tails, and a central vertex. Each wing contains α nodes. For our purposes, we let $\alpha \in \Omega(n)$. $n = \alpha\beta + \alpha + 1$. Let $\mathcal{T}_1$ be another labeled instance of the same tree, where the labels of the wings, are cyclically permuted. The order of the labels on a wing remains the same, as do other labels of the tree. This gives us $D(\mathcal{T}_1, \mathcal{T}_0) = \beta\alpha(\alpha + 1)$.

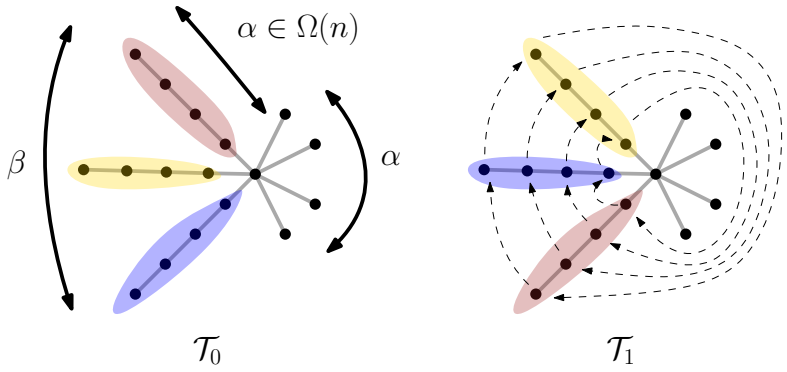


Figure 2.2: $\mathcal{T}_0$ is a n-node tree with β wings, α tails, and a central vertex. Each wing contains α nodes. $\mathcal{T}_1$ has the labels of the wings of $\mathcal{T}_0$ cyclically permuted. Adapted from [16].

Biniaz *et al.* [16] show that the optimal number of adjacent swaps to go from $\mathcal{T}_0$ to $\mathcal{T}_1$ is $opt(\alpha, \beta) = (\beta + 1)(\alpha(\alpha + 1)/2 + 2\alpha)$. Consider a time t_0, where $\mathcal{T}_0$ is the labeled configuration of the tree, with our hypothesis tree $\mathcal{H}_0$ being exact, i.e., $D(\mathcal{T}_0, \mathcal{H}_0) = 0$. Next, consider an adversarial evolver $\mathcal{E}$, which performs $opt(\alpha, \beta)$ number of swaps such that at time $t_1 = t_0 + opt(\alpha, \beta)$, $\mathcal{T}_1$ is the true labeling. Let $\mathcal{H}$ be the hypothesized labeling at time t_1. Since $\mathcal{A}$ has a speed-up of $2 - \delta$, and can affect the distance by at most 1 every step, we have $D(\mathcal{H}_1, \mathcal{T}_0) \leq (2 - \delta)\ opt(\alpha, \beta)$. Considering

$\beta = 2/\delta$, and $\alpha = \Omega(n)$ we have the following:

$$
\begin{aligned}
D(\mathcal{H}_1, \mathcal{T}_1) &\geq D(\mathcal{T}_1, \mathcal{T}_0) - D(\mathcal{H}_1, \mathcal{T}_0) && [D(\cdot,\cdot) \text{ is a metric}] \\
&\geq \beta\alpha(\alpha+1) - (2-\delta)\,(\beta+1)\left(\frac{\alpha(\alpha+1)}{2} + 2\alpha\right) \\
&\geq \left(\frac{2}{\delta} - \frac{(2-\delta)(2+\delta)}{2\delta}\right)\alpha(\alpha+1) - s(\delta)\alpha && [\text{Set } \beta = \tfrac{2}{\delta},\ s(\delta) \text{ is a constant}] \\
&\geq \frac{\delta}{2}\alpha(\alpha+1) - s(\delta)\alpha \ \in\ \Omega(n^2). && [\text{For } \alpha \in \Omega(n), \text{ and constant } \delta]
\end{aligned}
$$

□

2.10 Concluding Remarks

In this chapter, we have presented an efficient algorithm for tracking vertex labels in a tree. To accomplish this, we introduced a novel geometric framework for evolving data that adapts the standard evolving model to spatial domains by relaxing strict physical capacity constraints and relying on directional probing. Our analysis showed that in the limit, it is possible to maintain labels to within an average distance of $O(1)$ of their actual locations, subject to allowing a constant speedup factor $c \geq 2$ over the evolver.

However, many modern applications, such as tracking unmanned aerial vehicles or disease hotspots, operate in continuous space where updates require physical movement and exact coordinates are often unavailable. In the next chapter, we will leave the discrete world of trees and extend these concepts to the Euclidean plane. We tackle the challenge of tracking labels using *cone oracles*, where directional uncertainty plays a central role.

Chapter 3: Tracking Evolving Labels using Cone-Based Oracles

3.1 Chapter Overview

In the previous chapter, we presented a deterministic algorithm for tracking data on a discrete tree structure. However, many of the motivating applications for this dissertation, such as monitoring environmental sensors or tracking drone swarms, operate in continuous physical space, not fixed topologies.

In this chapter, we bridge the gap between discrete data structures and continuous geometry. We consider the problem of tracking labeled nodes in the Euclidean plane, where updates involve physical movement rather than instantaneous pointer swaps.

We introduce two major realistic constraints:

1. *Directional Uncertainty:* Real-world sensors often provide relative direction (e.g., "target is North-East") rather than precise coordinates. We model this using a *Cone-Based Oracle*.

2. *Local Memory (Cache Efficiency):* For massive datasets, an algorithm cannot constantly access global information. We utilize a *Theta Graph*, a sparse geometric spanner, to route updates locally, ensuring high cache efficiency.

We present a randomized algorithm that routes labels toward their true locations. We prove that with a speedup factor $c > 3$, the algorithm maintains an expected average distance of $O(n)$ from the

ground truth, effectively taming the continuous "evolver."

3.2 Introduction

The *evolving data framework*, proposed by Anagnostopoulos *et al.* [7], typically assumes that an algorithm can "probe" data to get exact values (e.g., comparing two numbers in a list). Chapter 2 adapted this to trees using precise edge pointers. However, in geometric settings, exact information is often expensive or unavailable.

Consider the task of tracking a swarm of *Unmanned Aerial Vehicles (UAVs)* [96]. A central controller might not know the exact GPS coordinates of every drone due to latency or jamming. However, a directional antenna or a camera feed might confirm that a specific drone is "somewhere within a 60-degree cone" relative to a reference point. Similarly, when tracking *disease hot-spots* via mobile testing units [63], we may know the general direction of a spreading infection cluster without knowing its precise center.

This chapter addresses the problem of maintaining a labeling in the plane under these conditions. An "evolver" continuously swaps labels of nearby nodes (modeling local movement or state changes), and we must update our hypothesis by physically moving labels along the edges of a sparse graph.

3.2.1 The Challenge of Memory and Locality

A key motivation for this chapter is *cache efficiency*. In massive geometric datasets (e.g., terabyte-scale spatial indices), loading the entire graph into memory is infeasible. Our algorithm relies on *local routing*: to move a label, we only need to load the immediate neighbors in a sparse graph (the Theta graph). This mimics the behavior of "Tiered Memory" systems [22, 98], where local

neighborhood queries are cheap ($o(1)$ after overhead), but jumping to a random location is expensive.

3.3 Theta Graphs and Geometric Routing

To handle continuous motion efficiently, we discretize the plane using a *Theta Graph*. A Theta graph is a sparse geometric spanner with a spanning ratio $t_\theta = 1/(1 - 2\sin(\theta_k/2))$, where $\theta_k = 2\pi/k$ is the angle induced by a set of cones around a point [79] (Figure 3.1(a)). Theta graphs enable a simple, memory-efficient routing scheme (Keil-Gutwin routing [54]):

> Let p be the current location of our label, and q the (unknown) true target location. The oracle tells us which cone $C_{p,q}$ around p contains q. We then move to the neighbor of p that lies within that cone and is "closest" in projection (see Figure 3.1(b)).

Crucially, this routing requires *only* local knowledge of p's neighbors and the approximate direction of q.

3.4 Problem Formulation

Let $P \in \mathbf{R}^2$ be a fixed point set of size n. Each point is assigned a unique label from the set $L = \{l_1, \ldots, l_n\}$ through a matching M.

The Evolver: At every time step, the evolver swaps labels between any pair of points that are less than a unit distance apart. This constraint models "local motion" or diffusion—labels don't teleport across the map; they diffuse through the crowd.

The Algorithm: We maintain a hypothesis mapping $H : L \to P$. We update H by moving a single label over the edges of the Theta graph G_θ.

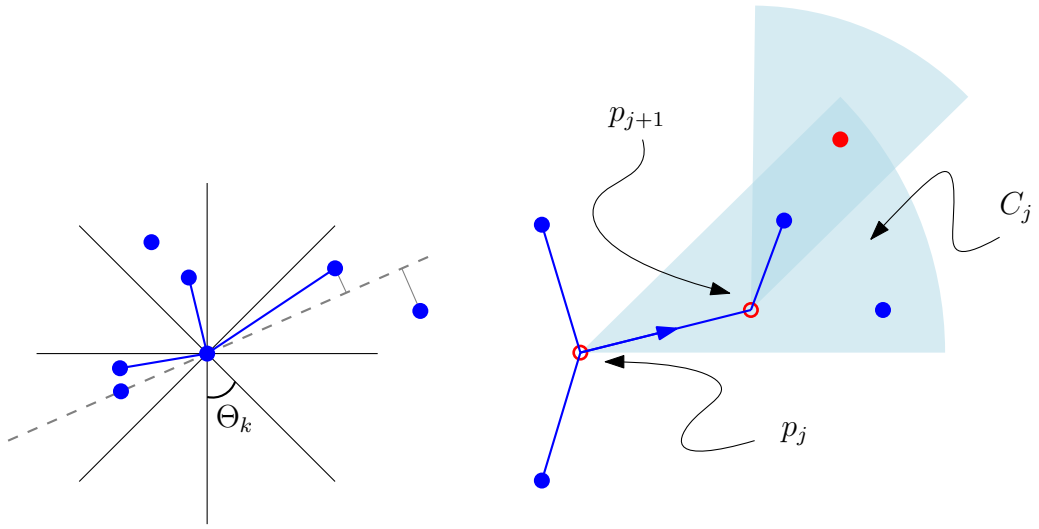


Figure 3.1: Theta Graph: Construction and Routing. (a) Construction: For each cone around a point, add an edge to the "nearest" neighbor in that cone (projected distance). (b) Routing: If the oracle indicates the target is in cone C_j, follow the edge inside it. Repeat at the next node.

The Metric: We define the error $\mathcal{D}_l$ for label l as the Euclidean distance between its hypothesized location and its actual location. The total error is $\mathcal{D}(M, H) = \sum_{l \in L} \mathcal{D}_l$. Without a competent algorithm, this error can grow to $O(n^2)$.

The Cone Oracle: We assume access to an oracle $\mathcal{O}(l, H(l))$. When queried, it returns the specific cone C_k around the current hypothesis $H(l)$ that contains the true location of l. Physically, this represents a directional sensor or a "hot/cold" gradient hint.

Cost Model: We assume a "Cached Memory Model."

- *Switching Cost:* Initiating a search for a new label takes significant time (memory overhead M).
- *Tracking Cost:* Once a label is loaded, subsequent moves along the graph take $o(1)$ time (cache hits).

Our goal is to find a speedup factor c, the ratio of our movement speed to the evolver's speed, that guarantees the total error remains $O(n)$.

3.5 Algorithm: TrackByCones

We employ a *Randomized Strategy* to defeat the evolver. If we updated labels in a fixed order (e.g., Round Robin), an adversarial evolver could predict our moves and disturb the labels we are about to check. By picking a label uniformly at random, we distribute our tracking effort and prevent the adversary from creating "worst-case" clusters of error. In contrast with Chapter 2, the analysis is much simpler since we only argue on the expected distance of the hypothesis to the truth. We however conjecture that there exists a deterministic algorithm as well with the exact speedup required by the randomized algorithm that maintains an optimal distance from the truth, under an adversarial evolver.

Rupert and Seidel [79] give an $O(n \log n)$ time algorithm to construct the theta graph in plane. We assume this is available to us with the required θ; smaller the θ, denser the graph and higher the memory requirements. Our main algorithm is described in Algorithm TRACKBYCONES. Every iteration we choose a random label l, and track it's true location. We assume we have an oracle which returns the cone around $H(l)$ that contains the true location of l. Using the returned cone, we find the corresponding edge using our constructed theta graph, and move l at a speed of ct_θ. We move on to another label, once the oracle returns NULL. We assume the evolver does not have access to our random bit generator.

Algorithm TRACKBYCONES Randomized Tracking using Cone Oracles

Require:
$\{H(l_1), H(l_2), \cdots, H(l_n)\}$ ▷ Initial hypothesized location of the labels
$\{e_{u,v}\}$ ▷ The edge set of the θ-graph

1: **while** true **do** ▷ Continuously run the algorithm
2: **for** l chosen randomly uniformly from $\{l_1, l_2, \cdots, l_n\}$ **do**
3: **while** $\mathcal{O}(l, H(l)) \neq NULL$ **do** ▷ Keep tracking l until found
4: Let $C_k \leftarrow \mathcal{O}(l, H(l))$ ▷ Cone returned by the oracle
5: Let $e_{l,k} = (H(l), v)$ be the edge incident on $H(l)$ inside C_k
6: Move l along $e_{l,k}$ with speed ct_θ ▷ c is a constant
7: Update $H(l) \leftarrow v$
8: **end while**
9: **end for**
10: **end while**

3.6 Analysis

Let the memory overhead be M, or in other words it takes M units of time to change a label and begin tracking it over the graph. Under a cached memory model, the oracle queries take negligible time compared to the switching cost of the labels. Hence lines 3, 4, 5, and 7 in Algorithm TRACKBYCONES take $o(1)$ time. We define a single iteration as picking a random label and tracking it, that is, a single loop of the *for loop* in line 2.

Let $\mathcal{D}_i$ be the overall distance at the start of the i^{th} iteration. Let $\mathcal{D}_l$ be the corresponding distance for label l. Since we are moving at the speed of ct_θ over a t_θ spanner, we spend D_l/c time in traversing Euclidean distance D_l. In that time the evolver could have moved the label by at most another D_l/c distance. So, in the worst case, the time spent by the algorithm in tracking l is at most

$$\frac{D_l}{c} + \frac{D_l}{c^2} + \cdots = \frac{D_l}{c-1}$$

In that time, the evolver can increase the overall distance $\mathcal{D}_i$ by at most $2D_l/(c-1)$. The algorithm reduced $\mathcal{D}_i$ by D_l by completely tracking l. Now since we chose l at random, in expectation we have

$\mathrm{E}[D_l] = \mathcal{D}_i/n$. Therefore,

$$\begin{aligned}
\mathrm{E}\left[\mathcal{D}_{i+1} \mid \mathcal{D}_i\right] &\leq \mathcal{D}_i - \frac{\mathcal{D}_i}{n} + \frac{2}{c-1} \cdot \frac{\mathcal{D}_i}{n} + M \\
\implies \mathrm{E}\left[\mathcal{D}_{i+1}\right] &\leq \left(1 - \frac{c-3}{c-1} \cdot \frac{1}{n}\right) \mathrm{E}\left[\mathcal{D}_i\right] + M \\
&\leq \left(1 - \frac{z}{n}\right) \mathrm{E}\left[\mathcal{D}_i\right] + M && \left[z = \frac{c-3}{c-1}\right] \\
&\leq \left(1 - \frac{z}{n}\right)^i \mathrm{E}\left[\mathcal{D}_1\right] + M \sum_{j=1}^{i} \left(1 - \frac{z}{n}\right)^i \\
&\leq \left(1 - \frac{z}{n}\right)^i O(n^2) + \frac{nM}{z} && [z > 0]
\end{aligned}$$

The above holds provided $z = (c-3)/(c-1) > 0$, which implies $c > 3$. Next we observe that for a sufficiently large i, we have $\mathrm{E}\left[\mathcal{D}_{i+1}\right] \in O(n)$. In fact the following suffices:

$$i = \frac{\ln n}{\ln n - \ln(n-z)} \sim n \ln n / z.$$

Therefore a speed-up factor of $3\,t_\theta$ is sufficient for our purposes. Hence, we have proved:

Theorem 3.1. *There exists a randomized algorithm with a constant speed-up factor, using cone based oracles, which in expectation maintains a hypothesized labelling with distance: $O(n)$, in the presence of an adversarial evolver which does not have access to the random bits.*

3.7 Concluding Remarks

In this chapter, we successfully moved the evolving data framework into the continuous domain. We showed that precise coordinates are not strictly necessary; *directional cues (cones)* combined with

sparse geometric graphs (Theta graphs) are sufficient to track moving targets with bounded error. This has significant implications for real-world systems with limited sensor precision or bandwidth constraints.

However, a fundamental limitation remains: both Chapter 2 and Chapter 3 treat the tracked objects as *singular points*. In many complex systems such as crowd dynamics, electron clouds, or uncertainty propagation, the "location" of an object is better described as a *probability distribution* rather than a single coordinate.

In *Chapter 4*, we will take the final step in our evolving data framework. We will abandon the concept of "point tracking" entirely and introduce a method to track *evolving distributions*, using information-theoretic measures (KL Divergence) to quantify our success.

Chapter 4: Evolving Distributions Under Local Motion

4.1 Chapter Overview

In the preceding chapters, we addressed the problem of tracking evolving data in increasingly complex domains. Chapter 2 established the fundamental speedup requirements for discrete trees, while Chapter 3 extended these concepts to point tracking in the Euclidean plane. However, both approaches shared a common simplification: they treated the tracked objects as *singular points* with precise coordinates, even if our view of them was obscured.

In this chapter, we confront the reality of physical systems where location itself is inherently uncertain. Whether tracking electron clouds, crowd density, or diffusing chemicals, the "state" of an object is better modeled as a *spatial probability distribution* rather than a single coordinate.

We introduce the (α, β)-*local-motion model*, where objects evolve not just by translation, but by reshaping their probability footprints based on local congestion. To measure the quality of our tracking, we abandon Euclidean distance in favor of the *Kullback-Leibler (KL) divergence* (relative entropy). This marks a significant shift from "geometric distance" to "informational distance," a theme that will dominate the remainder of this dissertation.

We present an algorithm that maintains a hypothesis close to the ground truth distributions. We prove that with a constant speedup factor, the algorithm guarantees a total divergence of $O(n)$ in the steady state, which is asymptotically optimal.

4.2 Introduction

Modern computational geometry is often tasked with managing data that is both massive and dynamic. As discussed in the Introduction (Chapter 0), the *evolving data framework* [8] provides a robust theoretical model for such problems. In this framework, an *evolver* changes the data invisibly, and an algorithm probes the system to maintain an accurate hypothesis.

In Chapter 3, we used a *Cone-Based Oracle* to handle directional uncertainty when tracking points. However, many real-world phenomena resist reduction to point coordinates. Consider:

- *Crowd Dynamics:* In a dense crowd, an individual's location is constrained by their neighbors. We effectively track "personal space bubbles" rather than pin-points [45].
- *Sensor Fusion:* GPS readings often come with a "circle of confusion" or a Gaussian error profile [89].
- *Quantum/Stochastic Systems:* The object itself may be delocalized.

In this chapter, we generalize the evolving framework to maintain a set of n *evolving probability distributions* in $\mathbb{R}^d$.

4.2.1 From Euclidean Distance to Information Geometry

In Chapters 1 and 2, we minimized Euclidean (or Tree) distance. However, in the space of probability distributions, Euclidean distance is often meaningless. Instead, we employ the *Kullback-Leibler (KL) divergence* (or relative entropy) [60].

The KL divergence, denoted $D_{\text{KL}}(P \parallel H)$, quantifies the "information loss" when a distribution H (hypothesis) is used to approximate P (truth). It answers the question: *"How many extra bits do I*

need to encode the location of this object if I use my hypothesis instead of the truth?"

This choice of metric is pivotal for the overall thesis. The KL divergence shares deep geometric properties with the *Hilbert Metric*, which we will introduce in Part II (Chapter 4) for classification tasks. Both metrics diverge as one approaches the boundary of the domain (certainty), making them highly sensitive to "edge cases" in data.

4.2.2 The Local-Motion Model

We introduce a motion model that respects the geometry of the data. In our *local-motion model*, the distance an object can move is not fixed (as in standard kinetic data structures) but is a fraction of the distance to its *nearest neighbor*. This captures the physical intuition that objects in sparse regions can move freely and fast, while objects in dense clusters ("traffic jams") are constrained to slow, local movements [12, 84].

Our contributions are as follows:

1. We formalize the tracking of probabilistic data using KL divergence within the evolving framework.

2. We present the TRACKBYZOOM algorithm, which dynamically adjusts the scale of "Hypothesis Balls" (Cauchy distributions) to match the evolving ground truth.

3. We prove that this algorithm achieves an optimal steady-state error of $O(n)$ with constant speedup.

4.3 Problem Formulation and Results

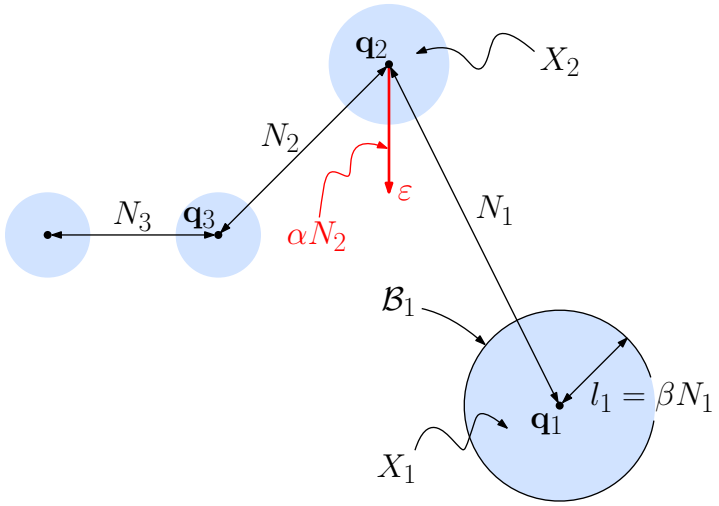
4.3.1 Objects

The objects that populate our system can be thought of as imprecise points [21, 65, 81] that are drawn independently from probability distributions that depend on the proximity to other objects. More formally, at any fixed time step, let $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\} \subset \mathbb{R}^d$ denote a point set of size n in d-dimensional Euclidean space, and let $[n]$ denote the index set $\{1, \ldots, n\}$. For each $\mathbf{q}_i \in Q$, let N_i denote the distance to its nearest neighbor in $Q \setminus \{\mathbf{q}_i\}$. Given $\mathbf{x} \in \mathbb{R}^d$ and nonnegative real r, let $\mathcal{B}(\mathbf{x}, r)$ denote the closed Euclidean ball of radius r centered at $\mathbf{x}$, and let $\mathrm{U}(\mathcal{B}(\mathbf{x}, r))$ denote the uniform probability distribution over this ball. Given a real β, where $0 < \beta < 1$, define the *β-local feature region* of $\mathbf{q}_i$ to be $\mathcal{B}(\mathbf{q}_i, \beta N_i)$. Let $P_i = P_i(\beta)$ denote the uniform probability distribution, $\mathrm{U}\left(\mathcal{B}(\mathbf{q}_i, \beta N_i)\right)$, and let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote this set of distributions. We refer to $\mathcal{P}$ as the *truth* or *ground truth*, as it represents the true state of the system, subject to the given imprecision. Together, Q and β define a set n independent random variables $\{X_1, \ldots, X_n\}$, with X_i distributed as P_i (see Figure 4.1a).

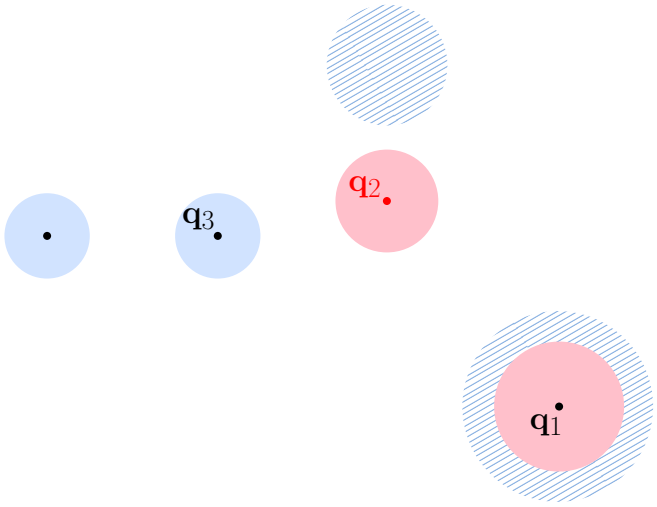
Remark 4.1. Our choice of using a uniform probability distribution is not critical to our approach. We later show that our results apply to a much broader class of distributions.

4.3.2 The Local-Motion Model

As mentioned in the introduction, there are many ways to model the realistic motion of a collection of objects in an environment. A natural requirement is that each object's motion is affected by the presence of nearby objects. Although there are many ways to incorporate this information (see,



(a) Illustration of the objects, and the notations used. The evolver's action ε is shown in red.



(b) Objects that were affected by ε are shown in pink. Tiled regions show the previous state.

Figure 4.1: The model and an action by the evolver. Shaded regions represent objects.

e.g., [84]), we have chosen a very simple model, where velocities are influenced by the distance to the nearest neighbor. We think of the objects of our system as moving continuously over time, and our algorithm queries their state at regular discrete *time steps*. From the algorithm's perspective, objects move, or "evolve," over time in the following manner. Given a parameter α, where $0 < \alpha < 1$, at each time step, an entity called *evolver* selects an object $i \in [n]$ and moves $\mathbf{q}_i$ by a distance of at most αN_i in any direction of its choosing (see Figure 4.1). While the value of α is known, the action of the evolver, including the choice of the object and the movement, is hidden from the algorithm. Throughout, we assume that the evolver is a *strong adversary*, which means that it has access to our algorithm and the input set [20]. For a nonnegative integer time step t, let $Q^{(t)}$ and $\mathcal{P}^{(t)}$ denote the underlying centers and distributions, respectively, at this time. To simplify notation, we omit the superscript when talking about the current time. We assume that there exists a *bounding region* in the form of a Euclidean ball $\mathcal{B}_0$ centered at the origin. At all times, the points $Q^{(t)}$ are restricted to lie within $\mathcal{B}_0$. The algorithm has knowledge of this ball. Define the system's *initial aspect ratio* to be $\Lambda_0 = (\text{radius}(\mathcal{B}_0))/(\min_{i\in[n]} N_i^{(0)})$. Given any positive constant c, let $c\mathcal{B}_0$ denote a factor-c expansion of this ball about the origin.

4.3.3 Oracle

Consider the state of the system at any fixed time t. Knowledge about the current state of the system is provided by an entity $\mathcal{O}$, called the *oracle*. It is given a Euclidean ball $\mathcal{B}(\mathbf{x}, r)$ and an object index $i \in [n]$. Recall that for each $i \in [n]$, X_i denotes a random variable distributed as P_i. The oracle is a function $\mathcal{O} : [n] \times \mathbb{R}^d \times \mathbb{R}^+ \to \{Y, N\} \times \{+, -\}$, where $\mathcal{O}(i, \mathbf{x}, r)$ returns:

- "Y" or "N" depending on whether $X_i \in \mathcal{B}(\mathbf{x}, r)$ and

- "+" or "−" depending on whether there exists $j \neq i$, such that $X_j \in \mathcal{B}(\mathbf{x}, r)$

The first element of the pair is used to estimate the location of the object i, and the second is used to estimate the distance to its nearest neighbor. Because X_i and X_j are random variables, so is $\mathcal{O}(i, \mathbf{x}, r)$. Consistent with previous applications of the evolving data framework (see, e.g., [10]), we intentionally made the oracle as weak as possible, implying that our algorithm can be used with stronger oracles.

4.3.4 Hypotheses and Distance

The algorithm maintains a *hypothesis* of the current object locations, which is defined to be a set $\mathcal{H} = \{H_1, \ldots, H_n\}$ of spatial probability distributions in $\mathbb{R}^d$. The *distance* between the truth $\mathcal{P}$ and the current hypothesis $\mathcal{H}$, denoted $\mathcal{D}(\mathcal{P}, \mathcal{H})$, is defined as the sum of n Kullback-Leibler divergences from the hypothesized distributions to the true ones. For $i \in [n]$, let

$$D_i \;=\; D_{\mathrm{KL}}(P_i \parallel H_i) \;=\; \int_{\mathbf{x} \in \mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i(\mathbf{x})} \mu(d\mathbf{x}),$$

where $\mu(\cdot)$ denotes the measure over $\mathcal{B}_i = \mathcal{B}(\mathbf{q}_i, \beta N_i)$ and define

$$\mathcal{D}(\mathcal{P}, \mathcal{H}) \;=\; \sum_{i=1}^{n} D_i \;=\; \sum_{i=1}^{n} D_{\mathrm{KL}}(P_i \parallel H_i). \tag{4.1}$$

As a baseline for comparisons, we introduce a *naïve hypothesis*, denoted $\mathcal{H}^*$, which assumes no information about the locations of the objects, other than the fact that they lie within the bounding region. Recall that $\mathcal{B}_0$ denotes this bounding region and $3\mathcal{B}_0$ denotes a factor-3 expansion about its center. Since all the points of Q lie within $\mathcal{B}_0$ and $0 < \beta < 1$, $3\mathcal{B}_0$ is guaranteed to contain all the β-local feature regions. For all $i \in [n]$, let H_i^* be the uniform distribution over $3\mathcal{B}_0$, and let

$\mathcal{H}^* = \{H_1^*, \ldots, H_n^*\}$. Regardless of the initial truth, the initial distance of the ith object satisfies the following bound.

$$\begin{aligned} D_i^* &= \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i^*(\mathbf{x})} \mu(d\mathbf{x}) &&= \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{1/(\beta N_i)^d}{1/(3\cdot \mathrm{radius}(\mathcal{B}_0))^d} \mu(d\mathbf{x}) \\ &&&= \log\left(\frac{3\cdot \mathrm{radius}(\mathcal{B}_0)}{\beta N_i}\right)^d \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x})\mu(d\mathbf{x}) \\ &&&\leq d\left(\log \Lambda_0 + \log\frac{3}{\beta}\right). \end{aligned}$$

Let $\mathcal{D} = \sum_{i\in[n]} D_i^*$. Under our assumption that d and β are constants, we have $\mathcal{D}^* \in \Theta(n \log \Lambda_0)$. The initial aspect ratio, Λ_0, depends on the initial configuration of the points of Q, and can be arbitrarily high. In the best case, when the points are uniformly distributed in $\mathcal{B}_0$, we have $\Lambda_0 = \Omega(n^{1/d})$ and hence $\mathcal{D} \in \Omega(n \log n)$. The objective of our algorithm is to maintain a significantly smaller bound on this distance. The combination of evolver, oracle, and distance function constitute a model of evolving motion, which we henceforth call the (α, β)-*local-motion model*.

4.3.5 Class of Algorithms

We assume that the algorithm that maintains the hypothesis runs in discrete steps over time. With each step, it may query the oracle a constant number of times, perform a constant amount of work, and then update the current set of hypotheses. The number of oracle queries is independent of n but can depend on dimension d, local feature scale factor β, and motion factor α. In our case, this work takes the form of updating the hypothesis for the object that was queried. In the purest form of the evolving data framework, the evolver and algorithm alternate [8]. Instead, similar to the generalized framework proposed in [3], we assume that there is a fixed *speed-up factor*, denoted σ. When amortized over the

entire run, the ratio of the number of steps taken by the algorithm and the evolver does not exceed σ.

4.3.6 Objective and Results

The objective of the algorithm is as follows. Given any starting ground-truth configuration, after an initial "burn-in" period, the algorithm guarantees that the hypothesis is within a bounded distance of the truth subject to model assumptions and the given speed-up factor. Our main result is that there exists an algorithm with constant speed-up factor σ that maintains a distance of $O(n)$ in steady state.

Theorem 4.1. *Consider a set of n evolving objects in $\mathbb{R}^d$ under the (α, β)-local-motion model, for constants α and β, where $0 < \alpha, \beta < \frac{1}{3}$. There exists an algorithm of constant speed-up σ, and burn-in time $t_0 \in O(n \log \Lambda_0 \log\log \Lambda_0)$ such that for all $t \geq t_0$, this algorithm maintains a distance of $O(n)$.*

The algorithm and its analysis will be proved in section 4.5. Given that no algorithm can guarantee an exact match between hypothesis and truth, it is natural to wonder how close this is to optimal. In section 4.4, we will show that it is asymptotically optimal by showing that for any algorithm and any constant speed-up factor, there exists an evolver that can force a distance of $\Omega(n)$ in steady state.

4.3.7 Why the KL Divergence?

The principal challenge in generalizing the evolving framework from simple 1-dimensional applications like sorting to a multidimensional setting is the lack of an obvious distance measure that captures how close the hypothesis is to the current state. Our approach is motivated by an information-theoretic perspective. The KL divergence $D_i = D_{\text{KL}}(P_i \parallel H_i)$ serves as a measure of how different the actual object distribution P_i is from our hypothesis H_i. (Note that the KL divergence is asymmetric,

which is to be expected, given the asymmetric roles of the truth and our hypothesis.)

As an example of this information-theoretic approach, consider the following application in coding theory and space quantization [30]. The objective is to communicate the location of an imprecise point X_i with any arbitrary resolution δ. From Shannon's source coding theorem [66], the theoretical lower bound for the expected number of bits required for communication is given by the *Shannon entropy* [85]

$$b_{P_i} = \sum_{C_\delta \in B_i} P_i(C_\delta) \log \frac{1}{P_i(C_\delta)},$$

where C_δ is a cell (a d-dimensional box) of size δ in $\mathbb{R}^d$, and $P(C)$, by a slight abuse of notation, is the probability associated with a cell C using the underlying probability distribution function P. This is roughly achieved by a Huffman coding [49] of the space around q_i, which intuitively assigns a number of bits proportional to the log of the inverse of the probability measure associated with an event, which in our case is X_i lying within a cell C_δ of size δ. Since the P_i's are not available to us, we use a similar strategy of encoding the space using H_i. Therefore, in expectation, we use roughly

$$b_{H_i} = \sum_{C_\delta \in B_i} P_i(C_\delta) \log \frac{1}{H_i(C_\delta)}$$

bits. When δ is arbitrarily small, the difference $(b_{H_i} - b_{P_i}) \sim D_i$. Therefore, $\mathcal{D}(\mathcal{P}, \mathcal{H}) = \sum_{i \in [n]} D_i$ roughly represents the expected number of additional bits needed to represent the location of X_i's individually using just H_i's, compared to the absolute theoretical limit.

In the next section, we present Theorem 4.2, which provides a lower bound on the distance achievable by any algorithm in our model. In section 4.5, we present the algorithm and analyze its steady-state performance. In section 4.7, we discuss possible relaxations to the assumptions made in

our model.

4.4 Lower Bound

In this section, we show that maintaining $\mathcal{D}(\mathcal{P}, \mathcal{H}) \in O(n)$ is the best we can hope to achieve. The proof follows a similar structure to other lower-bound proofs in the evolving data framework [3, 10,94]. Intuitively, over a period of time of length cn, for $c < 1$, the algorithm can inspect the locations of only a constant fraction of points. During this time, the evolver can move a sufficient number of uninspected points so that the overall distance increases by $\Omega(n)$. We formalize this in the following theorem.

Theorem 4.2. *For any algorithm $\mathcal{A}$, there exists a starting configuration $Q^{(0)}$ and an evolver (with knowledge of $\mathcal{A}$) in the local-motion model such that, for any positive integer t_0, there exists $t \geq t_0$, such that $\mathcal{D}^{(t)} = \mathcal{D}(\mathcal{P}^{(t)}, \mathcal{H}^{(t)}) \in \Omega(n)$.*

Proof. We construct a point set on the real line ($\mathbb{R}$), and we will mention at the end how to adapt the proof to any dimension d. Given the small constant β, the local feature constant, we define $Q^{(0)}$ to be the following set of $n = 2m$ points in $\mathbb{R}$.

$$Q^{(0)} = \bigcup_{i \in [m]} \left\{ a_i^{(0)} = 100i \right\} \cup \bigcup_{i \in [m]} \left\{ b_i^{(0)} = 100i + 1 \right\}.$$

Let $\mathcal{D}_{a,i} = \mathcal{D}_{2(k-1)+1},\ k \in [m]$, the distance corresponding to the first point in the tuple. We similarly define $P_{a,i}$ and $H_{a,i}$. Let N_i denote the current distance to a_i's nearest neighbor. The local feature interval for a_i, denoted I_i, is $100i + \beta N_i[-1, 1]$. Recall that $P_{a,i}$ is the uniform probability over I_i.

Therefore, $P_{a,i}(\mathbf{x}) = 1/|\mathcal{I}_i|$, where $|\mathcal{I}_i| = 1/\beta N_i$ is the diameter of $\mathcal{I}_i$. Now

$$\begin{aligned}\mathcal{D}_{a,i}^{(t_0)} &= \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log \frac{P_{a,i}^{(t_0)}(\mathbf{x})}{H_{a,i}^{(t_0)}(\mathbf{x})} d\mathbf{x} \\ &= \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \; - \; \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) \log H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}.\end{aligned}$$

By the concavity of the log function and Jensen's inequality, we have

$$\begin{aligned}\mathcal{D}_{a,i}^{(t_0)} &\geq -\log|\mathcal{I}_i| \int_{\mathbf{x}\in\mathcal{I}_i} P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \; - \; \log \int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) P_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x} \\ &= -\log|\mathcal{I}_i| - \log \frac{\int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}}{|\mathcal{I}_i|} \; = \; -\log \int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x}) d\mathbf{x}. \qquad (4.2)\end{aligned}$$

We may assume that $\mathcal{D}^{(t_0)} \in o(n)$ (for otherwise we can set $t = t_0$ and are done). This implies there are only $o(n)$ many a_i's such that $\mathcal{D}_{a,i}^{(t_0)} \in \Omega(1)$. Call the remaining $n/2 - o(n)$, a_i's the proximal set, denoted T_a. We are now ready to describe the evolver's actions. It chooses to stay dormant until time t_0. For a large enough constant M, consider the time interval $[t_0, t_0 + n/M]$. The algorithm $\mathcal{A}$, running at a speed-up factor of σ, can modify at most $\sigma n/M$ hypotheses in the time interval. The evolver chooses not to move any of those points. Therefore $\mathcal{A}$ can only reduce $\mathcal{D}^{(t_0)}$ by $o(n)$ over this time interval. Now there are at least $n/2 - \sigma n/M - o(n)$ members in the proximal set T_a, whose hypotheses were not altered by $\mathcal{A}$. Call this set the stable set, denoted S_a.

For $\kappa = \lceil \log(2/(1+\alpha)) \rceil$, the evolver selects some $n/(\kappa M)$ members from S_a. Call that set S'_a. To be specific, for all $a_i \in S'_a$, the evolver chooses to move a_i away from b_i some κ times, by a distance of exactly $\alpha N_{a,i}$, where $N_{a,i}$ is the distance of a_i from its current nearest neighbor. (Note that the nearest neighbor will remain b_i throughout these operations.) Given this value of κ, after the conclusion of these operations, the distance between a_i and b_i is at least 2. For $\beta < 1/3$, the local

feature of a_i changes to an interval $\mathcal{I}'_i$, where $\mathcal{I}'_i \cap \mathcal{I}_i = \emptyset$. Now, for $t = t_0 + n/M$, using a similar analysis as Eq. (4.2), we have

$$\mathcal{D}_{a,i}^{(t)} \;\geq\; -\log \int_{\mathbf{x}\in\mathcal{I}'_i} H_{a,i}^{(t)}(\mathbf{x})d\mathbf{x}. \tag{4.3}$$

Thus, for all $a_i \in S'_a$, we have $H_{a,i}^{(t)} = H_{a,i}^{(t_0)}$ and $D_{a,i}^{(t_0)} = o(1)$. Therefore,

$$\int_{\mathbf{x}\in\mathcal{I}_i} H_{a,i}^{(t_0)}(\mathbf{x})d\mathbf{x} \;\geq\; e^{-o(1)}.$$

If $\mathcal{D}_{a,i}^{(t)} \in o(1)$ as well, then similarly from Eq. (4.3), we have

$$\int_{\mathbf{x}\in\mathcal{I}'_i} H_{a,i}^{(t)}(\mathbf{x})d\mathbf{x} \;\geq\; e^{-o(1)}.$$

But, this yields a contradiction since $\mathcal{I}'_i \cap \mathcal{I}_i = \emptyset$ and $H_{a,i}^{(t)} = H_{a,i}^{(t_0)}$ implies that

$$\int_{\mathbf{x}\in\mathcal{I}'_i\cup\mathcal{I}_i} H_{a,i}^{(t)}(\mathbf{x})d\mathbf{x} \;\geq\; 2e^{-o(1)} \;>\; 1.$$

Therefore, for $a_i \in S'_a$, $\mathcal{D}_{a,i}^{(t)} \in \Omega(1)$, and since $|S'_a| = \Omega(n)$, we have $\mathcal{D}^{(t)} \in \Omega(n)$. For a general dimension d, we define Q, in the same way except all the points lie on a single axis. The evolver also moves these points along that particular axis. The rest of the analysis involves integration over a region of space rather an interval, but is straightforward and carries over to $\mathbb{R}^d$. □

4.5 Algorithm

In this section, we present our algorithm and analyze its performance. We begin by defining our hypothesis. For every index i, we define two parameters: $h_i \in \mathbb{R}$, and $\mathbf{k}_i = (k_{i,1}, \ldots k_{i,d}) \in \mathbb{R}^d$.

For $\mathbf{x} = (x_1, \ldots x_d) \in \mathbb{R}^d$ we set the hypothesis probability density for the ith point to be the d-fold product of independent Cauchy distributions, one per coordinate, with center at $\mathbf{k}_i$ and scaling parameter h_i [91]. That is,

$$H_i(\mathbf{x}) \;=\; f_{h_i,\mathbf{k}_i}(\mathbf{x}) \;=\; \frac{1}{\pi^d h_i^d} \prod_{j=1}^{d} \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)^{-1} \qquad \text{(4.4: Hypothesis Def.)}$$

(Note that this differs from the standard multivariate Cauchy [91].) If we let $f_{1,\mathbf{0}}(\mathbf{x})$ denote the probability density function for the standard d-fold product of Cauchy distributions (centered at the origin unit scale), we can express this equivalently as

$$f_{h_i,\mathbf{k}_i}(\mathbf{x}) \;=\; \frac{1}{h_i^d} f_{1,\mathbf{0}}\left(\frac{\mathbf{x} - \mathbf{k}_i}{h_i}\right), \quad \text{where} \quad f_{1,\mathbf{0}}(\mathbf{x}) \;=\; \frac{1}{\pi^d} \prod_{j=1}^{d} \left(1 + x_j^2\right)^{-1}.$$

It will be convenient to define the *hypothesis ball* $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$, which we use to illustrate H_i. Note that, unlike our truth probabilities P_i, our hypothesis distributions have unbounded support. Our algorithm modifies the parameters $\mathbf{k}_i$ and h_i in response to information received from the oracle.

Remark 4.2. As mentioned above, the truth P_i's have a bounded support and H_i's are defined over all of $\mathbb{R}^d$. This is in stark contrast to the purest form of the evolving framework [8], where the hypothesis and the truth have identical structures. We relax the framework and note that there is no reason for both of them to have the same structure, as long as we have a notion of distance between them.

4.5.1 Potential Function

Due to the subtleties of tracking the Kullback-Leibler divergence, we introduce a potential function Φ to aid in the analysis. For each object index $i \in [n]$, we define an individual potential function

Φ_i, which bounds the distance D_i to within a constant. The total potential Φ will be the sum of these functions. Let $s_i = \|\mathbf{q}_i - \mathbf{k}_i\|$ denote the Euclidean distance between the center of the actual distribution and the center of the hypothesis ball. Let $l_i = \beta N_i$ denote the radius of the local feature of $\mathbf{q}_i$. Recall that h_i is the radius of the hypothesis ball (see fig. 4.2). We define the individual and system *potentials* to be

$$\Phi_i = \Phi(P_i, H_i) = \log\left(\frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right) \quad \text{and} \quad \Phi = \Phi(\mathcal{P}, \mathcal{H}) = \sum_{i\in[n]} \Phi_i \tag{4.5}$$

The following lemma shows that, up to additive and scalar constants that depend on the dimension, distances can be bounded above by this potential.

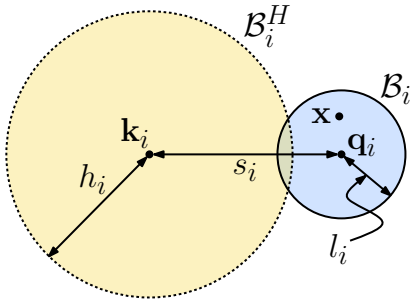


Figure 4.2: The definition of the individual potential Φ_i.

Lemma 4.1. *There exists a constant c_d (depending on dimension) such that for any $i \in [n]$, $D_i \leq c_d(1 + \Phi_i)$, and hence $\mathcal{D} \leq c_d(n + \Phi)$.*

Proof. Recall that $P_i(\mathbf{x})$ is the probability density function of a uniform distribution over a ball of radius l_i. Thus, for $\mathbf{x} \in \mathcal{B}_i$, $P_i(\mathbf{x}) = \omega_d / l_i^d$, where ω_d denotes the volume of the unit Euclidean ball in $\mathbb{R}^d$, and $\mathcal{B}_i$. Both D_i and Φ_i are scale-invariant, which implies that we may assume that space has been scaled so that both h_i and l_i are less than 1. By the definition of the multivariate Cauchy distribution,

we have

$$
\begin{aligned}
D_i \;&=\; \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i(\mathbf{x})} \mu(d\mathbf{x}) \\
&=\; \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{\omega_d / l_i^d}{(\pi^d h_i^d)^{-1} \prod_{j=1}^d \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)^{-1}} \mu(d\mathbf{x}) \\
&=\; \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \left(\omega_d \pi^d\right) \mu(d\mathbf{x}) \;+\; \int_{\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{h_i^d \prod_{j=1}^d \left(1 + \frac{(x_j - k_{i,j})^2}{h_i^2}\right)}{l_i^d} \mu(d\mathbf{x}). \qquad (4.6)
\end{aligned}
$$

For $\mathbf{x} \in \mathcal{B}_i$ and $j \in [d]$, by the triangle inequality, we have

$$
|x_j - k_{i,j}| \;\le\; \|\mathbf{x} - \mathbf{k}_i\| \;\le\; \|\mathbf{x} - \mathbf{q}_i\| + \|\mathbf{q}_i - \mathbf{k}_i\| \;\le\; l_i + s_i.
$$

Therefore,

$$
\begin{aligned}
D_i \;&\le\; \log \left(\omega_d \pi^d\right) \;+\; \int_{\mathbf{x} \in \mathcal{B}_i} P_i(\mathbf{x}) \log \left[\frac{h_i^d}{l_i^d} \prod_{j=1}^d \left(1 + \frac{(s_i + l_i)^2}{h_i^2}\right)\right] \mu(d\mathbf{x}) \\
&\le\; \log \left(\omega_d \pi^d\right) \;+\; \log \left[\frac{h_i^d}{l_i^d} \left(1 + \frac{(s_i + l_i)^2}{h_i^2}\right)^d\right] \int_{\mathbf{x} \in \mathcal{B}_i} P_i(\mathbf{x}) \mu(d\mathbf{x}) \\
&=\; \log \left(\omega_d \, \pi^d\right) \;+\; d \log \left(\frac{h_i^2 + (s_i + l_i)^2}{l_i \, h_i}\right) \;\le\; c_d \left(1 + \log \frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right),
\end{aligned}
$$

for a suitably chosen c_d that depends only on the dimension. □

Remark 4.3 (Potential function and approximating pairwise distances). The potential function Φ_i we define here is symmetric. It is remarkable that a symmetric function bounds an asymmetric function D_i under the choice of Cauchy distributions as hypotheses. However, that is not the goal of this chapter, as we have strict asymmetry between the hypothesis, and the truth (we use the former to maintain a

sketch of the latter). We intend to explore this potential function in the future, and find how it relates to approximating interesting Euclidean geometric structures. Nonetheless, we briefly mention here the consequences of bounding each Φ_i to within a constant. To demonstrate the value of our potential function, suppose that for $\forall i, \Phi_i < \log c$, for some constant c. This implies that $h_i < c\sqrt{l_i h_i}$, which further implies $h_i < c' l_i$ for some constant c'. Similarly, $s_i < c\sqrt{l_i h_i} < c'' l_i$ for some constant c''. By the triangle inequality, the distance between the centers of two hypothesis balls satisfies

$$\|\mathbf{k}_i - \mathbf{k}_j\| \;\leq\; s_i + s_j + \|\mathbf{q}_i - \mathbf{q}_j\| \;\leq\; c'' l_i + c'' l_j + \|\mathbf{q}_i - \mathbf{q}_j\|.$$

Since $l_i = N_i/\beta \leq \|\mathbf{q}_i - \mathbf{q}_j\|/\beta$, we have $\|\mathbf{k}_i - \mathbf{k}_j\| \leq c^* \|\mathbf{q}_i - \mathbf{q}_j\|$, for some constant c^*. Since $\|\mathbf{k}_i - \mathbf{k}_j\|$ is a constant approximation for $\|\mathbf{q}_i - \mathbf{q}_j\|$, it immediately follows that we can maintain a constant-weight approximation of structures like the Euclidean minimum spanning tree and the Euclidean traveling salesman tour on Q by constructing the same on $\mathbf{k}_i$'s, the centers of the hypotheses H_i's. □

An important feature of our choice of a potential function is that the evolver cannot change its value by more than a constant with each step. Recall that the evolver selects an object index i and moves q_i by a distance of most αN_i in any direction. This results in at most an α fraction change in the values of l_i and s_i, and hence the resulting change in Φ_i is bounded by a constant. However, note that the movement of q_i could potentially affect the local feature sizes of a number of other objects in the system. We can show by a packing argument that there is only a constant number of indices j, whose Φ_j value is affected. Furthermore, these values are only changed by a constant. This is encapsulated in the following lemma.

Lemma 4.2. *Each step of the evolver increases the potential Φ by at most a constant.*

Proof. Let's say the evolver chooses to move $\mathbf{q}_i$ at time 0. Now $l_i^{(0)}/\beta$ is the nearest neighbor distance of $\mathbf{q}_i$ at time 0. Therefore $\mathbf{q}_i$ moves by at most $(\alpha/\beta)l_i^{(0)}$ distance, implying $s_i^{(1)} \leq s_i^{(0)} + (\alpha/\beta)l_i^{(0)} \leq (1+\alpha/\beta)\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)$. Similarly $(1-\alpha)l_i^{(0)} \leq l_i^{(1)} \leq (1+\alpha)l_i^{(0)}$. Therefore,

$$\begin{aligned}
\Phi_i^{(1)} - \Phi_i^{(0)} &\leq \log \frac{\max\left(s_i^{(1)}, l_i^{(1)}, h_i^{(1)}\right)}{\sqrt{l_i^{(1)} h_i^{(1)}}} - \log \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&\leq \log \frac{\max\left(s_i^{(0)} + (\alpha/\beta)l_i^{(0)}, (1+\alpha)l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{(1-\alpha)l_i^{(0)} h_i^{(0)}}} - \log \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&\leq \log \left(\frac{1+\alpha/\beta}{\sqrt{1-\alpha}} \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \right) - \log \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&= \log \frac{1+\alpha/\beta}{\sqrt{1-\alpha}} \in O(1)
\end{aligned} \tag{4.7}$$

Let $\mathcal{N}_i$ be the set of indices of points whose nearest neighbor at time 0 was $\mathbf{q}_i^{(0)}$, or whose nearest neighbor at time 1 was $\mathbf{q}_i^{(1)}$. Only points with indices in this set have their potential function changed. If $j \in \mathcal{N}_i$, $s_j^{(0)}$, and $h_j^{(0)}$ do not change. Since $\mathbf{q}_i$ moves by at most $(\alpha/\beta)l_i^{(0)}$ distance, the nearest neighbor distance $N_j = l_j/\beta$ changes by at most $(\alpha/\beta)l_i^{(0)}$ as well. Therefore,

$$-(\alpha/\beta)l_i^{(0)} \leq N_j^{(1)} - N_j^{(0)} \leq (\alpha/\beta)l_i^{(0)} \implies -\alpha l_i^{(0)} \leq l_j^{(1)} - l_j^{(0)} \leq \alpha l_i^{(0)} \tag{4.8}$$

Since $(1-\alpha)l_i^{(0)} \leq l_i^{(1)}$, we also have

$$-\frac{\alpha}{1-\alpha} l_i^{(1)} \leq l_j^{(1)} - l_j^{(0)} \leq \frac{\alpha}{1-\alpha} l_i^{(1)} \tag{4.9}$$

If $\mathbf{q}_i^{(0)}$ was the nearest neighbor of $\mathbf{q}_j^{(0)}$, then $N_j^{(0)} \leq N_i^{(0)}$, and hence $l_j^{(0)} \leq l_i^{(0)}$. Using the last

fact, and dividing Eq. (4.8) by $l_j^{(0)}$, we have $(1-\alpha) \leq l_j^{(1)}/l_j^{(0)} \leq (1+\alpha)$. Similarly, If $\mathbf{q}_i^{(1)}$ was the nearest neighbor of $\mathbf{q}_j^{(1)}$, then using Eq. (4.9) we have $(1-\alpha/(1-\alpha)) \leq l_j^{(1)}/l_j^{(0)} \leq (1+\alpha/(1-\alpha))$. Therefore $l_j^{(1)}/l_j^{(0)} \in \Theta(1)$ for $j \in \mathcal{N}_i$. And hence,

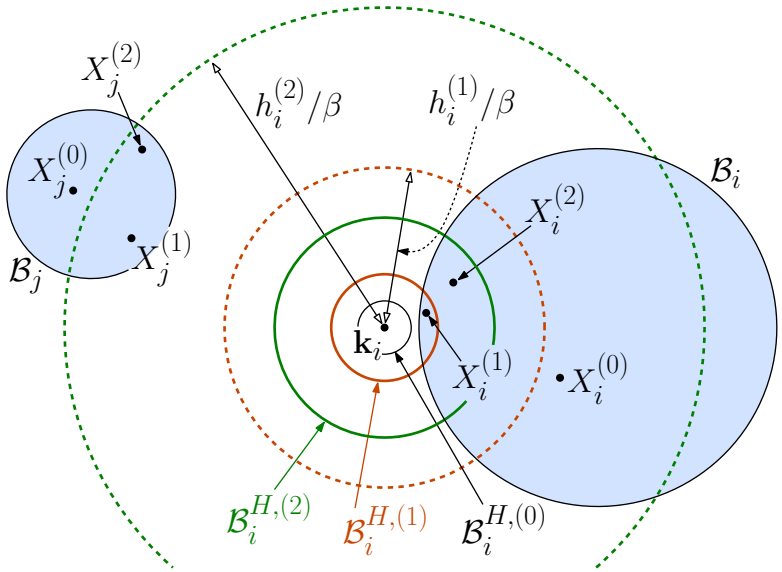
$$\begin{aligned}
\Phi_j^{(1)} - \Phi_j^{(0)} &= \log \frac{\max\left(s_i^{(1)}, l_i^{(1)}, h_i^{(1)}\right)}{\sqrt{l_i^{(1)} h_i^{(1)}}} - \log \frac{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&= \log \frac{\max\left(s_i^{(0)}, l_i^{(1)}, h_i^{(0)}\right)}{\max\left(s_i^{(0)}, l_i^{(0)}, h_i^{(0)}\right)} - \log \frac{\sqrt{l_i^{(1)} h_i^{(0)}}}{\sqrt{l_i^{(0)} h_i^{(0)}}} \\
&\leq \left|\log \frac{l_i^{(1)}}{l_i^{(0)}}\right| + \left|\log \frac{l_i^{(0)}}{l_i^{(1)}}\right| \in O(1), \quad \forall\, j \in \mathcal{N}_i
\end{aligned} \tag{4.10}$$

Finally, we show that $|\mathcal{N}_i| \in O(1)$. To that effect we solve a general problem: What is the maximum number of points in Q, whose nearest neighbor is $\mathbf{q}_1$? Without loss of generality, let $\mathbf{q}_2$ be the nearest neighbor of $\mathbf{q}_1$, such that $\|\mathbf{q}_2 - \mathbf{q}_1\| = 1$. Let $\mathbf{q}_3$'s nearest neighbor be $\mathbf{q}_1$. Consider the line segment $\overline{\mathbf{q}_1\mathbf{q}_3}$, and its intersection with $\mathcal{B}(\mathbf{q}_1, 1)$ the ball centered at $\mathbf{q}_1$, and passing through $\mathbf{q}_2$. Call the intersection point $\mathbf{q}_3'$. For any general point $\mathbf{q}_x \in Q$, we similarly define $\mathbf{q}_x'$. Note $\mathbf{q}_2' = \mathbf{q}_2$. By triangle inequality, we have $\|\mathbf{q}_3' - \mathbf{q}_2'\| \geq \|\mathbf{q}_3 - \mathbf{q}_2\| - \|\mathbf{q}_3 - \mathbf{q}_3'\| = \|\mathbf{q}_3 - \mathbf{q}_2\| - \|\mathbf{q}_3 - \mathbf{q}_1\| + \|\mathbf{q}_1 - \mathbf{q}_3'\|$. Since $\mathbf{q}_1$ is the nearest neighbor of $\mathbf{q}_3$, we have $\|\mathbf{q}_3 - \mathbf{q}_2\| \geq \|\mathbf{q}_3 - \mathbf{q}_1\|$. Therefore $\|\mathbf{q}_3' - \mathbf{q}_2'\| \geq \|\mathbf{q}_1 - \mathbf{q}_3'\| = 1$. Therefore, for any $x, y \in [n], x \neq y$, such that $\mathbf{q}_1$ is the nearest neighbor of $\mathbf{q}_x$ and $\mathbf{q}_y$, we have $\|\mathbf{q}_x' - \mathbf{q}_y'\| \geq 1$. Let $\mathcal{M}_1 = \{\mathbf{q}_x' \mid \mathbf{q}_1 \text{ is the nearest neighbor of } \mathbf{q}_x\}$. Draw a ball of radius 1/2 around $\mathbf{q}_1$, and every member of $\mathcal{M}_1$. Each of these balls are non-overlapping. However all of them are contained within a ball around $\mathbf{q}_1$ of radius $3/2$. Therefore $|\mathcal{M}_1| < (3/2)^d/(1/2)^d = 3^d \in O(1)$. Observing that $\mathcal{N}_i \leq 2\mathcal{M}_1$, and using Eq. (4.7), and 4.10 we have the result. □

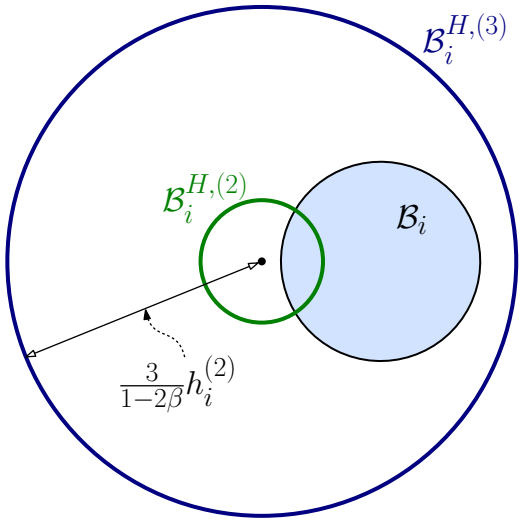
4.5.2 The Algorithm

Next, we present our algorithm. The algorithm runs in parallel to the evolver, running faster by a speed-up factor of σ (whose value will be derived in our analysis). Each *step* of the algorithm involves a probe of an object by the oracle, and following that, a possible modification of a hypothesis, H_i. The total running time of the algorithm is proportional to the number of steps. Let us begin with a high-level explanation. The details are provided in algorithm TRACKBYZOOM. Recall that we are given a set of n objects, each represented by X_i, a uniform probability distribution over a ball $\mathcal{B}_i = \mathcal{B}(\mathbf{q}_i, \beta N_i)$, where N_i is the distance to its nearest neighbor and β is the local feature scale. The points $\mathbf{q}_i$ are restricted to lie within a bounding ball $\mathcal{B}_0$ centered at the origin. The algorithm maintains a hypothesis in the form of n Cauchy distributions, each represented by a hypothesis ball $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$, where $\mathbf{k}_i$ is the center of the distribution and h_i is the distribution's scaling parameter. The algorithm begins with an initial hypothesis, where each hypothesis ball is just $\mathcal{B}_0$. This reflects the fact that we effectively assume nothing about the location of $\mathbf{q}_i$, except that it lies within the bounding ball. The algorithm then proceeds in a series of *iterations*, where each iteration handles all the n indices in order. The handling of each index involves two processes, called *zoom-out* and *zoom-in*.

In the zoom-out process, we query the oracle on index i to check whether (1) the sampled point X_i lies within its hypothesis ball and (2) at least one other X_j lies within a concentric ball whose radius is larger by a factor of $\frac{1}{\beta}$. If so, we expand the hypothesis ball by a factor of $3/(1-2\beta)$. (We show in Lemma 4.3 that this guarantees that the hypothesis ball $\mathcal{B}_i^H$ now contains the local feature ball $\mathcal{B}_i$.) We then proceed to the zoom-in process. If not, we double the hypothesis ball and repeat (see fig. 4.3).



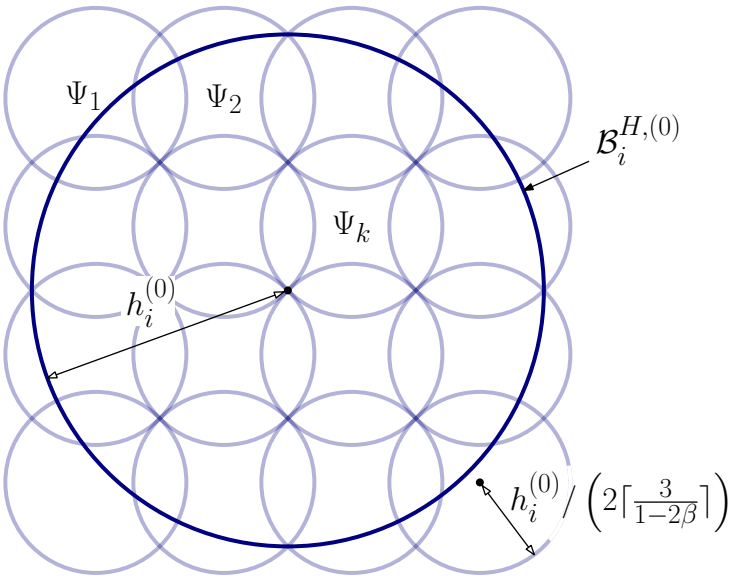
(a) Multiple stages of the zoom-out process starting at $t = 0$. Stage $t = 2$ is the first time $\mathcal{B}_i^{H,(t)}$ contains X_i, and its $1/\beta$-expansion contains X_j.



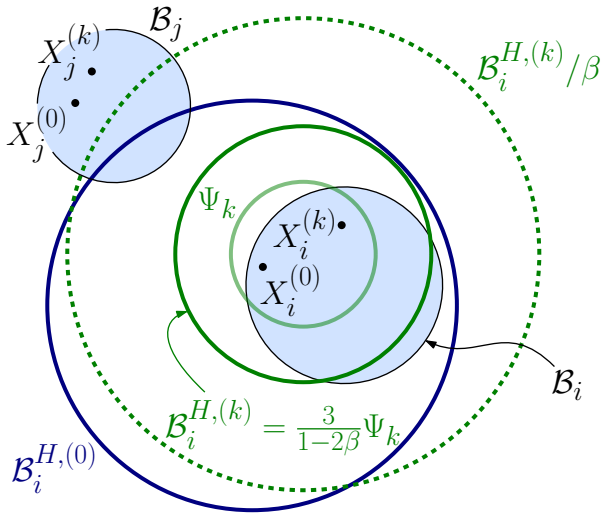
(b) $\mathcal{B}_i^{H,(3)}$ is the hypothesis ball at the end of the zoom-out process. Note that $\mathcal{B}_i \subset \mathcal{B}_i^{H,(3)}$.

Figure 4.3: The zoom-out process of TRACKBYZOOM (Line 6).

In the zoom-in process, we first check whether X_i lies within the hypothesis ball. (The evolver could have moved q_i.) If not, we return to the zoom-out process. If so, we next check the $\frac{1}{\beta}$-expansion of this ball. If there is no other X_j in this expanded ball, we accept the current hypothesis for i, and go on to the next point in the set. (We show in lemma 4.4 that Φ_i at this point in time is bounded by a constant). If there is such an X_j however, we may need to shrink the hypothesis ball for the current index. We cover the hypothesis ball with a collection of $O(1)$ balls whose radii are smaller by a constant factor, and we test whether X_i lies within any of them (see fig. 4.4). As soon as we find one, we shrink the hypothesis ball about this ball. We repeat this process until one of the earlier conditions applies (causing us to return to the zoom-out process or move on to the next point).



(a) Set of nested balls covering $\mathcal{B}_i^{H,(0)}$, which we call Ψ. $\text{radius}(\Psi_i) = \text{radius}(\mathcal{B}_i) / \left(2\lceil \frac{3}{1-2\beta} \rceil\right)$, and $|\Psi| \in O(1)$.



(b) Ψ_k is the first nested ball to contain X_i. $\mathcal{B}_i^{H,(k)}$ contains X_i, and its $1/\beta$-expansion does not contain X_j. TRACKBYZOOM moves on to index $i+1$.

Figure 4.4: Illustration of the zoom-in process in TRACKBYZOOM (Line 14).

4.5.3 Detailed Algorithm

Algorithm TRACKBYZOOM — Tracking Evolving Distributions

procedure TRACKBYZOOM($\mathcal{O}, n, \beta, \mathcal{B}_0$)

▷ Maintain hypothesis $\mathcal{H}$ by running the procedure at a speed-up factor σ, given the membership Oracle:$\mathcal{O}$, the input size:n, the local feature parameter:β, and the the maximum bounding ball centered at origin:$\mathcal{B}_0$

for $i \leftarrow 1$ to n **do**

▷ Initially set the hypotheses according to the given bounding ball

$h_i \leftarrow \text{radius}(\mathcal{B}_0)$, $\mathbf{k}_i \leftarrow \mathbf{0}$ ▷ Set $H_i(\mathbf{x}) \leftarrow f_{\mathbf{0},R_0}(\mathbf{x})$, as in Eq.(4.4: Hypothesis Def.)

end for

$i \leftarrow 1$ ▷ Start with the first point

ZOOM_OUT:

$\mathscr{O}_i \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i)$, $\mathscr{O}_i^* \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i/\beta)$ ▷ Store values to use later

▷ If $\mathcal{B}_i^H$ (hypothesis ball) contains X_i, and its $1/\beta$-factor expansion contains some X_j

if $\mathscr{O}_i = (Y, \cdot)$ **and** $\mathscr{O}_i^* = (Y, +)$ **then**

$h_i \leftarrow \frac{3}{1-2\beta} h_i$ ▷ Update $\mathcal{B}_i^H$ so that it contains $\mathcal{B}_i$ (local feature of $\mathbf{q}_i$)

go to ZOOM_IN

end if

$h_i \leftarrow 2h_i$ ▷ Zoom out by expanding $\mathcal{B}_i^H$

▷ Zoom out until $X_i \in \mathcal{B}_i^H$, and its $1/\beta$-expansion contains another X_j

go to ZOOM_OUT

Algorithm TRACKBYZOOM continued:

ZOOM_IN:

$\mathscr{O}_i \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i)$, $\mathscr{O}_i^* \leftarrow \mathcal{O}(i, \mathbf{k}_i, h_i/\beta)$ ▷ Store values to use later

if $\mathscr{O}_i = (N, \cdot)$ **then** ▷ X_i no longer in $\mathcal{B}_i^H$
  go to ZOOM_OUT
end if

▷ $X_i \in \mathcal{B}_i^H$, and its $1/\beta$-expansion doesn't contain another X_j
if $\mathscr{O}_i = (Y, \cdot)$ **and** $\mathscr{O}_i^* = (Y, -)$ **then**

  $i \leftarrow 1 + i \mod n$ ▷ Φ_i is a constant, ready to move to the next point

  go to ZOOM_OUT
end if

▷ If $\mathcal{B}_i^H$ (hypothesis ball) contains X_i, and its $1/\beta$-expansion contains some X_j
if $\mathscr{O}_i = (Y, \cdot)$ **and** $\mathscr{O}_i^* = (Y, +)$ **then**

  ▷ Ψ is the set of balls of radius $h_i / \left(2\lceil \frac{3}{1-2\beta} \rceil\right)$ which cover $\mathcal{B}_i^H$

  for $\mathcal{B}(\mathbf{x}, r) \in \Psi(\mathcal{B}_i^H, 2\lceil \frac{3}{1-2\beta} \rceil)$ **do**

    if $\mathcal{O}(i, \mathbf{x}, r) = (Y, \cdot)$ **then** ▷ Found the nested ball that contains X_i

      $\mathbf{k}_i \leftarrow \mathbf{x}$, $h_i \leftarrow \frac{3}{1-2\beta} r$ ▷ Expand the nested ball so that it contains B_i

    end if

  end for
end if

▷ Zoom in until $X_i \in \mathcal{B}_i^H$, and its $1/\beta$ expansion contains no other X_j
go to ZOOM_IN

end procedure

4.6 Analysis

In this section, we analyze the distance that the algorithm maintains with the ground truth. Recall that the initial hypotheses are based on the bounding ball $\mathcal{B}_0$, which is centered at the origin. Letting $R_0 = \text{radius}(\mathcal{B}_0)$, we have $H_i^{(0)}(\mathbf{x}) = f_{\mathbf{0},R_0}(\mathbf{x})$, as defined in Eq. (4.4: Hypothesis Def.). Using Eq. (4.5), the initial potential function satisfies:

$$\Phi_i^{(0)} = \log \sqrt{R_0/l_i^{(0)}}, \quad \text{and} \quad \Phi^{(0)} \in O(n \log \Lambda_0) \qquad (\Lambda_0\text{:Aspect Ratio}) \tag{4.11}$$

4.6.1 Two technical lemmas

Suppose the condition in line 8, and 23 are satisfied. We prove the following lemma to justify the expansion factor $\frac{3}{1-2\beta}$.

Lemma 4.3 (Expansion Factor)**.** *If* $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$ *and* $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, +)$*, then* $\mathcal{B}_i \subseteq \mathcal{B}\left(\mathbf{k}_i, \frac{3}{1-2\beta} h_i\right)$.

Proof. The fact that $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$ implies that $\mathcal{B}_i$ and $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$ intersect. Since $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, +)$, there exists $j \neq i$ such that $\|X_j - \mathbf{k}_i\| \leq h_i/\beta$ (see Figure 4.5). Since X_j is sampled from $\mathcal{B}_i = \mathcal{B}(\mathbf{q}_j, l_j)$, we have $\|X_j - \mathbf{q}_j\| \leq l_j$. By the definition of l_j and the triangle inequality, we have

$$l_j = \beta N_j \leq \beta \|\mathbf{q}_i - \mathbf{q}_j\| \leq \beta(\|\mathbf{q}_i - X_j\| + \|X_j - \mathbf{q}_j\|) \leq \beta(\|\mathbf{q}_i - X_j\| + l_j),$$

which implies that $\|X_j - \mathbf{q}_i\| \geq (1/\beta - 1) l_j$.

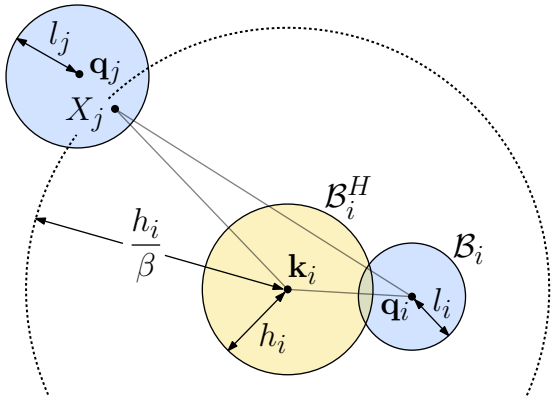


Figure 4.5: Proof of Lemma 4.3.

We assert that $\|X_j - \mathbf{q}_i\| \geq (1/\beta - 1)l_i$. To see why, first if $l_j \geq l_i$, then Thus,

$$\|X_j - \mathbf{q}_i\| \;\geq\; \left(\frac{1}{\beta} - 1\right) l_j \;\geq\; \left(\frac{1}{\beta} - 1\right) l_i.$$

On the other hand, if $l_i > l_j$, then by the triangle inequality, and the fact that $\|\mathbf{q}_i - \mathbf{q}_j\| \geq N_i = l_i/\beta$. we have

$$\|X_j - \mathbf{q}_i\| \;\geq\; \|\mathbf{q}_i - \mathbf{q}_j\| - \|X_j - \mathbf{q}_j\| \;\geq\; \frac{l_i}{\beta} - l_j \;>\; \left(\frac{1}{\beta} - 1\right) l_i.$$

By this assertion, the triangle inequality, and our earlier observations, we have,

$$\begin{aligned} l_i \left(\frac{1}{\beta} - 1\right) \;&\leq\; \|X_j - \mathbf{q}_i\| \;\leq\; \|X_j - \mathbf{k}_i\| + \|\mathbf{k}_i - \mathbf{q}_i\| \;\leq\; h_i/\beta + (h_i + l_i) \\ &\leq\; \left(\frac{1}{\beta} + 1\right) h_i + l_i, \end{aligned}$$

which implies that $l_i(1 - 2\beta) \leq (1 + \beta)h_i$, and hence $h_i + 2l_i \leq \frac{3}{1-2\beta} h_i$. Because $\mathcal{B}_i$ and $\mathcal{B}(\mathbf{k}_i, h_i)$ intersect, we have

$$\mathcal{B}_i \;\subseteq\; \mathcal{B}(\mathbf{k}_i, h_i + 2l_i) \;\subseteq\; \mathcal{B}\left(\mathbf{k}_i, \frac{3}{1-2\beta} h_i\right),$$

as desired. □

The next lemma applies whenever algorithm TRACKBYZOOM has completed its processing of an object on Line 20. It shows that the potential value for this object does not exceed a constant.

Lemma 4.4 (Done tracking X_i)**.** *There exists a constant ϕ_0, such that whenever algorithm* TRACKBYZOOM *completes its processing of any object i, $\Phi_i \leq \phi_0 \in O(1)$.*

Proof. The algorithm moves on to the next point when the condition in line 19 of the algorithm is satisfied i.e., $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$ and $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, -)$. Since $\mathcal{O}(i, \mathbf{k}_i, h_i) = (Y, \cdot)$, $\mathcal{B}_i^H = \mathcal{B}(\mathbf{k}_i, h_i)$ intersect. Moreover, if $\mathbf{q}_j$ is the current nearest neighbor of $\mathbf{q}_i$, $\mathcal{O}(i, \mathbf{k}_i, h_i/\beta) = (Y, -)$ implies X_j lies outside the $1/\beta$ expansion of $\mathcal{B}_i^H$ (see fig. 4.6).

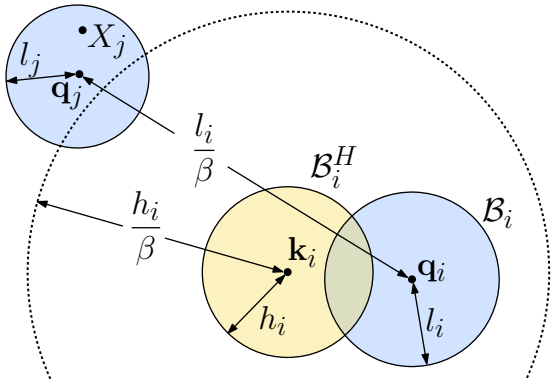


Figure 4.6: Proof of Lemma 4.4.

Now because $\mathbf{q}_j$ is the nearest neighbor of $\mathbf{q}_i$, $N_j \leq N_i$, hence $l_j \leq l_i$. By triangle inequality, the following series of inequality holds:

$$\frac{h_i}{\beta} \leq \|X_j - \mathbf{k}_i\| \leq \|X_j - \mathbf{q}_i\| + \|\mathbf{q}_i - \mathbf{k}_i\| \leq \left(\frac{l_i}{\beta} + l_j\right) + (h_i + l_i) \leq \left(\frac{1}{\beta} + 2\right) l_i + h_i,$$

implying that $\frac{1-\beta}{1+2\beta} h_i \leq l_i$. Now consider the hypothesis ball, $\mathcal{B}_i^{H^-}$, set by the algorithm in the previous step (via Line 26). From lemma 4.3, we have $\mathcal{B}_i \subseteq \mathcal{B}_i^{H^-}$. Let $h_i^- = \text{radius}\left(\mathcal{B}_i^{H^-}\right)$. Since the algorithm

shrinks the hypothesis ball by at least a factor of 2 during the zoom-in process, h_i^- is at least $2h_i$. And therefore $2h_i \geq l_i$. We also have $s_i \leq h_i + l_i$ (see fig. 4.6). Therefore,

$$\Phi_i = \log\left(\frac{\max(s_i, l_i, h_i)}{\sqrt{l_i h_i}}\right) \leq \log\left(\frac{l_i + h_i}{\sqrt{l_i h_i}}\right)$$
$$\leq \log\left(\frac{2h_i + h_i}{\sqrt{\frac{1-\beta}{1+2\beta} h_i \cdot h_i}}\right) \leq \phi_0, \quad \text{for } \phi_0 = \log\left(3 \cdot \sqrt{\frac{1+2\beta}{1-\beta}}\right)$$

□

4.6.2 Iterative changes to the Potential

Let us consider how algorithm TRACKBYZOOM affects the potential. For a particular index i, during the *zoom-out* process, we double the radius h_i of the hypothesis ball. If the Euclidean distance between the centers of the hypothesis, and the local-feature ball is much larger than h_i, then $\Phi = \log(\max(s_i, l_i, h_i)/\sqrt{l_i h_i}) = \log(s_i/\sqrt{l_i h_i})$ decreases by an additive constant. However, once h_i is greater than s_i, the algorithm risks increasing Φ. We show that the number of steps where Φ increases is a constant for every index i. However, during the zoom-in process, we maintain the invariant that the hypothesis ball contains the local-feature ball (lemma 4.3), which implies that $\max(s_i, l_i, h_i) = h_i$. Therefore, Φ decreases by a constant amount whenever we shrink the hypothesis ball by a constant factor. However, if the evolver moves q_i while index i is being processed, our algorithm may transition to the zoom-out process again. Hence, if the evolver executes some $\varepsilon^{\langle z \rangle}$ steps in the zth iteration, our algorithm may increase Φ by only $O\left(n + \varepsilon^{\langle z \rangle}\right)$. We obtain the following lemma.

Lemma 4.5 (Algorithm's contribution towards Φ)**.** *For any iteration z, let $\varepsilon^{\langle z \rangle}$ denote the number of steps executed by the evolver. Then* TRACKBYZOOM *increases Φ in $O\left(n + \varepsilon^{\langle z \rangle}\right)$ steps, each time by*

$O(1)$. *In each of the remaining steps of the zth iteration, it decreases Φ by $\Theta(1)$ in an amortized sense.*

Proof. For a particular index i, we first concentrate on the zoom-out routine of our algorithm TRACK-BYZOOM (see Line 6). Assume that the evolver leaves q_i untouched for the time being. We show later how to handle the case where the evolver moves q_i, while the algorithm is processing the index i. The algorithm increases h_i by a constant factor in Line 12. Therefore, $\Phi = \log(\max(s_i, l_i, h_i)/\sqrt{l_i h_i})$ decreases by a constant amount whenever $\max(s_i, l_i, h_i) \neq h_i$. Now suppose $\max(s_i, l_i, h_i) = h_i$ at time t, for the first time during the zoom-out routine. That implies a 2-expansion of $\mathcal{B}_i^H$ in line 12 contains the entire $\mathcal{B}_i$. Therefore, a further $1/\beta$-expansion definitely contains another X_j, $j \neq i$, satisfying the condition in line 8. We conclude that once $\max(s_i, l_i, h_i) = h_i$, the algorithm only increases Φ by a constant amount before moving on to the zoom-in routine of the algorithm (line 14). Now, we look at the zoom-in routine of the algorithm (line 14). The algorithm only zooms in as long as the condition in line 23 is satisfied. Since we expand $\mathcal{B}_i^H$ with an expansion factor $\frac{3}{1-2\beta}$ in line 26, using lemma 4.3 we conclude that $\mathcal{B}_i \subset \mathcal{B}_i^H$ in every step during the zoom-in routine. That implies $\max(s_i, l_i, h_i) = h_i$ throughout the zoom-in process, further implying $\Phi_i = \log\sqrt{h_i/l_i}$. In line 24 we subdivide the hypothesis ball into $|\Psi| = \left(2\lceil\frac{3}{1-2\beta}\rceil\sqrt{d}\right)^d \in O(1)$ balls and make constant number of queries to the oracle. Therefore, h_i reduces by a factor of $\frac{3}{1-2\beta}/\left(2\lceil\frac{3}{1-2\beta}\rceil\right) \geq 2$. And hence, Φ reduces by at least a constant amount. We amortize the reduction over the $|\Psi|$ calls to the Oracle, to denote a constant reduction in Φ in every zoom-in step.

If the evolver moved q_i while the algorithm was processing the index i, there is a possibility that X_i might move outside of $\mathcal{B}_i^H$ during the zoom-in routine. We take care of this condition in line 16 to initiate the zoom-out process. Since the algorithm possibly increases Φ by a constant amount when switching from zoom-out to zoom-in, we conclude: For every evolver's step, our algorithm might

increase Φ by at most a constant amount as well. □

Let $\Phi^{\langle z \rangle}$ be the potential function at the start of the zth iteration. And let $\Delta^{\langle z \rangle}$ be the time taken by the zth iteration of the algorithm. At the conclusion of the *zoom-in* process for a particular index i, the algorithm fixes a hypothesis H_i which is at a reasonable distance from the truth P_i. In fact, we show in lemma 4.4 that at this point in time, Φ_i is ϕ_0, a constant. Therefore, by lemma 4.5, TRACKBYZOOM running at a constant speed-up σ roughly spends $\sim \Phi_i^{\langle z \rangle}$ time (within constant factors), in reducing $\Phi_i^{\langle z \rangle}$ to ϕ_0. Summing over all indices, we see that the time taken by zth iteration, $\Delta^{\langle z \rangle}$, is $\sim \Phi^{\langle z \rangle}$. Intuitively, for a sufficiently large speed-up factor, the actions of the evolver can only affect $\Delta^{\langle z \rangle}$ by a small fraction of this amount. We summarize these observations in the following lemma.

Lemma 4.6 (Iteration time proportional to Potential)**.** *There exists a constant speed-up factor σ for* TRACKBYZOOM *such that* $\Delta^{\langle z \rangle} \in \Theta_\sigma\left(n + \Phi^{\langle z \rangle}\right)$.

Proof. By lemma 4.2, the evolver can only increase the potential by $\Phi_\varepsilon^{\langle z \rangle} = O(\Delta^{\langle z \rangle})$ during the iteration. By lemma 4.5 we observe that the algorithm increases Φ by some amount that is $O(n + \Delta^{\langle z \rangle})$. Since by lemma 4.4, the algorithm reduces each Φ_i to at most ϕ_0, it reduces the overall potential by at most $\Phi^{\langle z \rangle} - n\phi_0 + \Phi_\varepsilon^{\langle z \rangle} + O(n + \Delta^{\langle z \rangle}) = O(n + \Phi^{\langle z \rangle} + \Delta^{\langle z \rangle})$. For a speed-up factor σ, the algorithm TRACKBYZOOM spends $O(n + \Phi^{\langle z \rangle} + \Delta^{\langle z \rangle})/\sigma$ time to do so, implying $\Delta^{\langle z \rangle} \leq O(n + \Phi^{\langle z \rangle} + \Delta^{\langle z \rangle})/\sigma$, further implying $\Delta^{\langle z \rangle} \in O_\sigma(n + \Phi^{\langle z \rangle})$, for a sufficiently large constant σ. To see why $\Delta^{\langle z \rangle} \in \Omega(n + \Phi^{\langle z \rangle})$, we follow a similar logic, except to observe that the evolver can decrease the potential by at most $\Phi_\varepsilon^{\langle z \rangle} = O(\Delta^{\langle z \rangle})$ as well. By lemma 4.4, the algorithm reduces the overall potential Φ by at least $\Phi^{\langle z \rangle} - n\phi_0 - \Phi_\varepsilon^{\langle z \rangle} = \Omega(n + \Phi^{\langle z \rangle} - \Delta^{\langle z \rangle})$, which takes it at least $\Omega(n + \Phi^{\langle z \rangle} - \Delta^{\langle z \rangle})/\sigma$ time, implying that $\Delta^{\langle z \rangle} \geq \Omega(n + \Phi^{\langle z \rangle} - \Delta^{\langle z \rangle})/\sigma$, resulting in $\Delta^{\langle z \rangle} \in \Omega_\sigma(n + \Phi^{\langle z \rangle})$. □

We are now ready to derive how $\Phi^{\langle z \rangle}$ changes over the course of multiple iterations. We show

that for a sufficiently large (constant) speed-up factor σ, after $\sim \log\log\Lambda_0$ iterations Φ converges to $O(n)$. (Recall that Λ_0 is the initial aspect ratio.)

Lemma 4.7. *There exists a constant speed-up factor σ for algorithm* TRACKBYZOOM *and $z_0 \in \mathbb{Z}$, such that for all $z > z_0$, $\Phi^{\langle z\rangle} \in O(n)$*

Proof. By lemma 4.2, in the zth iteration the evolver takes at most $\varepsilon^{\langle z\rangle} = O(\Delta^{\langle z\rangle})$ steps and increases Φ by at most $O(\Delta^{\langle z\rangle})$. Therefore, by lemma 4.5, TRACKBYZOOM increases Φ by $O(n+\Delta^{\langle z\rangle})$. Since the algorithm runs at a speed-up factor σ it takes $\sigma\Delta^{\langle z\rangle}$ steps, and by lemma 4.5, it decreases Φ by at least $\lambda(\sigma\ \Delta^{\langle z\rangle} - O(n+\Delta^{\langle z\rangle}))$, for some $\lambda \in O(1)$. Accounting for all the increases and decreases we have:

$$\begin{aligned}\Phi^{\langle z+1\rangle} &\leq \Phi^{\langle z\rangle} + O(\Delta^{\langle z\rangle}) + O(n+\Delta^{\langle z\rangle}) - \lambda(\sigma\ \Delta^{\langle z\rangle} - O(n+\Delta^{\langle z\rangle}))\\ &\leq \Phi^{\langle z\rangle} + O(\Delta^{\langle z\rangle}) - \sigma\lambda\Delta^{\langle z\rangle} + O(n).\end{aligned}$$

There exists a sufficiently large $\sigma \in O(1)$ such that $\sigma\lambda\Delta^{\langle z\rangle} - O(\Delta^{\langle z\rangle}) \geq a\Delta^{\langle z\rangle}$, for $0 < a \in O(1)$. So,

$$\Phi^{\langle z+1\rangle} \leq \Phi^{\langle z\rangle} - a\Delta^{\langle z\rangle} + O(n).$$

By lemma 4.6, for a sufficiently large (constant) σ, there exists $c_\sigma > 0$ such that $\Delta^{\langle z\rangle} \geq c_\sigma\left(n + \Phi^{\langle z\rangle}\right)$. Therefore,

$$\Phi^{\langle z+1\rangle} \leq (1-ac_\sigma)\Phi^{\langle z\rangle} + (1-ac_\sigma)O(n) \leq (1-ac_\sigma)^{z+1}\Phi^{\langle 0\rangle} + \left(\sum_{y=1}^{z+1}(1-ac_\sigma)^y\right)O(n).$$

By Eq. (4.11), $\Phi^{\langle 0\rangle} = \Phi^{(0)} \in O(n\log\Lambda_0)$, and since there exists $b > 1$ such that $(1-ac_\sigma) < 1/b < 1$,

we have

$$\Phi^{\langle z+1\rangle} \leq \frac{O(n \log \Lambda_0)}{b^{z+1}} + O(n). \tag{4.12}$$

Observe that this is $O(n)$ whenever $z > \log\log \Lambda_0$. □

If $\Phi^{\langle z\rangle} \in O(n)$, then by lemma 4.6, $\Delta^{\langle z\rangle} \in \Theta(n)$. Thus, Φ increases by at most $O(n)$ throughout iteration z. Therefore, for any time t during iteration z, $\Phi^{(t)} \in O(n)$ as well. Using Eq. (4.12) we observe that for all $z > 1$, $\Phi^{\langle z\rangle} \leq O(n \log \Lambda_0)$. By lemma 4.6, this implies that $\Delta^{\langle z\rangle} \in \Theta(n \log \Lambda_0)$. Hence, for some $t_0 \in O(n \log \Lambda_0 \cdot \log\log \Lambda_0)$, we have $\Phi^{(t)} \in O(n)$, for all $t > t_0$. By combining Lemmas 4.7 and 4.1, we complete the proof of theorem 4.1. It is noteworthy that there is a trade-off between the speed-up factor σ and the motion parameter α. Recall that with each step, the evolver can move q_i by a distance of up to αN_i. If we reduce this parameter to $\alpha' = (1+\alpha)^{\frac{1}{c}} - 1$, then it takes the evolver at least c steps to effect the same change as before. Thus, by reducing the local-motion factor, it is possible to achieve a speed-up factor of $\sigma = 1$. Formally we have,

Corollary 4.1. *There exists a motion parameter $\alpha' < 1$ such that for a set of n evolving objects in $\mathbb{R}^d$ under the (α', β)-local-motion model, where $\beta < \frac{1}{3}$, algorithm* TRACKBYZOOM *(with speed-up factor 1) maintains a distance of $O(n)$ for all $t \geq t_0$, $t_0 \in O(n \log \Lambda_0 \log\log \Lambda_0)$*

4.7 Extensions and Future Works

4.7.1 Other Probability Distributions

The local-motion model introduced in Section 4.3 assumes a uniform probability distribution for each object. In this section we show that algorithm TRACKBYZOOM with an appropriate speed-up

factor, can be applied to a wider class of probability distributions. Let $\mathscr{P}$ be any bounded probability distribution restricted to the d-dimensional unit ball, $\mathcal{B}(\mathbf{0}, 1)$. That is, there is a constant ν and $\mathscr{P} : \mathcal{B}(\mathbf{0}, 1) \to [0, \nu)$. The probability distribution associated with $\mathbf{q}_i$ is defined to be

$$P_i(\mathbf{x}) \;=\; \frac{1}{l_i^d}\,\mathscr{P}\left(\frac{\mathbf{x}-\mathbf{q}_i}{l_i}\right) \;=\; \frac{\mathscr{P}(\widetilde{\mathbf{x}})}{l_i^d}, \qquad \widetilde{\mathbf{x}} = \left(\frac{\mathbf{x}-\mathbf{q}_i}{l_i}\right)$$

We define H_i as before (Eq. (4.4: Hypothesis Def.)) using this probability distribution. Similar to Eq. (4.6), the distance for the ith object is

$$D_i \;=\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log \frac{P_i(\mathbf{x})}{H_i(\mathbf{x})}\mu(d\mathbf{x}) \;=\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log P_i(\mathbf{x}) - \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log H_i(\mathbf{x}).$$

For $\mathbf{x} \in \mathcal{B}_i$, we have

$$H_i(\mathbf{x}) \;\geq\; \frac{1}{\pi^d h_i^d} \prod_{j=1}^{d} \left(1 + \frac{(s_i + l_i)^2}{h_i^2}\right)^{-1},$$

and therefore,

$$D_i \;\leq\; \int_{\mathbf{x}\in\mathcal{B}_i} P_i(\mathbf{x}) \log P_i(\mathbf{x})\mu(d\mathbf{x}) \;+\; \log\left(\frac{h_i^2 + (s_i + l_i)^2}{\pi h_i}\right)^d.$$

By the concavity of the log function and Jensen's inequality, we have $\int P_i(\mathbf{x}) \log P_i(\mathbf{x}) \leq \log \int P_i^2(\mathbf{x})$. Clearly, $P_i^2(\mathbf{x}) = \mathscr{P}^2(\widetilde{\mathbf{x}})/l_i^{2d}$. Thus,

$$D_i \;\leq\; \log \int_{\mathbf{x}\in\mathcal{B}_i} \frac{\mathscr{P}^2(\widetilde{\mathbf{x}})}{l_i^{2d}}\mu(d\mathbf{x}) + \log\left(\frac{h_i^2 + (s_i + l_i)^2}{\pi h_i}\right)^d.$$

Since $\mathscr{P}(\widetilde{\mathbf{x}}) < \nu$, and $\mathrm{vol}(\mathcal{B}_i) = \omega_d l_i^d$, where ω_d denotes the volume of the unit Euclidean ball in $\mathbb{R}^d$, we obtain

$$D_i \;\leq\; \log \frac{\omega_d \nu^2}{l_i^d} + \log \left(\frac{h_i^2 + (s_i + l_i)^2}{\pi h_i} \right)^d \;=\; \log \frac{\omega_d \nu^2}{\pi^d} + d \cdot \log \frac{h_i^2 + (s_i + l_i)^2}{h_i l_i}.$$

Using the same potential function as defined in Eq. (4.5), we have

$$D_i \;\in\; O(1) + O(\Phi_i).$$

Since we argued on the potential function Φ to conclude our main result, Theorem 4.1 also holds for any probability distribution that satisfies these criteria. Note many natural distributions, such as the uniform distribution, the truncated normal distribution, the truncated Cauchy distribution, belong to this family. (Truncated distribution here refers to the conditional distribution resulting from restricting the domain to $\mathcal{B}_i$: the volume associated with the ith object.)

4.7.2 Concluding Remarks and Transition to Part II

We have shown that it is possible to track evolving distributions with high accuracy using a constant speedup factor, provided we measure "distance" using the information-theoretic KL divergence. This concludes Part I of the dissertation, which focused on the *temporal* aspect of uncertainty: tracking moving targets.

However, tracking is only half the battle. In many applications, once we have bounded the uncertainty of an object (modeled as a distribution), we must perform geometric operations on it, such as *classification*. In Part II, we shift our focus from evolving distributions in time to classifying

distributions in space.

The transition between these two parts reveals a curious and mathematically profound duality in our choice of metrics. In this chapter, we tackled a fundamentally geometric problem (tracking a physical evolver in Euclidean space) by relying on a metric designed to compare probability distributions (the KL divergence). In Part II, we will invert this paradigm. We will tackle a fundamentally probabilistic problem (classifying distributional data within a probability simplex) by relying on a purely geometric measure: the *Hilbert metric*.

Despite their contrasting origins, these two metrics share a deep structural kinship. The KL divergence has shown us that geometry near the boundaries of a probability space is highly distorted, since informational distances explode as probabilities approach zero. In the next chapter, we will introduce the Hilbert metric, a projective geometry that naturally captures this exact boundary-sensitive behavior. By leveraging this geometric framework, we will build robust classifiers (Support Vector Machines) for the very distributional data we have learned to track here.

Part II

Spatial Uncertainty: Classification in Bounded Domains

Chapter 5: Support Vector Machines in Hilbert Metric

5.1 Chapter Overview

In Part I of this dissertation, we focused on *temporal uncertainty*, i.e. tracking objects and distributions as they evolved over time. We utilized the Kullback-Leibler (KL) divergence to measure the informational distance between a hypothesis and the ground truth.

In Part II, we shift our focus to *spatial uncertainty*. We consider the problem of *classification* within bounded domains, such as the probability simplex. Just as the KL divergence was the natural metric for tracking distributions, the *Hilbert metric* is the natural geometry for classifying them. Both metrics share a fundamental property: they expand the geometry near the boundaries of the domain, offering robustness where Euclidean metrics fail.

In this chapter, we lay the theoretical groundwork for classification in this geometry. We define the *Linear Support Vector Machine (SVM)* problem in the Hilbert metric. We prove that this problem is of *LP-type* (Linear Programming type), which allows us to derive exact algorithms. We present efficient solutions for convex polytopes in d-dimensional space. While these exact algorithms provide a rigorous baseline, their dependence on the combinatorial complexity of the domain motivates the approximation techniques we will develop in Chapter 6.

5.2 Introduction

In machine learning, *binary classification* is a supervised learning task where an algorithm learns to map input data (feature vectors in $\mathbb{R}^d$) to one of two mutually exclusive classes, often denoted as *positive* and *negative*. The primary objective is to train a model that can accurately predict the class of a new, unseen data point. Binary classification is one of the most fundamental geometric problems in machine learning [18, 31, 80, 87]. Support vector machines (SVMs) were introduced by Vapnik and Chervonenkis [90] as a robust binary classification algorithm [29]. Given two sets of *training points* A and B, SVM computes a hyperplane that separates the two sets such that the minimum distance from each point set to the separating hyperplane, called the *margin*, is maximized. They take two classes of data that are embedded as points in a multidimensional feature space. Once embedded, the algorithm attempts to separate them using a kernel function, which is chosen to maximize the separation distance between the two classes. Throughout, we assume linear kernels, which means that points are separated by a hyperplane.

Two sets of points $A, B \subset \mathbb{R}^d$ are said to be *linearly separable* if there exists a $(d-1)$-dimensional hyperplane h such that A and B lie on opposite sides of h, possibly on h itself (see Figure 5.1(a)). Define $\mathrm{Sep}(A, B)$ to be the set of separating hyperplanes. The minimum distance from either set to a separating hyperplane is called its *margin*. Given two linearly separable point sets A and B in $\mathbb{R}^d$, the (linear) *SVM problem* is that of computing the hyperplane $h \in \mathrm{Sep}(A, B)$ that maximizes the margin (see Figure 5.1(b)). The name arises from the fact that, assuming general position, there are $d+1$ points that achieve the margin exactly, called the *support vectors*.

SVMs are heavily relied upon across a diverse set of fields such as computer vision [57, 88], natural language processing [34, 51], and computational biology [13, 48]. Traditionally, these mod-

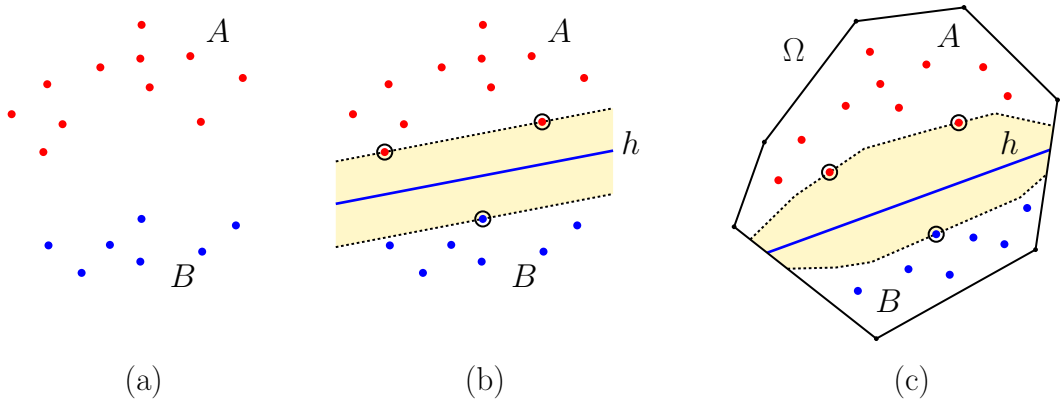


Figure 5.1: (a) The point sets, (b) Euclidean SVM, (c) and the Hilbert SVM.

els are built upon Euclidean geometry. However, there has been a massive recent surge of interest in non-Euclidean spaces, particularly hyperbolic geometries. This interest stems from the unique exponential volume growth property of hyperbolic space, which allows it to embed trees and complex hierarchical structures with minimal distortion [58,59]. Consequently, hyperbolic representations have become highly effective for modeling word hypernymy and linguistic structures in natural language processing [36,70], capturing complex user-item interactions and implicit hierarchies in recommendation systems [24], and mapping structural connectivity in brain networks in neuroscience [6]. SVMs have been successfully adapted to these hyperbolic settings to respect this intrinsic geometry [27].

While unbounded spaces (like the hyperbolic Poincare disk) excel at representing hierarchical data, a vast number of critical domains in modern machine learning are intrinsically constrained within *bounded* convex polytopes. In this chapter, we study the SVM problem from the perspective of the Hilbert geometry and the closely related Funk geometry. In 1895, David Hilbert introduced the Hilbert metric [46]. The Hilbert metric (defined formally in Section 5.3) generalizes the Cayley-Klein model of hyperbolic geometry to arbitrary bounded convex sets. It boasts several desirable properties, such as having straight-line geodesics and being invariant under projective transformations [75]. In bounded domains, distances must be acutely sensitive to the proximity of a point to the boundary of the feasible

region. The Hilbert metric naturally imposes this penalty, diverging to infinity as points approach the boundary.

5.2.1 Why Hilbert Geometry? The Link to Uncertainty and Machine Learning

In Chapter 4, we modeled uncertain objects as probability distributions and used the KL divergence to track them. Here, we address the problem of *classifying* such objects. The motivation for developing classification algorithms specifically tailored to bounded polytopal boundaries under the Hilbert metric is driven by several applications in contemporary machine learning:

- *Compositional Data and the Probability Simplex:* Compositional data, such as market shares, genomic relative abundances, and topic distributions, consist of nonnegative vectors that sum to one. These vectors naturally reside within a bounded polytope known as the *probability simplex*. Euclidean distances are notoriously ill-suited here because they fail to capture relative variations near the boundaries (i.e., when probabilities approach 0 or 1). The Hilbert metric implicitly respects these compositional constraints and boundary sensitivities, serving as a powerful intrinsic geometry for clustering and classification in the simplex [72, 73]. Thus, the Hilbert SVM is the natural classifier for the distributional data structures we investigated in Part I.

- *Optimal Transport and the Birkhoff Polytope:* In optimal transport, matching, and assignment problems, the set of feasible transport plans (doubly stochastic matrices) forms a bounded convex domain known as the Birkhoff polytope [64]. Solving machine learning problems over permutations or transport plans frequently requires navigating this highly structured polytope [19, 83]. An SVM formulated in the Hilbert metric provides a theoretically sound mechanism for large-margin classification of transport matrices by explicitly respecting the linear constraints

that define the Birkhoff polytope.

- *Deep Learning on Symmetric Spaces and Siegel Networks:* There is a growing interest in representation learning on symmetric spaces, such as the bounded Siegel disk or the cone of symmetric positive-definite matrices. These spaces are foundational in advanced architectures like Siegel networks, and are crucial for applications in quantum information theory, medical imaging, and deep learning [50, 53, 61, 78]. The Hilbert metric naturally models the intrinsic geometry of these bounded domains, particularly when they are approximated by high-dimensional bounding facets.

- *Combinatorial Reasoning and Tropical Geometry:* Recent advances at the intersection of machine learning and combinatorial optimization utilize tropical geometry to map discrete algorithms into continuous, piecewise-linear, polyhedral structures. In tropical geometry, feasible solutions and decision boundaries trace the faces of polyhedral complexes [44, 97]. Classifying or routing information within these spaces requires a metric that aligns with polyhedral boundaries. A Hilbert SVM provides an exact, geometrically grounded decision boundary for neural algorithmic reasoning over such combinatorial domains.

5.2.2 Problem Statement and Contributions

The input to our problem consists of two discrete, linearly separable point sets A and B in the interior of a convex body Ω, and the objective is to compute a separating hyperplane h that maximizes the minimum Hilbert distance from every point of $A \cup B$ to h. Changing how distances are measured results in very different criteria for the optimum separating hyperplane (see Figure 5.1(c)). In this chapter, we explore these differences and develop an efficient algorithm for the Hilbert SVM problem

for convex polygons in the plane. We also generalize our results to the closely related Funk geometry.

The results for the planar case were established in [2]. A complicated binary search on the points to find the tangent line, resulted in the following result. We mention it here for the sake of completion:

Proposition 5.1. *The Hilbert SVM problem, and the Funk SVM problem for a set of n points in an m-sided convex polygon in $\mathbb{R}^2$ can be solved in $O(n \log^2 m)$ and $O(n \log m)$ time respectively.*

The main focus of this chapter is solving the SVM problems in the multidimensional case, where a binary search like approach as in [2] is infeasible. For our multidimensional case, Ω is a full-dimensional convex polytope of total combinatorial complexity M. By this, we mean that Ω is the convex hull of a finite set of points in general position, and M is the total number of faces of all dimensions on its boundary. The following are our main results:

Theorem 5.1. *For any fixed $d \geq 3$, the Hilbert SVM problem for a set of n points in a convex polytope in $\mathbb{R}^d$ is of LP-type and can be solved in $O(nM^3)$ time, where M is the total combinatorial complexity of Ω.*

Theorem 5.2. *For any fixed $d \geq 3$, the Funk SVM problem for a set of n points in a convex polytope in $\mathbb{R}^d$ is of LP-type and can be solved in $O(nM_0)$ time, where M_0 is the number of vertices of Ω.*

Note that it is possible to test whether A and B are linearly separable through a standard reduction to linear programming. Through an analysis of the geometry of separating hyperplanes in the Funk and Hilbert geometries, we show that in $\mathbb{R}^d$, these SVM problems are of LP-type [86] of combinatorial dimension $d + 1$. By standard results, in any fixed dimension d, LP-type problems can be solved with $O(n)$ violation tests and evaluations of the objective function on point sets of constant size. (See, e.g., Sharir and Welzl [86], Seidel [82], and Clarkson [28], or the deterministic algorithm of

Chazelle and Matoušek [25].) Our main technical contribution is a detailed analysis of the geometric structure of the distance between points and lines and hyperplanes in the Funk and Hilbert geometries. We use this analysis to show how to compute the violation tests and objective function evaluations that are required by algorithms for solving LP-type problems.

The remainder of the chapter is organized as follows. In Section 5.3, we present preliminary definitions and facts about the Hilbert geometry and SVMs. In Section 5.4, we present the proof that the Hilbert SVM problem is of LP-type of combinatorial dimension three. In Section 5.5, we present algorithms for computing the primitives of violation testing and computing the objective function. Finally, in Section 5.6, we present concluding remarks.

5.3 Mathematical Preliminaries

In this section, we provide a number of definitions and useful facts. Throughout, for our planar results, Ω is an m-sided convex polygon. For our multidimensional results, Ω is a full-dimensional convex polytope of total combinatorial complexity m. By this, we mean that Ω is the convex hull of a finite set of points in general position, and m is the total number of faces of all dimensions on its boundary. Let $\partial\Omega$ denote its boundary, and let $\mathrm{int}(\Omega)$ denote its interior. Given two points $p, q \in \Omega$, let $\overline{pq}$ denote the *chord* of Ω passing through these points, that is, the intersection of Ω with the line through p and q. Given a set $S \subset \mathbb{R}^2$, let $\mathrm{conv}(S)$ denote its convex hull. Given two points p and q, let $\overleftrightarrow{pq}$ denote the line that passes through them, that is, their affine closure.

5.3.1 The Funk and Hilbert Distances

In this section, we will introduce the Funk, reverse-Funk, and Hilbert geometries over Ω. For any two points $p, q \in \text{int}(\Omega)$, if $p = q$ all the distances are defined to be zero. Otherwise, let p' and q' denote the endpoints of the chord $\overline{pq}$ ordered as $\langle p', p, q, q' \rangle$ (see Figure 5.2(a)). These distances are defined in terms of the ratios of distances between pairs of these points.

$$\text{Funk:} \quad d_\Omega^F(p,q) = \ln \frac{\|p-q'\|}{\|q-q'\|} \tag{5.1}$$

$$\text{Reverse Funk:} \quad d_\Omega^{rF}(p,q) = d_\Omega^F(q,p) = \ln \frac{\|q-p'\|}{\|p-p'\|} \tag{5.2}$$

$$\begin{aligned} \text{Hilbert:} \quad d_\Omega^H(p,q) &= \frac{d_\Omega^F(p,q) + d_\Omega^{rF}(p,q)}{2} \\ &= \frac{1}{2} \ln \left(\frac{\|q-p'\|}{\|p-p'\|} \frac{\|p-q'\|}{\|q-q'\|} \right). \end{aligned} \tag{5.3}$$

Intuitively, $d_\Omega^F(p,q)$ is a measure of how far q is from p relative to the boundary that lies beyond q, and $d_\Omega^{rF}(p,q)$ is just the reverse of this. The Hilbert distance $d_\Omega^H(p,q)$ symmetrizes these by taking their arithmetic mean. These distance functions are all nonnegative and satisfy the triangle inequality [75]. The Hilbert distance defines a *metric* over $\text{int}(\Omega)$. The Funk and reverse Funk are asymmetric, and so they each define a *weak metric* over $\text{int}(\Omega)$. Observe that as either point approaches the boundary of Ω, the Hilbert distance approaches ∞. (This is true for one of the two points in Funk and reverse-Funk.) The Funk and reverse-Funk distances are invariant under invertible affine transformations. The Hilbert distance has the additional property that it is invariant under invertible projective transformations.

Letting d_Ω^* denote any of the above and given a point $p \in \Omega$ and scalar $r \geq 0$, the associated

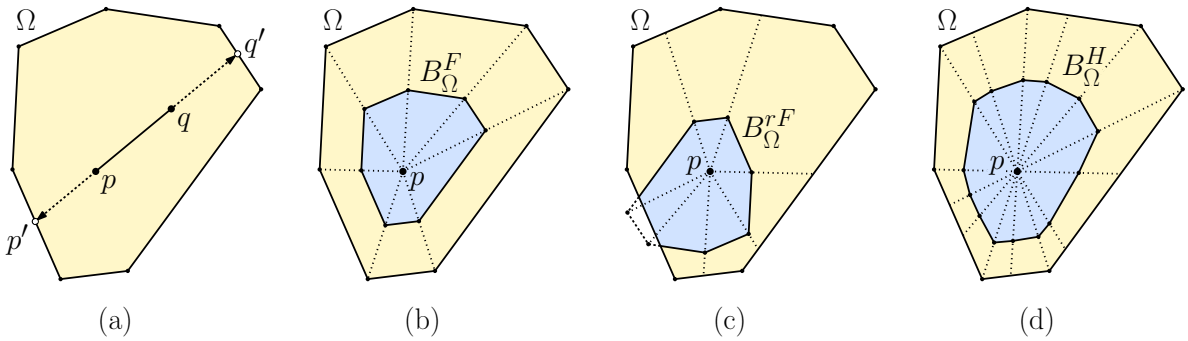


Figure 5.2: (a) Defining distance and balls for (b) Funk, (c) reverse Funk, and (d) Hilbert.

metric balls are defined in the standard way.

$$B^*_\Omega(p, r) \;=\; \{q \in \Omega \;:\; d^*_\Omega(p, q) \leq r\}.$$

In particular, letting $\lambda^F(r) = 1 - e^{-r}$ and $\lambda^{rF}(r) = e^r - 1$, it is easy to verify that the Funk and reverse-Funk balls are

$$B^F_\Omega(p, r) \;=\; p + \lambda^F(r)(\Omega - p) \tag{5.4}$$

$$B^{rF}_\Omega(p, r) \;=\; \Omega \cap \big(p - \lambda^{rF}(r)(\Omega - p)\big)\,. \tag{5.5}$$

Thus, the Funk ball is a scaling of Ω by a factor of $\lambda^F(r)$ about p [74] (see Figure 5.2(b)), and the reverse-Funk ball is the intersection of Ω with a scaling of $-\Omega$ by a factor of $\lambda^{rF}(r)$ about p (see Figure 5.2(c)). Unlike the Funk ball, this scaling may generally extend outside Ω. The Hilbert ball about a point is a bit more complicated to describe. Let us first consider the planar case. Given a point $p \in \mathrm{int}(\Omega)$, define a *spoke* to be the chord $\overline{pv}$, for some vertex v of Ω, and define a *half-spoke* to be either of the subsegments of the spoke whose endpoint is p. The m vertices of Ω define m spokes, which induce $2m$ half-spokes joining p to the boundary of Ω (see Figure 5.2(d)). Nielsen and Shao

showed that $B_\Omega^H(p,r)$ is a convex polygon whose vertices are the $2m$ points lying on these half-spokes at Hilbert distance r from p. Furthermore, the edges of the ball satisfy the following concurrency condition [71].

Lemma 5.1. *Consider a convex polygon Ω in $\mathbb{R}^2$ and an edge e on the boundary of some Hilbert ball $B_\Omega^H(p,r)$. Let e' and e'' denote the edges of Ω that lie within the double-wedge formed by the two half-spokes that bound e (see Figure 5.3). Then the linear extensions of e, e', and e'' are concurrent.*

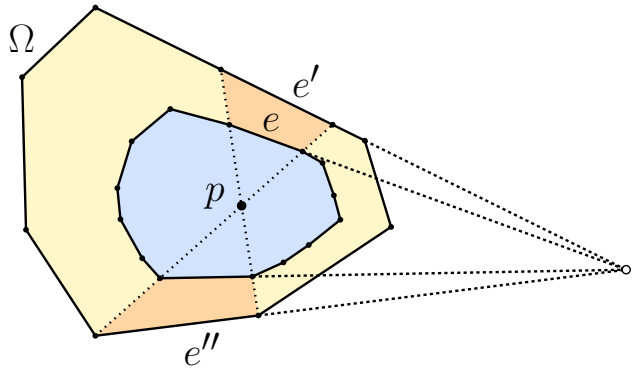


Figure 5.3: Concurrency of Hilbert ball edges.

This lemma generalizes to $\mathbb{R}^d$, where e, e' and e'' are facets, but the planar version suffices for our results. Since all distances will be defined with respect to Ω henceforth, we will frequently omit explicit references to Ω when discussing distances and metric balls. The concept of spokes can be easily generalized to the case where Ω is a convex polytope in $\mathbb{R}^d$, and $p \in \text{int}(\Omega)$. In particular, for $0 \leq k \leq \lfloor (d-1)/2 \rfloor$, consider two faces f and f' of Ω of respective dimensions k and $d-k-1$ such that p lies in the convex hull of $f \cup f'$. It follows from straightforward conditions on dimensionality (see, e.g., [32]) that, assuming general position, for any two such faces, there is at most one line passing through p that intersects both faces. Define a *spoke* of p to be the chord of Ω contained in any such line, and again a *half-spoke* is the portion of the chord lying on one side of p. (The planar case arises when $d = 2$ and $k = 0$.)

Lemma 5.2. *Consider a convex polytope Ω in $\mathbb{R}^d$ of combinatorial complexity M and a point $p \in \text{int}(\Omega)$, for any $r > 0$, the Hilbert ball $B^H_\Omega(p, r)$ is a polytope with $O(M^2)$ vertices, each of which lies on a half-spoke emanating from p. When $d = 2$, this reduces to $O(m)$.*

Proof. The planar case follows from the analysis of Nielsen and Shao [71]. For the higher-dimensional cases, we appeal to a variational argument. Consider any vertex x of $B^H_\Omega(p, r)$. Suppose to the contrary that the chord $\overline{px}$ is not a spoke. Consider the points y and y' where this chord intersects $\partial\Omega$. It follows that there are nonempty affine neighborhoods around y and y', denoted N_y and $N_{y'}$, respectively, $N_y, N_{y'} \subset \partial\Omega$, such that lines passing through p and intersecting N_y also intersect $N_{y'}$. This implies that there is a nonempty affine neighborhood around x such that $d^H_\Omega(p, x') = r$, for all x' in this neighborhood. Hence, x is not a vertex of $B^H_\Omega(p, r)$. By the observations made above, the number of spokes passing through any point $p \in \text{int}(\Omega)$ is bounded by the square of the number of faces of Ω, that is, $O(M^2)$. □

5.3.2 On Hilbert and Funk Balls

In this section, we present a number of technical results regarding the geometry and construction of Hilbert and Funk balls for lines and hyperplanes. Due to space limitations, some details are omitted and appear in the full version of the paper. Let us consider how to compute the reverse-Funk ball of radius r for a hyperplane h that intersects a polytope Ω. (Later, we will consider how this applies to the planar case, where h is a line.) Consider any point q that lies on the boundary of $B^{rF}(h, r)$ (see Figure 5.4(a)). This means that the closest point of q on h lies at Funk distance r. Letting p denote this point, we have $d^F(q, p) = r$. Consider the supporting hyperplane h' of Ω that is parallel to h and on the opposite side from q. Let q' be a vertex of Ω on h' (see Figure 5.4(b)). By convexity, the segment

$\overline{q'q}$ intersects h at some point p within Ω. Define $\mathrm{cp}^F(q,h)$ to be this point. By Eq. (5.4) (noting that q here plays the role of p in the equation), we have $\|p-q\|/\|q'-q\| = \lambda^F(r)$. It follows that the set of points q that lie on the boundary of $B^{rF}(h,r)$ define a hyperplane h'' that is parallel to h, where the relative distances between these hyperplanes is determined by $\lambda^F(r)$. By applying the same logic to the points on the opposite side of h, it follows that $B^{rF}(h,r)$ is the intersection of Ω with a slab bounded by two hyperplanes that are parallel to h.

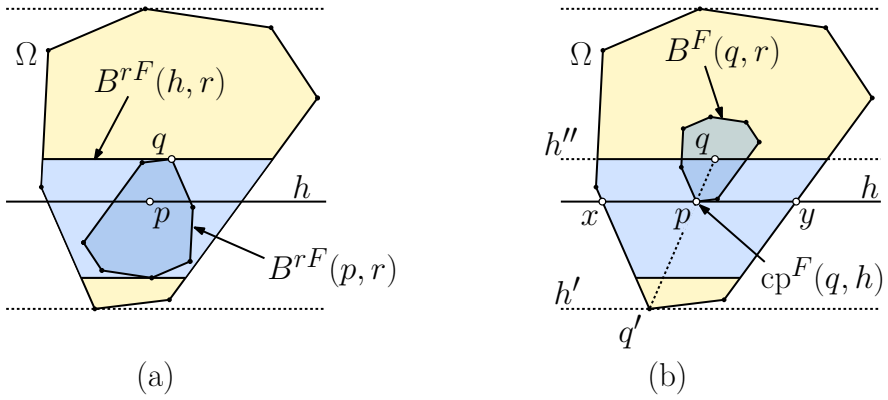


Figure 5.4: (a) The reverse-Funk ball around a line and (b) the closest point to q in the Funk distance.

The times to compute $\mathrm{cp}^F(q,h)$ and $d^F(q,h)$ are dominated by the time needed to compute the supporting hyperplanes of Ω that are parallel to h. If Ω is a convex polytope in $\mathbb{R}^d$ with M_0 vertices, this can be done in $O(M_0)$ time.

Lemma 5.3. *Consider a convex polytope Ω in $\mathbb{R}^d$ and a $(d-1)$-dimensional hyperplane h that intersects the interior of Ω and $q \in \mathrm{int}(\Omega)$. Then*

(i) *$B^{rF}(h,r)$ is the intersection of Ω with a slab that is parallel to h.*

(ii) *$\mathrm{cp}^F(q,h)$ is the point p on $h \cap \Omega$ that minimizes the Funk distance $d^F(q,p)$.*

If Ω is a convex polytope in $\mathbb{R}^d$ with M_0 vertices, $\mathrm{cp}^F(q,h)$ and $d^F(q,h)$ can be computed in $O(M_0)$ time.

Next, we consider the Hilbert ball of radius $r \geq 0$ about a line ℓ that intersects the interior of Ω, $B^H(\ell, r)$. The procedure for constructing such a ball in $\mathbb{R}^2$ was given by Gezalyan and Mount [40, Lemma 6], but we sketch it here for the sake of completeness. First, extend each of the edges of Ω until the extension intersects ℓ. For each such point s, consider the triangle defined by the convex hull of the edge e that was extended to form s and the vertex v where the supporting line for Ω passing through s hits the opposite side of Ω (see Figure 5.5(a)). These triangles subdivide Ω. Each such triangle has two chords that cross ℓ. On each such chord, create two points at Hilbert distance r from the crossing point with ℓ. These points are in convex position, and by the projective properties of the Hilbert distance, the line segments connecting them through each triangle are part of the Hilbert ball. Together, these points constitute the vertices of the convex hull of the Hilbert ball of the line (see Figure 5.5(b)).

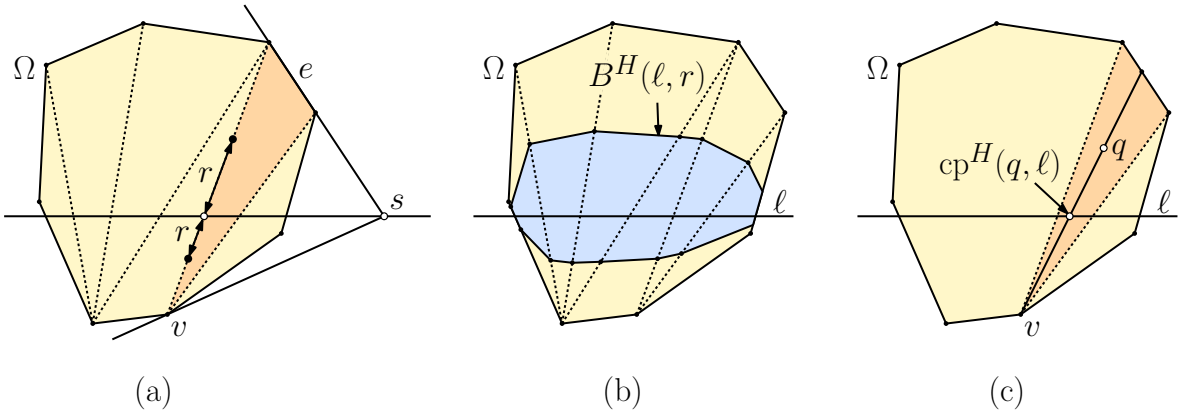


Figure 5.5: The Hilbert ball around a line.

Given a point $q \in \Omega$, we can compute its closest point on $\ell \cap \Omega$ as follows. Determine the triangle of the above construction that contains q (see Figure 5.5(c)). Consider the chord passing through q and the appropriate vertex v of the triangle. Let $\mathrm{cp}^H(q, \ell)$ denote the intersection point of this chord with ℓ. The chord that joins q and $\mathrm{cp}^H(q, \ell)$ is called the *closest-point spoke* for q with respect to ℓ. The following was proved in [40]

Proposition 5.2. *Consider a convex polygon Ω in $\mathbb{R}^2$ and a line ℓ that intersects the interior of Ω and $q \in \text{int}(\Omega)$. Then*

(i) *$B^H(\ell, r)$ is generated by the above procedure*

(ii) *$\text{cp}^H(q, \ell)$ is the point p on $\ell \cap \Omega$ that minimizes the Hilbert distance $d^H(p, q)$.*

Assuming Ω has m-sides, $\text{cp}^H(q, \ell)$ and $d^H(q, \ell)$ can be computed in $O(\log^2 m)$ time.

For the multi-dimensional case, however, a similar approach seems untenable owing to the total complexity of Ω. We instead keep all possible extreme points (corners) of the d-dimensional Hilbert ball as a proxy (Section 5.4.2)

5.3.3 Support Vector Machines in the Euclidean Geometry

Before discussing SVMs in the Hilbert setting, it is illustrative to explore the differences with the Euclidean case. Given two point sets $A, B \in \mathbb{R}^d$ (see Figure 5.6(a)) which are assumed to be linearly separable, the objective is to compute a separating hyperplane h that maximizes the *margin*, defined to be the minimum distance of either point set to h (see Figure 5.6(b)). The points achieving the margin are called the *support vectors*. Local minimality considerations imply that, assuming general position, the number of support vectors does not exceed $d + 1$ [23].

Define a *slab* to be the region bounded between two parallel hyperplanes, the Euclidean SVM problem can be interpreted as the problem of computing the slab of maximum orthogonal width that separates A and B. The SVM hyperplane lies midway between the slab's bounding hyperplanes. The margin r is just half the slab's width (see Figure 5.6(b)). An alternative and equivalent interpretation is to consider the maximum radius r such that by placing balls of radius r about the points of A and B, the resulting sets of balls are linearly separable (see Figure 5.6(c)). By replacing Euclidean balls with

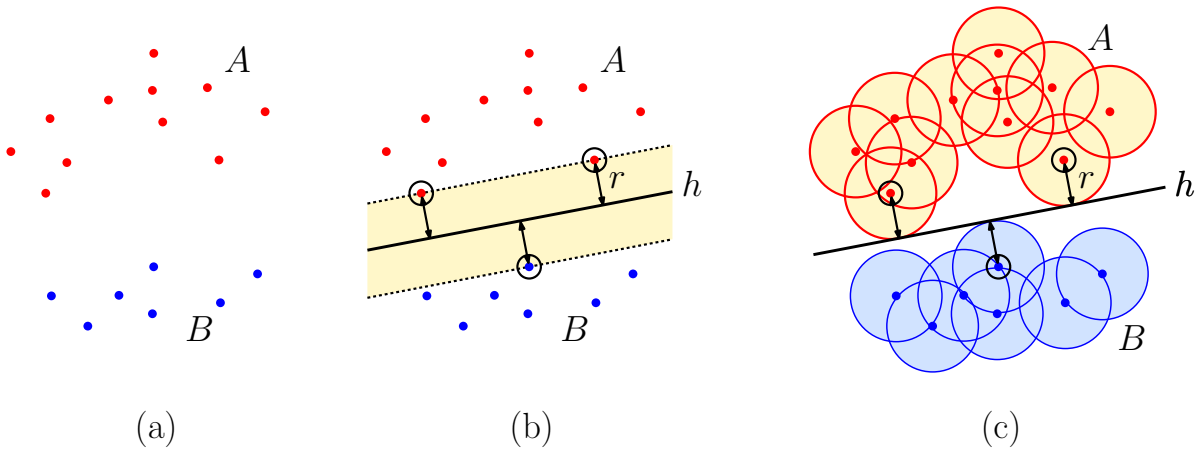


Figure 5.6: The Euclidean SVM problem: (a) Point sets A and B, (b) the SVM hyperplane and support vectors, (c) separating maximal balls.

metric balls, this observation can be readily generalized to any normed space. The problem can be shown to be of LP-type [86]. This implies the existence of a linear-time solution in spaces of constant dimension. There are a number of techniques to accelerate the running time in high dimensions. An important distinction with SVM in the context of the Hilbert and Funk distances is that the shape of metric balls varies with the location of the center point.

5.4 SVMs in the Funk and Hilbert Geometries

In this section, we present algorithms for solving the SVM problem in the Funk and Hilbert geometries. We consider both the planar and multidimensional cases. Recall that Ω is a full-dimensional convex polytope of total combinatorial complexity M. In both cases, we are given two linearly separable point sets $A, B \subset \text{int}(\Omega)$ (see Figure 5.7(a)). The objective is to compute a separating hyperplane h that maximizes the minimum Hilbert (or Funk) distance from each point of $A \cup B$ to h. Throughout, let $n = |A| + |B|$. Although we treat both n and m as asymptotic quantities, we think of m more as a large constant, since it defines a fixed geometry. Just as in the Euclidean case (see Section 5.3.3), there

are two equivalent formulations of the problem. In the first, we seek the separating hyperplane h of maximum radius r such that $B^H(h, r)$ contains no points of $A \cup B$ in its interior (see Figure 5.7(b)). The other is to determine the maximum radius r such that the union of balls $\bigcup_{a \in A} B^H(a, r)$ is linearly separable from $\bigcup_{b \in B} B^H(b, r)$ (see Figure 5.7(c)). As in the Euclidean SVM problem, the support vectors are associated with the points of A and B whose distance to the SVM separator equals the margin. We augment the concept of support vector to include not only the point but the half-spoke that identifies the vertex of the Hilbert or Funk ball that lies on the separating hyperplane. Note that when dealing with the Funk distance, we use the forward sense of the distance when computing balls centered at the points, and we use the reverse-Funk when computing the ball centered about the hyperplane.

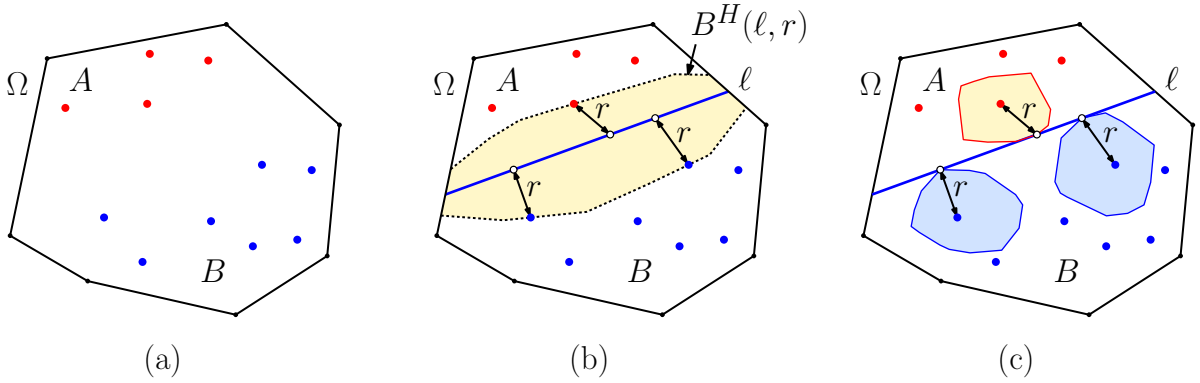


Figure 5.7: The Hilbert SVM problem.

5.4.1 Local Optimality

In $\mathbb{R}^d$, we refer to the hyperplane h that achieves the maximum Hilbert margin as the *Hilbert SVM hyperplane* for A and B, and we define the *Funk SVM hyperplane* analogously for the Funk distance. The following lemma provides a characterization of this hyperplane.

Lemma 5.4. *Given a convex polytope Ω in $\mathbb{R}^d$ and two linearly separable point sets $A, B \subset \Omega$ in general position, the Hilbert (resp., Funk) SVM hyperplane h satisfies the following condition. Let*

r denote the margin achieved by h. There exist $d+1$ pairs $\{(p_i, s_i)\}_{i=0}^{d}$, each consisting of a point $p_i \in A \cup B$ and a half-spoke s_i of p_i. For $0 \le i \le d$, let q_i denote the unique point at Hilbert (resp., Funk) distance r from p_i along s_i. The points $Q = \{q_i\}_{i=0}^{d}$ all lie on h, and furthermore, if we partition Q as $Q_a = Q \cap A$ and $Q_b = Q \cap B$, then $\mathrm{conv}(Q_a) \cap \mathrm{conv}(Q_b) \neq \emptyset$.

Proof. For any $r \ge 0$, let $B^H(A, r) = \bigcup_{a \in A} B^H(a, r)$ denote the set of points of Ω whose distance from any point A is at most r, and define $B^H(B, r)$ similarly for B. As mentioned above, SVM is equivalent to determining the maximum r such that $B^H(A, r)$ and $B^H(B, r)$ are linearly separable. As the value of r increases towards the optimum margin, the set of hyperplanes that separate these sets of balls becomes progressively smaller, until there is a unique separating hyperplane. Because Hilbert balls are convex polytopes, when this critical event occurs, the separating hyperplane is constrained by $d+1$ contact points, which are generally vertices of these balls (see Figure 5.8(a) and (b)).

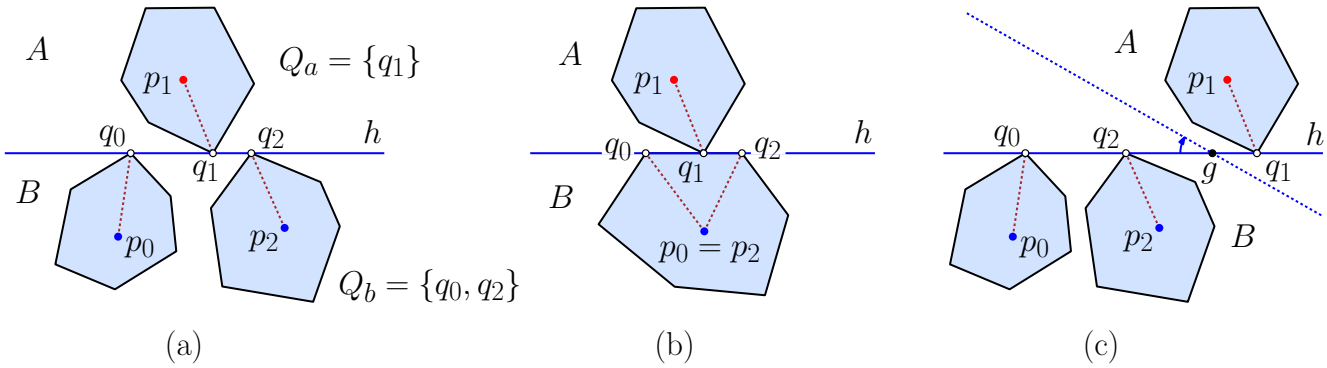


Figure 5.8: Criteria for local optimality for the Hilbert SVM.

By Lemma 5.2, these vertices lie on the half-spokes of points from A and B and each vertex is at distance r from its respective point. Let $\{(p_i, s_i\}_{i=0}^{d}$ denote these point-spoke combinations, and let Q_a and Q_b be as defined in the statement of the lemma. If $\mathrm{conv}(Q_a) \cap \mathrm{conv}(Q_b) = \emptyset$ (see Figure 5.8(c)), then there exists a $(d-2)$-dimensional affine subspace $g \subset h$ that separates them. By rotating h about g, we can break the contact with these metric balls, implying that it would be possible to increase the

margin, a contradiction. □

5.4.2 LP-Type Problems

In this section, we show that our Hilbert and Funk SVM problems possess properties similar to that of low-dimensional linear programming [82], which leads to an efficient algorithmic solution. In particular, we show that these problems have a property called *LP-type* [86]. This type of problem requires a finite set S as input and an objective function f to be maximized, which induces a total ordering over subsets of S. Generally, an LP-type problem is defined as follows.

Definition 5.1 (LP-type [86]). An optimization problem over a set S and objective f is of *LP-type* if:

- Monotonicity: For sets $S' \subseteq S'' \subseteq S$, $f(S') \geq f(S'') \geq f(S)$.
- Locality: For sets $S' \subseteq S'' \subseteq S$ and every $x \in S$, if $f(S') = f(S'') = f(S' \cup \{x\})$, then $f(S') = f(S'' \cup \{x\})$.

A *basis* of an LP-type problem is a set $S' \subseteq S$ with the property that every proper subset of S' has a larger objective-function value than S' itself. The *combinatorial dimension* of an LP-type problem is defined to be the maximum cardinality of a basis. A natural choice for S would be $A \cup B$ and a basis would consist of $d + 1$ points of this set. While this suffices for the simpler Funk distance and the Hilbert distance in the plane, we will need more information for the multidimensional Hilbert SVM problem. In particular, we will use pairs consisting of a point of $A \cup B$ and a half-spoke for this point. As seen in Lemma 5.4, these entities locally define the SVM hyperplane. Given A, B, and Ω, define $V = V_\Omega(A, B)$ to be the set of pairs (p, s), where $p \in A \cup B$, and s is any half-spoke emanating from p. Such a pair is called a *point-spoke*. In the multidimensional setting, the role of

the set S in the definition of LP-type will be played by V. We define the objective function for the multidimensional Hilbert SVM problem as follows. Recall that $\mathrm{Sep}(A, B)$ denotes the set of $(d-1)$-dimensional hyperplanes that separate A and B. Given a hyperplane $h \in \mathrm{Sep}(A, B)$ and a point-spoke $v = (p, s)$, define $d_\Omega^H(v, h)$ to be the Hilbert distance from p to h measured along the half-spoke s. If s does not intersect h, define the distance to be ∞. In the planar case, given any set $S' \subset A \cup B$, we define $f(S')$ to be

$$f(S') \;=\; \sup_{\ell \in \mathrm{Sep}(A,B)} \min_{p \in S'} d_\Omega^H(p, \ell).$$

In the multidimensional case, given any $h \in \mathrm{Sep}(A, B)$ and $V' \subset V$, define

$$d_\Omega^H(V', h) \;=\; \min_{v \in V'} d_\Omega^H(v, h) \qquad \text{and} \qquad f(V') \;=\; \sup_{h \in \mathrm{Sep}(A,B)} d_\Omega^H(V', h).$$

Given $p \in \mathrm{int}(\Omega)$, define $\sigma(p)$ to be the set of half-spokes of $p \in \mathrm{int}(\Omega)$. By convexity of Hilbert balls, for any $p \in A \cup B$ the half-spoke $s \in \sigma(p)$ that defines the distance $d_\Omega^H(p, h)$ to any hyperplane h is the one that minimizes the value of $d_\Omega^H((p, s), h)$. Thus, the objective function for our LP-type problem is the following.

$$f(V) \;=\; \sup_{h \in \mathrm{Sep}(A,B)} d_\Omega^H(V, h) \;=\; \sup_{h \in \mathrm{Sep}(A,B)} \min_{p \in A \cup B} \min_{s \in \sigma(p)} d_\Omega^H((p, s), h).$$

Our next result shows that Hilbert SVM satisfies these conditions.

Lemma 5.5. *Given a convex polytope Ω in $\mathbb{R}^d$ and a set of linearly separable points $A, B \subset \Omega$, the Hilbert SVM problem and the Funk SVM problems are both LP-type problems of combinatorial dimension $d + 1$.*

Proof. We proved in Lemma 5.4 that for either distance measure, any optimal solution is determined by a basis consisting of $d+1$ point-spoke pairs, which implies that the combinatorial dimension is $d+1$.

Monotonicity holds trivially by the fact that adding more point-spokes to a set can never increase the minimum distance from that set to any hyperplane.

To establish locality, suppose that for some sets, $S' \subseteq S'' \subseteq S$, we have $f(S') = f(S'') = f(S' \cup \{x\})$. We assert that the maximum-margin separating hyperplane is unique. To see why, suppose to the contrary that S' and S'' define two distinct separating hyperplanes, h' and h'', respectively, both with equal margins. These hyperplanes intersect in a $(d-2)$-affine subspace g (possibly external to Ω). Every strict convex combination h of these two hyperplanes yields a separating hyperplane that passes through g. Assuming general position (and particularly that no $d+1$ point-spokes are equidistant to g in terms of the Hilbert/Funk distance), it follows that h does not contact all $d+1$ basis point-spokes. Since $d+1$ point-spokes are needed to fix a hyperplane, there exists a perturbation of h that yields a strictly larger margin, a contradiction. Therefore, it follows that the support vectors that form this basis are all members of S'. We may assume that $x \notin S'$, since otherwise the conclusion holds trivially. Since $x \notin S'$, the metric ball centered at x whose radius is equal to the margin does not intersect the separating hyperplane. Therefore, adding x to S'' does not change the margin value, which implies that $f(S') = f(S'' \cup \{x\})$, as desired. □

An alternative combinatorial algorithm for LP-type problems with a slightly better running time was presented by Sharir and Welzl [86]. However Seidel's algorithm [82] is easy to describe and efficient in practice. It takes as input two sets, a set S of inserted points, and a set X (which is initially empty) of elements known to belong to the optimal basis. It then considers the remaining

elements one-by-one in a random order. Whenever a point conflicts with the current optimum (that is, its addition alters the objective function's value) this point is added to X and the algorithm is invoked recursively. The pseudo-code is presented in Algorithm LP-TYPE-SVM.

Algorithm LP-TYPE-SVM — Randomized incremental algorithm for Hilbert SVM

procedure HILBERTSVM(A, B, f, X) ▷ Solve SVM over A, B with basis subset X

$R \leftarrow \emptyset$
$V \leftarrow X$ ▷ V stores the solution basis

for $x \in$ random permutation of $A \cup B$ **do**

if $(f(V) \neq f(V \cup \{x\}))$ **then** ▷ Violation test

$V \leftarrow$ HILBERTSVM($R, f, X \cup \{x\}$) ▷ Add x to basis and recurse

end if

$R \leftarrow R \cup \{x\}$

return V ▷ Return the final basis

end for
end procedure

It can be seen as a consequence of Seidel's algorithm that in some ith iteration of the algorithm, the violation test fails only when x is one of the $d + 1 - |X|$ remaining elements of the basis. The probability of this occurring is at most $\frac{d+1-|X|}{i}$. Since our ground-set, the set of point spoke pairs: V is of size $O(nM^2)$, the analysis from Seidel's paper shows that the overall expected number of violation tests which our algorithm performs is $O(nM^2d!)$ for the Hilbert SVM case.

5.5 Solving the LP-Type Problems

As mentioned above, in any fixed-dimensional space, an LP-type problem over a ground set of size n can be solved in $O(n)$ time assuming free access to two primitives. The first involves evaluating the *objective function* for a given basis set. Second, the *violation test* checks whether a given point causes the objective function to decrease. In this section, we will explore the running times of these primitives.

Given point sets A and B of total size n, recall that the ground set S for the Funk SVM consists of $A \cup B$. Thus, $|S| = n$. For the multidimensional Hilbert SVM, the ground set consists of all point-spoke pairs V. Assuming that Ω has total combinatorial complexity M, $|V| = O(nM)$.

The overall running time to solve the SVM problems will be the product of the size of the ground set and the running time of the two primitives. Our algorithms for computing the objective function will return both the SVM hyperplane h and the margin r as a witness. This implies that the violation tests are relatively easy.

In the case of the Funk distance and the two-dimensional Hilbert distance, we are given a point $p \in S$ to test. In the Funk case, by Lemma 5.3, we can compute the distance from p to h in $O(\log M_0)$ time in $\mathbb{R}^2$ and in $O(M_0)$ time in the multidimensional case.

In the two-dimensional Hilbert case, by Lemma 5.2, we can compute the distance from p to h in $O(\log^2 M)$ time. In all cases, there is a violation if and only if this distance exceeds r.

The remaining case is the multidimensional Hilbert geometry. In this case, we are given a point-spoke pair (p, s) to test. In $O(M)$ time, we can determine the facets hit by the endpoints of the associated chord. We can then determine the point of intersection of this chord with the hyperplane and compute the distance from p to the hyperplane along this spoke in constant time. Again, a violation

occurs if this distance exceeds r. All of these tests can be performed within the time bounds needed for the objective function evaluation, so they do not dominate the overall running time.

5.5.1 The Funk Objective Function

We will first discuss the objective function for the Funk SVM, since it is the easier of the two. In the planar case, Lemma 5.4 implies that a basis consists of three points, say $\{a, b, c\}$. From the lemma, we know that two points must come from one set and one from the other, so there is no loss in generality in assuming that $a, c \in A$ and $b \in B$. Let ℓ be the SVM line we seek. Consider the supporting line of Ω that lies on the opposite side of ℓ as a and c. Let v denote the vertex of Ω incident to this supporting line. By Lemma 5.3(ii), $\mathrm{cp}^F(a, \ell)$ and $\mathrm{cp}^F(c, \ell)$ lie on the segments $\overline{av}$ and $\overline{cv}$, respectively (see Figure 5.9(a)). Since $d^F(a, \ell) = d^F(c, \ell)$, it follows that ℓ is parallel to $\overline{ac}$.

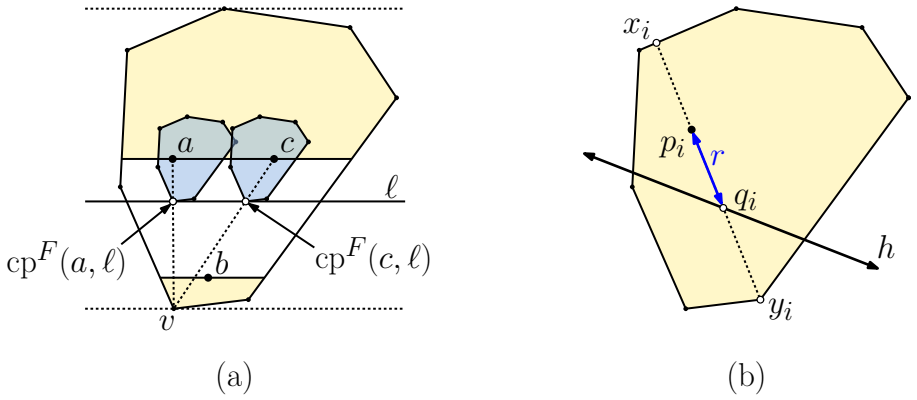


Figure 5.9: Computing the (a) Funk objective function and (b) Hilbert objective function.

Given the two points a and c, we can compute the vertex v in $O(\log m)$ time by binary search on the boundary of Ω. We can compute the parallel supporting line on the opposite side of Ω in $O(\log m)$ time, which is used to compute b's Funk distance. From these two lines, it is straightforward to compute the location of the parallel line ℓ that equalizes the Funk distances among all three points. By combining this with the observations from Section 5.5, we can compute both the violation test and

the objective function for the planar Funk SVM in time $O(\log m)$. Given that the ground set $A \cup B$ has n points, the overall running time is $O(n \log m)$, and Proposition 5.1 follows.

Next, let us consider the d-dimensional case, for any fixed $d \geq 3$. A basis consists of $d+1$ points $\{p_i\}_{i=0}^{d} \subseteq A \cup B$. By Lemma 5.3(i), the points p_i lie on two sides of a slab that is parallel to the desired SVM hyperplane. Let h denote this hyperplane. To compute h, observe that it can be represented as $\{x \in \mathbb{R}^d : \langle x, w \rangle = 1\}$, for some nonzero vector $w \in \mathbb{R}^d$. Thus, there exist reals α and β such that the points p_i above the slab satisfy $\langle p_i, w \rangle - \alpha = 0$ and the points below the slab satisfy $\langle p_i, w \rangle - \beta = 0$. This yields a system of $d+1$ homogeneous linear equations in the $d+2$ variables $\{\alpha, \beta, w_1, \ldots, w_d\}$. We can solve these equations to determine w up to a non-zero scale factor. This fixes the orientation of h, but not its exact location. To compute its location, we first determine the two supporting hyperplanes of Ω parallel to h. This can be done in $O(M_0)$ time, where M_0 denotes the number of vertices of Ω. Given these hyperplanes, we take any basis point of A and any basis point of B, and through a straightforward calculation, in $O(1)$ time we can determine the parallel hyperplane h that equalizes the Funk distance to each.

In summary, assuming that the dimension d is fixed, both the violation test and the objective function can be computed in $O(M_0)$ time. Given that the ground set $A \cup B$ has $O(n)$ points, the overall running time is $O(nM_0)$. This establishes Theorem 5.2(ii).

5.5.2 The Multidimensional Hilbert Objective Function

Next, let us consider how to compute the objective function for the d-dimensional Hilbert case, for any fixed $d \geq 3$. A basis consists of $d+1$ point-spoke pairs $\{p_i, s_i\}_{i=0}^{d}$. We claim that we can compute h in $O(M)$ time.

To see this, observe that h can be represented as $\{x \in \mathbb{R}^d : \langle x, w\rangle = 1\}$ for some nonzero vector $w \in \mathbb{R}^d$. For a given point-spoke pair (p_i, s_i), in $O(M)$ time we can compute the points x_i and y_i, where the extension of the half-spoke intersects $\partial\Omega$. Let us assume this is oriented so that $y_i - x_i$ is in the direction of s_i (see Figure 5.9(b)). For a given distance r, let $q_i = q_i(r)$ denote the point along this spoke that is at Hilbert distance r from p. To simplify the calculation, let $\widehat{r} = \exp(2r)$, and let us assume that the chord $\overline{x_i y_i}$ is affinely mapped to the interval $[0, 1]$ on the real line, which maps p_i and q_i to points in this interval (which we will continue to refer to as p_i and q_i). By Eq. (5.3), for $0 \le i \le d$

$$\widehat{r} \;=\; \frac{q_i}{p_i} \cdot \frac{1 - p_i}{1 - q_i} \quad \text{or equivalently} \quad \widehat{r}(p_i \cdot (1 - q_i)) \;=\; q_i \cdot (1 - p_i).$$

Together with the constraints $\widehat{r} \ge 1$ (from the fact that $r \ge 0$) and $\{0 \le q_i \le 1\}_{i=0}^{d}$ (from the requirement that the q_i's lie in the interval $[0, 1]$), this yields a system of $d + 1$ algebraic equations of degree 2 in the $d + 1$ unknowns $\{\widehat{r}, w_1, \ldots, w_d\}$. (Note that the p_i values are fixed.) Since d is a fixed constant, we can solve this system in $O(1)$ time. If this system has a feasible solution, then the objective function value is $r = \frac{1}{2} \ln \widehat{r}$, and the values of the q_i's determine the hyperplane by specifying the points where the separating hyperplane crosses each of the half-spokes.

The time to compute the objective function is dominated by the $O(M)$ time needed to compute the endpoints of each of the half-spoke extensions. By Lemma 5.2, each point gives rise to $O(M^2)$ half-spokes, which implies that the ground set V is of size $O(nM^2)$. Thus, the overall running time is $O(nM^3)$, and this establishes Theorem 5.1.

5.6 Concluding Remarks and Transition to Chapter 6

In this chapter, we presented an efficient algorithm for solving the SVM problem in the Hilbert and Funk geometries for a set of n points in convex polytopes of combinatorial complexity M. Our results rely on the fact that the Hilbert SVM problem has LP-type structure. However our multidimensional results are polynomial in the total combinatorial complexity of the polytope Ω, which generally scales exponentially with the dimension.

This dependency on M reveals a fundamental limitation of the exact approach: as the complexity of the domain grows (e.g., in high-dimensional probability simplices), exact computation becomes infeasible. The number of vertices on the Hilbert ball grows quadratically in the number of faces of all dimensions of the polytope, making the basis size for our LP-type problem prohibitively large for high-dimensions.

This naturally motivates the need for *approximation*. In the next chapter (Chapter 6), we will abandon the requirement for an exact SVM. Instead, we will introduce a polynomial-time approximation scheme that allows us to trade exactness for speed. We will show that by relaxing the problem to allow for an ϵ-approximate margin, we can solve the Hilbert SVM problem with a complexity that is polynomial in the number of points, the number of bounding hyperplanes of Ω, and d, thereby unlocking high-dimensional applications.

Chapter 6: Improved Classifiers in High Dimensional Hilbert Metrics

6.1 Chapter Outline

In the previous chapter (Chap. 5), we established that the Support Vector Machine (SVM) problem in Hilbert geometry is solvable via LP-type algorithms. However, we also identified a critical limitation: exact solutions rely on the combinatorial complexity of the Hilbert ball, which grows exponentially with dimension. This renders the exact approach infeasible for high-dimensional applications, such as classifying probability distributions over large domains.

In this chapter, we overcome this "curse of dimensionality." We shift our strategy from *combinatorial geometry* to *convex optimization*. By utilizing *Birkhoff's formulation* of the Hilbert metric, we show that the SVM problem can be modeled as a series of linear programming feasibility tests.

We present an approximation algorithm that computes a large-margin separator in time polynomial in the number of points, the number of bounding facets, and the dimension. This is a significant improvement over the exact algorithms of Chapter 5. We also extend this efficient framework to the Funk metric, introduce *soft-margin classifiers* for non-separable data, and propose a nearest-neighbor classifier for the embedding space.

6.2 Introduction

In machine learning, *binary classification* is a supervised learning task where an algorithm learns to map input data to one of two mutually exclusive classes. The primary objective is to train a model that can accurately predict which of the two predefined classes a new, unseen data point belongs to. Binary classification is one of the most fundamental geometric problems in machine learning [18, 31, 80, 87]. As detailed in Chapter 5, Support Vector Machines (SVMs) [29, 90] solve this by computing a hyperplane that separates training sets P^+ and P^- with a maximum *margin*.

While Chapter 5 explored the theoretical structure of SVMs in Hilbert geometry, this chapter focuses on *scalability*. SVMs are widely used in high-dimensional fields such as computer vision [57, 88], natural language processing [34, 51], and computational biology [13, 48]. In these domains, the dimensionality d of the feature space is often large. The algorithms developed in Chapter 5, which depend on the complexity of the Hilbert ball's boundary ($O(m^2)$ vertices in the plane, but exploding in high dimensions), are impractical here.

To make classification in such spaces practical, we abandon the search for an exact basis (the LP-type approach). Instead, we treat the problem as numerical optimization. By approximating the margin to within an additive error ϵ, we derive an algorithm with a runtime that is polynomial in the dimension d. To that effect, we leverage a different characterization of the Hilbert metric.

6.2.1 Related Work and Context

In the Euclidean metric, SVMs are typically solved by formulating them as convex optimization problems, specifically quadratic programming, which efficiently maximize the margin between classes [14, 23]. This formulation frequently exploits Lagrangian duality, allowing the problem to be

expressed purely in terms of inner products, which can then be elegantly generalized via the kernel trick [47]. However, generalizing this dual formulation to non-Euclidean spaces, such as hyperbolic or Hilbert geometries, is highly non-trivial. Substituting the Euclidean inner product or distance with a non-Euclidean metric generally destroys the positive semi-definiteness of the kernel or the convexity of the objective function. Consequently, the standard trick of solving the dual problem gives rise to a non-convex optimization problem when applied naively to these geometries.

Because of this inherent non-convexity, previous attempts to formulate SVMs and linear classifiers in hyperbolic spaces have faced significant computational hurdles. Cho *et al.* [27] were among the first to consider large-margin classification in hyperbolic space. While they demonstrated that their approach significantly outperforms Euclidean classifiers on hierarchical benchmarks by respecting the data's intrinsic geometry, their method directly optimized a non-convex margin objective using Riemannian gradient descent. Without careful initialization, such approaches are highly susceptible to local minima and provide neither a global optimum guarantee nor a worst-case polynomial running time. Other continuous optimization techniques for representation learning in hyperbolic spaces similarly rely on iterative local methods lacking rigorous theoretical convergence guarantees [38, 76]. Furthermore, as highlighted by Mishne *et al.* [68], training hyperbolic learning models via Riemannian optimization frequently suffers from severe numerical instability and convergence issues, leading to catastrophic floating-point failures. By formulating our Hilbert SVM strictly as a sequence of LP-based problems, our approach completely obviates these Riemannian convergence and numerical stability bottlenecks.

Another potential avenue for solving SVMs in the Hilbert metric is to exploit isometric embeddings. Extending results of de la Harpe [33], Vernicos [92] presented an elegant isometry mapping the polytopal Hilbert geometry into a finite-dimensional normed vector space. While one might be

tempted to map the input points into this embedding space to solve a standard SVM, this approach is fundamentally flawed for linear classification. Because the mapping is highly non-linear, a simple linear classifier (a hyperplane) computed in the embedding space does not translate back to a linear hyperplane in the original input space. Furthermore, computing the complex intersection of the resulting decision boundary with the feasible image of the polytope is analytically and computationally impractical.

As demonstrated in *Chapter 5*, the Hilbert SVM problem can be modeled as an LP-type problem [2]. While that formulation proved that the problem is computationally tractable in fixed dimensions, the algorithm's running time grows polynomially with the total complexity of the polytope defining the metric. Unfortunately, the total complexity of this polytope generally grows exponentially in the dimension of the space. As such, the exact method suffers from the curse of "dimensionality", remaining unsuitable for high dimensions, where most applications in machine learning lie. In contrast, in this chapter, our running times are polynomial in dimension.

6.2.2 Our Contributions

In this chapter, we explore an alternative algebraic viewpoint of Hilbert geometry to develop an efficient, globally approximate algorithm for the Hilbert SVM problem. The defining feature of our approach is that the running time scales only *polynomially* with the dimension of the space, successfully circumventing the exponential bottleneck that plagues prior vertex-based methods while avoiding the local-minima pitfalls of Riemannian optimization.

Rather than relying on non-convex gradient descent or explicit vertex enumeration, our algorithm leverages Birkhoff's formulation of the Hilbert metric to reduce the maximum-margin problem

to a sequence of linear programming (LP) feasibility tests. Because we rely on numerical LP solvers, we assume the input coordinates and bounding hyperplane coefficients are represented as bounded B-bit rational numbers, and we compute the maximum margin to within a user-supplied additive approximation error ε.

The following theorem formalizes our main result:

Theorem 6.1. *Consider a polytope Ω in $\mathbb{R}^d$ defined by m hyperplanes, and two linearly separable point sets P^+ and P^- of total size n contained within Ω's interior. Assume further that all of the defining coordinates and coefficients are representable as B-bit rational numbers. Then there exists an algorithm running in time $O(\text{poly}\,(n, m, d, B, \log(1/\varepsilon))$ that computes a separating hyperplane whose margin in the Hilbert metric defined by Ω is within an additive error ε of the optimum.*

This polynomial dependence on the ambient dimension d is a crucial advancement for modern machine learning applications. Furthermore, our framework is highly extensible:

- *Asymmetric Metrics:* We generalize our LP-feasibility results to the closely related asymmetric Funk geometry, which is useful for directed spaces and specific optimal transport problems (Section 6.3.1).

- *Soft-Margin Classification:* To handle real-world datasets that contain noise or are not perfectly linearly separable, we provide a robust soft-margin formulation for the Hilbert SVM that scales penalties based on boundary proximity (Section 6.5.1).

- *Nearest-Neighbor Classifiers:* While isometric embeddings fail for hyperplane SVMs, we leverage Vernicos' mapping to design an efficient approximate nearest-neighbor-based classifier for non-linear decision boundaries (Section 6.5.2).

6.3 Mathematical Preliminaries

In this section, we provide the specific definitions and facts utilized for the optimization framework. We will use different fonts, specifically p and $\mathbf{p}$, to distinguish between a point $p \in \mathbb{R}^d$ and its associated d-dimensional column vector $\mathbf{p}$. We represent a $(d-1)$-dimensional flat, that is a hyperplane, L in $\mathbb{R}^d$ as a pair $(\mathbf{w}, c) \in \mathbb{R}^d \times \mathbb{R}$ represented by the function $f_L(x) = \mathbf{w}^\top \mathbf{x} + c$. A hyperplane partitions space into three sets, L^-, L, L^+, where a point $\mathbf{x} \in \mathbb{R}^d$ belongs to each of these sets depending on whether $f_L(x)$ is < 0, $= 0$, or > 0, respectively. Given a set of m hyperplanes, $\mathcal{L} = \{L_i : (\mathbf{w}_i, c_i)\}_{i=1}^m$ let $\Omega = \Omega(\mathcal{L})$ denote the closure of the space bounded by the positive halfspaces L_i^+, that is, $\Omega = \mathrm{cl}\big(\bigcap_{i=1}^m L_i^+\big)$. Let $\partial\Omega$ denote the boundary of Ω. Throughout, we assume that Ω is bounded and full dimensional. Given a nonnegative integer m, let $[m]$ denote the index set $\{1, \ldots, m\}$. Given two points $p, q \in \mathrm{int}(\Omega)$, let $\overline{pq}$ denote the *chord* of Ω passing through these points, that is, the intersection of Ω with the line through p and q. Let $\overleftrightarrow{pq}$ denote the line that passes through them, and $\overrightarrow{pq}$ denote the ray starting from p and passing through q. Given $p, q \in \mathbb{R}^d$, let $d(p, q)$ denote their Euclidean distance. Given a hyperplane L, let $d(p, L)$ denote the minimum Euclidean (perpendicular) distance from p to L.

6.3.1 Hilbert Geometry Definitions

For completeness, we restate the definitions of the Funk and Hilbert distances, which were introduced in Chapter 5. Given a convex body Ω and any two points $p, q \in \mathrm{int}(\Omega)$, if $p = q$ all the distances are defined to be zero. Otherwise, let p' and q' denote the endpoints of the chord $\overline{pq}$ ordered

as $\langle p', p, q, q' \rangle$ (see Figure 6.1). The Funk and Hilbert distances are defined:

$$\text{Funk:} \quad d_\Omega^F(p,q) := \ln \frac{\|p-q'\|}{\|q-q'\|} \tag{6.1}$$

$$\text{Reverse Funk:} \quad d_\Omega^{rF}(p,q) := d_\Omega^F(q,p) = \ln \frac{\|q-p'\|}{\|p-p'\|} \tag{6.2}$$

$$\text{Hilbert:} \quad d_\Omega^H(p,q) := \frac{d_\Omega^F(p,q) + d_\Omega^{rF}(p,q)}{2} = \frac{1}{2} \ln \frac{\|q-p'\|}{\|p-p'\|} \frac{\|p-q'\|}{\|q-q'\|}. \tag{6.3}$$

Intuitively, $d_\Omega^F(p,q)$ is a measure of how far q is from p relative to the boundary that lies beyond q, and $d_\Omega^{rF}(p,q)$ is just the reverse of this. The Hilbert distance $d_\Omega^H(p,q)$ symmetrizes these by taking their arithmetic mean.

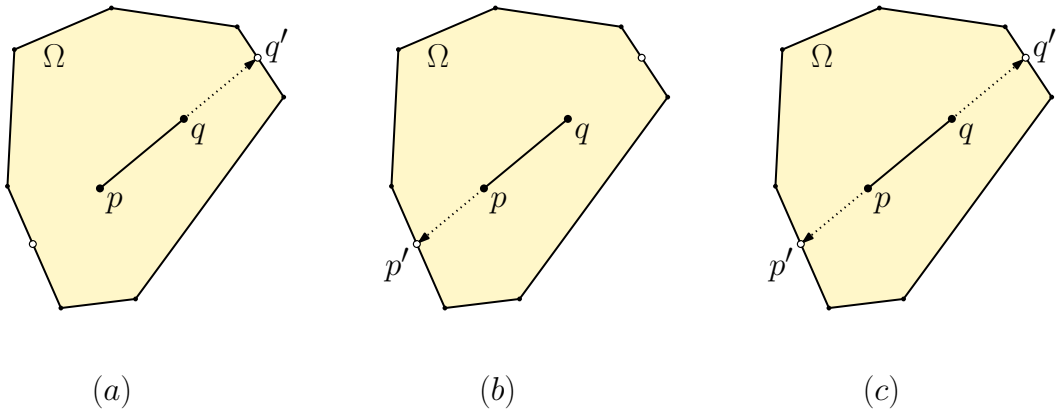


Figure 6.1: The (a) forward Funk, (b) reverse Funk, (c) and Hilbert distances between p and q in Ω.

These distance functions are all nonnegative and satisfy the triangle inequality [75]. The Hilbert distance defines a *metric* over $\text{int}(\Omega)$. The Funk and reverse Funk are asymmetric and hence define *weak metrics* over $\text{int}(\Omega)$. Observe that as either point approaches the boundary of Ω, the Hilbert distance approaches ∞. The Hilbert distance is invariant under invertible projective transformations.

6.3.2 Birkhoff's Formulation and Metric Balls

While Chapter 5 relied on the cross-ratio definition of the metric, here we utilize an alternative, equivalent definition provided by Birkhoff [17]. This formulation expresses distances in terms of the bounding hyperplanes of Ω, which allows us to formulate the SVM problem as a linear program.

Proposition 6.1 (Birkhoff's Formulation)**.** *Given a set of m hyperplanes, $\mathcal{L} = \{L_i : (\mathbf{w}_i, c_i)\}_{i=1}^m$, then for any pair of points $p, q \in \text{int}(\Omega(\mathcal{L}))$:*

$$\begin{aligned}
d_\Omega^F(p,q) &= \max_{i\in[m]} \log \frac{d(p,L_i)}{d(q,L_i)} = \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i} \\
d_\Omega^{rF}(p,q) &= \max_{i\in[m]} \log \frac{d(q,L_i)}{d(p,L_i)} = \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{q} + c_i}{\mathbf{w}_i^\top \mathbf{p} + c_i} \\
d_\Omega^H(p,q) &= \frac{1}{2} \max_{j,k\in[m]} \log \frac{d(p,L_j)}{d(q,L_j)} \frac{d(q,L_k)}{d(p,L_k)} = \frac{1}{2} \max_{j,k\in[m]} \log \left(\frac{\mathbf{w}_j^\top \mathbf{p} + c_j}{\mathbf{w}_j^\top \mathbf{q} + c_j} \frac{\mathbf{w}_k^\top \mathbf{q} + c_k}{\mathbf{w}_k^\top \mathbf{p} + c_k} \right).
\end{aligned}$$

This follows from the fact that Ω is convex, and observing that if $\overrightarrow{pq}$ intersects $\partial\Omega$ in q', on the bounding hyperplane L_k, then by similar triangles

$$\frac{d(p,q')}{d(q,q')} = \frac{d(p,L_i)}{d(q,L_i)}.$$

Intuitively, this provides a way to define the metrics without explicitly calculating where the line $\overleftrightarrow{pq}$ intersects $\partial\Omega$. One consequence of the proposition is that the metrics can be reduced to a sup norm over $\mathbb{R}^t$ for some finite dimension t, using a non-linear mapping function [62]. The other useful consequence is its application in constructing metric balls in high dimensions. Letting d_Ω^* denote any of the above metrics and given a point $p \in \text{int}(\Omega)$ and scalar $r \geq 0$, the associated metric balls are

defined in the standard way.

$$B^*_\Omega(p,r) \;:=\; \{q \in \Omega \;:\; d^*_\Omega(p,q) \le r\}.$$

It is well known that the Funk metric about a point p is a scaling of Ω about p by an appropriate factor that depends on the radius. This can be derived directly from Proposition 6.1:

$$\begin{aligned}
B^F_\Omega(p,r) &= \left\{q \in \Omega \;:\; d^F_\Omega(p,q) \le r\right\} \\
&= \left\{q \in \Omega \;:\; \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i} \le r\right\} \\
&= \left\{q \in \Omega \;:\; \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i} \le r,\ \forall i \in [m]\right\} \\
&= \left\{q \in \Omega \;:\; 0 \le \mathbf{w}_i^\top \mathbf{q} + c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i),\ \forall i \in [m]\right\} \\
&= \bigcap_{i\in[m]} \left\{q \in \Omega \;:\; 0 \le \mathbf{w}_i^\top \mathbf{q} + c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i)\right\}. \qquad (6.4)
\end{aligned}$$

Fixing p and r, let $c_i^* := c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i)$, and let S_i^F be the hyperplane represented by $(\mathbf{w}_i, c_i^*)$ (See Figure 6.2(a)). Observe that S_i^F is parallel to L_i. We have the following:

Lemma 6.1 (Funk Balls)**.** *Given a set of m hyperplanes, $\mathcal{L}$ as in Proposition 6.1, the Funk ball $B^F_{\Omega(\mathcal{L})}(p,r)$ is a polytope bounded by the m hyperplanes, $S_i^F \;:\; (\mathbf{w}_i, c_i^*)$, for $i \in [m]$.*

A similar result can be proved for reverse-Funk balls (see Figure 6.2(b)). We next use Proposition 6.1 to demonstrate the Hilbert balls are convex polytopes, but the number of bounding facets is

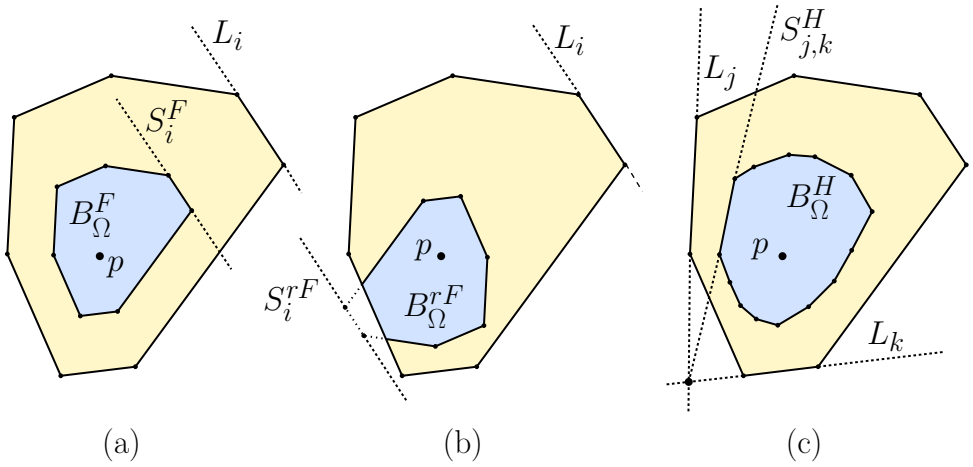


Figure 6.2: Defining balls and their bounding hyperplanes for (a) Funk, (b) Reverse Funk, and (c) Hilbert.

larger by the square. Following the same reasoning as in Eq. (6.4), we have

$$\begin{aligned} B_\Omega^H(p,r) &= \{q \in \Omega \,:\, d_\Omega^H(p,q) \le r\} \\ &= \left\{ q \in \Omega \,:\, \frac{1}{2} \max_{j,k\in[m]} \log \left(\frac{\mathbf{w}_j^\top \mathbf{p} + c_j}{\mathbf{w}_j^\top \mathbf{q} + c_j} \frac{\mathbf{w}_k^\top \mathbf{q} + c_k}{\mathbf{w}_k^\top \mathbf{p} + c_k} \right) \le r \right\} \\ &= \bigcap_{j,k\in[m]} \left\{ q \in \Omega \,:\, \left(\mathbf{w}_j^\top \mathbf{p} + c_j\right)\left(\mathbf{w}_k^\top \mathbf{q} + c_k\right) \le e^{2r} \left(\mathbf{w}_k^\top \mathbf{p} + c_k\right)\left(\mathbf{w}_j^\top \mathbf{q} + c_j\right) \right\}. \end{aligned}$$

Fixing p and r, let $\alpha_{j,k} := e^{2r}\left(\mathbf{w}_k^\top \mathbf{p} + c_k\right)$ and $\alpha'_{j,k} := \left(\mathbf{w}_j^\top \mathbf{p} + c_j\right)$. We have

$$B_\Omega^H(p,r) = \bigcap_{j,k\in[m]} \left\{ q \in \Omega \,:\, 0 \le \left(\alpha_{j,k}\mathbf{w}_j - \alpha'_{j,k}\mathbf{w}_k\right)^\top \mathbf{q} + \left(\alpha_{j,k}c_j - \alpha'_{j,k}c_k\right) \right\}. \tag{6.5}$$

Let $\mathbf{w}^*_{j,k} = \left(\alpha_{j,k}\mathbf{w}_j - \alpha'_{j,k}\mathbf{w}_k\right)$, and $c^*_{j,k} = \left(\alpha_{j,k}c_j - \alpha'_{j,k}c_k\right)$. Let $S^H_{j,k}$ be the hyperplane represented by $(\mathbf{w}^*_{j,k}, c^*_{j,k})$. Observe that $S^H_{j,k}$ is a linear combination of L_j, and L_k, and therefore $S^H_{j,k}$ passes through the intersection of L_j and L_k (see Figure 6.2(c)). In summary, we have the following: Setting $S^H_{j,k} \,:\, (\mathbf{w}_{j,k}, c_{j,k})$, we observe that $S^H_{j,k}$ is a linear combination of $L_j \,:\, (\mathbf{w}_j, c_j)$, and $L_k \,:\, (\mathbf{w}_k, c_k)$. Therefore $L_j \cap L_k \subset S^H_{j,k}$, that is $S^H_{j,k}$ passes through the intersection of L_j and L_k. See Figure 6.2(c).

Therefore we have the following:

Lemma 6.2 (Hilbert Balls)**.** *Given a set of m hyperplanes, $\mathcal{L}$ as in Proposition 6.1, the Hilbert ball $B^H_{\Omega(\mathcal{L})}(p, r)$ is a polytope bounded by the at most $m(m-1)$ hyperplanes: $S^H_{j,k}$, for $j, k \in [m]$.*

6.4 Hilbert SVM

In this section, we present our solution to the SVM problem in the Hilbert geometry. Recall that our objective is to compute a maximum-margin hyperplane (with respect to the Hilbert distance) that separates two point sets P^+ and P^- of total size n. We are given an index set $I = \{1, \ldots, n\}$, and a partition of I into I^+, and I^-, that is $I^+ \cup I^- = I$, $I^+ \cap I^- = \emptyset$, along with the associated d-dimensional point sets: $P^{\pm} = \{p_i : i \in I\}$, $P^+ = \{p_i : i \in I^+\}$, $P^- = \{p_i : i \in I^-\}$. Throughout this section, we assume P^+ and P^- are linearly separable, that is, there is a hyperplane K such that $P^+ \subset K^+$, and $P^- \subset K^-$. (In Section 6.5.1, we will consider the general case.)

Define the Hilbert distance of a point $p \in \Omega$ to a hyperplane L to be $d^H_\Omega(L, p) = \min_{x \in L} d^H_\Omega(x, p)$. We define the *margin* of P^+ and P^- with respect to a separating L as $\min_{p \in P^{\pm}} d^H_\Omega(L, p)$, the minimum distance of any point to L. Finding the maximum margin of any separating hyperplane can be expressed formally as

$$\max\big\{\gamma : \exists K \text{ such that } P^+ \subset K^+, P^- \subset K^-, \text{ and } d^H_\Omega(K, p) \geq \gamma,\ \forall p \in P^{\pm}\big\}. \tag{6.6}$$

Our algorithm will yield an absolute approximation to the optimal solution. In particular, given an *approximation parameter* $\varepsilon > 0$, our algorithm will produce a separating hyperplane that achieves a margin of at least $\gamma - \varepsilon$, where γ is the optimal margin. Assuming K is represented by the pair $(\mathbf{w}, c)$,

this can be expressed equivalently as the following optimization problem:

$$\begin{aligned} \text{maximize:} \quad & \gamma \\ \text{subject to:} \quad & \mathbf{w}^\top \mathbf{p} + c \ > \ 0, \quad \forall p \in P^+ \\ & \mathbf{w}^\top \mathbf{p} + c \ < \ 0, \quad \forall p \in P^- \\ & d_\Omega^H(K, p) \ \geq \ \gamma, \quad \forall p \in P^\pm, \quad K \ : \ (\mathbf{w}, c). \end{aligned} \tag{6.7}$$

It is not immediately straightforward as to how to solve Opt. (6.7). One difficulty lies in finding a reasonable representation for $d_\Omega^H(K, p)$, the Hilbert distance of a point to a hyperplane. This was done in Chapter 5, but the size of the representation given there depended on the number of vertices in the Hilbert ball, which grows exponentially with the dimension. Other ways of formulating the SVM problem in hyperbolic spaces usually mirror that in the Euclidean setting, framing it as a non-convex optimization problem, with no theoretical guarantees on the runtime [27]. However, by exploiting the geometry of the polytopal Hilbert metric and using the alternative viewpoints of the metrics as presented in Proposition 6.1, we are able to significantly improve upon the prior attempts. We attempt to solve the problem via series of LP-feasibility problems over a range of values of γ. If the input parameters are given to us as B-bit rational numbers, we show that we only have to solve $O\left(L \log(1/\varepsilon)\right)$ many LPs, where L is the total bit size of the input.

6.4.1 Overall Algorithm

We assume every input parameter is given to us as a rational number, expressed as a fraction of two nonzero B-bit integers. We call this representation a *B-bit rational number*. Each of the n points

$p \in P^{\pm}$ is represented as a d-vector $\mathbf{p}$ whose coordinates are B-bit rational numbers. The domain Ω is defined by a set $\mathcal{L}$ of m hyperplanes, each of whose $(\mathbf{w}, c)$ representation is given as a sequence of $d+1$ B-bit rational numbers. We say that the resulting SVM problem instance has *size parameters* (d, n, m, B). The total *bit complexity* of such an instance is $O(d(n+m)B)$. Before describing the overall algorithm we first establish an upper-bound on γ the maximum margin in terms of the bit size of the input. Let K be the corresponding separating hyperplane for which γ is achieved. Observe that for an arbitrary $p_+ \in P^+$, and $p_- \in P^-$, we have $d^H_\Omega(p_-, p_+) \geq d^H_\Omega(p_-, K) + d^H_\Omega(p_+, K) \geq 2\gamma$. Now let the line joining p_- and p_+ intersect $\partial\Omega$ in bounding hyperplanes $L_- : (\mathbf{w}_-, c_-)$, and $L_+ : (\mathbf{w}_+, c_+)$. Therefore,

$$d^H_\Omega(p_-, p_+) \;=\; \frac{1}{2}\log\frac{d\,(p_+, L_-)\,d\,(p_-, L_+)}{d\,(p_+, L_+)\,d\,(p_-, L_-)} \;=\; \frac{1}{2}\log\frac{\left(\mathbf{w}_-^\top\,\mathbf{p}_+ + c_-\right)\left(\mathbf{w}_+^\top\,\mathbf{p}_- + c_+\right)}{\left(\mathbf{w}_-^\top\,\mathbf{p}_- + c_-\right)\left(\mathbf{w}_+^\top\,\mathbf{p}_+ + c_+\right)}.$$

Consider any one of the terms in the numerator, say, $(\mathbf{w}_-^\top\,\mathbf{p}_+ + c_-)$. Now, $\mathbf{w}_-$, and p_+ are given by d B-bit rational numbers, therefore their inner product can have an absolute value of at most $d \cdot 2^{2B}$. Hence, the entire term $\mathbf{w}_-^\top\,\mathbf{p}_+ + c_-$ has an absolute value at most $d \cdot 2^{2B} + 2^B$. Recalling that the Hilbert distance is the logarithm of ratios, up to constant factors, $d^H_\Omega(p_-, p_+)$ is at most $(8B + 4\log d)\log 2 = O(B + \log d)$, implying that $\gamma \leq M \in O(B + \log d)$. Thus, we have:

Lemma 6.3. *Given an SVM instance with size parameters (d, n, m, B), the separation margin γ for any hyperplane is at most $M(d, n, m, B) = O(B + \log d)$.*

Our overall algorithm is as follows. Since we are interested in the optimum margin up to an accuracy of ε, we check if P^+ and P^- are separable with a margin of at least r, where r is an element from the range: $\Gamma = [\varepsilon, 2\varepsilon, \ldots, M]$, where $M = M(d, n, m, B)$. To check separation for a particular value r we will conduct an (approximate) LP-based feasibility test. We employ this test as part of

a parametric binary search over the range Γ. This implies that we only need to solve $O(\log B + \log\log d + \log(1/\varepsilon))$ many feasibility tests. We next describe this LP-based subroutine.

6.4.2 LP-feasibility

We consider the problem of determining whether it is possible to achieve a given margin γ, up to an additive error of ε. For a particular margin r, recall that $B_\Omega^H(p, r)$ is the Hilbert ball of radius r centered at p. By Lemma 6.2, $B_\Omega^H(p, r)$ is a polytope bounded by $m(m-1)$ hyperplanes. (In Section 6.4.3, we will show how to construct them efficiently.) Let $\mathcal{B}(r)^+ = \{B_\Omega^H(p^+, r) \mid p^+ \in P^+\}$, the collection of the Hilbert balls of radius r around the positively labeled points, and define $\mathcal{B}(r)^-$ analogously for P^-. The feasibility of γ reduces to determining whether the collections of balls $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ are linearly separable. For $i \in [n]$, let $\mathbf{A}(r, i)$ be an $m(m-1) \times d$ matrix and let $\mathbf{b}(r, i)$ be an $m(m-1) \times 1$ vector such that

$$\mathbf{x} \in B_\Omega^H(p_i, r) \iff \mathbf{A}(r, i)\mathbf{x} + \mathbf{b}(r, i) \geq 0.$$

We can construct $\mathbf{A}(r, i)$ and $\mathbf{b}(r, i)$ from the hyperplanes $S_{j,k}^H : (\mathbf{w}_{j,k}^*, c_{j,k}^*)$ described in Lemma 6.2. In particular, for $t \in [m(m-1)]$, if $(\mathbf{w}_t^*, c_t^*)$ denotes the t-th bounding hyperplane of $B_\Omega^H(p_i, r)$, then $\mathbf{w}_t^*$ is the t-th row of $\mathbf{A}(r, i)$ and c_t^* is the t-th element of $\mathbf{b}(r, i)$. Therefore, finding whether $\mathcal{B}(r)^+$

and $\mathcal{B}(r)^-$ are linearly separable is equivalent to the following decision problem:

$$\exists K \,:\, (\mathbf{w}, c), \quad \text{such that}$$

$$\forall p_i \in P^+, \qquad \forall \mathbf{x} : \mathbf{A}(r,i)\mathbf{x} + \mathbf{b}(r,i) \geq 0, \quad \mathbf{w}^\top \mathbf{x} + c \geq 0, \tag{6.8}$$

$$\forall p_i \in P^-, \qquad \forall \mathbf{x} : \mathbf{A}(r,i)\mathbf{x} + \mathbf{b}(r,i) \geq 0, \quad \mathbf{w}^\top \mathbf{x} + c \leq 0. \tag{6.9}$$

For fixed r and i, let $\mathbf{A} = \mathbf{A}(r,i)$ and $\mathbf{b} = \mathbf{b}(r,i)$. Eq. (6.8) can be rewritten as:

$$\min_{\mathbf{x}:\, \mathbf{A}\mathbf{x}+\mathbf{b} \geq 0} \mathbf{w}^\top \mathbf{x} + c \;\geq\; 0.$$

By standard LP duality [39], we have

$$\min_{\mathbf{x}:\, \mathbf{A}\mathbf{x}+\mathbf{b} \geq 0} \mathbf{w}^\top \mathbf{x} \quad = \quad \max_{\mathbf{y}:\, \mathbf{A}^\top \mathbf{y} = \mathbf{w},\, \mathbf{y} \geq \mathbf{0}} -\mathbf{b}^\top \mathbf{y}.$$

Therefore we can simplify Eq. (6.8) as follows:

$$\begin{aligned}
&\forall \mathbf{x} : \mathbf{A}\mathbf{x} + \mathbf{b} \geq 0, \quad \mathbf{w}^\top \mathbf{x} + c \;\geq\; 0 \\
&\quad \Longleftrightarrow \min_{\mathbf{x}:\, \mathbf{A}\mathbf{x}+\mathbf{b} \geq 0} \mathbf{w}^\top \mathbf{x} + c \;\geq\; 0 \\
&\quad \Longleftrightarrow \max_{\mathbf{y}:\, \mathbf{A}^\top \mathbf{y} = \mathbf{w},\, \mathbf{y} \geq \mathbf{0}} -\mathbf{b}^\top \mathbf{y} + c \;\geq\; 0 \\
&\quad \Longleftrightarrow \exists\, \mathbf{y}(i) :\; \mathbf{A}(r,i)^\top \mathbf{y}(i) \;=\; \mathbf{w}, \;\; \mathbf{y}(i) \geq \mathbf{0}, \;\; \mathbf{b}(r,i)^\top \mathbf{y}(i) \leq c.
\end{aligned}$$

Analogously, Eq. (6.9) can be simplified to

$$\exists\, \mathbf{y}(i):\ \mathbf{A}(r,i)^\top \mathbf{y}(i) \ =\ \mathbf{w},\ \ \mathbf{y}(i) \geq \mathbf{0},\ \ \mathbf{b}(r,i)^\top \mathbf{y}(i) \geq c.$$

Hence, to determine whether the collections $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ are linearly separable we solve the following LP-feasibility problem in the parameters $\mathbf{w}_{d\times 1}$, c, and $\mathbf{y}(i)_{m(m-1)\times 1}$, for $i \in [n]$.

$$\begin{aligned}
\text{Minimize:} \qquad & 0 \\
\text{Subject to:} \qquad & \mathbf{b}(r,i)^\top \mathbf{y}(i) \ \leq\ c, \quad \forall i \in I^+ \\
& \mathbf{b}(r,i)^\top \mathbf{y}(i) \ \geq\ c, \quad \forall i \in I^- \\
& \mathbf{A}(r,i)^\top \mathbf{y}(i) \ =\ \mathbf{w}, \quad \forall i \in I \\
& \mathbf{y}(i) \ \geq\ \mathbf{0}, \quad \forall i \in I \\
& \mathbf{1}^\top \mathbf{w} \ =\ 1.
\end{aligned} \tag{6.10}$$

We add the final constraint to avoid the trivial solution in which all the parameters are zero.

Remark 6.1 (Intuitive Description). Opt. (6.10) can be understood intuitively by observing the following. $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ being linearly separable implies there exists a hyperplane $K : (\mathbf{w}, c)$, such that for every $B^+ \in \mathcal{B}(r)^+$, there is a hyperplane K_{B^+} that is parallel to and above K, and is tangent to some ball in B^+. Similarly, for every $B^- \in \mathcal{B}(r)^-$, there is a hyperplane K_{B^-} that is parallel to and below K, and is tangent to some ball in B^- (see Figure 6.3).

A hyperplane L that is tangent to a polytope Ω, with bounding hyperplanes $L_1, \ldots, L_m$, can be written as a positive linear combination (often referred to as the conical combination) of the bounding hyperplanes. The vector $\mathbf{y}(i)$ in Opt. (6.10) corresponds to the factors for this conical combination.

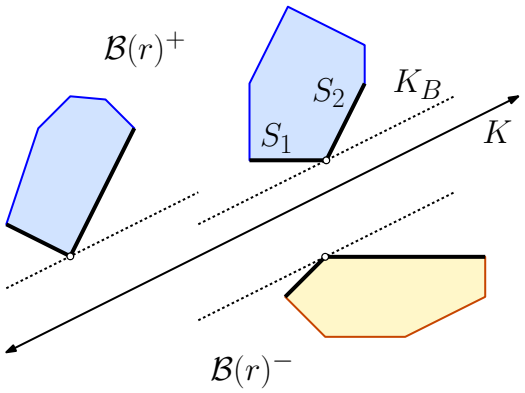


Figure 6.3: Intuitive description of the dual: K separates $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$. K_B is a conical combination of S_1 and S_2 while lying strictly above K.

The third constraint in Opt. (6.10) guarantees a hyperplane K_B parallel to K, along with being a conical combination of the supporting hyperplanes for each $B \in \mathcal{B}(r)^+ \cup \mathcal{B}(r)^-$. The first two constraints ensure K_B is above or below K, depending on whether $B \in \mathcal{B}(r)^+$ or $\mathcal{B}(r)^-$.

Remark 6.2 (LP for Funk). The LP feasibility formulation for the Funk metric is similar to Opt. (6.10). We briefly mention the required modifications. By the construction given in Lemma 6.1, for $i \in [n]$, let $\mathbf{A}(r,i)$ be an $m \times d$ matrix and let $\mathbf{b}(r,i)$ be an $m \times 1$ vector such that

$$\mathbf{x} \in B^F_\Omega(p_i, r) \iff \mathbf{A}(r,i)\mathbf{x} + \mathbf{b}(r,i) \geq 0.$$

For this case, $\mathbf{y}(i)$ will be an $m \times 1$ vector. The remainder of the analysis and formulation is exactly the same as in the case of the Hilbert metric.

6.4.3 Constructing Hilbert Balls

In this section we present a procedure for constructing the bounding hyperplanes for a Hilbert ball, $B^H_\Omega(p_i, r)$. Recall that $\mathcal{L} = \{L_i\}_{i=1}^m$ denotes the set of hyperplanes bounding Ω. First, we take two bounding hyperplanes of Ω: $L_j : (\mathbf{w}_j, c_j)$ and $L_k : (\mathbf{w}_k, c_k)$, $j \neq k$, $j, k \in [m]$. For $p_i \in P^\pm$, we

find $S(i,r)_{j,k} \ : \ (\mathbf{u}(i,r)_{(j,k)}, t(i,r)_{j,k})$, the hyperplane at a Hilbert-distance of r from p_i with respect to the boundary defined by L_j and L_k, such that there exists a line ℓ that intersects L_j, $S(i,r)_{j,k}$, p_i, and L_k in that order (see Figure 6.4).

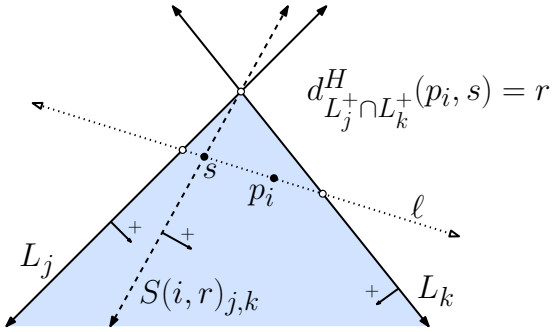


Figure 6.4: Constructing $B^H_\Omega(p_i, r)$, the Hilbert ball of radius r around p_i in Ω

It is clear that $\exists\, \alpha : 0 \leq \alpha \leq 1$, such that

$$S(i,r)_{j,k} \ = \ \alpha L_j - (1-\alpha) L_k. \tag{6.11}$$

For any point s on $S(i,r)_{j,k}$, $s \in L_j^+ \cap L_k^+$, it is easy to see that

$$\frac{d(s, L_k)}{d(s, L_j)} \ = \ \frac{\alpha}{1-\alpha}.$$

Solving for α using the following:

$$\exp(2 \cdot d^H_\Omega\left(p_i, S(i,r)_{j,k}\right)) \ = \ \frac{\mathbf{w}_j^\top \mathbf{p}_i + c_j}{\mathbf{w}_k^\top \mathbf{p}_i + c_k} \cdot \frac{\alpha}{1-\alpha} \ = \ \exp(2r). \tag{6.12}$$

For $p_i \in \text{int}(\Omega)$, r is finite and positive: $0 < r < \infty$. Therefore Eq. (6.12) can be equivalently written as a linear equation in α. Since for $j, k \in [m]$, $j \neq k$, $i \in [n]$, $S(i,r)_{j,k}$ is a bounding hyperplane for $B^H_\Omega(p_i, r)$, we can use Eqs. (6.11), and (6.12) to construct $B^H_\Omega(p_i, r)$, for all $p_i \in P^\pm$ in $O(ndm^2)$

time.

Remark 6.3 (Funk Balls). We can use a similar approach to construct $B^F_\Omega(p_i, r)$, the Funk ball of radius r around p_i. Recall that the Funk ball around p_i is a scaled copy of Ω around p_i. Therefore for a bounding hyperplane of Ω, $L_j : (\mathbf{w}_j, c_j)$, $j \in [m]$, we have $S(i,r)_j = (\mathbf{w}_j, c^*_j(i,r))$ as a bounding hyperplane of $B^F_\Omega(p_i, r)$ for some constant $c^*_j(i,r)$. We solve for $c^*_j(i,r)$ using the following equation:

$$\exp(d^F_\Omega(p_i, S(i,r)_j)) \;=\; \frac{\mathbf{w}_j^\top \mathbf{p}_i + c_j}{\mathbf{w}_j^\top \mathbf{p}_i + c^*_j(i,r)} \;=\; \exp(r). \tag{6.13}$$

As before, for any $r \geq 0$, this can be expressed as a linear function in $c^*(i,r)$. Therefore, we can construct $B^F_\Omega(p_i, r)$, for all $p_i \in P^\pm$ in $O(nmd)$ time.

6.4.4 Complexity Analysis

Recall that to obtain γ within an additive error of ε, we solve $O(\log B + \log\log d + \log(1/\varepsilon))$ LPs of the form Opt. (6.10). We can use any standard interior-point method, such as Khachiyan's Ellipsoid method [55] or Karmarkar's method [52] to solve Opt. (6.10) in time that is polynomial in the number of variables, number of inequalities, and bit-length of each variable. We first establish the bit-length required to encode a parameter in our LP-feasibilty problem (Opt. (6.10)). Recall that we are searching for r over the range $\Gamma = [\varepsilon, 2\varepsilon, \ldots, M]$, with $M = M(d, n, m, B) = O(B + \log d)$. Therefore, the quantity e^{2r} can take on values in the range $\Gamma_e = [e^{2\varepsilon}, e^{4\varepsilon}, \ldots, e^{2M}]$. Instead of fixing r's value exactly over Γ, it is sufficient for our purposes if we search for r over $[r_1, r_2, \ldots, r_{M/\varepsilon}]$, where $(i-1)\varepsilon \leq r_i < i\varepsilon$. The minimum difference between two elements in Γ_e is at least $e^{4\varepsilon} - e^{2\varepsilon} > \varepsilon$. (This can be verified using Taylor's series for e^x, at $x = 0$.) Therefore, to represent Γ_e, it is sufficient for our purposes to find a bit-representation which allows for a maximum value e^{2M} up to a resolution of ε.

This can be done by using $O(M)$ bits for the integer part, and $O(\log(1/\varepsilon))$ bits for the fractional part. Hence, it is sufficient to represent the elements of Γ_e as $O(M + \log(1/\varepsilon)) = O(B + \log d + \log(1/\varepsilon))$-bit rational numbers. Using Eq. (6.12), we solve for and represent α, using $O(B + \log d + \log(1/\varepsilon))$-bit rational numbers. Therefore, using Eq. (6.11) every parameter of $S(i, r)_{j,k}$, which acts as a bounding hyperplane for $B^H_\Omega(p_i, r)$, can also be represented using $O(B + \log d + \log(1/\varepsilon))$-bit rational numbers. Now, the number of variables in Opt. (6.10) is $O((n + d)m^2)$, with the number of inequalities being $O(n(m^2 + d))$. Hence, we can determine if Opt. (6.10) is feasible in $O(\text{poly}(n, m, d, B + \log d + \log(1/\varepsilon))$ time, for a fixed r. And since we solve $O(\log B + \log\log d + \log(1/\varepsilon))$ such LPs, our overall running time is $O(\text{poly}(n, m, d, B, \log(1/\varepsilon))$. This proves our main result in Theorem 6.1. Recall that the input bit-length is $d(n + m)B$ implying the overall running time is also polynomial in the input bit-length and the log of the desired accuracy.

6.5 Extensions

In this section, we will explore two extensions to binary classification and SVM problem in the Hilbert geometry. The first is a soft-margin classifier, which is applicable when the point sets are not linearly separable, and the second is a binary classifier based on nearest neighbors.

6.5.1 A Soft-Margin Classifier

Up to this point we assumed P^+, and P^- are linearly separable. If they are not, a common approach is to consider a hyperplane classifier with appropriate penalties associated with misclassification of points in the training set. In this section we propose a formulation for such a "soft" classifier. We modify our LP-feasibility problem: Opt (6.10) to introduce penalties in the form of slack variables.

Reiterating the observation from Remark 6.1, we note that the first two constraints of Opt (6.10) enforce the condition that, for $p \in P^+$, $B^H_\Omega(p, r)$ is completely above the separator $K : (\mathbf{w}, c)$. Instead, we only require that $B^H_\Omega(p, r)$ is above $K' : (\mathbf{w}, c + \xi)$ for some positive ξ. Here ξ works as a penalty for the improper separation of $B^H_\Omega(p, r)$. We similarly set individual penalties for each point. Next, we want the following desirable property: points that are closer to the boundary of Ω should pay a heavier penalty for being mis-classified. It is because for a fixed point q, $d^H_\Omega(p_i, q)$ scales inversely with p_i's distance to the boundary. Therefore we scale the penalty associated with p_i by ω_i, defined as the inverse of its distance to the boundary:

$$\omega_i = \frac{1}{\min_{j \in [m]} d^H_\Omega(L_j, p_i)}$$

Finally we propose the following LP in the parameters $\mathbf{w}$, c, and ξ_i, $\mathbf{y}(i)$, for $i \in [n]$, to minimize the overall weighted penalty for a fixed Hilbert radius r:

$$\begin{aligned}
\text{Minimize:} \qquad \Xi_r &= \sum_{i \in [n]} \omega_i \xi_i \\
\text{Subject to:} \qquad \mathbf{b}(r, i)^\top \mathbf{y}(i) &\leq c + \xi_i, \quad \forall i \in I^+ \\
\mathbf{b}(r, i)^\top \mathbf{y}(i) &\geq c - \xi_i, \quad \forall i \in I^- \\
\mathbf{A}(r, i)^\top \mathbf{y}(i) &= \mathbf{w}, \quad \forall i \in I \\
\mathbf{y}(i) &\geq \mathbf{0}, \quad \forall i \in I \\
\mathbf{1}^\top \mathbf{w} &= 1 \\
\xi_i &\geq 0, \quad \forall i \in I. \qquad (6.14)
\end{aligned}$$

Let the optimum value derived from Opt. (6.14) be Ξ_r, and the corresponding separating hyperplane; $K_r = (\mathbf{w}, c)$. Recall that r attains values over $\Gamma_r = [r_1, r_2, \ldots, r_{M/\varepsilon}]$, where r_i's are any values satisfying $(i-1)\varepsilon \leq r_i < i\varepsilon$, and $M = O(B + \log d)$. To find an appropriate classifier K^*, we take

$$K^* \;=\; K_s, \qquad \text{where} \qquad s \;=\; \arg\max_{i \in [M/\varepsilon]} \left(r_i - C\,\frac{\Xi_{r_i}}{n} \right).$$

Note that C is a user defined-constant, a weight associated with improper separation. A higher value would be partial towards proper separation, while a lower value towards a larger margin. A slight drawback of this approach is that it no longer suffices to perform a binary search on the range Γ_r. Rather, we need to perform a grid search, i.e. solve Opt. (6.14) for every value of $r \in \Gamma_r$ in the worst case. This implies the overall running time for the soft-margin classifier is $O(\text{poly}(n, m, d, B, 1/\varepsilon))$.

6.5.2 A Nearest Neighbor-based Classifier

In this section, we consider an alternative approach to binary classification in the Hilbert geometry. Rather than using a separating hyperplane, we select two sites q^+ and q^-, and a point p is classified according to which of these two sites it is closer to. In the Euclidean setting, the bisector between two points is a hyperplane, and this is no different from SVM. However, in Hilbert, bisectors are generally not hyperplanes. (An analysis of the structure of Hilbert bisectors in 2-dimensional case was presented in [40], where it was shown that they are piecewise conics.)

As before, we assume that Hilbert geometry is defined with respect to a polytope Ω in $\mathbb{R}^d$ defined by m bounding hyperplanes $\mathcal{L} = \{L_i\}_{i=1}^m$. Extending results of de la Harpe [33], Vernicos presented an elegant isometry from the polytopal Hilbert geometry (Ω, d_Ω^H) to a finite dimensional normed vector space [92]. In particular, he presented an isometric mapping $f_\Omega : (\Omega, d_\Omega^H) \to (\mathbb{R}^{m-1}, ||\cdot||_{\Sigma_{2m}})$, where

$||\cdot||_{\Sigma_{2m}}$ is a norm defined by a regular polytope Σ bounded by $2m$ facets in $\mathbb{R}^{m-1}$. In other words, the embedding space is $\mathbb{R}^{m-1}$ with Σ_{2m} as the unit ball. Although one might be tempted to solve the SVM problem in the embedding space, the result would be far from ideal. Since f_Ω is nonlinear, a linear classifier $\widehat{K}$ in the embedding space would not translate to a simple classifier in the input space. Moreover, since f_Ω is not a bijection it is not clear how complex the intersection of $\widehat{K}$ with the image of Ω, $f_\Omega(\Omega)$, would be. Instead, we propose a simple nearest neighbor-based classifier in the embedding space. Given Ω, let q_i, Q^+, and Q^- denote the images of p_i, P^+, and P^- under the embedding f_Ω. (q_i for all $i \in [n]$ can be computed in $O(m^2)$ time by solving m linear equations). We find two representative centers c^+, and c^-, for Q^+, and Q^- respectively such that the following is maximized:

$$\beta \;=\; \max_{c^+,c^-\in\mathbb{R}^{m-1}} \left(\min_{q^+\in Q^+} \frac{\min_{q^-\in Q^-}\|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma}{\|\mathbf{q}^+ - \mathbf{c}^+\|_\Sigma}, \min_{q^-\in Q^-} \frac{\min_{q^+\in Q^+}\|\mathbf{q}^- - \mathbf{q}^+\|_\Sigma}{\|\mathbf{q}^- - \mathbf{c}^-\|_\Sigma} \right) \tag{6.15}$$

Stated differently, β is the value such that any point q is at least β times away from its nearest neighbor in the opposite class, as from its representative. This implies that the points can be correctly classified using an approximate nearest-neighbor algorithm that achieves an approximation factor of at most β. This can be rewritten as an optimization problem over the variables $\mathbf{c}^+$, $\mathbf{c}^-$, and $1/\beta$

$$\begin{aligned}
&\text{Minimize:} && \frac{1}{\beta} \\
&\text{Subject to:} && \|\mathbf{q}^+ - \mathbf{c}^+\|_\Sigma \;\le\; \frac{1}{\beta}\|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma, \quad \forall q^+ \in Q^+, \forall q^- \in Q^- \\
& && \|\mathbf{q}^- - \mathbf{c}^-\|_\Sigma \;\le\; \frac{1}{\beta}\|\mathbf{q}^- - \mathbf{q}^+\|_\Sigma, \quad \forall q^+ \in Q^+, \forall q^- \in Q^-
\end{aligned} \tag{6.16}$$

We note that $\|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma$ in the above optimization is a pre-determined constant. If we let

$W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{2m}\}$ denote the directions determining the norm polytope Σ, then the condition that $\|\mathbf{x}\|_\Sigma \leq t$ can be expressed as a set of linear inequalities: $\mathbf{w}_i^\top \mathbf{x} \leq t$, for $i \in [2m]$. Therefore, Opt. (6.16) is an LP in $O(m)$ variables, and $O(nm)$ inequalities.

6.6 Chapter Summary

In this chapter, we addressed the computational bottleneck identified in Chapter 5: the "curse of dimensionality" inherent in exact Hilbert geometry algorithms. While the LP-type framework provided a rigorous theoretical foundation, its dependence on the combinatorial complexity of the Hilbert ball rendered it impractical for high-dimensional feature spaces.

We overcame this limitation by adopting a numerical optimization approach. By using *Birkhoff's formulation* we characterize Hilbert balls as a intersection of half-spaces rather than a collection of vertices. This insight allowed us to reformulate the SVM problem as a sequence of linear programming feasibility tests. The resulting approximation algorithm achieves a running time that is polynomial in the dimension d, making Hilbert-based classification feasible for large-scale applications.

Furthermore, we extended this framework to practical machine learning scenarios where data may not be linearly separable. We introduced a *soft-margin classifier* that weights penalties based on boundary proximity, and a *nearest-neighbor classifier* that operates in the embedding space.

With these tools, we have established a robust and efficient set of algorithms for classifying data within bounded convex domains. In the final chapter, we will step back to synthesize the results from both Part I (Tracking) and Part II (Classification), offering a unified perspective on the geometry of uncertainty.

Chapter 7: Conclusion and Future Directions

7.1 Summary of Contributions

In this dissertation, we have explored the algorithmic challenges posed by data that is massive, dynamic, and inherently uncertain. We approached this problem through two distinct but complementary geometric lenses: the *temporal tracking* of evolving data and the *spatial classification* of data in bounded domains.

7.1.1 Tracking Temporal Uncertainty (Part I)

Part I addressed the challenge of maintaining accurate representations of data that changes continuously and invisibly.

- *Discrete Structures:* In Chapter 2, we established the fundamental limits of tracking on tree topologies. We proved that a simple algorithm with a constant speedup factor $c \geq 2$ is sufficient to maintain labels within an average distance of $O(1)$ from the ground truth, matching the lower bounds for random evolvers.

- *Continuous Geometry:* In Chapter 3, we extended this framework to the Euclidean plane. By utilizing a *cone-based oracle* to handle directional uncertainty and a *Theta-graph* for local routing, we demonstrated that a speedup of $c > 3t_\theta$, where t_θ is the spanning ratio of the graph,

suffices to track moving points with bounded expected error.

- *Probabilistic Data:* In Chapter 4, we moved beyond point tracking to track evolving *probability distributions*. We introduced the local-motion model to capture crowd dynamics and utilized the Kullback-Leibler (KL) divergence to bound the information loss of our hypothesis. This established the crucial link between geometric location and information-theoretic uncertainty.

7.1.2 Classifying Spatial Uncertainty (Part II)

In **Part II**, we addressed the challenge of classifying data that is constrained within fixed, convex domains, a scenario where Euclidean metrics often fail to capture the semantic significance of boundary proximity.

- *Exact Classification:* In Chapter 5, we formulated the Support Vector Machine (SVM) problem in the Hilbert metric. We proved that this problem is of *LP-type*, enabling efficient exact solutions ($O(nM^3)$) for low-dimensional polytopes.

- *Scalable Approximation:* In Chapter 6, we overcame the "curse of dimensionality" inherent in the exact approach. By exploiting Birkhoff's formulation of the Hilbert metric, we recast the problem as a sequence of linear programming feasibility tests. This yielded a polynomial-time approximation scheme that makes Hilbert-based classification practical for high-dimensional applications in machine learning.

7.2 The Unified View: Robustness vs. Tracking

While Parts I and II address different algorithmic tasks, they provide a unified toolkit for handling uncertainty in geometric systems.

Part I deals with *dynamic variation*. When the ground truth moves significantly, no static model can remain valid; an algorithm must actively *track* the changes to minimize divergence. The KL divergence served as our metric for this temporal lag.

Part II deals with *static ambiguity*. When data points are noisy or reside near the "edge cases" of a domain (e.g., the boundary of a probability simplex), we rely on the *geometric robustness* of the Hilbert metric. A large-margin Hilbert separator ensures that small perturbations in the data do not alter the classification result.

Together, these results demonstrate that non-Euclidean geometries, whether information theoretic (KL) or projective (Hilbert), are essential for robust computing in modern, uncertain environments.

7.3 Future Directions

The work presented in this dissertation opens several promising avenues for future research, particularly at the intersection of temporal tracking and geometric classification.

7.3.1 Online Hilbert SVMs

The most direct synthesis of Part I and Part II is the problem of **classifying evolving distributions**. Suppose we have two sets of probability distributions, P^+ and P^-, moving inside a simplex according to the local-motion model defined in Chapter 4.

- *Problem:* Can we maintain an approximate Hilbert SVM separator (from Chapter 6) without recomputing it from scratch at every time step?

- *Conjecture:* Since the local-motion model bounds the displacement of the distributions, the op-

timal separator likely changes continuously. Dynamic linear programming techniques or kinetic data structures could potentially be adapted to the Hilbert geometry to maintain the separator efficiently.

7.3.2 Dynamic Data Structures for Hilbert Geometry

Our current algorithms for Hilbert SVMs are static. In many applications, such as real-time sensor networks, points are inserted or deleted continuously.

- *Problem:* Design dynamic data structures that support nearest-neighbor queries or margin maintenance in the Hilbert metric.
- *Opportunity:* The concurrency properties of Hilbert balls (Chapter 5) suggests that Voronoi diagrams in Hilbert geometry have specific structural properties that might support local updates, similar to their Euclidean counterparts.

7.3.3 Beyond Linear Separators

In Chapter 6, we introduced a soft-margin classifier for non-linearly separable data. However, modern machine learning relies heavily on non-linear decision boundaries.

- *Challenge:* Extend the Hilbert classification framework to *Kernel SVMs*.
- *Open Question:* Is there a "Hilbert Kernel" that allows us to map points into a higher dimensional Hilbert space where they become linearly separable, while preserving the boundary sensitive properties of the original metric?

7.3.4 Hyperbolic Neural Networks

Recent advances in geometric deep learning have shown that embedding hierarchies into hyperbolic space improves performance. Since the Hilbert metric is a generalization of hyperbolic geometry (specifically the Cayley-Klein model), it offers a more flexible embedding space that can adapt to asymmetric data structures (via the Funk metric).

- *Application:* Investigating the use of the differentiable approximation of the Hilbert metric (from Chapter 6) as a loss function in deep neural networks for hierarchical classification.

In conclusion, this dissertation has laid the algorithmic foundations for handling data that is both moving and bounded. As computational applications increasingly interact with the physical world, the ability to track efficiently and classify robustly within non-Euclidean constraints will only become more critical.

Bibliography

[1] Aditya Acharya, Auguste H Gezalyan, and David M Mount. Classifiers in high dimensional hilbert metrics. *20th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, 2026. doi:10.4230/LIPIcs.SWAT.2026.32.

[2] Aditya Acharya, Auguste H. Gezalyan, Julian Vanecek, David M. Mount, and Sunil Arya. Support vector machines in the hilbert geometry. In *19th International Symposium on Algorithms and Data Structures, WADS 2025*, pages 3:1–3:20, 2025. doi:10.4230/LIPIcs.WADS.2025.3.

[3] Aditya Acharya and David M. Mount. Optimally tracking labels on an evolving tree. In *Proceedings of the 34th Canadian Conference on Computational Geometry, CCCG 2022*, pages 1–8, 2022. URL: https://dblp.org/db/conf/cccg/cccg2022.html.

[4] Aditya Acharya and David M. Mount. Tracking evolving labels using cone based oracles. *Young Researcher's Forum at Symposium on Computational Geometry*, 2023. doi:10.48550/arXiv.2306.03306.

[5] Aditya Acharya and David M. Mount. Evolving Distributions Under Local Motion. In *19th International Symposium on Algorithms and Data Structures (WADS 2025)*, pages 4:1–4:20, 2025. doi:10.4230/LIPIcs.WADS.2025.4.

[6] Antoine Allard and M. Ángeles Serrano. Navigable maps of structural brain networks across species. *PLOS Computational Biology*, 16:1–20, 2020. doi:10.1371/journal.pcbi.1007584.

[7] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. doi:10.1016/j.tcs.2010.10.003.

[8] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theor. Comput. Sci.*, 412(24):2564–2576, 2011. doi:10.1016/j.tcs.2010.10.003.

[9] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, ITCS '12, pages 149–160, New York, NY, USA, 2012. ACM. doi:10.1145/2090236.2090249.

[10] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 149–160. ACM, 2012. doi:10.1145/2090236.2090249.

[11] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, page 24–32, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2339530.2339539.

[12] Mira Beermann and Anna Sieben. The connection between stress, density, and speed in crowds. *Scientific Reports*, 13:13626, 2023. doi:10.1038/s41598-023-39006-8.

[13] Asa Ben-Hur, Cheng Soon Ong, Sören Sonnenburg, Bernhard Schölkopf, and Gunnar Rätsch. Support vector machines and kernels for computational biology. *PLoS Comput. Biology*, 4(10):e1000173, 2008. doi:10.1371/journal.pcbi.1000173.

[14] Kristin P. Bennett and Colin Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explor. Newsl.*, 2, 2000. doi:10.1145/380995.380999.

[15] Juan Jose Besa, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Optimally Sorting Evolving Data. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 81:1–81:13, 2018. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9085, doi:10.4230/LIPIcs.ICALP.2018.81.

[16] Ahmad Biniaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. *Discret. Math. Theor. Comput. Sci.*, 24(2), 2022. doi:10.46298/dmtcs.8383.

[17] Garrett Birkhoff. Extensions of Jentzsch's theorem. *Trans. Amer. Math. Society*, 85:219–227, 1957.

[18] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2016. doi:10.1007/978-0-387-45528-0.

[19] Mathieu Blondel, André F. T. Martins, and Vlad Niculae. Learning with Fenchel-Young losses. *J. Mach. Learn. Research*, 21:1–69, 2020. URL: http://jmlr.org/papers/v21/19-021.html.

[20] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

[21] Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011. doi:10.1007/s00453-010-9430-0.

[22] Aydin Buluç, Scott Beamer, Kamesh Madduri, Krste Asanovic, and David A. Patterson. Distributed-memory breadth-first search on massive graphs. *CoRR*, abs/1705.04590, 2017. URL: http://arxiv.org/abs/1705.04590, arXiv:1705.04590, doi:10.48550/arXiv.1705.04590.

[23] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998. doi:10.1023/A:1009715923555.

[24] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4869–4880, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/0415740eaa4d9decbc8da001d3fd805f-Abstract.html, doi:10.48550/arXiv.1910.12933.

[25] Bernard Chazelle and Jiří Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21:579–597, 1996. doi:10.1006/jagm.1996.0060.

[26] Yi-Jen Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992. doi:10.1109/5.163409.

[27] H. Cho, B. DeMeo, J. Peng, and B. Berger. Large-margin classification in hyperbolic space. In *Proc. 22nd Internat. Conf. Artif. Intell. and Stat.*, pages 1832–1840. PMLR, 2019. URL: http://proceedings.mlr.press/v89/cho19a.html, doi:10.48550/arXiv.1806.00437.

[28] K. L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. Assoc. Comput. Mach.*, 42:488–499, 1995. doi:10.1145/201019.201036.

[29] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20:273–297, 1995. doi:10.1007/BF00994018.

[30] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2012. doi:10.1002/047174882X.

[31] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. doi:10.1017/CBO9780511801389.

[32] S. Das, S. R. Dev, and Sarvottamananda. A worst-case optimal algorithm to compute the Minkowski sum of convex polytopes. *Discrete Appl. Math.*, 350:44–61, 2024. doi:10.1016/j.dam.2024.02.004.

[33] P. de la Harpe. On Hilbert's metric for simplices. In G. A. Niblo and M. A. Roller, editors, *Geometric Group Theory*, pages 97–119. Cambridge University Press, 1993. doi:10.1017/CBO9780511661860.009.

[34] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Mach. Learn.*, 46:161–190, 2002. doi:10.1023/A:1012454411458.

[35] Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. *Dynamic Graph Algorithms*, page 9. Chapman & Hall/CRC, 2 edition, 2010.

[36] B. Dhingra, C. Shallue, M. Norouzi, A. Dai, and G. Dahl. Embedding text in hyperbolic spaces. In *Proc. Twelfth Workshop Graph-Based Methods for Nat. Lang. Proc. (TextGraphs-12)*, pages 59–69, 2018. doi:10.48550/arXiv.1806.04313.

[37] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. In *Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy, June 6-8, 2005*, SCG '05, pages 142–149, New York, NY, USA, 2005. ACM. doi:10.1145/1064092.1064116.

[38] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5350–5360, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/dbab2adc8f9d078009ee3fa810bea142-Abstract.html, doi:10.48550/arXiv.1805.09112.

[39] Bernd Gärtner and Jirí Matousek. *Understanding and using linear programming*. Universitext. Springer, 2007. doi:10.1007/978-3-540-30717-4.

[40] Auguste H. Gezalyan and David M. Mount. Voronoi diagrams in the hilbert metric. In *39th International Symposium on Computational Geometry, SoCG 2023, Dallas, Texas, USA, June 12-15, 2023*, volume 258 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.35.

[41] George Giakkoupis, Marcos Kiwi, and Dimitrios Los. Naively sorting evolving data is optimal and robust. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2217–2242. IEEE, 2024. doi:10.1109/FOCS61266.2024.00130.

[42] Gilad Goraly and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58(3):610–636, nov 2010. doi:10.1007/s00453-009-9290-7.

[43] Daniel Graf. How to sort by walking and swapping on paths and trees. *Algorithmica*, 78(4):1151–1181, aug 2017. doi:10.1007/s00453-017-0282-8.

[44] B. Hashemi, K. Pasque, C. Teska, and R. Yoshida. Tropical attention: Neural algorithmic reasoning for combinatorial algorithms. In *Proc. 39th Annu. Conf. Neural Inf. Process. Systems*, 2025. doi:10.48550/arXiv.2505.17190.

[45] O. Hesham and G. Wainer. Context-sensitive personal space for dense crowd simulation. In *Proc. Symp. Simulation for Architecture and Urban Design*, pages 1–8, 2017. URL: https://dl.acm.org/doi/10.5555/3289787.3289806.

[46] D. Hilbert. Ueber die gerade Linie als kürzeste Verbindung zweier Punkte. *Math. Annalen*, 46:91–96, 1895. doi:10.1007/BF02096204.

[47] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36:1171–1220, 2008. doi:10.1214/009053607000000677.

[48] S. Huang, N. Cai, P. P. Pacheco, S. Narrandes, Y. Wang, and W. Xu. Applications of support vector machine (SVM) learning in cancer genomics. *Cancer Genomics & Proteomics*, 15:41–51, 2018. doi:10.21873/cgp.20063.

[49] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40:1098–1101, 1952. doi:10.1109/JRPROC.1952.273898.

[50] M. Ito, Y. Seo, T. Yamazaki, and M. Yanagida. Geometric properties of positive definite matrices cone with respect to the Thompson metric. *Linear Algebra Appl.*, 435:2054–2064, 2011. doi:10.1016/j.laa.2011.03.063.

[51] Thorsten Joachims. *Learning to classify text using support vector machines - methods, theory and algorithms*, volume 668 of *The Kluwer international series in engineering and computer science*. Kluwer, 2002. doi:10.1007/978-1-4615-0907-3.

[52] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. doi:10.1007/BF02579150.

[53] Jacek Karwowski and Frank Nielsen. Hilbert geometry of the symmetric positive-definite bicone: Application to the geometry of the extended gaussian family. *CoRR*, abs/2508.14369, 2025. arXiv:2508.14369, doi:10.48550/arXiv.2508.14369.

[54] J Mark Keil and Carl A Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992. doi:10.1007/BF02187821.

[55] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk*, 244:1093–1096, 1979. URL: https://www.mathnet.ru/eng/dan42319.

[56] Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 241–250. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715921.

[57] Scott Krig. *Computer Vision Metrics - Textbook Edition*. Springer, 2016. doi:10.1007/978-3-319-33762-3.

[58] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *CoRR*, abs/1006.5169(3):036106, 2010. URL: http://arxiv.org/abs/1006.5169, arXiv:1006.5169, doi:10.48550/arXiv.1006.5169.

[59] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Amin Vahdat, and Marián Boguñá. On curvature and temperature of complex networks. *CoRR*, abs/0903.2584(3):035101, 2009. URL: http://arxiv.org/abs/0903.2584, arXiv:0903.2584, doi:10.48550/arXiv.0903.2584.

[60] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22:79 – 86, 1951. doi:10.1214/aoms/1177729694.

[61] Wu Lin, Valentin Duruisseaux, Melvin Leok, Frank Nielsen, Mohammad Emtiyaz Khan, and Mark Schmidt. Simplifying momentum-based positive-definite submanifold optimization with applications to deep learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202, pages 21026–21050. PMLR, 2023. URL: https://proceedings.mlr.press/v202/lin23c.html, doi:10.48550/arXiv.2302.09738.

[62] B. C. Lins. *Asymptotic behavior and Denjoy-Wolff theorems for Hilbert metric nonexpansive maps*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2007.

[63] Kanglin Liu, Changchun Liu, Xi Xiang, and Zhili Tian. Testing facility location and dynamic capacity planning for pandemics with demand uncertainty. *Eur. J. Oper. Res.*, 304(1):150–168, 2023. doi:10.1016/j.ejor.2021.11.028.

[64] J. A. De Loera and E. D. Kim. *Combinatorics and geometry of transportation polytopes: An update*, volume 625, pages 37–76. AMS, 2013. URL: http://www.ams.org/books/conm/625/12491.

[65] Maarten Löffler and Marc J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56:235–269, 2010. doi:10.1007/s00453-008-9174-2.

[66] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. URL: https://books.google.com/books?id=AKuMj4PN_EMC.

[67] Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and Hardness of Token Swapping. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57, pages 66:1–66:15, 2016. URL: http://drops.dagstuhl.de/opus/volltexte/2016/6408, doi:10.4230/LIPIcs.ESA.2016.66.

[68] G. Mishne, Z. Wan, Y. Wang, and S. Yang. The numerical stability of hyperbolic representation learning. In *Proc. 40th Internat. Conf. Mach. Learn.*, pages 24925–24949, 2023. URL: https://proceedings.mlr.press/v202/mishne23a.html.

[69] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge University Press, 2017. URL: https://books.google.com/books?id=E9UlDwAAQBAJ.

[70] M. Nickel and D. Kiela. Poincaré embeddings for learning hierarchical representations. *Adv. Neural Inf. Proc. Syst.*, 30:6341–6350, 2017. doi:10.48550/arXiv.1705.08039.

[71] F. Nielsen and L. Shao. On balls in a Hilbert polygonal geometry (multimedia contribution). In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 67:1–67:4, 2017. doi:10.4230/LIPIcs.SoCG.2017.67.

[72] F. Nielsen and K. Sun. Clustering in Hilbert's projective geometry: The case studies of the probability simplex and the elliptope of correlation matrices. In Frank Nielsen, editor, *Geometric Structures of Information*, pages 297–331. Springer Internat. Pub., 2019. doi:10.1007/978-3-030-02520-5_11.

[73] Frank Nielsen and Ke Sun. Non-linear embeddings in hilbert simplex geometry. In *Topological, Algebraic and Geometric Learning Workshops 2023, 28 July 2023, Honolulu, HI, USA*, volume 221 of *Proceedings of Machine Learning Research*, pages 254–266. PMLR, 2023. URL: https://proceedings.mlr.press/v221/nielsen23a.html, arXiv:arXiv:2203.11434.

[74] A. Papadopoulos and M. Troyanov. From Funk to Hilbert geometry. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 33–68. European Mathematical Society Publishing House, 2014. doi:10.4171/147-1/2.

[75] A. Papadopoulos and M. Troyanov. *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*. European Mathematical Society Publishing House, 2014. doi:10.4171/147.

[76] W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao. Hyperbolic deep neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(12):10023–10044, 2021. doi:10.1109/TPAMI.2021.3136921.

[77] Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990. URL: https://www.sciencedirect.com/science/article/pii/S0747717108800016, doi:10.1016/S0747-7171(08)80001-6.

[78] D. Reeb, M. J. Kastoryano, and M. M. Wolf. Hilbert's projective metric in quantum information theory. *J. Math. Physics*, 52(8), 2011. doi:10.1063/1.3615729.

[79] Jim Rupert and Raimund Seidel. Approximating the d-dimensional complete euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG 1991)*, pages 207–210, 1991.

[80] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 4th edition, 2020. URL: http://aima.cs.berkeley.edu/.

[81] David Salesin, Jorge Stolfi, and Leonidas J. Guibas. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proceedings of the Fifth Annual Symposium on Computational Geometry, Saarbrücken, Germany, June 5-7, 1989*, pages 208–217. ACM, 1989. doi:10.1145/73833.73857.

[82] Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discret. Comput. Geom.*, 6:423–434, 1991. doi:10.1007/BF02574699.

[83] Thibault Séjourné, Jean Feydy, François-Xavier Vialard, Alain Trouvé, and Gabriel Peyré. Sinkhorn divergences for unbalanced optimal transport. *CoRR*, abs/1910.12958, 2019. URL: http://arxiv.org/abs/1910.12958, arXiv:1910.12958, doi:10.48550/arXiv.1910.12958.

[84] A. Seyfried, B. Steffen, W. Klingsch, and M. Boltes. The fundamental diagram of pedestrian movement revisited. *J. Stat. Mech. Theory Exp.*, 2005:P10002, 2005. doi:10.1088/1742-5468/2005/10/P10002.

[85] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4):623–656, 1948. doi:10.1002/j.1538-7305.1948.tb00917.x.

[86] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577, pages 569–579. Springer, 1992. URL: https://doi.org/10.1007/3-540-55210-3_213, doi:10.1007/3-540-55210-3_213.

[87] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Information science and statistics. Springer, 2008. doi:10.1007/978-0-387-77242-4.

[88] Richard Szeliski. *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science. Springer, 2022. doi:10.1007/978-3-030-34372-9.

[89] S. M. Tomkiewicz, M. R. Fuller, J. G. Kie, and K. K. Bates. Global positioning system and associated technologies in animal behaviour and ecological research. *Phil. Trans. R. Soc. B*, 365:2163–2176, 2010. doi:10.1098/rstb.2010.0090.

[90] V. N. Vapnik and A. Y. Chervonenkis. On a class of algorithms of learning pattern recognition. *Automation and Remote Control*, 25(6):937–945, 1964. (In Russian). URL: http://mi.mathnet.ru/at11678.

[91] S. Verdú. The Cauchy distribution in information theory. *Entropy*, 25:346, 2023. doi:10.3390/e25020346.

[92] C. Vernicos. On the Hilbert geometry of convex polytopes. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 111–126. European Mathematical Society Publishing House, 2014. doi:10.48550/arXiv.1406.0733.

[93] Juan José Besa Vial, William E Devanny, David Eppstein, Michael T Goodrich, and Timothy Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 87–96. SIAM, 2018. doi:10.1137/1.9781611975055.8.

[94] Juan José Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018*, pages 87–96. SIAM, 2018. doi:10.1137/1.9781611975055.8.

[95] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theor. Comput. Sci.*, 586:81–94, 2015. Fun with Algorithms. doi:10.1016/j.tcs.2015.01.052.

[96] Ugur Zengin and Atilla Dogan. Real-time target tracking for autonomous uavs in adversarial environments: A gradient search algorithm. *IEEE Trans. Robotics*, 23(2):294–307, 2007. doi:10.1109/TRO.2006.889490.

[97] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80, pages 5819–5827. PMLR, 2018. URL: http://proceedings.mlr.press/v80/zhang18i.html, doi:10.48550/arXiv1805.07091.

[98] Ying Zhang, Yu-Ling Hsueh, Wang-Chien Lee, and Yi-Hao Jhang. Efficient cache-supported path planning on roads. *IEEE Transactions on Knowledge and Data Engineering*, 28(4):951–964, 2015. doi:10.1109/TKDE.2015.2507581.

[99] Yiming Zou, Gang Zeng, Yuyi Wang, Xingwu Liu, Xiaoming Sun, Jialin Zhang, and Qiang Li. Shortest paths on evolving graphs. In Hien T. Nguyen and Vaclav Snasel, editors, *Computational Social Networks*, pages 1–13, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-42345-6_1.