Thesis for the degree
Master of Science

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

עבודת גמר (תזה) לתואר
מוסמך למדעים

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
Sharon Stein

מאת
שרון שטיין

אלגוריתמים מהירים לפירוקים מפרידים במרחבים אוקלידיים במימד גבוה

Fast Algorithms for Separating Decompositions in High-Dimensional Euclidean Spaces

Advisor:
Prof. Robert Krauthgamer

מנחה:
פרופ' רוברט קראוטגמר

February 2025

שבט ה'תשפ"ה

Abstract

This thesis presents a fast algorithm for constructing separating decompositions in high-dimensional Euclidean spaces, significantly improving upon existing methods in terms of time complexity. Separating decompositions are fundamental in various areas of computer science, ranging from graph spanners to tree embeddings. Our main contribution is an algorithm that constructs an $(O(\sqrt{d}), \Delta)$-separating decomposition for $V \subseteq \mathbb{R}^d$ sized n, where $d = \Theta(\log n)$, in $O(n^{1.51})$ time. This represents a substantial improvement over the naive implementation in time $\tilde{O}(n^3)$.

We begin by providing an improved implementation of a known algorithm [CCG+98], that decreases the time complexity from $\tilde{O}(n^3)$ to $\tilde{O}(n^2)$. We then introduce a faster algorithm that leverages Locality Sensitive Hashing (LSH) techniques, achieving an expected running time of $\tilde{O}(n^{1+1/c^2+o(1)})$ for $c = 1.4$. This approach incurs an additive error in the separation probability. Finally, we present a method to eliminate this additive error while maintaining the improved time complexity.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Robert Krauthgamer, for his invaluable guidance, patience, and support throughout my research journey. His insights, both professional and personal, have greatly influenced my academic development, and I truly appreciate his mentorship.

I also thank my fellow students for their collaboration, discussions, and shared efforts, with special appreciation to Yotam Kenneth for his support.

Finally, I extend my heartfelt appreciation to my family and friends for their encouragement and support during this process. A special thank you to my partner, Dvir Plaksin, for always being there for me.

1 Introduction

Graph partitioning is a fundamental problem in computer science and mathematics, involving the division of a graph into disjoint subsets of vertices, often with the goal of minimizing the number of edges between the subsets while satisfying constraints such as balancing the sizes or some other measure of the sets. In the context of metric spaces, graph partitioning can be formulated similarly by considering a complete graph whose vertices represent points in the metric space, and edge weights correspond to pairwise distances between points.

Low-diameter partitioning focuses on dividing the metric space into sets called clusters, where each cluster has a bounded diameter. This genre of problems has various applications ranging from graph spanners [HPIS13, FN20] to tree embeddings [AP90, LS93, Bar96, CCG+98, WLB+00, FRT03]. A probabilistic approach to such partitioning uses a probability distribution over partitions. Each partition in the distribution ensures that the clusters have a bounded diameter, so distant elements are separated. Meanwhile nearby points are grouped together with high (or at least non-negligible) probability.

Low diameter graph decompositions were first introduced by Linial and Saks [LS93] and later popularized by Bartal [Bar96]. We focus on one of the most central variants in the literature, known as the *separating decomposition*, which is sometimes also referred to as a *Lipschitz decomposition*.

Definition 1.1 (Separating Decomposition)**.** *Let (X, ρ) be a metric space. A distribution F over partitions of X is called a* (β, Δ)-separating decomposition *if the following conditions are satisfied:*

1. ***Diameter:*** *For every partition $P \in supp(F)$ and every cluster $C \in P$, we have $diam(C) \leq \Delta$.*

2. ***Separation Parameter:*** *For every $x, y \in X$,*

$$\Pr_{P \sim F}[P(x) \neq P(y)] \leq \beta \frac{\rho(x, y)}{\Delta},$$

where $P(x)$ denotes the cluster of P containing $x \in X$.

The parameter β measures the quality of the decomposition, with lower values of β indicating better partitions. For general n-point metric spaces, one can achieve $\beta = O(\log n)$ and this bound is tight, as shown by Bartal [Bar96]. For planar graphs, $\beta = O(1)$ is achievable [KPR93, Rao99], and for d-dimensional Euclidean spaces, $\beta = O(\sqrt{d})$ [CCG+98] and this bound is tight.

For shortest path metrics on general graphs, several fast decomposition algorithms have been proposed to efficiently construct partitions. Notably, Mendel and Schwob [MS09] achieve time complexity $O(m \log n + n \log^2 n)$, where m is the number of edges and n is the number of vertices, which is particularly effective for sparse graphs. Miller, Peng and Chen [MPX13] achieve linear time complexity $O(m)$ and their algorithm can leverage parallelism to handle large-scale graphs efficiently. Both algorithms achieve separation parameter $\beta = O(\log n)$. However, for Euclidean spaces, where we consider

the complete graph with $m = \Theta(n^2)$, the running time of these algorithms becomes $O(n^2)$ or more, while achieving $\beta = O(\log n)$ regardless of the dimension d.

This thesis focuses on developing and analyzing fast algorithms for constructing partitions of high-dimensional (dimension $d = \Omega(\log n)$) Euclidean spaces. The algorithm presented in [CCG+98] lacks an explicit implementation and a naive one results in time complexity of $\tilde{O}(n^3)$ for n points. Our main theorem, presented below (Theorem 1.1), significantly improves upon this bound.

To simplify the problem, we leverage the Johnson-Lindenstrauss (JL) Lemma [JL84]. The JL Lemma states that every set of n points in a high-dimensional Euclidean space can be embedded into $\mathbb{R}^d$, where $d = O(\log n)$, while preserving pairwise distances within a small multiplicative error. This dimensionality reduction ensures that the problem remains well-posed and manageable in $O(\log n)$-dimensional space, with only a constant factor impact on the final separation parameter. Moreover, the dimensionality reduction can be performed efficiently with high probability of success. By using fast JL transforms [AC09, KN14], the dimensionality reduction step can be applies on each point in time proportional to the original dimension.

Theorem 1.1. *Let $V \subseteq \mathbb{R}^d$ of size n for $d = \Theta(\log n)$ and $\Delta > 0$. Then there exist an algorithm that reports a partition P sampled from an $(O(\sqrt{d}), \Delta)$-separating decomposition of V in $O(n^{1+1/c^2+o(1)}) = O(n^{1.51})$ time where $c = 1.4$.*

To achieve our result, we first present an explicit implementation of the known algorithm that improves the running time to $\tilde{O}(n^2)$. We then develop an even faster algorithm that incurs an additive error (in the separation probability), by utilizing Locality Sensitive Hashing (LSH) [IM98] techniques to quickly identify points that are likely to be in the same cluster, in order to reduce the number of distance computations. Finally, we present a method to eliminate the additive error while maintaining the improved time complexity, by effectively proving that the additive error is negligible for sufficiently distant pairs of points and forcing close pairs of points to cluster together.

For low dimension (i.e. $d = o(\log n)$), a similar result can be achieved by padding the points with zeros to embed them into $\mathbb{R}^{O(\log n)}$ and running the same algorithm. The running time remains unchanged, but the separation parameter is $\beta = O(\sqrt{\log n})$ instead of $O(\sqrt{d})$. Alternatively, one could use the algorithm from [CCG+98], which provides a better $\beta = O(\sqrt{d})$ with polynomial running time $\tilde{O}(n^2)$, or use grid-based methods to achieve $\beta = O(d)$ in linear time.

In Section 6, we explore applications of our fast partitioning algorithm. We demonstrate how our methods improve the efficiency of approximation algorithms for the Minimum Communication Cost Spanning Tree (MCST) problem and the construction of high-dimensional Euclidean tree covers, offering running time improvements over previous approaches.

Future work in this area could explore extending these techniques to other metric spaces, investigating lower bounds on the time complexity for constructing separating decompositions and applying these fast partitioning methods to specific problems in machine learning or network design.

2 Running Time Analysis for [CCG+98]

Theorem 2.1 ([CCG+98]). *Let $d > 1$, for every n-point set $V \subseteq \mathbb{R}^d$ and for every $\Delta > 0$ there exists an $(O(\sqrt{d}), \Delta)$-separating decomposition of $(V, \|.\|_2)$.*

The algorithm for constructing such a decomposition is not provided with an explicit implementation in [CCG+98]. A naive implementation results in a time complexity of $\tilde{O}(n^3)$ in the worst case. We omit the dependence on d, as we assume $d = O(\log n)$ using the Johnson-Lindenstrauss Lemma [JL84], ensuring that any factor of d remains polylogarithmic in n and does not affect the overall asymptotic complexity. Specifically, the original algorithm draws balls of radius $r = \Delta/2$, denoted $B(v, r)$, around each point $v \in V$ and repeatedly selects points uniformly at random from the region defined by the union of these balls, $\bigcup_{v \in V} B(v, r)$. Each time a point z is chosen, all points within distance r from z are grouped into a cluster, i.e., $B(z, r) \cap V$, and removed from V. The process continues until all points are clustered.

The original algorithm does not explicitly define how to sample a random point z from the union of balls $\bigcup_{v \in V} B(v, r)$. However, to implement this step correctly, one must ensure a uniform distribution over this region. Since there is no direct way to sample from a union of arbitrary balls efficiently, our implementation first selects a random point $v \in V$ and then chooses $z \in B(v, r)$ uniformly at random. To account for the varying densities of different regions, a rejection sampling step is required: z is rejected (discarded) with probability proportional to $1 - 1/|B(z, r) \cap V|$ and a new point is picked. This ensures that the final choice of z follows the correct distribution over the entire union. Each selection of z requires $\tilde{O}(n)$ time for computing the intersection. The rejection sampling process takes $\tilde{O}(n)$ time since each z is rejected w.p. at most $1 - 1/n$. Since the algorithm requires up to $O(n)$ iterations to cluster all points, the total running time is $\tilde{O}(n^3)$.

In this section, we present an explicit algorithm that improves upon the naive $\tilde{O}(n^3)$ worst-case complexity by incorporating specific optimizations to achieve a time complexity of $\tilde{O}(n^2)$ with high probability. The key idea is to gradually compute $B(z_v, r) \cap V$ while evaluating the rejection sampling on the fly to avoid unnecessary computations. In other words, rather than first computing all distances from a randomly selected z and then applying rejection sampling, we combine both steps. A random $\alpha \in [0, 1]$ is chosen to determine the rejection threshold. The algorithm iterates through V in random order and counts how many points lie in $B(z_v, r)$, and stops if the count exceeds $1/\alpha$. If so, z is immediately rejected, avoiding redundant distance computations.

Theorem 2.2. *Algorithm 1 reports an $(O(\sqrt{d}), \Delta)$-separating decomposition of V and w.h.p. runs in $\tilde{O}(n^2)$ time.*

The key to proving this theorem is the following lemma.

Lemma 2.3. *The number of iterations of the outer loop (starting in line 2) is w.h.p. at most $O(n \log n)$.*

Proof of Theorem 2.2 assuming Lemma 2.3. For each iteration in the algorithm, the running time is dominated by the computation of I_z (the scan in line 5), which in the worst case is bounded by n. It follows that the overall running time is w.h.p. at most $O(n^2 \log n)$.

Algorithm 1 Separating Decomposition

Require: $V \subseteq \mathbb{R}^d$ of size n and $\Delta > 0$

1: $r \leftarrow \frac{\Delta}{2}$, $P \leftarrow \emptyset$
2: **while** $V \neq \emptyset$ **do**
3: choose $v \in V$, $z_v \in B(v, r)$ and $\alpha \in [0, 1]$ uniformly at random
4: $I_z \leftarrow \emptyset$
5: **scan** every $u \in V$ in a random order π ▷ the scan includes rejection sampling and might terminate early
6: **if** $\|u - z_v\| \leq r$ **then**
7: $I_z \leftarrow I_z \cup \{u\}$
8: **if** $|I_z| > \frac{1}{\alpha}$ **then**
9: terminate the scan ▷ reject z_v
10: **if** the scan completed **then** ▷ upon completion, $I_z = B(z_v, r) \cap V$
11: $V \leftarrow V \backslash I_z$, $P \leftarrow P \cup \{I_z\}$
12: report P

The correctness of the algorithm is proved as in [CCG+98] (Section 3.1). Since $\Pr_\alpha[|B(z_v, r) \cap V| > 1/\alpha] = 1 - 1/|B(z_v, r) \cap V|$, it's easy to see that by choosing z from a random ball and using rejection sampling, the distribution of z is uniform over the union of all balls. It follows that if $u, v \in V$ and since z is chosen uniformly from $\bigcup_{v \in V} B(v, r)$,

$$\Pr[P(u) \neq P(v)] = \frac{Vol(B(u, r) \triangle B(v, r))}{Vol(B(u, r) \cup B(v, r))} \leq O(\sqrt{d}) \frac{\|u - v\|}{\Delta},$$

where the last inequality is from [CCG+98]. □

2.1 Bounding the Number of Iterations (Proof of Lemma 2.3)

We now prove Lemma 2.3 by showing that the expected size of the set V decreases with each iteration. Let V_j denote the set V after j iterations of the outer loop (line 2), and after the iterations terminate, define $V_j = \emptyset$. Let n_j denote the size of V_j, i.e., $n_j = |V_j|$. We now aim to show that, at every time $t \geq 0$, the expected size of the set after j additional iterations, i.e., n_{t+j}, decreases by at least j from the initial size n_t (modulo boundary conditions).

Lemma 2.4.

$$\forall t, j \geq 0, \quad \mathbb{E}[n_{t+j} | n_t] \leq \max(n_t - j, 1).$$

Proof of Lemma 2.3 assuming Lemma 2.4. Let $t \geq 0$. By Markov's inequality

$$\Pr[n_{t+n} \geq 2 | n_t] \leq \frac{\mathbb{E}[n_{t+n} | n_t]}{2} \leq \frac{\max(n_t - n, 1)}{2} \leq \frac{1}{2}$$

Thus,

$$\Pr[n_{n \log n} \geq 2] = \Pr[n_n \geq 2 | n_0] \Pr[n_{2n} \geq 2 | n_n] \dots \Pr[n_{n \log n} \geq 2 | n_{n(\log n - 1)}] \leq \frac{1}{2^{\log n}} \leq \frac{1}{n},$$

which means that after $O(n \log n)$ iterations, with high probability $|V|$, i.e., the number of points left to cluster is at most 1. If $|V| = 0$, the algorithm terminates successfully. If $|V| = 1$, the way the algorithm is designed ensures that it will cluster the remaining point in the next iteration and then terminate.

□

Proof of Lemma 2.4. Let $t \geq 0$ and proceed by induction on j. In the base case, $j = 0$: trivially $\mathbb{E}[n_t | n_t] = n_t \leq \max(n_t, 1)$. For the inductive step, first consider $1 \leq j < n_t$, and assume that $\mathbb{E}[n_{t+j-1} | n_t] = n_t - (j-1)$. If $n_{t+j-1} \leq 1$, then clearly $\mathbb{E}[n_{t+j} | n_t, n_{t+j-1} \leq 1] \leq 1$. Now assume $n_{t+j-1} \geq 2$ and consider iteration $t + j$. The scan in line 5 will be completed if and only if $|B(z_v, r) \cap V_{t+j-1}| \leq 1/\alpha$ which happens w.p. $\frac{1}{|B(z_v,r) \cap V_{t+j-1}|}$. Thus,

$$\mathbb{E}_{v,z_v,\alpha}[n_{t+j-1} - n_{t+j} \mid n_t, n_{t+j-1} \geq 2] = \mathbb{E}_{v,z_v}[\frac{1}{|B(z_v, r) \cap V_{t+j-1}|} \cdot |B(z_v, r) \cap V_{t+j-1}| + (1 - \frac{1}{|B(z_v, r) \cap V_{t+j-1}|}) \cdot 0 \mid n_t, n_{t+j-1} \geq 2] = 1$$

Thus,

$$\mathbb{E}[n_{t+j} \mid n_t, n_{t+j-1} \geq 2] = \mathbb{E}[n_{t+j-1} \mid n_t, n_{t+j-1} \geq 2] - 1$$

By the law of total expectation, and since the size of the set is non-increasing, $n_{t+j-1} \geq 2$ implies $n_{t+j-2} \geq 2$

$$\begin{aligned} &= \mathbb{E}[\mathbb{E}[n_{t+j-1} \mid n_t, n_{t+j-1} \geq 2, n_{t+j-2} \geq 2]] - 1 \\ &= \mathbb{E}[n_t - (j-1) \mid n_t] - 1 \\ &= n_t - j \\ &\leq \max(n_t - j, 1). \end{aligned}$$

In the remaining case, since the size of the set is non-increasing,

$$\forall j \geq n_t, \quad \mathbb{E}[n_{t+j} \mid n_t, n_{t+j-1} \geq 2] \leq \mathbb{E}[n_{t+n_t-1} \mid n_t, n_{t+n_t-2} \geq 2] = 1 \leq \max(n_t - j, 1).$$

□

2.2 Time Analysis for Each Iteration

The following lemma shows that we can tighten the bound on the running time of each iteration in some cases.

Lemma 2.5. *Given $v \in V, z_v \in B(v, r)$, the expected time required for the j^{th} iteration of the outer loop (line 2) of Algorithm 1 is at most $O(\frac{n_j}{c_{j,z_v}} \log n)$ where $c_{j,z_v} = |B(z_v, r) \cap V_j|$*

The expectation is taken over the randomness in the selection of $\alpha \in [0, 1]$ and the random permutation π used.

Proof. Let T be a random variable representing the number of points scanned before the j^{th} iteration is terminated. Then $T = \min(X + \frac{1}{\alpha}, n_j)$ where X represents the number of points encountered in $V_j \backslash B(z_v, r)$. The scan terminates when it finds more than $\frac{1}{\alpha}$ points in $B(z_v, r) \cap V_j$ which implies that $X \sim NHG(n_j, n_j - c_{j,z_v}, \frac{1}{\alpha})$ (Negative Hyper-Geometric distribution; see [JKK05], Section 6.2.2).

$$\mathbb{E}[X] = \frac{1}{\alpha} \frac{n_j - c_{j,z_v}}{n_j - (n_j - c_{j,z_v}) + 1} = \frac{1}{\alpha} \frac{n_j - c_{j,z_v}}{c_{j,z_v} + 1} \leq \frac{1}{\alpha} \frac{n_j - c_{j,z_v}}{c_{j,z_v}}.$$

Now, by the law of total expectation

$$\begin{aligned}
\mathbb{E}_{\alpha,\pi}[T \mid v, z_v, V_j] &= \mathbb{E}_\alpha \mathbb{E}_\pi[T \mid \alpha, v, z_v, V_j] \\
&= \mathbb{E}_\alpha \mathbb{E}_\pi[\min(X + \frac{1}{\alpha}, n_j) \mid \alpha, v, z_v, V_j]
\end{aligned}$$

since $\min(\cdot, n_j)$ is a concave function, by Jensen's inequality

$$\begin{aligned}
&\leq \mathbb{E}_\alpha[\min(\mathbb{E}_\pi[X] + \frac{1}{\alpha}, n_j) \mid v, z_v, V_j] \\
&\leq \mathbb{E}_\alpha[\min(\frac{1}{\alpha} \frac{n_j}{c_{j,z_v}}, n_j) \mid v, z_v, V_j] \\
&= \frac{n_j}{c_{j,z_v}} \int_0^1 \min(\frac{1}{a}, c_{j,z_v})\, da \\
&= \frac{n_j}{c_{j,z_v}} (\int_0^{\frac{1}{c_{j,z_v}}} c_{j,z_v}\, da + \int_{\frac{1}{c_{j,z_v}}}^1 \frac{1}{a}\, da) \\
&= \frac{n_j}{c_{j,z_v}} (1 + \ln c_{j,z_v}) \\
&\leq \frac{n_j}{c_{j,z_v}} (1 + \ln n)
\end{aligned}$$

□

3 Fast Algorithm with Additive Error

In this section, we present a faster algorithm (Theorem 3.2) that incurs an additive error in the separation probability. This is achieved by utilizing Locality Sensitive Hashing (LSH) [IM98] to quickly identify points that may belong to the same cluster, thereby reducing the number of distance computations. We then combine this approach with Lemma 2.5 to improve the (worst-case) time bound.

In Algorithm 1, the computation of I_z takes $O(n)$ time in the worst case because it might need to examine all points in V. To improve the running time, our aim is to limit the set of points to only those that are sufficiently close to z, and find these points quickly.

Definition 3.1 (Locality Sensitive Hashing [IM98])**.** *A family $\mathcal{H}$ of functions $h : \mathbb{R}^d \to \mathbb{Z}$ is called* (r, cr, P_1, P_2)-sensitive *if for every two points $p, q \in \mathbb{R}^d$:*

- *If $\|p - q\| \leq r$ then $\Pr_{h \in \mathcal{H}}[h(q) = h(p)] \geq p_1$,*

- *If* $\|p-q\| \geq cr$ *then* $\Pr_{h\in\mathcal{H}}[h(q)=h(p)] \leq p_2$.

We assume $p_1 > p_2$ *which is needed for the LSH family to be useful and define* $\rho = \frac{\log 1/p_1}{\log 1/p_2}$.

Theorem 3.1 ([AI06]). *For every scale* $r > 0$*, dimension* $d > 0$*, and* $c > 1$*, there exists an* $(r, cr, \frac{1}{n^\rho}, \frac{1}{n})$*-sensitive family of hash functions for* $V \subseteq \mathbb{R}^d$ *of size* n*, where* $\rho = \frac{1}{c^2} + o(1)$ *and each function can be evaluated in time* $\tau = O(dn^{o(1)} \log n)$.

Definition 3.2 (Separating Decomposition with Additive Error). *Let* (X, ρ) *be a metric space. A distribution* F *over partitions of* X *is called a* (β, Δ)-separating decomposition with additive error $\gamma \in [0,1]$ *if the following conditions are satisfied:*

1. ***Diameter:*** *For every partition* $P \in supp(F)$ *and every cluster* $C \in P$*, we have* $diam(C) \leq \Delta$.
2. ***Separation Parameter:*** *For every* $x, y \in X$,

$$\Pr_{P\sim F}[P(x) \neq P(y)] \leq \beta\frac{\rho(x,y)}{\Delta} + \gamma,$$

where $P(x)$ *denotes the cluster of* P *containing* $x \in X$.

Algorithm 2 follows the same overall structure of Algorithm 1, the key difference is how the scan for computing I_z is performed. In Algorithm 1, for each chosen z, the scan iterates over all points in V to find those within distance r. Algorithm 2 uses LSH to reduce the number of points examined. Instead of scanning the entire set V, it scans only the points found by the LSH, which contain points that are close to z and points that "might be" close to z.

Algorithm 2 Fast Separating Decomposition with Additive Error

Require: $V \subseteq \mathbb{R}^d$ of size n and $\Delta > 0$

$r \leftarrow \frac{\Delta}{2}$, $P \leftarrow \emptyset$
pick $k = O(n^\rho \log n)$ hash functions $h_1, \ldots, h_k \in \mathcal{H}$ as in Theorem 3.1 and evaluate them on all points in V
while $V \neq \emptyset$ **do**
    choose $v \in V$, $z_v \in B(v,r)$ and $\alpha \in [0,1]$ uniformly at random
    $B_{z_v} \leftarrow \bigcup_{i=1}^{k} h_i^{-1}(h_i(z_v))$ where $h^{-1}(\cdot)$ is the preimage of h in V
    $I_z \leftarrow \emptyset$
    scan every $u \in B_{z_v}$ in a random order π
        if $\|u - z_v\| \leq r$ **then**
            $I_z \leftarrow I_z \cup \{u\}$
            if $|I_z| > \frac{1}{\alpha}$ **then**
                terminate the scan ▷ reject z_v
    if the scan completed **then** ▷ upon completion, $I_z = B(z_v, r) \cap B_{z_v}$
        $V \leftarrow V \backslash I_z$, $P \leftarrow P \cup \{I_z\}$
        remove all $v \in I_z$ from h_j for all j
report P

Theorem 3.2. *When Algorithm 2 is executed on $V \subseteq \mathbb{R}^d$ of size n, it reports a sample from an $(O(\sqrt{d}), \Delta)$-separating decomposition with additive error $\frac{1}{n^4}$ of V. If $d = \Theta(\log n)$ then it runs in expected $\tilde{O}(n^{1+1/c^2+o(1)})$ time for $c = 1.4$.*

We prove this theorem in subsections 3.1 and 3.2.

3.1 Additive Error Analysis

Lemma 3.3. *Algorithm 2 reports a sample from an $(O(\sqrt{d}), \Delta)$-separating decomposition with additive error $\frac{1}{n^4}$ of V.*

Proof. First, the output P always is a partition of V since it contains disjoint subsets whose union is exactly V. By line 8 of the algorithm, the diameter requirement is met.

The rejection sampling process ensures that the chosen z is uniformly distributed over the union of balls centered at points in V with radius r. Given two points $x, y \in V$, they will be assigned to different clusters if the first point z^* chosen from $B(x,r) \cup B(y,r)$ is in $B(x,r) \triangle B(y,r)$ and not in $B(x,r) \cap B(y,r)$, or if $z^* \in B(x,r) \cap B(y,r)$ but one of x, y is not in B_{z^*}.

$$\begin{aligned}\Pr[P(x) \neq P(y)] = \Pr[z^* \in B(x,r) \triangle B(y,r)] \\ + \Pr[P(x) \neq P(y) \mid z^* \in B(x,r) \cap B(y,r)] \Pr[z^* \in B(x,r) \cap B(y,r)]\end{aligned}$$

Since the distribution of z^* is uniform over $B(x,r) \cup B(y,r)$, the two events where one of x, y is not in B_{z^*} are symmetric.

$$\begin{aligned}\Pr[P(x) \neq P(y)] &= \Pr[z^* \in B(x,r) \triangle B(y,r)] \\ &\quad + 2\Pr[x \notin B_{z^*}, y \in B_{z^*} \mid z^* \in B(x,r) \cap B(y,r)] \Pr[z^* \in B(x,r) \cap B(y,r)] \\ &\leq \frac{Vol(B(x,r) \triangle B(y,r))}{Vol(B(x,r) \cup B(y,r))} \\ &\quad + 2\Pr[x \notin \bigcup_{i=1}^{k} h_i^{-1}(z^*) \mid z^* \in B(x,r) \cap B(y,r)] \frac{Vol(B(x,r) \cap B(y,r))}{Vol(B(x,r) \cup B(y,r))}\end{aligned}$$

By the computations from [CCG+98] (Section 3.1)

$$\begin{aligned}&\leq O(\sqrt{d})\frac{\|x-y\|}{\Delta} + 2\Pr[\forall i, h_i(x) \neq h_i(z^*) \mid z^* \in B(x,r) \cap B(y,r)] \\ &= O(\sqrt{d})\frac{\|x-y\|}{\Delta} + 2\Pr[h(x) \neq h(z^*) \mid z^* \in B(x,r) \cap B(y,r)]^k \\ &\leq O(\sqrt{d})\frac{\|x-y\|}{\Delta} + 2(1 - \frac{1}{n^\rho})^k\end{aligned}$$

Choosing $k = 5n^\rho \ln n$,

$$\leq O(\sqrt{d})\frac{\|x-y\|}{\Delta} + \frac{1}{n^4}.$$

□

3.2 Running Time Analysis

In this section we prove the following Lemma.

Lemma 3.4. *Algorithm 2 runs in expected* $\tilde{O}(dn^{1+1/c^2+o(1)} + n^2 e^{-\Omega(d)})$ *time where* $c = 1.4$.

Recall Lemma 2.5 which analyzes the expected time per iteration in Algorithm 1, where the scan is performed over the entire set V_j. In this case, $n_j = |V_j|$ represents the number of remaining unclustered points, and $c_{j,z_v} = |B(z_v, r) \cap V_j|$ denotes the number of points within distance r of z_v. In Algorithm 2, instead of scanning the entire set V, we only scan over B_{z_v}, the set of points that share a locality-sensitive hash value with z_v. This change reduces the size of the search space from $|V_j|$ to $|B_{z_v}|$. By applying the same analysis as in Lemma 2.5, we obtain Lemma 3.5, where the expected iteration time is now determined by the ratio $|B_{z_v}|/|B_{z_v} \cap B(z_v, r)|$.

Lemma 3.5. *Given* $v \in V, z_v \in B(v, r)$, *the expected time required for the* j^{th} *iteration of the outer loop (line 2) of Algorithm 2 is at most* $O(\frac{|B_{z_v}|}{c_{h,j,z_v}} \log n)$, *where* $c_{h,j,z_v} = |B_{z_v} \cap B(z_v, r)|$.

We prove the following lemma at the end of this section through a direct calculation. The intuition behind the lemma is based on geometric properties in high-dimensional spaces. With high probability, for a random point $z \in B(v, r)$, the distance $\|z - v\|$ will be close to r, and the inner product $\langle z - v, v - u\rangle$ will be close to zero, meaning that the vectors $z - v$ and $v - u$ are nearly orthogonal. This orthogonality leads to the approximation $\|z-u\|^2 \approx \|z-v\|^2+\|v-u\|^2$, which in turn implies that approximately $\|z-u\|^2 \geq 2r^2$. Therefore, the probability that z lies within the range $r \leq \|z-u\| \leq cr$ is exponentially small in d.

Lemma 3.6. *Let* $r > 0$, *let* $1 < c \leq 1.4$ *and let* $u, v \in \mathbb{R}^d$ *such that* $r \leq \|u - v\| \leq cr$. *Then for* $z \in B(v, r)$ *chosen randomly,* $\Pr_z[r < \|z - u\| \leq cr] \leq e^{-\Omega(d)}$.

Proof of Lemma 3.4. First, let us analyze the time for iteration j of the outer loop (line 2). In the j^{th} iteration, after selecting a random point $z_v \in B(v, r)$, the algorithm scans points from the preimage of the hash functions, denoted as B_{z_v}. Ideally, B_{z_v} should contain only the points within $B(z_v, r)$. However, due to the probabilistic nature of LSH, additional points may be included. Specifically, points that lie in the range $r < \|u - z_v\| \leq cr$, and distant points with $\|u - z_v\| > cr$. Our goal is to bound the contribution of these errors to the overall running time.

$$\forall i \in [k], \quad h_i^{-1}(h_i(z_v)) = \{u \in h_i^{-1}(h_i(z_v)) \mid \|u - z_v\| \leq r\} \cup \{u \in h_i^{-1}(h_i(z_v)) \mid r < \|u - z_v\| \leq cr\} \cup \{u \in h_i^{-1}(h_i(z_v)) \mid \|u - z_v\| > cr\}$$

Which means,

$$B_{z_v} = \bigcup_{i=1}^{k} h_i^{-1}(h_i(z_v)) \subseteq (B_{z_v} \cap B(z_v, r)) \cup \{u \in V_j \mid r < \|u - z_v\| \leq cr\} \cup \bigcup_{i=1}^{k} \{u \in h_i^{-1}(h_i(z_v)) \mid \|u - z_v\| > cr\}$$

Denote

$$Err_{z_v} = \{u \in V_j \mid r < \|u - z_v\| \leq cr\}$$
$$Err_h(h_i) = \{u \in h_i^{-1}(h_i(z_v)) \mid \|u - z_v\| > cr\}$$

We can bound these as follows. Using Lemma 3.6,

$$\begin{aligned}\mathbb{E}_{z_v}[|Err_{z_v}|] &= \mathbb{E}_{z_v}[|\{u \in V_j \mid r < \|u - z_v\| \leq cr\}|] \\ &= \sum_{u \in V_j} \Pr_{z_v}[r \leq \|u - z_v\| \leq cr] \leq ne^{-\Omega(d)}.\end{aligned}$$

Using Theorem 3.1 and the law of total expectation

$$\begin{aligned}\mathbb{E}_{z_v,h}[|\bigcup_{i\in[k]} Err_h(h_i)|] &= \mathbb{E}_{z_v}\mathbb{E}_h[|\bigcup_{i\in[k]} Err_h(h_i)| \mid z_v] \\ &= \mathbb{E}_{z_v}\mathbb{E}_h[|\bigcup_{i\in[k]} \{u \in h_i^{-1}(h_i(z_v)) \mid \|u - z_v\| > cr\}| \mid z_v] \\ &= \sum_{u \in V_j} \Pr[\exists i, h_i(u) = h_i(z_v) \mid \|u - z_v\| > cr] \\ &\leq n \cdot \frac{k}{n} = k.\end{aligned}$$

Now,

$$|B_{z_v}| \leq c_{h,j,z_v} + |Err_{z_v}| + |\bigcup_{i\in[k]} Err_h(h_i)|.$$

By Lemma 3.5, and since $v \in B_{z_v} \cap B(z_v, r)$ we know that $c_{h,j,z_v} \geq 1$,

$$\begin{aligned}\mathbb{E}_{h,z_v}[\text{time for the } j^{th} \text{ iteration} \mid V_j] &\leq O(\mathbb{E}_{h,z_v}[\frac{|B_{z_v}|}{c_{h,j,z_v}} \log n \mid V_j]) \\ &\leq O((1 + \mathbb{E}_{h,z_v}|Err_{z_v}| + \mathbb{E}_{h,z_v}|\bigcup_{i\in[k]} Err_h(h_i)|) \cdot \log n) \\ &\leq O((ne^{-\Omega(d)} + k) \log n) \\ &\leq O((ne^{-\Omega(d)} + n^{\rho} \log n) \log n)\end{aligned}$$

By Theorem 3.1, the preprocessing time for evaluating the LSH functions is $O(kn\tau)$ where $\tau = O(dn^{o(1)} \log n)$.

The overall running time consists of the preprocessing time for the LSH functions, plus the number of iterations multiplied by the time required for each iteration.

$$\tilde{O}(dn^{1+\rho}) + \tilde{O}(n) \cdot \tilde{O}(ne^{-\Omega(d)} + n^{\rho}) = \tilde{O}(dn^{1+\rho} + n^2 e^{-\Omega(d)}).$$

□

Proof of Lemma 3.6. Without loss of generality, we can assume $v = (0, ..., 0)$ and $u = (ar, 0, ..., 0)$ where $1 \leq a \leq c$. It is known that a random $z \in B(v, r)$ can be chosen also by picking $z = rx^{\frac{1}{d}} \frac{z'}{\|z'\|}$ where $z' \sim N(0, \frac{1}{d}I), x \sim U[0, 1]$. Now, $\frac{\|z'\|^2}{1/d} \sim \chi^2_d$ and by known tail bounds for the chi-squared distribution [LM00],

$$\forall y > 0, \quad \Pr_{z'}[\frac{\|z'\|^2}{1/d} \geq 2\sqrt{dy} + 2y + d] < e^{-y},$$

$$\forall y > 0, \quad \Pr_{z'}[\frac{\|z'\|^2}{1/d} \leq -2\sqrt{dy} + d] < e^{-y}$$

and taking $y = \frac{d}{40{,}000}$,

$$\Pr[\frac{\|z'\|^2}{1/d} \geq \frac{d}{100} + \frac{d}{20,000} + d] < e^{-\Omega(d)} \Rightarrow \Pr[\|z'\|^2 \geq 1.01005] < e^{-\Omega(d)}.$$

$$\Pr[\frac{\|z'\|^2}{1/d} \leq -\frac{d}{100} + d] \Rightarrow \Pr[\|z'\|^2 \leq 0.99] < e^{-\Omega(d)}$$

which means that w.h.p. $\|z'\|^2 \in (0.99, 1.02)$. Additionally,

$$\Pr_x[x^{\frac{1}{d}} < e^{-0.01}] = \Pr_x[x < e^{-0.01d}] = e^{-0.01d} = e^{-\Omega(d)}.$$

and thus w.h.p. $x^{\frac{1}{d}} \in (e^{-0.01}, 1)$. We shall assume henceforth that all these high-probability events indeed occur. Now,

$$\|z - u\|^2 = (ar - z_1)^2 + \sum_{i=2}^{d} z_i^2 = a^2r^2 - 2arz_1 + \sum_{i=1}^{d} z_i^2$$

so we would like to estimate z_1 and $\sum_{i=1}^{d} z_i^2$. By using the tail bound for gaussians,

$$\begin{aligned}\Pr[z_1 > \frac{r}{200}] &= \Pr[x^{\frac{1}{d}} \frac{z_1'}{\|z'\|} > \frac{1}{200}] \\ &\leq \Pr[z_1' > \frac{\sqrt{0.99}}{200}] \\ &\leq e^{-(\frac{\sqrt{0.99d}}{200})^2/2} = e^{-\Omega(d)}\end{aligned}$$

then w.h.p. also $z_1 < \frac{r}{200}$. By the events we assumed before to occur, we have

$$\sum_{i=1}^{d} z_i^2 = \|z\|^2 = r^2 x^{\frac{2}{d}} \geq \frac{r^2}{e^{0.02}}.$$

Since $z \in B(v,r)$ taking $a = 1$ will create a lower bound for the distance. Combining everything,

$$\begin{aligned}\|z-u\|^2 &= a^2r^2 - 2arz_1 + \sum_{i=1}^{d} z_i^2 \\ &\geq r^2 - \frac{r^2}{100} + e^{-0.02}r^2 \\ &> 1.97r^2 \\ &> c^2r^2.\end{aligned}$$

We conclude that

$$\Pr_z[r < \|z-u\| \leq cr] \leq \Pr_z[\|z-u\| \leq cr] \leq e^{-\Omega(d)}.$$

□

4 Removing the Additive Error

We adopt a black-box approach to address the additive error. Given an algorithm that constructs a separating decomposition with a sufficiently small additive error, we first show that for sufficiently distant points, the additive error is negligible. This implies that by forcing close pairs to cluster together, we can create a partition with no additive error.

To achieve this, we repeat the given algorithm until the output is a separating decomposition where all sufficiently close pairs are clustered together, thereby constructing a partition with no additive error.

To keep the construction relatively fast, we require an efficient method to verify whether all sufficiently close pairs are clustered together for each partition sampled during the repetitions of the given algorithm (the naive approach, which involves checking all pairs explicitly, has a time complexity of $O(n^2)$).

We outline the fast verification process, addressing the challenge of ensuring both the efficiency of each verification step and the overall number of iterations required to achieve the desired result.

The following lemma demonstrates that the additive error becomes negligible for pairs that are sufficiently far apart.

Lemma 4.1. *Let $V \subseteq \mathbb{R}^d$ of size n and let P be a partition of V chosen from a $(O(\sqrt{d}), \Delta)$-Separating Decomposition with additive error $\frac{1}{n^4}$. Then,*

$$\forall u,v \in V : \|u-v\| > \frac{1}{\sqrt{d}} \cdot \frac{\Delta}{n^4}, \quad \Pr_P[P(u) \neq P(v)] \leq O(\sqrt{d})\frac{\|u-v\|}{\Delta}$$

Proof. If $\|u-v\| > \frac{1}{\sqrt{d}} \cdot \frac{\Delta}{n^4}$, then,

$$\Pr_P[P(u) \neq P(v)] \leq O(\sqrt{d})\frac{\|u-v\|}{\Delta} + \frac{1}{n^4} \leq O(\sqrt{d})\frac{\|u-v\|}{\Delta} + \sqrt{d}\frac{\|u-v\|}{\Delta} \leq O(\sqrt{d})\frac{\|u-v\|}{\Delta}.$$

□

After showing that the additive error for sufficiently far points is negligible, we aim to design a verification process to determine whether all close pairs are clustered together. To achieve this, we first define a set that facilitates the identification of close pairs in the input. We allow a relaxed threshold, meaning that the set may include some farther pairs, in order to enable fast construction of the set.

Definition 4.1 ((c,r)-Close Pairs Set)**.** *Let $V \subseteq \mathbb{R}^d$, $r > 0$ and $c \geq 1$. A (c,r)-*Close Pairs Set*, denoted $S_{c,r}$, is a set of unordered pairs from V that satisfies the following conditions:*

- *if $\|u - v\| \leq r$, then $(u,v) \in S_{c,r}$;*
- *if $\|u - v\| > cr$, then $(u,v) \notin S_{c,r}$.*

Given a partition P, the next step is to determine whether two points from the set belong to the same cluster in P.

To verify certain properties of the pairs in a Close Pairs Set, we use a symmetric and transitive predicate q. For instance, q can determine whether two points belong to the same connected component in a graph. Specifically, in our case, this predicate is used to verify whether two points from the set belong to the same cluster in P.

We prove the following lemma in Section 4.1.

Lemma 4.2. *There exists a deterministic algorithm that given $V \subseteq \mathbb{R}^d$ and $0 < r < \frac{1}{2d+2}$ runs in $\tilde{O}(dn)$ time and constructs (implicitly) an $(O(d^{3/2}), r)$-Close Pairs Set. Given a symmetric and transitive predicate $q : V \times V \to \{0,1\}$, the algorithm can then determine whether all pairs in the set satisfy q in $\tilde{O}(dn)$ time.*

We would like to use Lemma 4.2 with $r = \frac{\Delta}{n^4}$, without loss of generality $\Delta = 1$, which means that $r = \frac{\Delta}{n^4} = \frac{1}{n^4} < \frac{1}{2d+2}$.

Algorithm 3 Fast Separating Decomposition

Require: $V \subseteq \mathbb{R}^d$ of size n, $\Delta > 0$ and an algorithm A that reports an $(O(\sqrt{d}), \Delta)$-Separating Decomposition with additive error $\frac{1}{n^4}$

1: construct an $(O(d^{3/2}), \frac{\Delta}{n^4})$-Close Pairs Set of V, denoted S ▷ use Lemma 4.2
2: **while** true **do**
3: run algorithm A to sample a partition P of V
4: **if** for every $(u,v) \in S$, $P(u) = P(v)$ **then** ▷ use Lemma 4.2
5: report P

Lemma 4.3. *Given a sample P as in line 3, the probability that Algorithm 3 reports this P in line 5 is at least $1 - \frac{1}{n}$.*

Proof. Let $u, v \in S$, Since $\|u - v\| \leq O(d^{3/2})\frac{\Delta}{n^4}$,

$$\Pr_P[P(u) \neq P(v)] \leq O(\sqrt{d})\frac{\|u-v\|}{\Delta} + \frac{1}{n^4} \leq \frac{O(d^2)}{n^4} < \frac{1}{n^3}$$

by taking the union over all $O(n^2)$ possible pairs we get that $\Pr_P[P \text{ is not reported}] < \frac{1}{n}$. □

Theorem 4.4. *Algorithm 3 reports a partition P sampled from an $(O(\sqrt{d}), \Delta)$-separating decomposition of V.*

Proof. Since the given algorithm respects the diameter constraint, Algorithm 3 does as well. In order to bound the probability a pair of points separate by the partition reported by the end of Algorithm 3, we need to analyze the conditional probability associated with the reported partition. Let $u, v \in V$. If $\|u - v\| \leq \frac{\Delta}{n^4}$ then by the constraint on line 4, by the end of Algorithm 3, $\Pr[P(u) \neq P(v) \mid P \text{ is reported}] = 0$. If $\|u - v\| > \frac{\Delta}{n^4}$, then (using Lemma 4.1 and Lemma 4.3),

$$\begin{aligned}
\Pr_P[P(u) \neq P(v) \mid P \text{ is reported}] &= \frac{\Pr_P[P(u) \neq P(v) \wedge P \text{ is reported}]}{\Pr_P[P \text{ is reported}]} \\
&\leq \frac{\Pr_P[P(u) \neq P(v)]}{1 - 1/n} \\
&\leq O(\sqrt{d}) \frac{\|u - v\|}{\Delta}.
\end{aligned}$$

□

Theorem 4.5. *Let $d = \Theta(\log n)$ and let the algorithm used by Algorithm 3 be Algorithm 2, then Algorithm 3 runs in expected $\tilde{O}(n^{1+1/c^2+o(1)})$ time where $c = 1.4$.*

Proof. Since (by Lemma 4.2) the construction and verification of whether the $(O(d^{3/2}), r)$-Close Pairs Set satisfies q takes $\tilde{O}(dn) = \tilde{O}(n)$ time, the running time of Algorithm 3 is dominated by running time of Algorithm 2 multiplied by the expected number of iterations needed for success.

According to Theorem 3.2, Algorithm 2 has an expected running time of $\tilde{O}(n^{1+\rho})$. The number of iterations follows a geometric distribution, and by Lemma 4.3, the success probability of each iteration is at least $1 - \frac{1}{n}$. This means the expected number of iterations is at most $\frac{1}{1-1/n} \leq 2$. As a result, the expected running time of Algorithm 3 is also $\tilde{O}(n^{1+\rho})$. □

4.1 Fast Scanning of an $(O(d^{3/2}), r)$-Close Pairs Set

In this section, we prove Lemma 4.2 by introducing an alternative definition that represents a Close Pairs Set and could also be reported quickly.

Definition 4.2 ((c, r)-Close Pairs Buckets)**.** *Let $V \subseteq \mathbb{R}^d$, $r > 0$, $c > 1$ $m_1, m_2 > 0$. Define (c, r)-*Close Pairs Buckets *$\{B_{i,j}\}_{i \in [m_1], j \in [m_2]}$ with $B_{i,j} \subseteq V$ such that:*

- *For every i, $\{B_{i,j}\}_{j \in [m_2]}$ is a partition of V*
- *If $\|u - v\| \leq r$, then there are i and j such that $u, v \in B_{i,j}$.*
- *If $\|u - v\| > cr$, then there are no i and j such that both $u, v \in B_{i,j}$*

Lemma 4.6. *Let $V \subseteq \mathbb{R}^d$, $r > 0$, $c \geq 1$ $m_1, m_2 > 0$ and let $\{B_{i,j}\}_{i < m_1, j < m_2}$ be (c, r)-Close Pairs Buckets. Then the set $S = \bigcup_{i,j} B_{i,j} \times B_{i,j}$ is a (c, r)-Close Pairs Set.*

Proof. Since the buckets are subsets of V, S is a subset of $V \times V$. Let $u, v \in V$. If $\|u - v\| \leq r$ then there exists i, j such that $u, v \in B_{i,j}$ which means $(u, v) \in S$. if $\|u - v\| > cr$ then there is no i, j such that $u, v \in B_{i,j}$, thus $(u, v) \notin S$. □

Lemma 4.7. *There exists an algorithm that that given $V \subseteq \mathbb{R}^d$ and $0 < r < \frac{1}{2d+2}$ constructs $(O(d^{3/2}), r)$-Close Pairs Buckets in $\tilde{O}(dn)$ time for $m_1 = O(d)$ and some $m_2 \leq n$.*

Consider a uniform grid over $\mathbb{R}^d$ with a fixed side-length τ where each grid cell is indexed by a point $j \in \mathbb{Z}^d$. A point $p \in \mathbb{R}^d$ belongs to a grid cell j if and only if $\forall i \in [d]$, $\lfloor p_i/\tau \rfloor = j_i$.

Definition 4.3. *A point $p \in \mathbb{R}^d$ is central in its τ-grid cell if for each $i \in [d]$, we have $\alpha\tau < p_i - \lfloor p_i/\tau \rfloor \tau < (1 - \alpha)\tau$.*

Observation 4.1. *Let $p, q \in \mathbb{R}^d$. If q is central in its τ-grid cell and $\|p - q\|_\infty \leq \alpha\tau$, then p and q belong to the same τ-grid cell.*

Lemma 4.8 ([Cha97])**.** *For every point $p \in \mathbb{R}^d$ and $l \in \mathbb{N}$, there exists $i \in \{0, 1, \ldots, d\}$ such that $p + v^{(i)}$ for $v^{(i)} = (\frac{i}{d+1}, \ldots, \frac{i}{d+1}) \in \mathbb{R}^d$ is $\frac{1}{2d+2}$-central in its 2^{-l}-grid cell.*

We now aim to construct buckets as shifted grids over $\mathbb{R}^d$. A shifted grid (by $s \in [0, 1]^d$) is one where the grid cells are defined as $j + s$ for every $j \in \mathbb{Z}^d$. Its easy to see that instead of shifting the points, we can shift the grids altogether. This implies the following corollary.

Corollary 4.9. *For every point $p \in \mathbb{R}^d$ and $l \in \mathbb{N}$, there exists $i \in \{0, 1, \ldots, d\}$ such that p is $\frac{1}{2d+2}$-central in its $v^{(i)}$-shifted 2^{-l}-grid cell.*

Proof of Lemma 4.7. Given $0 < r < \frac{1}{2d+2}$, let l be such that $(2d+2)r \in [2^{-(l+1)}, 2^{-l})$ and set $\tau = 2^{-l}$. Define $d + 1$ hash functions $h_0, h_1, \ldots, h_d$ each corresponding to a shifted side-length τ grid in $\mathbb{R}^d$. For each $i \in \{0, \ldots, d\}$, and for any point $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, the hash function $h_i : V \to \mathbb{Z}^d$ is defined by $h_i(x) = (g_i(x_1), \ldots, g_i(x_d))$ where $g_i(y) = \lfloor y/\tau + \frac{i}{d+1} \rfloor$.

For every $0 \leq i \leq d$ and $j \in \mathbb{Z}^d$ define $B_{i,j} = h_i^{-1}(j)$. We'll now show that $\{B_{i,j} \neq \emptyset\}_{i,j}$ are $(O(d^{3/2}), r)$-Close Pairs Buckets. First, $\{B_{i,j}\}_j$ is partition of V. Now consider any two points $u, v \in V$. If $\|u - v\| \leq r$, then

$$\|u - v\|_\infty \leq \|u - v\| \leq r \leq \frac{1}{2d + 2}\tau.$$

By Corollary 4.9, there exists $i \in \{0, \ldots, d\}$ such that u is $\frac{1}{2d+2}$-central in the $\frac{i}{d+1}$ shifted τ-sided grid and by Observation 4.1, we know that u and v are located in the same τ-grid cell, and hence $h_i(u) = h_i(v)$ and both $u, v \in B_{i,h_i(u)}$. Next, the grid cells have diameter at most $\sqrt{d}\tau$,

$$\sqrt{d}\tau \leq 2\sqrt{d}(2d + 2)r \leq O(d^{3/2})r.$$

Thus, for u, v with a distance exceeding the grid cell diameter, which is $O(d^{3/2})r$, u, v are guaranteed not to belong to the same grid cell. Then for all i, we have $h_i(u) \neq h_i(v)$, and consequently, u, v will not both belong to any bucket.

For every $0 \leq i \leq d$, the cost of hashing n points is $\tilde{O}(n)$ and since $\{B_{i,j}\}_j$ is a partition of V, it takes $\tilde{O}(dn)$ time to construct all the buckets. □

Lemma 4.10. *Let $V \subseteq \mathbb{R}^d$, let $r > 0$, $c > 1$, $m_1, m_2 > 0$, let S be a (c, r)-Close Pairs Set generated by $m_1 \cdot m_2$ (c, r)-Close Pairs Buckets and let $q : V \times V \to \{0, 1\}$ be a symmetric and transitive predicate. Then one can determine whether all pairs in the set S satisfy q in $\tilde{O}(m_1 n)$ time.*

Proof. We proceed as follows. For each bucket $B_{i,j}$, iterate through its elements, comparing each element with the next one. If the predicate q returns 0 for any pair, immediately reject. If the iteration is complete over all buckets without the predicate returning 0, accept. We now prove that the algorithm accepts if and only if all pairs in S satisfy q.

- If all pairs in S satisfy q, it is clear that the algorithm accepts, since no pair in the same bucket will return 0 for q.
- If the algorithm accepts, consider any pair $(u, v) \in S$. By the construction of S, there exists a bucket $B_{i,j}$ such that $u, v \in B_{i,j}$. Denote the elements of $B_{i,j}$ as $w_1, w_2, \ldots, w_k$. Since the algorithm accepts, for all $1 \leq k_0 < k$, $q(w_{k_0}, w_{k_0+1}) = 1$. By the symmetry and transitivity of q, this implies that also for $u, v \in B_{i,j}$, $q(u, v) = 1$.

For every $i \in [m_1]$, $|\bigcup_j B_{i,j}| = n$, which means that the running time is $\tilde{O}(m_1 n)$. □

Proof of Lemma 4.2. Using the construction in Lemma 4.7, construct $(O(d^{3/2}), r)$-Close Pairs Buckets. By Lemma 4.10, we can verify whether all pairs in the $(O(d^{3/2}), r)$-Close Pairs Buckets satisfy q in $\tilde{O}(m_1 n) = \tilde{O}(dn)$ time. By Lemma 4.6, the constructed buckets form a (c, r)-Close Pairs Set. Therefore, we achieve the desired construction and verification in $\tilde{O}(dn)$ time. □

5 Hashing Data Structure

In the previous sections, we assume the use of a typical dictionary hashing structure that provides $\tilde{O}(1)$ insertion and deletion operations. This guarantees the expected time bounds. Under this assumption, we now prove some additional guarantees, which, although less typical, are crucial for our time bounds.

Theorem 5.1. *A hash function h has the following properties, given $v \in V$ and their corresponding hash $h(v)$:*

1. *the insertion time for $v, h(v)$ is $\tilde{O}(1)$*
2. *the deletion time for $v, h(v)$ is $\tilde{O}(1)$*
3. *the cost of choosing a random point from $h^{-1}(h(v))$ is $\tilde{O}(1)$*

Proof. We can construct the hash functions in the following way to maintain the desired properties: First, initialize two dictionaries. The first dictionary stores points from V as keys, and the values are the hash values with an additional index. The second dictionary has the hash values as keys, and its values are arrays containing the corresponding points from V, along with the size of each array.

Formally, given a hash function h, let d_1 and d_2 be the two dictionaries, defined as follows:

$$\forall v \in V,\ d_1(v) = (h(v), \text{index}),$$
$$\forall h_0 \in h^{-1}(V),\ d_2(h_0) = (\text{array}, \text{size}),$$

where the array in $d_2(h(v))$ contains the point v, which is indexed by $d_1(v)$.

1. **Insertion:** To insert $v \in V$, we first compute $h(v)$ and append v to the array $d_2(h(v))$ at the end, updating the stored size. Then, we add v to d_1 with the appropriate index.

2. **Deletion:** To delete $v \in V$, we remove v from d_1. Next, to delete v from the array in $\tilde{O}(1)$ we replace v with the last entry of the array, decrease the saved size, and adjust the index in d_1 of the point that was swapped into v's original position.

3. **Randomness:** Since the points corresponding to each hash value are stored in an array, we can generate a random permutation of the indices to select a random point efficiently.

□

6 Applications

6.1 Minimum Communication Cost Spanning Tree

A well-known problem in network optimization is the Minimum Communication Cost Spanning Tree (MCST) problem, first presented by Hu [Hu74]. Its goal is to construct a spanning tree that minimizes the total communication cost in an input graph. More precisely, given a graph $G = (V, E)$ with edge weights $w(e) \geq 0$ for each edge $e \in E$, and a requirement value $y_{u,v} \geq 0$ for each pair of vertices $u, v \in V$, the communication cost of a spanning tree T is defined as

$$C(T) = \sum_{u,v \in V} y_{u,v} \cdot d_T(u, v),$$

where $d_T(u, v)$ is the distance between u and v in the tree. The objective is to find a spanning tree T^* that minimizes $C(T)$.

When working with metric spaces, the input graph is assumed to be a complete graph whose edge weights represent distances. In such cases, every tree on the given set of vertices is a spanning tree, as the graph contains all possible edges.

A widely used approach for approximating metric spaces is to embed them into tree metrics that dominate the original space, meaning that distances in the tree never underestimate those in the original metric. The quality of such an embedding is quantified by its distortion, defined as the worst-case multiplicative stretch, i.e., the maximum ratio of distances in the tree to their corresponding distances in the original space.

Bartal [Bar96] introduced probabilistic embedding into k-hierarchically well-separated trees (k-HSTs), which approximate metric spaces with polylogarithmic distortion (later improved to logarithmic [FRT03]). It is formally defined as follows.

Definition 6.1 ([Bar96]). *A* k-hierarchically well-separated tree *(k-HST) is a rooted weighted tree* $T = (V_T, E_T)$ *satisfying the following properties:*

1. *For every node* $v \in V_T$*, all edges connecting* v *to a child are of equal weight.*
2. *The edge weight along a path from the root to a leaf decrease by a factor of at least* k*.*

Using the following randomized k-HST embedding for an n-point metric (X, d_X), Bartal was able to achieve expected distortion $O(\log^2 n)$, which was later improved to $O(\log n)$ [FRT03].

- partition X into clusters $X_1, \ldots, X_k$ such that $\text{diam}(X_i) \leq \text{diam}(X)/k$ using a separating decomposition
- recursively construct trees $T_1, \ldots, T_k$ for $X_1, \ldots, X_k$
- create a tree T by joining the roots of each T_i to a new root r by edges of length $\text{diam}(X)$

The expected distortion of the resulting tree is equal to the separation parameter (β) of the separating decomposition used multiplied by the recursion depth $\log_k(\text{diam}(X))$ (assuming without loss of generality that all pairwise distances ≥ 1) which can be made $O(\log n)$ after a small optimization. Consequently, the expected distortion can be improved to $O(\log n)$ for planar graphs [KPR93, Rao99] and $O(\sqrt{d}\log n)$ for d-dimensional Euclidean graphs [CCG+98].

By linearity of expectation and the fact that the optimal tree T^* is a spanning tree, the approximation factor for the MCST follows directly from the distortion of the HST.

Corollary 6.1. *Let* (X, d_X) *be an* n*-point metric space, let* T *be a randomized* 2*-HST embedding for* X *with expected distortion* α*, and let* $\{y_{u,v}\}$ *be requirement values for every pair* $u, v \in X$*. Then*

$$\mathbb{E}_T[\sum_{u,v} y_{u,v} \cdot d_T(u,v)] \leq \alpha \sum_{u,v} y_{u,v} \cdot d_X(u,v) \leq \alpha C(T^*)$$

where T^* *is an optimal MCST.*

Our fast partitioning scheme for Euclidean metric spaces follows the same approach used by Bartal, allowing us to substitute our method for his while achieving the same results. Since our construction is faster, we can compute an $O(\sqrt{d} \cdot \log(\text{diam}(X)))$-approximation for the MCST more efficiently. Specifically, the construction requires $O(\log(\text{diam}(X)))$ levels, and using our fast construction, the total time is improved to $\tilde{O}(n^{1+1/c^2+o(1)})$ time where $c = 1.4$.

Corollary 6.2. *There exists a randomized algorithm that given* $X \subseteq \mathbb{R}^d$ *of size* n *where* $d = \Theta(\log n)$*, embeds* X *into the set of leaves of a* 2*-HST with expected distortion* $O(\sqrt{d} \cdot \log(\textit{diam}(X)))$ *in* $\tilde{O}(n^{1+1/c^2+o(1)})$ *time, where* $c = 1.4$*.*

Corollary 6.3. *There exists a randomized algorithm that given* $X \subseteq \mathbb{R}^d$ *of size* n *where* $d = \Theta(\log n)$*, computes an* $O(\sqrt{d} \cdot \log(\textit{diam}(X)))$*-approximation for the Minimum Communication Cost Spanning Tree (MCST) problem in* $\tilde{O}(n^{1+1/c^2+o(1)})$ *time, where* $c = 1.4$*.*

[FRT03] improved the distortion to $O(\log n)$ with a similar tree construction but a different partitioning method, which, according to their Section 2.3, takes $O(n^2)$ time. This result implies an $O(\log n)$ approximation for the MCST problem in polynomial time.

6.2 Tree Covers for High-Dimensional Euclidean Spaces

Let (X, d_X) be a metric space. A *tree cover* for X is a collection of trees $\mathcal{F}$. Each tree in $\mathcal{F}$ has X as its vertex set and has edge weights, such that

$$\forall x, y \in X, \quad d_X(x, y) \leq d_T(x, y).$$

A tree cover $\mathcal{F}$ has stretch $\alpha \geq 1$ if for every $x, y \in X$, there exists a tree T in $\mathcal{F}$ that preserves the distance between x, y up to α factor i.e.

$$\exists T \in \mathcal{F}, \quad d_T(x, y) \leq \alpha d_X(x, y).$$

We call such $\mathcal{F}$ an *α-tree cover of X*.

Tree covers have gained research attention due to their algorithmic importance [GKR05, CGMZ05, GHR06, BFN19, CCL+24]. They have been generalized to various metric spaces and graphs. A key measure of quality for a tree cover is its size $|\mathcal{F}|$. A small tree cover is particularly useful for solving distance-related problems by simplifying them to computations on a few trees.

In many scenarios, constructing a tree cover with the desired stretch property can be achieved probabilistically. By using randomized partitioning techniques, such as (β, Δ)-separating decompositions, we can construct a collection of trees, forming a tree cover, where with high probability for every pair of points, at least one tree preserves the distance between them within an $O(\beta)$ factor.

The next lemma is relatively simple and directly builds on existing methods in the literature. Previous papers have used similar techniques to achieve related results [CGMZ05, HPIS13, FN20], though we have not found this specific statement in the literature, which is why it is included here.

Lemma 6.4. *Let (X, d_X) be an n-point metric space. If there exists an algorithm A that reports a partition sampled from a (β, Δ)-separating decomposition of X for every $\Delta > 0$, then there exists an algorithm that constructs an $O(\beta)$-tree cover of X of size $O(\log n \log(\mathit{diam}(X)))$.*

Proof. First, using algorithm A, for every $\Delta > 0$ we can sample from a (β, Δ)-separating decomposition.

For every $i \in \{0, \ldots, \log(\mathrm{diam}(X))\}$ sample $k = O(\log n)$ partitions from a $(\beta, 2^{i+2}\beta)$-separating decomposition, denoted F_i. For each partition construct a tree as follows. For each cluster, select an arbitrary point, and connect it to all other points in the cluster using the original metric distance edges. Then, select one of these selected points to serve as the root of the tree, and connect the other selected points (one from each cluster) to the root using the original metric distance edges.

Observe that the number of trees is as the number of partitions sampled, $O(\log n \log(\mathrm{diam}(X))$. Let us show that this construction is an $O(\beta)$-tree cover for X. Let $x, y \in X$, there

exists i such that $d_X(x,y) \in [2^i, 2^{i+1})$. For each partition drawn from F_i, we have

$$\Pr_{P \sim F_i}[P(x) \neq P(y)] \leq \beta \cdot \frac{d_X(x,y)}{2^{i+2}\beta} \leq \frac{1}{2}.$$

Since we sample $k = O(\log n)$ partitions from F_i, then with high probability there exists a partition P, sampled from F_i where x, y are clustered together. Conditioned on this event, we get that in the tree created by P, denoted T_P:

$$d_{T_P}(x,y) \leq 2\text{diam}(P(x)) \leq O(\beta) d_X(x,y).$$

By taking a union bound over all pairs of points we see that this is an $O(\beta)$-tree cover for X with high probability. □

Corollary 6.5. *Let $d = \Theta(\log n)$ and let algorithm A be Algorithm 3, then it takes $\tilde{O}(n^{1+1/c^2+o(1)})$ time where $c = 1.4$ to construct an $O(\sqrt{d})$-tree cover of X of size $O(\log n \log(diam(X)))$*

Proof. It takes $\tilde{O}(n)$ time to connect the vertices for each tree since only one linear scan is required. Thus, the overall time complexity is dominated by the construction of the partitions and is $\tilde{O}(n^{1+1/c^2+o(1)})$ time where $c = 1.4$ by using our fast construction (Theorem 4.5). □

6.3 High-Dimensional Euclidean Spanners

The notion of tree covers is closely related to the well studied notion of spanners. In the context of metric spaces, a spanner with stretch $\alpha \geq 1$ for the metric (X, d_X), is a graph G with $X = V_G$, such that

$$\forall x, y \in X, \quad d_X(x,y) \leq d_G(x,y) \leq \alpha \cdot d_X(x,y).$$

It is often desired that the spanner would be a sparse graph.

Spanners are natural and useful representations of a metric, and as such they have been a subject of extensive research. In particular, it is known that every n-point metric admits a $(2k-1)$-spanner of size $O(n^{1+1/k})$ for any integer $k > 0$ [PS89], and assuming the girth conjecture of Erdos [Erd64], this bound is tight. For simpler metrics that are induced by a set of n points in a low-dimensional Euclidean space say of dimension d, the stretch bound can be improved considerably: there exists a $(1+\epsilon)$-spanner with only $O(n(1/\epsilon)^{O(d)})$ edges [Sal91, Vai91]. For a constant dimension d and fixed ϵ, this gives a bound that is linear in n.

However, spanners for high-dimensional Euclidean spaces have received relatively little attention due to the curse of dimensionality. As the dimension d increases, standard techniques for constructing spanners in low-dimensional spaces often fail to scale effectively, making the construction of sparse spanners with low stretch a challenging task. This complexity has historically limited the study of high-dimensional Euclidean spanners compared to their low-dimensional counterparts.

Har-Peled, Indyk and Sidiropoulos [HPIS13] introduced an algorithm for constructing spanners of high-dimensional Euclidean graphs. The running time of their algorithm is not explicitly stated.

Theorem 6.6. *([HPIS13]) Let X be a set of n points in Euclidean space. Then, there exists a spanner for $(X, \|.\|_2)$ with stretch $O(\sqrt{\log n})$ and with $O(n \log n \log(diam(X)))$ edges.*

The algorithm requires $O(\log n \log(\text{diam}(X)))$ partitions of X. For each partition, it iterates through every cluster, selects a vertex, and connects all other vertices in the cluster to the selected vertex using Euclidean distance edges. The spanner is the union of all edges accumulated across the partitions. It takes $\tilde{O}(n)$ time to connect the vertices for each partition since only one scan is required. Given the partitions, it takes $\tilde{O}(n)$ time to construct the spanner. Thus, the overall time complexity is dominated by the construction of the partitions.

The algorithm has two variations. The first variation (Section 2 in [HPIS13]) uses the $(O(\sqrt{d}), \Delta)$-separating decomposition from [CCG+98], while the second variation (Section 3) employs the Locality Sensitive Hashing (LSH) construction introduced by [AI06] to create a partition. In both variations, the algorithm requires $O(\log n \log(\text{diam}(X)))$ random partitions to achieve the desired results. In the LSH-based variation, taking $c = \sqrt{\log n}$ to construct an $O(\sqrt{\log n})$-spanner, it takes $\tilde{O}(n)$ time to process each LSH function [AI06], resulting in overall running time $\tilde{O}(n)$. This makes the LSH-based variation efficient and fast.

Using our fast construction, the variation based on the separating decomposition becomes faster, with overall time complexity $\tilde{O}(n^{1+1/c^2+o(1)})$, where $c = 1.4$. While this is not faster than the LSH-based variation, it significantly improves the efficiency of the method based on the separating decomposition.

References

[AC09] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, 2006.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 503–513. IEEE, 1990.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996.

[BFN19] Yair Bartal, Nova Fandina, and Ofer Neiman. Covering metric spaces by few trees. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[CCG+98] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pages 379–388. IEEE, 1998.

[CCL+24] Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenkovic, Shay Solomon, and Cuong Than. Optimal euclidean tree covers. *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, 2024.

[CGMZ05] Hubert TH Chan, Anupam Gupta, Bruce M Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In *Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[Cha97] Timothy M Chan. Approximate nearest neighbor queries revisited. In *Proceedings of the thirteenth annual symposium on computational geometry*, pages 352–358, 1997.

[Erd64] Paul Erdös. Extremal problems in graph theory. *Publ. House Cszechoslovak Acad. Sci., Prague*, pages 29–36, 1964.

[FN20] Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. *Discrete Computational Geometry*, 64(3):796–822, 2020.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

[GHR06] Anupam Gupta, Mohammad T Hajiaghayi, and Harald Räcke. Oblivious network design. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 970–979, 2006.

[GKR05] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM Journal on Computing*, 34(2):453–474, 2005.

[HPIS13] Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 804–809. SIAM, 2013.

[Hu74] Te C Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[JKK05] Norman L Johnson, Adrienne W Kemp, and Samuel Kotz. *Univariate discrete distributions*, volume 444. John Wiley & Sons, 2005.

[JL84] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *Conference on Modern Analysis and Probability*, volume 26, pages 189–206. American Mathematical Society, 1984.

[KN14] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[KPR93] Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 682–690, 1993.

[LM00] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of statistics*, pages 1302–1338, 2000.

[LS93] Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

[MPX13] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 196–203, 2013.

[MS09] Manor Mendel and Chaya Schwob. Fast ckr partitions of sparse graphs. *Chicago Journal OF Theoretical Computer Science*, 2:1–18, 2009.

[PS89] David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

[Rao99] Satish Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 300–306, 1999.

[Sal91] Jeffrey S Salowe. Construction of multidimensional spanner graphs, with applications to minimum spanning trees. In *Proceedings of the seventh annual symposium on Computational geometry*, pages 256–261, 1991.

[Vai91] Pravin M Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.

[WLB+00] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, Ramamurthy Ravi, and Chuan Yi Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3):761–778, 2000.