Faster Relational Algorithms Using Geometric Data Structures*

ARYAN ESMAILPOUR, Department of Computer Science, University of Illinois Chicago, USA
STAVROS SINTOS, Department of Computer Science, University of Illinois Chicago, USA

Optimization tasks over relational data, such as clustering, often suffer from the prohibitive cost of join operations, which are necessary to access the full dataset. While geometric data structures like BBD trees yield fast approximation algorithms in the standard computational setting, their application to relational data remains unclear due to the size of the join output. In this paper, we introduce a framework that leverages geometric insights to design faster algorithms when the data is stored as the results of a join query in a relational database. Our core contribution is the development of the RBBD tree, a randomized variant of the BBD tree tailored for relational settings. Instead of completely constructing the RBBD tree, by leveraging efficient sampling and counting techniques over relational joins, we enable on-the-fly efficient expansion of the RBBD tree, maintaining only the necessary parts. This allows us to simulate geometric query procedures without materializing the join result. As an application, we present algorithms that improve the state-of-the-art for relational k-center/means/median clustering by a factor of k in running time while maintaining the same approximation guarantees. Our method is general and can be applied to various optimization problems in the relational setting.

CCS Concepts: • **Theory of computation → Data structures and algorithms for data management**.

Additional Key Words and Phrases: relational data, clustering, k-center, k-median, k-means, BBD tree

ACM Reference Format:
Aryan Esmailpour and Stavros Sintos. 2026. Faster Relational Algorithms Using Geometric Data Structures. *Proc. ACM Manag. Data* 4, 2 (PODS), Article 102 (May 2026), 27 pages. https://doi.org/10.1145/3801898

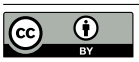
1 Introduction

Optimization problems, such as clustering, are traditionally studied in computer science under the assumption of full access to the input data. However, in real-world applications, data is typically stored in *relational databases*, where information is distributed across multiple interrelated tables. Each table contains a set of tuples, and tuples from different tables may join if they have the same value on the shared attributes. This relational structure introduces a fundamental challenge: full access to the data is only possible after performing a join operation across the relevant tables. Such joins are computationally expensive, as the output size of a join can be polynomially larger than the size of the input tables [47, 48]. Nonetheless, relational data is ubiquitous in practice. According to [1, 3, 51], the majority of modern database systems and data science tasks involve relational data.

Machine learning applications over relational data typically involve two stages: (1) data preparation, where multiple tables are joined to construct the feature dataset, and (2) modeling, where machine learning algorithms (for example, clustering algorithms in unsupervised learning) are

*This work was partially supported by NSF grant IIS-2348919.

Authors' Contact Information: Aryan Esmailpour, Department of Computer Science, University of Illinois Chicago, Chicago, USA, aesmai2@uic.edu; Stavros Sintos, Department of Computer Science, University of Illinois Chicago, Chicago, USA, stavros@uic.edu.



ACM 2836-6573/2026/5-ART102
https://doi.org/10.1145/3801898

applied to the resulting data. However, this two-step pipeline can be prohibitively expensive in practice, largely due to the high cost of evaluating large joins.

Consider a machine-learning workflow in which feature engineering over relational data is followed by clustering, based on the Yelp review dataset [2], which has been used for k-means clustering in prior work [22, 26]. The database consists of five relations: (1) Review(user_id, business_id, stars, date) describing reviews, (2) User(user_id, review_count, fans, yelping_since) containing reviewer attributes, (3) Business(business_id, city, state, avg_rating) with business metadata, (4) Category(business_id, category) listing business categories (e.g., "Restaurant", "Coffee Shop"), and (5) Attributes(business_id, attribute) recording additional characteristics (e.g., "OpenLate", "HasWiFi"). A typical feature-engineering query joins these relations: Q = Review $\bowtie$ User $\bowtie$ Business $\bowtie$ Category $\bowtie$ Attributes. Each tuple in the join result corresponds to a review enriched with numerical attributes (e.g., review_count, avg_rating) and categorical attributes (e.g., category, attribute). Clustering is then performed directly on the joined tuples, grouping review events by joint user attributes, business characteristics, and semantic labels. Because businesses often have multiple category and attribute labels, the number of join results is much larger than any base relation. As reported in [26], the Yelp tables contain roughly 8 million tuples in total, while the fully materialized join contains about 22 million tuples. This blow-up highlights the cost of materializing joins before learning and motivates algorithms that operate directly on the base tables without explicitly constructing the join results.

Recent work has aimed to circumvent materializing the full join output by designing *relational algorithms* for optimization tasks such as clustering [22, 26, 46]. However, these approaches often suffer from large constant approximation ratios or expensive runtimes. Moreover, each known method typically relies on a different specialized technique tailored to a specific problem.

In this paper, we introduce a general framework that enables faster algorithms for a broad class of optimization problems on relational data. Given a join query and a database instance, our approach can be used to efficiently solve problems such as clustering, diversity maximization, and fairness-aware selection. Many of these problems have well-understood and near-linear time approximation algorithms in the standard computational setting, where the input is a single flat table. However, naively applying these techniques in the relational setting is inefficient due to the size of the join output. Our key idea is to simulate the behavior of geometric data structures—particularly those based on hierarchical tree structures such as the *BBD tree* [13], *without* materializing the entire join result. We present our approach by implementing a relational version of the BBD tree, since it best serves the clustering problems we aim to solve. However, the same techniques can be adopted to derive a relational version of other commonly used geometric data structures such as quad trees [36], kd-trees [27], or partition trees [21]. We introduce a new randomized data structure, the *RBBD tree*, which maintains the desirable properties of the BBD tree but is tailored for relational data. Crucially, we show that given only one node of an RBBD tree (defined over the join results of an acyclic join query), its children can be computed in roughly $O(N)$ time, where N is the size of the input database. This enables efficient simulation of geometric queries over the join results, including range counting, range reporting, and range sampling queries, as these typically require accessing only a logarithmic number of tree nodes.

Our method is general: any geometric algorithm that interacts with the input data through σ queries to a BBD tree can be transformed into a randomized relational algorithm (over an acyclic join query) with total runtime $O(\sigma \cdot N)$ (ignoring logarithmic factors). As many geometric algorithms fall into this category, our framework yields efficient relational counterparts for a variety of tasks. As an application, we show that the running time of the state-of-the-art algorithms for all relational k-center, k-median, and k-means clustering can be improved by a factor of k.

While in many applications k is considered small, k-clustering for large k has become increasingly relevant in modern applications, such as product quantization for nearest neighbor search in vector databases [42]. This has motivated extensive algorithmic studies of large-k-clustering across various computational models [16, 25, 30, 33, 35]. Furthermore, in the relational setting k can be even larger than N, because the number of join results might be orders of magnitude larger than N. In the literature on relational clustering [8, 26, 31, 46], existing algorithms typically assume that k may be large, and successive work has focused on improving the dependency on k. We present the first relational clustering algorithms with linear dependency on both k and N for acyclic queries.

1.1 Notation and problem definition

Join Queries. We are given a database schema $\mathbf{R}$ over a set of d attributes $\mathbf{A} = \{A_1, \ldots, A_d\}$. The database schema $\mathbf{R}$ contains m relations $R_1, \ldots, R_m$. Let $\mathbf{A}_j \subseteq \mathbf{A}$ be the set of attributes associated with relation $R_j \in \mathbf{R}$. For an attribute $A \in \mathbf{A}$, let $\mathsf{dom}(A)$ be the domain of attribute A. Let $\mathsf{dom}(X) = \prod_{A \in X} \mathsf{dom}(A)$ be be the domain of a set of attributes $X \subseteq \mathbf{A}$. Let $\mathbf{D}$ be a database instance over the database schema $\mathbf{R}$. For simplicity, we assume that each relation $R_j \in \mathbf{R}$ contains N tuples in $\mathbf{D}$. Throughout the paper, we consider data complexity i.e., m and d are constants, while N is a large integer. We use R_j to denote both the relation and the set of tuples from $\mathbf{D}$ stored in the relation. For a subset of attributes $B \subseteq \mathbf{A}$ and a set $Y \subset \mathbb{R}^d$, let $\pi_B(Y)$ be the set containing the projection of the tuples in Y onto the attributes B. Notice that two different tuples in Y might have the same projection on B, however, $\pi_B(Y)$ is defined as a set, so the projected tuple is stored once.

Following the related work on relational clustering [22, 26, 46], we are given a join query $\boldsymbol{q} := R_1 \bowtie \ldots \bowtie R_m$. The set of results of a join query $\boldsymbol{q}$ over the database instance $\mathbf{D}$ is defined as $\boldsymbol{q}(\mathbf{D}) = \{t \in \mathsf{dom}(\mathbf{A}) \mid \forall j \in [1, m] : \pi_{\mathbf{A}_j}(t) \in R_j\}$. A join query $\boldsymbol{q}$ is acyclic if there exists a tree, called a join tree, such that the nodes of the tree are the relations in $\mathbf{R}$ and for every attribute $A \in \mathbf{A}$, the set of nodes/relations that contain A form a connected component. All our algorithms are presented assuming that $\boldsymbol{q}$ is an acyclic join query, however, in the end we extend the results to any general join query. Let $\rho^\star(\boldsymbol{q})$ be the fractional edge cover of query $\boldsymbol{q}$, which is a parameter that bounds the number of join results $\boldsymbol{q}(\mathbf{D})$ over any database instance: for every database instance $\mathbf{D}'$ with $O(N)$ tuples, it holds that $|\boldsymbol{q}(\mathbf{D}')| = O(N^{\rho^\star(\boldsymbol{q})}) = O(N^m)$ as shown in [15].

We use the notation $\mathsf{fhw}(\boldsymbol{q})$ to denote the *fractional hypertree width* [38] of the query $\boldsymbol{q}$. The fractional hypertree width roughly measures how close $\boldsymbol{q}$ is to being acyclic. For every acyclic join query $\boldsymbol{q}$, we have $\mathsf{fhw}(\boldsymbol{q}) = 1$. Given a cyclic join query $\boldsymbol{q}$, we convert it to an equivalent acyclic query such that each relation is the result of a (possibly cyclic) join query with fractional edge cover at most $\mathsf{fhw}(\boldsymbol{q})$. Hence, a cyclic join query over a database instance with $O(N)$ tuples per relation can be converted, in $O(N^{\mathsf{fhw}(\boldsymbol{q})})$ time, to an equivalent acyclic join query over a database instance with $O(N^{\mathsf{fhw}(\boldsymbol{q})})$ tuples per relation [15]. A more formal definition of fhw is given in the full version of the paper [32]. If $\boldsymbol{q}$ is clear from the context, we write fhw instead of $\mathsf{fhw}(\boldsymbol{q})$.

For tuples $p, q \in \mathsf{dom}(\mathbf{A})$, let $\phi(p, q) = ||p - q||_2 = \left(\sum_{j=1}^{d} (\pi_{A_j}(p) - \pi_{A_j}(q))^2\right)^{1/2}$ denote their Euclidean distance. If $A_j \in \mathbf{A}$ is a categorical attribute, we apply the *one-hot encoding* and define $(\pi_{A_j}(p) - \pi_{A_j}(q))^2 = 1$ if $\pi_{A_j}(p) \neq \pi_{A_j}(q)$, and 0 otherwise. We focus on Euclidean distance for clarity and consistency with prior work, but our algorithms extend to any ℓ_p metric.

Throughout the paper, we use the term *standard computational setting* when we consider the input data to be stored in a single flat table. In contrast, we use the term *relational setting* when the data is the result of a join query as described above.

Clustering. In this paper, we focus on the three classical clustering objectives: k-center, k-median, and k-means clustering. These formulations capture different ways of measuring the quality of clustering and are among the most widely used models in both theory and practice [14, 37, 40, 44, 49].

We start with some useful general definitions. Let P be a set of tuples in $\mathsf{dom}(\mathbf{A})$ and let $C \subset \mathsf{dom}(\mathbf{A})$ be a set of k tuples. For a tuple $p \in \mathsf{dom}(\mathbf{A})$, let $\phi(p, C) = \min_{c \in C} \phi(p, c)$. We define

$$\mathbf{u}_C(P) = \max_{p \in P} \phi(p, C), \quad \mathbf{v}_C(P) = \sum_{p \in P} \phi(p, C), \quad \text{and} \quad \mu_C(P) = \sum_{p \in P} \phi^2(p, C).$$

k-center clustering: Given a set of tuples P in $\mathsf{dom}(\mathbf{A})$ and a parameter k, the goal is to find a set $C \subseteq P$ with $|C| = k$ such that $\mathbf{u}_C(P)$ is minimized. Let $\mathsf{OPT}(P) = \arg\min_{S \subseteq P, |S|=k} \mathbf{u}_S(P)$ be a set of k centers with the minimum $\mathbf{u}_{\mathsf{OPT}(P)}(P)$. Let $\mathsf{kCenterAlg}_\gamma$ be a (known) γ-approximation algorithm for the k-center clustering problem in the standard computational setting that runs in $T^{\mathsf{cen}}_\gamma(|P|)$ time, where γ is a constant.

k-median clustering: Given a set of tuples P in $\mathsf{dom}(\mathbf{A})$ and a parameter k, the goal is to find a set $C \subseteq P$ with $|C| = k$ such that $\mathbf{v}_C(P)$ is minimized. Let $\mathsf{OPT}(P) = \arg\min_{S \subseteq P, |S|=k} \mathbf{v}_S(P)$ be a set of k centers with the minimum $\mathbf{v}_{\mathsf{OPT}(P)}(P)$. Let $\mathsf{kMedianAlg}_\gamma$ be a (known) γ-approximation algorithm for the k-median problem in the standard computational setting that runs in $T^{\mathsf{med}}_\gamma(|P|)$ time, where γ is a constant.

k-means clustering: Given a set of tuples P in $\mathsf{dom}(\mathbf{A})$ and a parameter k, the goal is to find a set $C \subseteq P$ with $|C| = k$ such that $\mu_C(P)$ is minimized. Let $\mathsf{OPT}(P) = \arg\min_{S \subseteq P, |S|=k} \mu_S(P)$ be a set of k centers with the minimum $\mu_{\mathsf{OPT}(P)}(P)$. Let $\mathsf{kMeansAlg}_\gamma$ be a (known) γ-approximation algorithm for the k-means problem in the standard computational setting that runs in $T^{\mathsf{mean}}_\gamma(|P|)$ time, where γ is a constant.

We use the same notation $\mathsf{OPT}(\cdot)$ for the optimum solution for k-center/median/means clustering. It is always clear from the context whether we are referring to k-center, k-median, or k-means clustering. Sometimes we call the value of $\mathbf{u}_C(P)$ (or $\mathbf{v}_C(P), \mu_C(P)$) as the clustering cost of C in P.

Relative approximation. We call a set $C \subset \mathsf{dom}(\mathbf{A})$ an α-approximation for the k-center (resp. k-median, k-means) clustering over a tuple set $P \subset \mathsf{dom}(\mathbf{A})$, if $\mathbf{u}_C(P) \leq \alpha \cdot \mathbf{u}_{\mathsf{OPT}(P)}(P)$ (resp. $\mathbf{v}_C(P) \leq \alpha \cdot \mathbf{v}_{\mathsf{OPT}(P)}(P)$, $\mu_C(P) \leq \alpha \cdot \mu_{\mathsf{OPT}(P)}(P)$). An α-approximation algorithm on P always returns a set of centers which is α-approximation over P.

Coreset. A set $C \subseteq \mathsf{dom}(\mathbf{A})$, with $|C| \ll |P|$, is a coreset for the k-center/median/means clustering over P, if every α-approximation algorithm executed on C returns a set of centers which is a $(1+\varepsilon)\alpha$-approximation for P. Throughout the paper, we assume that $\varepsilon \in (0, 1)$.

In this paper, we study k-center, k-median and k-means clustering on the results of a join query.

Definition 1.1. [Relational k-center/median/means clustering] Given a database instance $\mathbf{D}$, a join query $\boldsymbol{q}$, and a positive integer parameter k, the goal is to compute a set $\mathcal{S} \subseteq \boldsymbol{q}(\mathbf{D})$ of size $|\mathcal{S}| \leq k$ such that $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$ (resp. $\mathbf{v}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$, $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$ is minimized.

Remark. While we focus on k-clustering over full join results, all our algorithms extend straightforwardly to k-clustering over the results of project–join queries under *bag semantics*, where duplicates are preserved (for more details see the full version [32]). This setting is particularly useful in applications where the goal is to cluster all tuples in the join result only with respect to a subset of attributes. As discussed in [32], the ability to perform clustering on project–join queries under bag semantics is also a key ingredient for handling clustering over joins that involve both numerical and categorical attributes.

1.2 Related work

For the relational k-center problem assuming only numerical attributes, Agarwal et al. [8] propose a $(2+\varepsilon)$-approximation in $\tilde{O}(\min\{k^2 N^{\mathsf{fhw}}, kN^{\mathsf{fhw}} + k^{d/2}\})$ time.

For the relational k-means problem, Khamis et al. [4], gave an efficient implementation of the Lloyd's k-means heuristic. Relative approximation algorithms are also known. Curtin et al. [26] construct a weighted set of tuples (*grid-coreset*) such that an approximation algorithm for the weighted

Problem	Method	Approximation	Running time
k-center	[8]	$(1+\varepsilon)\gamma$	$\tilde{O}(k^2 N^{\text{fhw}} + T_\gamma^{\text{cen}}(k^2))$
	[8]	$(1+\varepsilon)\gamma$	$\tilde{O}(kN^{\text{fhw}} + k^{\lceil d/2 \rceil + 1} + T_\gamma^{\text{cen}}(k))$
	Theorem 5.2	$(1+\varepsilon)\gamma$	$\tilde{O}(kN^{\text{fhw}} + T_\gamma^{\text{cen}}(k))$
k-median	[31]	$(2+\varepsilon)\gamma$	$\tilde{O}(k^2 N^{\text{fhw}} + k^4 + T_\gamma^{\text{med}}(k^2))$
	Theorem 6.6	$(2+\varepsilon)\gamma$	$\tilde{O}(kN^{\text{fhw}} + T_\gamma^{\text{med}}(k))$
k-means	[26]	$\gamma^2 + 4\gamma\sqrt{\gamma} + 4\gamma$	$\tilde{O}(k^m N^{\text{fhw}} + T_\gamma^{\text{mean}}(k^m))$
	[46]	$320 + 644(1+\varepsilon)\gamma$	$\tilde{O}(k^4 N^{\text{fhw}} + T_\gamma^{\text{mean}}(k))$
	[31]	$(4+\varepsilon)\gamma$	$\tilde{O}(k^2 N^{\text{fhw}} + k^4 + T_\gamma^{\text{mean}}(k^2))$
	Theorem 6.6	$(4+\varepsilon)\gamma$	$\tilde{O}(kN^{\text{fhw}} + T_\gamma^{\text{mean}}(k))$

Table 1. Comparison of our clustering algorithms with state-of-the-art relative approximation methods for datasets where all attributes are numerical. The running time is shown in data complexity. We assume that $\varepsilon \in (0,1)$ is an arbitrary constant. The notation $\tilde{O}$ is used to hide $\log^{O(1)} N$ factors from the running time. $T_\gamma^{\text{cen}}(y)$ (resp. $T_\gamma^{\text{med}}(y)$, $T_\gamma^{\text{mean}}(y)$) is the running time of a known γ-approximation algorithm over y points for the k-center (resp. k-median, k-means) clustering in the standard computational setting. fhw is the fractional hypertree width of $\boldsymbol{q}$. Recall that d is the number of attributes in query $\boldsymbol{q}$.

k-means clustering in the grid-coreset returns an approximation solution to the relational k-means clustering. If all attributes are numerical then their algorithm runs in $\tilde{O}(k^m N^{\text{fhw}} + T_\gamma^{\text{mean}}(k^m))$ time and has a $(\gamma^2 + 4\gamma\sqrt{\gamma} + 4\gamma)$-approximation factor. In the general case, having both numerical and categorical attributes, the running time is $\tilde{O}(k^d N^{\text{fhw}} + T_\gamma^{\text{mean}}(k^d))$. All subsequent works focus exclusively on the case where all attributes are numerical. Moseley et al. [46] designed a relational implementation of the k-means++ algorithm [9, 11]. For a constant $\varepsilon \in (0,1)$, their algorithm runs in $\tilde{O}(k^4 N^{\text{fhw}} + k^2 N^{\text{fhw}} + T_\gamma^{\text{mean}}(k))$ expected time and has a $(320 + 644(1+\varepsilon)\gamma)$-approximation factor. Finally, Esmailpour and Sintos [31] used the hierarchical aggregation method [22] to derive an $O(1)$-approximation randomized algorithm in $\tilde{O}(k^2 N^{\text{fhw}})$ time. Then, they construct a geometric coreset in $\tilde{O}(k^2 N^{\text{fhw}} + k^4)$ time, leading to a $(4+\varepsilon)\gamma$-approximation algorithm with high probability in $\tilde{O}(k^2 N^{\text{fhw}} + k^4 + T_\gamma^{\text{mean}}(k^2))$ time. The algorithm in [31] can be extended for the relational k-median clustering getting a $(2+\varepsilon)\gamma$-approximation algorithm in $\tilde{O}(k^2 N^{\text{fhw}} + k^4 + T_\gamma^{\text{med}}(k^2))$ time. Additive approximation algorithms are also known for relational clustering problems [22]. Recently, Surianarayanan et al. [55] studied the relational k-center clustering problem with outliers.

Finally, there is a lot of recent work on relational algorithms for learning problems such as, linear regression and factorization [6, 43, 50, 54], SVMs [5, 7, 58], Independent Gaussian Mixture models [23, 24]. Database research has explored various combinatorial problems defined over relational data, such as ranked enumeration [28, 29, 56], quantiles [57], direct access [20], diversity [8, 10, 45], and top-k [56]. A survey is given by Schleich et al. [53].

1.3 Our results

Unless otherwise stated, we assume that $\texttt{dom}(A) = \mathbb{R}$ for each $A \in \mathbf{A}$ and the query $\boldsymbol{q}$ is acyclic. In the full version [32], we show how our algorithms extend when the attribute set $\mathbf{A}$ contains both numerical and categorical domains, while in Section 7 we extend our results to cyclic queries.

First, in Section 3 we give the high-level ideas of our new method. We highlight the differences of our approach with the previously known algorithms, and we propose a new geometric insight on how to improve the running time.

• In Section 4 we present a variation of the BBD tree called Randomized BBD tree (or simply RBBD tree) that has similar properties to the BBD tree in the standard computational setting. However, we show that given a node of the RBBD tree, its children can be constructed in almost

linear time with respect to the size of the input database in the relational setting. This is the key idea to design faster algorithms for various optimization problems in the relational setting.

• In Section 5 we show efficient algorithms for the relational k-center clustering. First, we propose an $O(1)$-approximation algorithm that runs in $\tilde{O}(kN)$ time, with high probability. The result of our new algorithm is given as input to the coreset construction from [8] to get a $(1+\varepsilon)\gamma$-approximation algorithm in $\tilde{O}(kN + T_\gamma^{\mathsf{cen}}(k))$ time.

• In Section 6 we design efficient algorithms for the relational k-median/means clustering. We get our intuition from [40] and we propose a geometric approach using our RBBD tree in the relational setting to design an $O(1)$-approximation algorithm for the relational k-median/means clustering that returns $O(k \log^3 N)$ centers. The returned set is given as input to the coreset construction method (in the relational setting) in [31] to derive the final result. Interestingly, our new observations in this work can be used to accelerate the coreset construction of [31]. In the end, we get a $(2+\varepsilon)\gamma$-approximation (resp. $(4+\varepsilon)\gamma$-approximation) for the relational k-median (resp. k-means) clustering in $\tilde{O}(kN + T_\gamma^{\mathsf{med}}(k))$ (resp. $\tilde{O}(kN + T_\gamma^{\mathsf{mean}}(k))$) time, with high probability.

• Due to the generality of our approach, in Section 7 and in the full version [32], we show how our new method can be used to get faster algorithms for other optimization problems, such as diversity and fairness problems, in the relational setting.

The running times and approximation factors of our new relational clustering algorithms, along with a comparison to the state-of-the-art, are shown in Table 1.

2 Preliminaries

We revisit various known tools, algorithms, and data structures that we use in the next sections. For a point $p \in \mathbb{R}^d$ and $r \in \mathbb{R}$, let $\mathcal{B}(p,r) = \{x \in \mathbb{R}^d \mid \phi(p,x) \leq r\}$ be the ball of center p and radius r.

Box. A box ρ is an axis-parallel d-dimensional hyper-rectangle that is defined as the product of d intervals over the attributes, i.e., $\rho = I_1 \times \ldots \times I_d$, where $I_i = [a_i, b_i]$ for $a_i, b_i \in \mathbb{R}$.

ε-sample. Let $(P, \mathcal{R})$ be a set system where P is a set of n points in $\mathbb{R}^d$ and $\mathcal{R}$ is the set of all (infinitely many) possible boxes in $\mathbb{R}^d$. For a parameter $\varepsilon \in (0,1)$, a set $\mathcal{N} \subseteq P$ is an ε-sample if for every $\rho \in \mathcal{R}$, $|\frac{|P \cap \rho|}{|P|} - \frac{|\mathcal{N} \cap \rho|}{|\mathcal{N}|}| \leq \varepsilon$. It is known [18, 39] that a set of $O(\varepsilon^{-2} \log \psi^{-1})$ uniform random samples from P is an ε-sample of $(P, \mathcal{R})$ with probability at least $1 - \psi$.

Oracles for aggregation queries. We show known oracles that can be used to access some tuples from the join results or compute some statistics on the join results without explicitly evaluating the join query. In the next sections, we aim to count or sample from the join results in $\boldsymbol{q}(\mathbf{D})$ that lie inside a box. More formally, let ρ be a box in $\mathbb{R}^d$. The goal is i) count the number of tuples $|\boldsymbol{q}(\mathbf{D}) \cap \rho|$, ii) sample uniformly at random from $\boldsymbol{q}(\mathbf{D}) \cap \rho$, iii) report all tuples in $\boldsymbol{q}(\mathbf{D}) \cap \rho$, and iv) pick any arbitrary tuple from $\boldsymbol{q}(\mathbf{D}) \cap \rho$. The next lemma follows from [31].

Lemma 2.1 ([31]). *Let $\mathbf{D}$ be a database instance of N tuples (with numerical attributes) and let $\boldsymbol{q}$ be an acyclic query over $\mathbf{D}$. Let ρ be a box in $\mathbb{R}^d$. There exists an algorithm*

(1) $\mathsf{CountRect}(\boldsymbol{q}, \mathbf{D}, \rho)$ *to count $|\boldsymbol{q}(\mathbf{D}) \cap \rho|$ in $O(N)$ time,*
(2) $\mathsf{SampleRect}(\boldsymbol{q}, \mathbf{D}, \rho, z)$ *to sample z points from $\boldsymbol{q}(\mathbf{D}) \cap \rho$ in $O(N + z \log N)$ time,*
(3) $\mathsf{ReportRect}(\boldsymbol{q}, \mathbf{D}, \rho)$ *to report all points in $\boldsymbol{q}(\mathbf{D}) \cap \rho$ in $O(N + |\rho \cap \boldsymbol{q}(\mathbf{D})|)$ time,*
(4) $\mathsf{ReprRect}(\boldsymbol{q}, \mathbf{D}, \rho)$ *to find a representative tuple from $\boldsymbol{q}(\mathbf{D}) \cap \rho$ in $O(N)$ time.*

The running times of the oracles hold assuming perfect hashing.[1]

[1]We assume perfect hashing for simplicity. A standard randomized hash table yields the same asymptotic bounds with high probability; perfect hashing assumption is used only to avoid carrying additional (routine) randomness through the analysis.

BBD tree. The main geometric data structure we employ is the *BBD tree* [12, 13], a variant of the quadtree [36]. A BBD tree $\mathcal{T}$ on a set P of n points in $\mathbb{R}^d$ is a binary tree of height $O(\log n)$ with n leaves, where each point in P is stored in exactly one leaf node. Let $\square$ denote the axis-aligned box with the smallest volume containing P. Each node u of $\mathcal{T}$ is associated with a region $\square_u \subseteq \mathbb{R}^d$, which is either a box or the difference of two boxes (a box with a hole). This region contains precisely the points from P that are stored in the leaves of the subtree rooted at u. If `root` is the root node of $\mathcal{T}$, then $\square_{\texttt{root}} = \square$. Let $P_u = P \cap \square_u$ denote the points contained in the subtree rooted at u. If $|P_u| = 1$, then u is a leaf. Otherwise, u has two children, say w and z, such that $\square_w$ and $\square_z$ form a partition of $\square_u$. The regions associated with the nodes of $\mathcal{T}$ thus induce a hierarchical partition of $\mathbb{R}^d$. For illustration, consider the set of 23 points shown in Figure 1. A partially constructed (and, for simplicity, slightly simplified) BBD tree of this set is shown in Figure 2. The blue segments in Figure 1 depict the hierarchical partition induced by the constructed BBD tree. For the node v in Figure 2, the region $\square_v$ is the box defined by opposite corners C and J, while for node w, the region $\square_w$ is the box defined by opposite corners C and K, excluding the inner box defined by opposite corners L and M. For every node u of the BBD tree in Figure 2, we also show the number of points in P_u. For example, $|P_w| = 4$ because the box defined by $[C, K]$ contains 8 points, but the hole box $[L, M]$ contains 4 points. A BBD tree has $O(n)$ space and can be constructed in $O(n \log n)$ time.

A BBD tree requires $O(n)$ space and can be constructed in $O(n \log n)$ time. Given a parameter $\varepsilon \in (0, 1)$ and a ball $\mathcal{B}(x, r)$ in $\mathbb{R}^d$, the BBD tree supports a query procedure, denoted $\mathcal{T}(x, r)$, that returns a set of *canonical nodes* $\mathcal{U}(x, r) = u_1, \ldots, u_\kappa$, with $\kappa = O(\log n + \varepsilon^{-d+1})$, in $O(\log n + \varepsilon^{-d+1})$ time. These nodes satisfy $\square_{u_i} \cap \square_{u_j} = \emptyset$ for all distinct i, j, and $\mathcal{B}(x, r) \subseteq \bigcup_{1 \le i \le \kappa} \square_{u_i} \subseteq \mathcal{B}(x, (1+\varepsilon)r)$. In other words, the BBD tree returns a set of boxes (with or without holes) such that the union of them completely covers the ball $\mathcal{B}(x, r)$ and might cover some part of $\mathcal{B}(x, (1 + \varepsilon)r) \setminus \mathcal{B}(x, r)$. The query procedure is straightforward once the tree is built. Starting from the root of $\mathcal{T}$ and an initially empty set $\mathcal{U}(x, r)$, the algorithm recursively traverses the tree. When visiting a node u, if $\square_u \subseteq \mathcal{B}(x, (1 + \varepsilon)r)$, the node u is added to $\mathcal{U}(x, r)$. If $\square_u \cap \mathcal{B}(x, r) = \emptyset$, the traversal of this branch terminates. Otherwise, the procedure continues recursively with both children of u. For example, in Figure 1, let $\mathcal{B}(x, r)$ denote the inner red circle and $\mathcal{B}(x, (1 + \varepsilon)r)$ the outer dashed circle. The corresponding canonical nodes $\mathcal{U}(x, r)$ are highlighted as red nodes in Figure 2. We note that the boxes in $\mathcal{U}(x, r)$ cover all points in the inner red circle and, in addition, cover a point from P in the annulus $\mathcal{B}(x, (1 + \varepsilon)r) \setminus \mathcal{B}(x, r)$, because the box with opposite corners N and S lies entirely within the outer dashed circle. BBD trees can be used for different types of queries, such as reporting, counting, or sampling queries.

3 Overview

In this section, we describe the intuition and the high-level ideas of our approach. In the next sections, we formally prove the claims made in this section.

Previous methods. Most of the previously known methods for relational clustering [8, 22, 26, 31, 46] follow the same high-level approach. They first design a constant approximation algorithm with more than k centers. Let Z be that set. Then Z is used to construct a small coreset C for the relational clustering problem. Finally, a known clustering algorithm in the standard computational setting is executed on C to derive the final solution. Interestingly, the coreset construction is relatively fast. In most cases, the running time is dominated by the first part, i.e., constructing any constant approximation solution with potentially more than k points. For example, consider the state–of–the–art algorithms for the relational k-center [8] and relational k-median/means clustering [31]. In these papers, they construct $O(1)$-approximation algorithms with $\tilde{O}(k^2)$ centers in $\tilde{O}(k^2 N)$ time using a *hierarchical aggregation method*. At a high level, they place all relations from $\mathbf{R}$ as leaf nodes in a balanced binary tree. For a leaf node u_1 run a standard clustering algorithm

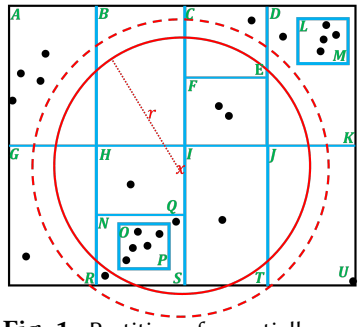


Fig. 1. Partition of a partially constructed BBD tree over the set of black points. Red circle represents the query ball $\mathcal{B}(x, r)$ while the larger dashed circle represents $\mathcal{B}(x, (1+\varepsilon)r)$.

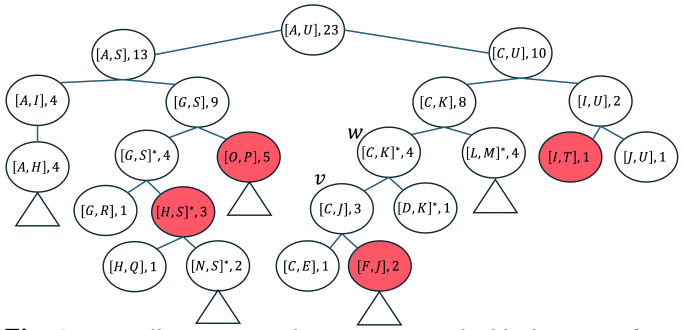


Fig. 2. Partially constructed BBD tree over the black points of Figure 1. A node u of the form $[X, Y], c$ has a region $\square_u$ defined by the opposite corners X, Y, and $|P_u| = |\square_u \cap P| = c$. Nodes with $[X, Y]^*, c$ correspond to nodes whose region is a box with a hole. The triangles below certain nodes denote subtrees omitted for simplicity and clarity. Red nodes indicate the set of canonical nodes $\mathcal{U}(x, r)$ corresponding to the ball $\mathcal{B}(x, r)$ shown in Figure 1.

in the standard computational setting, considering only the tuples of the relation stored in u_1. Let $\mathcal{S}_1$ be the returned set of k centers. Equivalently, they get $\mathcal{S}_2$ from the sibling node u_2. Then, they show that $\mathcal{S}_{1\times 2} = \mathcal{S}_1 \times \mathcal{S}_2$ is an $O(1)$-approximation solution for the clustering problem on the result of joining the tables in leaf nodes u_1 and u_2. Then they construct a coreset based on $\mathcal{S}_{1\times 2}$ in time $\tilde{O}(|\mathcal{S}_{1\times 2}| \cdot N)$ and continue recursively on higher levels of the tree. While this method is useful to design $O(1)$-approximation algorithms, the drawback is that it constructs intermediate sets of size $\Theta(k^2)$ (for example $\mathcal{S}_{1\times 2}$), leading to $\Omega(k^2 \cdot N)$ time algorithms.

Our approach. We give the high-level idea of our approach using the k-center clustering as an example. Let P be a set of n points in $\mathbb{R}^d$ and k be a positive integer parameter in the standard computational setting. In computational geometry, there are various efficient constant approximation algorithms for the k-center clustering. While these algorithms are fast in the standard computational setting, it is not clear how to efficiently extend them to the relational setting. Consider the following simple algorithm for the k-center problem on P. Let $r = \mathbf{u}_{\mathrm{OPT}(P)}(P)$ be the optimum cost, and assume that r is known upfront. In the first iteration, we add in $\mathcal{S}$ any arbitrary point $p_1 \in P$. We remove all points in $\mathcal{B}(p_1, 2r) \cap P$ from P, i.e., all points in P within distance $2r$ from p_1, and we continue with the next iteration. In the i-th iteration we choose an arbitrary point $p_i \in P \setminus \bigcup_{j<i} \mathcal{B}(p_j, 2r)$, we remove all points $\mathcal{B}(p_i, 2r) \cap P$ and we continue for a total number of k iterations. It is easy to verify that this algorithm returns a 2-approximation for the k-center problem on P. Assuming that the optimum r is known, a naive implementation of this algorithm takes $O(nk)$ time. If r is not known, we need to execute a binary search on the distances of P (using a WSPD [19, 41]), leading to $O(nk \log n)$ running time. However, there is a faster implementation of this algorithm in the standard computational setting. We build a BBD tree $\mathcal{T}$ over P. For every node u of $\mathcal{T}$, we store a representative point $u.rep$ from the points stored in the leaf nodes of the subtree rooted at u, i.e., $u.rep \in P_u$ (see Section 4). In the first iteration $p_1 = \texttt{root}.rep$, is the representative point in the root. Then we run the query $\mathcal{T}(p_1, 2r)$ and we get the set of canonical nodes $\mathcal{U}(p_1, 2r)$. We mark every node in $\mathcal{U}(p_1, 2r)$ as 'inactive'. Then we start from the deepest node in $\mathcal{U}(p_1, 2r)$ and we update the representative points of the active nodes to make sure that no representative point lies in a leaf node of a subtree rooted at an inactive node. We continue similarly in the next $k-1$ iterations, visiting only active nodes. This implementation returns a set $\mathcal{S} \subseteq P$ such

that $\mathbf{u}_{\mathcal{S}}(P) \leq 2(1+\varepsilon)\mathbf{u}_{\mathrm{OPT}(P)}(P)$ with running time $O(n \log n + k(\varepsilon^{-d+1} + \log n)) = O(n \log n)$, in standard computational setting.

A straightforward implementation of the BBD tree based algorithm for k-center clustering will be expensive in the relational setting. Just to construct the BBD tree over all join results, we need $\Omega(|\boldsymbol{q}(\mathbf{D})|)$ time. However, we make the following simple but key observation. **We do not really need to construct the entire tree upfront. What if we construct the tree on the fly?** Consider again the standard computational setting. Initially, assume that we have only constructed the root node `root` of the tree $\mathcal{T}$. The goal is to compute $\mathcal{U}(p_i, 2r)$. Let v be a node of the constructed part of $\mathcal{T}$ that the query procedure traverses. If the children of v have not been constructed, we construct the left child v_1 and the right child v_2, and then we continue with the regular query procedure. Given a node v of the BBD tree, if we can construct its two children v_1, v_2 in $O(t)$ time, then the execution time is bounded by $O(k \cdot t \cdot (\log n + \varepsilon^{-d+1}))$. While this is a loose upper bound in the standard computational setting, we use it to bound the query procedure in the relational setting.

Our core idea is the on-the-fly construction of a new variant of the BBD tree in the relational setting. This enables the design of several new relational oracles, which in turn yield the fastest known approximation algorithms for a variety of optimization problems. While the underlying idea is conceptually simple, constructing a BBD tree in the relational setting requires multiple steps, combining techniques from computational geometry (BBD trees), database theory (relational counting and sampling oracles), and probability analysis (ε-samples). A key advantage of our approach is generality. Many classical algorithms from standard settings can be directly translated to relational settings with minimal effort using the proposed data structure. This sharply contrasts with previous work on relational clustering (for example [8, 31, 46]), where algorithms are specifically tailored for particular objectives and often differ significantly from their classical counterparts.

In the next section, we propose a new variation of the BBD tree, called randomized BBD tree (or RBBD tree in short) such that if a node v of an RBBD tree over $\boldsymbol{q}(\mathbf{D})$ is given, then the children v_1, v_2 can be computed in $O(N + \varepsilon^{-2} \log N)$ time in the relational setting.

4 RBBD tree

Before we describe the RBBD tree, we give an overview of how standard BBD trees are constructed as described in [13]. Then, we introduce a randomized variation of the BBD tree called RBBD tree. Although the RBBD tree offers no advantage over the BBD tree in the standard computational setting, we show that it can be beneficial in the relational setting, as the children of any node in the RBBD tree over $\boldsymbol{q}(\mathbf{D})$ can be constructed efficiently.

4.1 More details on the construction of a BBD tree

Let P be a set of n points in $\mathbb{R}^d$. Recall that every node u of the BBD tree is associated with a region $\square_u$, which is either a box or a box with a hole, and recall that $P_u = \square_u \cap P$. In the latter case, let $\square_u^+$ be the outer box and $\square_u^-$ be the inner box (hole). A *midpoint box* is defined as any box that can be obtained by a recursive application of the following rule, starting from the initial bounding box $\square_{\texttt{root}}$, then: Let ρ be a midpoint box and let ξ be the longest side of ρ. Split ρ into two identical boxes by a hyperplane which is orthogonal to the ξ-th coordinate axis, passing through the center of ρ. See Figure 3 for an example of midpoint boxes. The tree is constructed in a way that if $\square_u$ is a box, then $\square_u$ is a midpoint box, while if $\square_u$ is a box with a hole, then both $\square_u^+$ and $\square_u^-$ are midpoint boxes, for any node u of the tree. The children of u are constructed using one of the following two methods, *fair split* and *centroid shrink*. At the root of the tree, they apply a fair split to construct its children. In any other case, if u was generated by fair split (resp. centroid shrink) its children are generated by centroid shrink (resp. fair split).

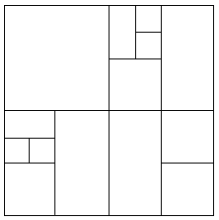


Fig. 3. All the cells are midpoint boxes.

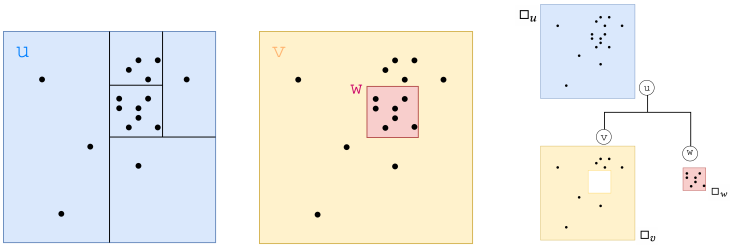


Fig. 4. Illustration of finding the children v, w of a node u by the centroid shrink process. The region $\square_v$ is a box with a hole while $\square_w$ is a box.

Fair split. They compute a hyperplane that passes through the center of $\square_u$ and is orthogonal to the longest side of $\square_u$ (or $\square_u^+$ if $\square_u$ is a box with a hole). Notice that even if $\square_u$ contains a hole, then the hyperplane splitting $\square_u$ into two boxes will not intersect $\square_u^-$ because both $\square_u^-$ and $\square_u^+$ are midpoint boxes. In this case, if w, v are the children nodes of u, then $\square_w$ will be a box, and $\square_v$ will be a box with a hole, having $\square_v^- = \square_u^-$. The fair split is executed in $O(1)$ time.

Centroid shrink. On a high level, the goal is to find a midpoint box ρ^* inside $\square_u$ such that $|\rho^* \cap P| \leq \frac{2}{3}|P_u|$ and $|(\square_u \setminus \rho^*) \cap P| \leq \frac{2}{3}|P_u|$. In order to compute ρ^* their algorithm works in iterations. We describe the construction assuming that $\square_u$ is a box without a hole. In Appendix A.1, we discuss the construction when $\square_u$ is a box with a hole. Initially, let $\rho = \square_u$. They compute the smallest midpoint box that contains all points in $P_u \cap \rho$. Let $\rho' \subseteq \rho$ be this box. They apply $O(d)$ fair splits on ρ' until they find a non-trivial fair split, i.e., a fair split that splits the points in $P_u \cap \rho'$ into two non-empty subsets. Let $\hat{\rho} \subset \rho'$ be one of the two boxes created by the non-trivial fair split such that $|\hat{\rho} \cap P_u| \geq \frac{|\rho' \cap P_u|}{2}$. We say that $\hat{\rho}$ is the majority midpoint box. The key observation is that $|\hat{\rho} \cap P_u| \leq |\rho \cap P_u| - 1$. If $|\hat{\rho} \cap P_u| > \frac{2}{3}|P_u|$, they set $\rho = \hat{\rho}$ and they continue recursively. If $|\hat{\rho} \cap P_u| \leq \frac{2}{3}|P_u|$, then $\rho^* = \hat{\rho}$ and then they construct two children nodes w, v of u such that $\square_w = \rho^*$ is a box (without a hole), and $\square_v$ is a box with a hole with $\square_v^+ = \square_u$ and $\square_v^- = \rho^* = \square_w$, as shown in Figure 4. As shown in [13], the smallest midpoint box $\rho' \subseteq \rho$ that contains all points in $P_u \cap \rho$ can be found in $O(1)$ time. Since in every recursive iteration $|\hat{\rho} \cap P_u| \leq |\rho' \cap P_u| - 1$, the centroid shrink procedure is executed in $O(|P_u|)$ time.

4.2 Construction of the RBBD tree

We define a new variation of the BBD tree called RBBD tree in the standard computational setting over a set of n points $P \subset \mathbb{R}^d$. The construction of the RBBD tree shares most of the steps and properties with the BBD tree. For example, all constructed boxes (both inner and outer) are midpoint boxes. The fair split is identical to BBD tree's fair split. The only difference is the definition and execution of the centroid shrink procedure, given a node u.

We define the *randomized centroid shrink*. Let u be a node. We get a set $\mathcal{N}_u$ of $\Theta(\varepsilon^{-2} \log n)$ uniform random samples from $\square_u \cap P_u$. If $\square_u$ is a box without a hole, then we simply sample in the box $\square_u$. If $\square_u$ is a box with a hole, we sample in the region between $\square_u^+$ and $\square_u^-$. Then the procedure is similar to the centroid shrink from the BBD tree using $\mathcal{N}_u$ instead of P_u. We recursively compute the smallest midpoint box ρ' that contains $\mathcal{N}_u$, and we apply $O(d)$ fair splits until we find a non-trivial fair split. We recursively continue the construction on the box $\hat{\rho}$ created by the non-trivial fair split procedure that contains most of the points in $\mathcal{N}_u$ (majority midpoint box). We repeat this recursive approach until we find a majority midpoint box $\hat{\rho}$ (after a non-trivial fair split) that contains at most $\frac{2}{3}|\mathcal{N}_u|$ samples from $\mathcal{N}_u$. If $\square_u$ is a box with a hole, the procedure is shown in Appendix A.1. In Appendix A.2, we prove the next theorem using the properties of the BBD and RBBD trees.

Theorem 4.1. *Given a set P of n points in $\mathbb{R}^d$ a parameter $\varepsilon \in (0,1)$, and a constant parameter $\zeta > 1$, the RBBD tree has height $O(\log n)$, it can be constructed in $O(n \log n + n \cdot (f(n,\varepsilon) + \varepsilon^{-2} \log n))$ time with space $O(n)$, where $f(n,\varepsilon)$ denotes the running time to sample $\Theta(\varepsilon^{-2} \log n^{\zeta})$ points from a box (or a box with a hole), such that given a query ball $\mathcal{B}(p,r)$ for a point $p \in \mathbb{R}^d$ and a radius r, a set $\mathcal{U}$ of $O(\log n + \varepsilon^{-d+1})$ canonical nodes can be found in $O(\log n + \varepsilon^{-d+1})$ time such that $\mathcal{B}(p,r) \subseteq P \cap \bigcup_{u \in \mathcal{U}} \square_u \subseteq \mathcal{B}(p,(1+\varepsilon)r)$. The height, the space, the construction time of the tree, the size of $\mathcal{U}$, and the query time hold with probability at least $1 - \frac{1}{n^{\zeta-1}}$.*

4.3 RBBD tree in the relational setting

Let $\boldsymbol{q}$ be an acyclic join query over a database instance $\mathbf{D}$. Due to the size of $\boldsymbol{q}(\mathbf{D})$, we cannot construct the entire RBBD tree as described in the previous subsection. Instead, we show how we can construct the children of a node u, given that u has been constructed and it is a node of an RBBD tree over $\boldsymbol{q}(\mathbf{D})$. More specifically, we show how we can implement the fair split and the randomized centroid shrink of the RBBD tree efficiently in the relational setting. Furthermore, for every new node v of the RBBD tree we show how to compute the count $v.c = |\square_v \cap \boldsymbol{q}(\mathbf{D})|$ and a representative $v.rep \in \square_v \cap \boldsymbol{q}(\mathbf{D})$. This information is useful for the design of our algorithms.

Root node. We first show how to construct the root node `root` of the RBBD tree, which is always a box without a hole. For every attribute $A_i \in \mathbf{A}$ we compute $\pi_{A_i}(\mathbf{D})$ and we keep the maximum M_i and minimum value m_i. We define the $\square_{\texttt{root}}$ as a box without a hole with opposite corners $(m_1, \ldots, m_d)$ and $(M_1, \ldots, M_d)$. This procedure is executed straightforwardly in $O(N)$ time.

Fair split. The implementation of the fair split in the relational setting is identical to the fair split in the standard computational setting. We only need to identify the longest side of a box along with its center in $O(1)$ time.

Randomized centroid shrink. We only need to show how to implement the sampling procedure since all the other steps can be executed similarly to the standard computational setting. We need to sample uniformly at random $\Lambda = \Theta(\varepsilon^{-2} \log(N^{m+4})) = \Theta(\varepsilon^{-2} \log N)$ points from $\square_u \cap \boldsymbol{q}(\mathbf{D})$. If $\square_u$ is a box (without a hole) then we simply call $\mathsf{SampleRect}(\boldsymbol{q}, \mathbf{D}, \square_u, \Lambda)$ from Lemma 2.1 and we get the set $\mathcal{N}_u$ in $O(N + \varepsilon^{-2} \log^2 N)$ time. In Appendix A.3, we show how we sample in the same running time from $\square_u \cap \boldsymbol{q}(\mathbf{D})$, when $\square_u$ is a box with a hole.

Count and representative tuples in newly constructed nodes. If $\square_u$ is a box without a hole, we call $u.c = \mathsf{CountRect}(\boldsymbol{q}, \mathbf{D}, \square_u)$ and $u.rep = \mathsf{ReprRect}(\boldsymbol{q}, \mathbf{D}, \square_u)$ from Lemma 2.1. Both of these procedures run in $O(N)$ time. In Appendix A.4, we compute the count and the representative point in the same running time, when $\square_u$ is a box with a hole.

Lemma 4.2. *Let $\boldsymbol{q}$ be an acyclic join query over a database instance $\mathbf{D}$ of size N. Given a node u of an RBBD tree over $\boldsymbol{q}(\mathbf{D})$, the fair split procedure is executed in $O(N)$ time, while the randomized centroid shrink procedure is executed in $O(N + \varepsilon^{-2} \log^2 N)$ time.*

4.4 Relational oracles using a partially constructed RBBD tree

Using Lemma 4.2 and Theorem 4.1, we derive new oracles that will allow us to solve optimization problems in the relational setting, efficiently. Let $\mathcal{T}$ be a partially constructed RBBD tree over $\boldsymbol{q}(\mathbf{D})$, i.e., $\mathcal{T}$ consists of a connected subset of nodes (including the root, denoted by `root`) of an RBBD tree over $\boldsymbol{q}(\mathbf{D})$. Initially, $\mathcal{T}$ consists only of the root, as constructed in Section 4.3. For every constructed node $u \in \mathcal{T}$, we store i) $\square_u$ the region corresponding to node u which is either a box or a box with a hole, and ii) $u.a$ a boolean variable which is 1 if u is an *active* node, and 0 if it is an *inactive* node. A point $p \in \boldsymbol{q}(\mathbf{D})$ is called an *inactive point* if there exists a node $u \in \mathcal{T}$ such that $p \in \square_u$ and $u.a = 0$. A point which is not inactive is *active*. While we do not compute it explicitly, for convenience we use $Q' \subseteq \boldsymbol{q}(\mathbf{D})$ to denote the set of current active points in $\boldsymbol{q}(\mathbf{D})$. Furthermore,

in each node $u \in \mathcal{T}$, we store $u.rep$ which is an arbitrary representative point from $\square_u \cap Q'$. Finally, for each node $u \in \mathcal{T}$, we store the number of active points in the region $\square_u$, i.e., $u.c = |\square_u \cap Q'|$.

Given such a partially constructed tree $\mathcal{T}$, we show how to implement five oracles in the relational setting while maintaining $\mathcal{T}$ (new nodes might need to be constructed). The running time of every oracle in this section holds with probability at least $1 - \frac{1}{N^{m+3}}$. Due to space requirements, we show the proofs of all lemmas in Appendix A.5.

Make points in a ball inactive. Given a ball $\mathcal{B}(x, r)$, where $x \in \mathbb{R}^d$ and r is a non-negative real number, the goal is to (implicitly) make all points in $\mathcal{B}(x, r) \cap Q'$ inactive. Let $\mathcal{T}.\mathsf{inactive}(x, r)$ be this oracle, which works as follows. We first execute a query procedure similar to the query procedure on a BBD tree. Let $\mathcal{U}(x, r)$ be the set of canonical nodes we will compute. Let u be a node of $\mathcal{T}$ that the query procedure processes. Initially, $u = \mathtt{root}$ and $\mathcal{U}(x, r) = \emptyset$. If $u.a = 0$, we stop the execution of the query through this branch of the tree. In the rest, we assume that $u.a = 1$. If the children of u have not been constructed in $\mathcal{T}$ we use Lemma 4.2 to construct its children. Next, we continue checking whether $\square_u$ intersects $\mathcal{B}(x, r)$. If $\square_u \subseteq \mathcal{B}(x, (1+\varepsilon)r)$, then we add u in $\mathcal{U}(x, r)$ and we stop the execution through this branch of the tree. If $\square_u \cap \mathcal{B}(x, r) = \emptyset$, we stop the execution of the query through this branch of the tree. In any other case, we continue the query procedure in both children of u recursively. For every added node $u \in \mathcal{U}(x, r)$, we set $u.a = 0$. Before we finish the procedure, we need to update the representative points and the counts of the ancestor nodes of $\mathcal{U}(x, r)$. Starting from the parent node v of the deepest node in $\mathcal{U}(x, r)$, we run recursively: Let u, w be v's two children. If $u.a = w.a = 0$, then we set $v.a = 0$. If $u.a = 1$ and $w.a = 0$ (resp. $u.a = 0$ and $w.a = 1$), we set $v.c = u.c$ and $v.rep = u.rep$ (resp. $v.c = w.c$ and $v.rep = w.rep$). If $u.a = w.a = 1$, we set $v.c = u.c + w.c$ and $v.rep = u.rep$. After we process all canonical nodes' ancestors in one level of the tree, we continue the update of the nodes in the level above.

Lemma 4.3. *$\mathcal{T}.\mathsf{inactive}(x, r)$ makes all points in $\mathcal{B}(x, r) \cap Q'$ inactive and might make some points in $\mathcal{B}(x, (1+\varepsilon)r) \cap Q'$ inactive. Furthermore, in the end of the execution of the oracle, for every node $u \in \mathcal{T}$ the values $u.rep, u.c$ are correct. The running time is $O((\varepsilon^{-d+1} + \log N)(N + \varepsilon^{-2}\log^2 N))$.*

Approximately count active points in a ball. Given a ball $\mathcal{B}(x, r)$ the goal is to compute $|\mathcal{B}(x, r) \cap Q'|$ approximately. We define the oracle $\mathcal{T}.\mathsf{count}(x, r)$, and run the same query procedure as in $\mathcal{T}.\mathsf{inactive}(x, r)$ to compute $\mathcal{U}(x, r)$. We return $\sum_{u \in \mathcal{U}(x,r)} u.c$.

Lemma 4.4. *$\mathcal{T}.\mathsf{count}(x, r)$ returns a number c such that $|\mathcal{B}(x, r) \cap Q'| \leq c \leq |\mathcal{B}(x, (1+\varepsilon)r) \cap Q'|$ in $O((\varepsilon^{-d+1} + \log N)(N + \varepsilon^{-2}\log^2 N))$ time.*

Get a representative from Q'. Next, the goal is to return any point from Q'. We define the oracle $\mathcal{T}.\mathsf{rep}()$. If $\mathtt{root}.a = 0$ then we return $\emptyset$, i.e., $Q' = \emptyset$. If $\mathtt{root}.a = 1$, then we return $\mathtt{root}.rep$.

Lemma 4.5. *$\mathcal{T}.\mathsf{rep}()$ returns a point in Q' if $Q' \neq \emptyset$, and $\emptyset$ otherwise, in $O(1)$ time.*

Report all points in Q'. Assume that we aim to report all points in Q'. We define the oracle $\mathcal{T}.\mathsf{report}()$. Starting from the root node $\mathtt{root}$ we execute a depth-first search over $\mathcal{T}$. If we reach a node u such that $u.a = 0$ we stop the traversal through this branch of the tree. If we reach a node u that we have not constructed its children or u is a leaf node of the RBBD tree, we run $\mathsf{ReportRect}(\boldsymbol{q}, \mathbf{D}, \square_u)$. We continue the depth-first search until we visit all constructed nodes in $\mathcal{T}$.

Lemma 4.6. *$\mathcal{T}.\mathsf{report}()$ returns Q' in time $O(|\mathcal{T}| \cdot N + |Q'|)$, where $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$.*

Sample z points from Q'. The goal is to sample (with replacement) z points from Q', uniformly at random. We describe the oracle $\mathcal{T}.\mathsf{sample}(z)$. We use $\mathcal{T}$ to get each sample one by one. If $\mathtt{root}.a = 0$ we return no sample because $Q' = \emptyset$. Otherwise we set $u = \mathtt{root}$ and we act as follows. If u has two active children v, w then we move to the node v with probability $\frac{v.c}{v.c + w.c}$ and to node w with

probability $\frac{w.c}{v.c+w.c}$. If $v.a = 1$ and $w.a = 0$ (resp. $v.a = 0$ and $w.a = 1$) we move to the node v (resp. w). We recursively repeat this process until we reach a leaf node of the RBBD tree or an active node whose children have not been constructed in $\mathcal{T}$. If we have reached a leaf node, then we trivially add the point stored in the leaf node to the sample set. In case we reach a node u whose children have not been constructed in $\mathcal{T}$, we sample one point from $\square_u \cap \boldsymbol{q}(\mathbf{D})$ as described in Section 4.3.

Lemma 4.7. *$\mathcal{T}$.sample(z) returns z samples uniformly at random from Q' in $O(z \cdot N)$ time.*

5 k-Center Clustering

In Section 3, we already discussed the high-level idea of solving the relational k-center clustering efficiently. Here, we describe all the details and the formal proofs.

5.1 A constant approximation algorithm

We first describe the main part of the algorithm, where given a distance r, if r is sufficiently large, it returns a set $\mathcal{S}$ such that $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq (2+\varepsilon)r$. Later, we run a binary search over the distances in $\boldsymbol{q}(\mathbf{D})$ to get a constant approximation for the relational k-center clustering.

Main algorithm. Let $\mathcal{T}$ consist of the root node `root` of an RBBD tree over $\boldsymbol{q}(\mathbf{D})$ as shown in Subsection 4.3. Initially, we set $\mathcal{S} = \emptyset$. We run the following steps for at most k iterations. In the i-th iteration, we set $p_i = \mathcal{T}$.rep() and we add p_i in $\mathcal{S}$. Then we call $\mathcal{T}$.inactive($p_i, 2r$). If after k iterations $\mathcal{T}$.rep() $\neq \emptyset$ then we return an empty set $\mathcal{S} = \emptyset$. On the other hand, if $\mathcal{T}$.rep() $= \emptyset$ within the k iterations, then we return $\mathcal{S}$.

Analysis. If $r \geq \mathbf{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, we show that the main algorithm returns a non-empty set $\mathcal{S}$. From Lemma 4.3, for every new point p_i all (active) points in $\boldsymbol{q}(\mathbf{D})$ within distance $2r$ become inactive (along with some points within distance $2(1+\varepsilon)r$) by $\mathcal{T}$.inactive($p_i, 2r$). Hence, in each iteration, all the points of at least one optimum cluster become inactive, as discussed in Section 3. So after k iterations all points in $\boldsymbol{q}(\mathbf{D})$ are inactive (i.e., $\mathcal{T}.rep() = \emptyset$) and $\mathcal{S} \neq \emptyset$. Hence, $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq 2(1+\varepsilon)r$. By initially setting $\varepsilon \leftarrow \varepsilon/2$, we have that if $r \geq \mathbf{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\mathbf{D})$ then $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq (2+\varepsilon)r$.

We run the $\mathcal{T}$.inactive($\cdot,\cdot$) and the $\mathcal{T}$.rep() oracles $O(k)$ times. Each oracle runs in $O((\varepsilon^{-d+1} + \log N)(N + \varepsilon^{-2}\log^2 N))$ time, with probability at least $1 - \frac{1}{N^{m+3}}$, as shown in Section 4.4. Overall, the main algorithm runs in $O\left(k\left(\varepsilon^{-d+1} + \log N\right)\left(N + \varepsilon^{-2}\log^2 N\right)\right) = O(k \cdot N \cdot \log N + k \cdot \log^3 N) = \tilde{O}(k \cdot N)$ time assuming that $\varepsilon \in (0,1)$ is an arbitrary constant, with probability at least $1 - \frac{1}{N^2}$.

Complete algorithm. Assume that we run a binary search on the pairwise distances of $\boldsymbol{q}(\mathbf{D})$. For each distance r selected by the binary search we run the main algorithm. If it returns a non-empty set $\mathcal{S}$ then we continue the binary search for smaller values of r. Otherwise, we continue with larger values of r. In the end, the last computed non-empty set $\mathcal{S}$ from the binary search is returned as the final set of centers. From the analysis of the main algorithm along with the discussion in Section 3, such algorithm returns a set $\mathcal{S}$ such that $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq (2+\varepsilon)\mathbf{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$. Unfortunately, we cannot sort all pairwise distances in $\boldsymbol{q}(\mathbf{D})$ and then run a binary search on them because we would need $\Omega(|\boldsymbol{q}(\mathbf{D})|^2)$ time. While it is not straightforward to run a binary search on ℓ_2 pairwise distances, the key idea is to run a binary search over the ℓ_∞ pairwise distances of points in $\boldsymbol{q}(\mathbf{D})$. We show the details of the binary search in Appendix B. Using the fact that for any pair of points $t, p \in \mathbb{R}^d$, $||t-p||_\infty \leq ||t-p||_2 \leq \sqrt{d}||t-p||_\infty$, we prove the next theorem in Appendix B.

Theorem 5.1. *Given an acyclic join query $\boldsymbol{q}$ over d attributes, a database instance $\mathbf{D}$ of size N, a parameter k, and an arbitrarily small constant parameter $\varepsilon \in (0,1)$, there exists an algorithm that computes a set $\mathcal{S} \subseteq \boldsymbol{q}(\mathbf{D})$ of k points and a real number $r_{\mathcal{S}}$ such that $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq (2\sqrt{d}+\varepsilon)\mathbf{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$. The running time of the algorithm is $O(kN\log^2 N + k\log^4 N)$ with probability at least $1 - \frac{1}{N}$.*

Notice that our algorithm from Theorem 5.1 returns an estimation $r_{\mathcal{S}}$ of $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$. This is useful in order to derive a better approximation for relational k-center clustering in Section 5.2 and a constant approximation for the relational k-median/means clustering as shown in Section 6.

5.2 Better approximation algorithm

Using our result from Theorem 5.1, along with the coreset construction from [8], we get the fastest known $(2+\varepsilon)$-approximation algorithm for the relational k-center clustering problem. First, we execute the algorithm from Theorem 5.1 and we get $\mathcal{S}$ and $r_{\mathcal{S}}$. Both $\mathcal{S}$ and $r_{\mathcal{S}}$ are given as input to the coreset construction in [8] and we get the main result of this section.

Theorem 5.2. *Given an acyclic join query $\boldsymbol{q}$ over d attributes, a database instance $\mathbf{D}$ of size N, a parameter k, and an arbitrarily small constant parameter $\varepsilon \in (0,1)$, there exists an algorithm that computes a set $\mathcal{S} \subseteq \boldsymbol{q}(\mathbf{D})$ of k points such that $\boldsymbol{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq (1+\varepsilon)\gamma \boldsymbol{u}_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$. The running time of the algorithm is $O(kN\log^2 N + k\log^4 N + T_{\gamma}^{\mathsf{cen}}(k))$ with probability at least $1-\frac{1}{N}$.*

For example, if Feder and Greene algorithm [34] is executed on the coreset constructed by [8] then we get a $2(1+\varepsilon)$-approximation solution in $O(k\varepsilon^{-d}\log k)$ additional time. By setting $\varepsilon \leftarrow \varepsilon/2$, the approximation factor becomes $2+\varepsilon$. The overall asymptotic complexity for arbitrarily small constant $\varepsilon \in (0,1)$ is $O(kN\log^2 N + k\log^4 N)$. In [32] we show an alternative $(2+\varepsilon)$-approximation algorithm with the same running time without constructing the coreset from [8].

6 k-Median and k-Means Clustering

We focus on the relational k-means clustering, while the algorithm for k-median follows almost verbatim. In this section, we first describe a constant approximation algorithm returning $\tilde{O}(k)$ centers. Then, the returned set is given as input to the coreset construction in [31] to derive the final approximation algorithm and get exactly k centers (Subsection 6.2). Our new tools can also improve the coreset construction from [31], leading to an even faster algorithm.

6.1 Constant approximation algorithm with $\tilde{O}(k)$ centers

High level idea. We first show a constant approximation algorithm for the relational k-median/means problem on $\boldsymbol{q}(\mathbf{D})$ that runs in only $\tilde{O}(kN)$ time, returning a set of $O(k\log^3 N)$ centers. We propose a modification of the algorithm in [40] in the relational setting. However, the algorithm of [40] is heavily based on the properties of the standard computational setting, and it is not trivial how to extend it in the relational setting. We use our new tools for constructing an RBBD tree on–the–fly to implement an efficient constant approximation for the relational k-means/median problem.

We first execute our relational k-center algorithm on $\boldsymbol{q}(\mathbf{D})$ in order to get a polynomial upper and lower bound on the optimum k-means/median cost. At first, all the points in $\boldsymbol{q}(\mathbf{D})$ are considered active. Then, the algorithm works in iterations. In the i-th iteration, we get a small set X_i of $\tilde{O}(k)$ samples from the active points. Next, tuples in $\boldsymbol{q}(\mathbf{D})$ that are close enough to X_i are considered assigned to their closest point in X_i and are implicitly removed (made inactive) from $\boldsymbol{q}(\mathbf{D})$. This procedure is repeated until every point in $\boldsymbol{q}(\mathbf{D})$ is removed. We note that some points that are assigned to the samples might pay a k-median/means cost much larger than the cost they pay in the optimum solution $\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))$. These points are called *bad points*, while the rest points are called *good points*. Similarly to [40], we show that the number of good points in each iteration is much larger than the number of bad points, allowing us to charge the cost of bad points to the good points. In contrast to [40], where the authors assign each point (in the standard computational setting) to a center, we use our machinery for implementing an RBBD tree in the relational setting to implicitly assign groups of points in $\boldsymbol{q}(\mathbf{D})$ to approximately closest centers.

Algorithm. First, we compute $|\boldsymbol{q}(\mathbf{D})|$ using Lemma 2.1. We construct the root node `root` of the RBBD tree $\mathcal{T}$ on $\boldsymbol{q}(\mathbf{D})$ as described in Section 4.3. Let $\mathcal{S}$ be the set of points we will return as a solution to k-means clustering, and let $r_{\mathcal{S}}$ be the estimation of the cost $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$. Initially, we set $r_{\mathcal{S}} = 0$ and $\mathcal{S} = \emptyset$. Our algorithm runs in iterations. Let $Q'_i \subseteq \boldsymbol{q}(\mathbf{D})$ be the set of active points in $\boldsymbol{q}(\mathbf{D})$ at the beginning of the i-th iteration. This set of points is not explicitly constructed. Instead, we only use Q'_i to make our algorithm more intuitive. Initially, $Q'_1 = \boldsymbol{q}(\mathbf{D})$. Furthermore, let $\tau_i = |Q'_i|$, so initially $\tau_1 = |\boldsymbol{q}(\mathbf{D})|$. In contrast to Q'_i, τ_i will be explicitly stored in every step of the algorithm.

Algorithm 1: Rel-k-Means$(\boldsymbol{q}, \mathbf{D}, k, \varepsilon)$

```
Compute |q(D)| using Lemma 2.1;
Build the root of RBBD tree T over q(D);
(V, L) ← Rel-k-Center(q, D, k, ε);
M ← 2 log(|q(D)|) + 3;
a_0 ← 0; b_0 ← L/(4|q(D)|); a_∞ ← 2L|q(D)|;
  b_∞ ← ∞;
a_j ← 2^{j-1} L/(4|q(D)|),
  b_j ← 2^j L/(4|q(D)|)  ∀j ∈ [M];
S ← ∅; r_S ← 0; τ_1 ← |q(D)|; i ← 1;
while τ_i > 480 · k log^2 |q(D)| do
    X_i ← T.sample(240k log^2 |q(D)|);
    S ← S ∪ X_i;
    T' ← copy(T);
    for j ∈ {0, 1, ..., M, ∞} do
        c_{i,j} ← 0;
        foreach x ∈ X_i do
            c_{i,j} ← c_{i,j} + T'.count(x, b_j);
            T'.inactive(x, b_j);
    β_i ← max{j : c_{i,j} > τ_i / (10 log |q(D)|)};
    Reset T to state of T' before defining
      c_{i,β_i+1};
    τ_{i+1} ← τ_i − Σ_{ℓ∈[β_i]∪{0}} c_{i,ℓ};
    r_S ← r_S + Σ_{ℓ∈[β_i]∪{0}} c_{i,ℓ} · ((1+ε)b_ℓ)^2;
    i ← i + 1;
S ← S ∪ T.report();
return S;
```

• **Compute estimation of $\mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$.** We run our algorithm for the relational k-center problem from Theorem 5.2. Let V denote the set of points and L the estimated cost returned by the algorithm. It is easy to verify that $\left(\frac{L}{2+\varepsilon}\right)^2 \leq \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$ and $\mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D})) \leq |\boldsymbol{q}(\mathbf{D})| \cdot L^2$, by definition. So, it holds that $\frac{L^2}{9} \leq \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D})) \leq |\boldsymbol{q}(\mathbf{D})| \cdot L^2$.

After computing L, the algorithm runs in iterations. In iteration i, we first check if $\tau_i \leq 480 \cdot k \log^2(|\boldsymbol{q}(\mathbf{D})|)$. If yes then we add in $\mathcal{S}$ all remaining active points, i.e., $\mathcal{S} = \mathcal{S} \cup \mathcal{T}.\mathsf{report}()$.

• **Get a set of $\Theta(k \log^2 |\boldsymbol{q}(\mathbf{D})|)$ samples from Q'_i uniformly at random.** In the i'th iteration, we get a set X_i of $240 \cdot k \log^2 |\boldsymbol{q}(\mathbf{D})|$ samples from Q'_i uniformly at random. In particular, we set $X_i = \mathcal{T}.\mathsf{sample}(240k \log^2 |\boldsymbol{q}(\mathbf{D})|)$ and we update $\mathcal{S} = \mathcal{S} \cup X_i$.

• **Partition Q'_i based on their distances from X_i.** We implicitly partition the points in Q'_i in classes based on their distance from the set X_i. Let $a_0 = 0$, $b_0 = \frac{L}{4|\boldsymbol{q}(\mathbf{D})|}$, $a_\infty = 2L|\boldsymbol{q}(\mathbf{D})|$, $b_\infty = \infty$, and $a_j = 2^{j-1}\frac{L}{4|\boldsymbol{q}(\mathbf{D})|}$, $b_j = 2^j \frac{L}{4|\boldsymbol{q}(\mathbf{D})|}$ for every $j = 1, \ldots, M$, where $M = 2\log(|\boldsymbol{q}(\mathbf{D})|) + 3$. We construct a copy $\mathcal{T}'$ of $\mathcal{T}$ and repeat the next steps for every $j = 0, 1, \ldots, M, \infty$. We define $c_{i,j}$ and initially set $c_{i,j} = 0$. We go through the points $x \in X_i$ one by one and we set $c_{i,j} = c_{i,j} + \mathcal{T}'.\mathsf{count}(x, b_j)$ and then we call $\mathcal{T}'.\mathsf{inactive}(x, b_j)$.

Let $Q'_{i,j} \subseteq Q'_i$ be the set of points that were active just before the definition of $c_{i,j}$ and became inactive after applying $\mathcal{T}'.\mathsf{inactive}(x, b_j)$ for all $x \in X_i$. While we do not compute $Q'_{i,j}$ explicitly in the algorithm, in the analysis we show that for every iteration i:

(1) for every distinct pair $j_1, j_2 \in [M] \cup \{0, \infty\}$, $Q'_{i,j_1} \cap Q'_{i,j_2} = \emptyset$, while $\bigcup_{j \in [M] \cup \{0,\infty\}} Q'_{i,j} = Q'_i$,
(2) for every $j \in [M] \cup \{0, \infty\}$, the set $Q'_{i,j} \subseteq Q'_i$ contains all points of Q'_i with distance greater than $(1+\varepsilon)a_j$ and at most b_j from X_i, might contain some points with distance at most $(1+\varepsilon)b_j$ from X_i, and might contain some points with distance at least a_j from X_i, and
(3) for every $j \in [M] \cup \{0, \infty\}$, $c_{i,j} = |Q'_{i,j}|$.

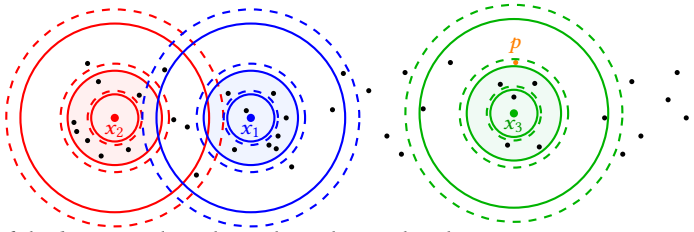


Fig. 5. Example of the k-means algorithm, where three colored points x_1, x_2, x_3 represent the points in the random set X_i. Solid circles represent $\mathcal{B}(x_\ell, b_j)$, and dashed circles $\mathcal{B}(x_\ell, (1+\varepsilon)b_j)$ for $\ell \in [3], j \in \{0,1,2\}$. If $\beta_i = 1$, the points in the shaded area belong in $\hat{Q}_i$, i.e., they are considered covered and they become inactive. The orange point p may or may not become inactive since it lies in $\mathcal{B}(x_3, (1+\varepsilon)b_1) \setminus \mathcal{B}(x_3, b_1)$.

• **Decide what subset of Q'_i should be covered by X_i.** Let β_i be the largest index such that $c_{i,\beta_i} > \frac{\tau_i}{10 \log |\boldsymbol{q}(\mathbf{D})|}$ (we will show that $\beta_i \leq M$ exists with high probability). In the analysis we show that with high probability, the k-means cost we pay assigning all points in the first $\beta_i + 1$ sets, $\hat{Q}_i = \bigcup_{\ell \in [\beta_i] \cup \{0\}} Q'_{i,\ell}$ to the centers X_i is bounded (approximately) by the cost that the optimum solution pays assigning $\hat{Q}_i$, i.e., $\mu_{X_i}(\hat{Q}_i) \leq O(1) \cdot \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)$. We delete $\mathcal{T}'$ and we repeat the procedure from the previous bullet using the tree $\mathcal{T}'$ up to $j = \beta_i$. In other words, $\mathcal{T}$ gets the instance of $\mathcal{T}'$ at the moment just before c_{i,β_i+1} is defined. In this version of $\mathcal{T}$, each point in $\hat{Q}_i$ is inactive. We show an example in Figure 5. We observe that in the next iteration $Q'_{i+1} \leftarrow Q'_i \setminus \hat{Q}_i$. We update $\tau_{i+1} = \tau_i - \sum_{\ell \in [\beta_i] \cup \{0\}} c_{i,\ell}$ and $r_{\mathcal{S}} = r_{\mathcal{S}} + \sum_{\ell \in [\beta_i] \cup \{0\}} c_{i,\ell}((1+\varepsilon)b_\ell)^2$.

The pseudocode of our method is shown in Algorithm 1.

Analysis. The proofs of the next lemmas are shown in Appendix C.

Lemma 6.1. *For every iteration i, the family of sets $Q'_{i,0}, \ldots, Q'_{i,1}, \ldots, Q'_{i,M}, Q'_{i,\infty}$ satisfy the three properties in the third bullet.*

Next, we show that 1) the number of iterations of the algorithm is $O(\log N)$, so $|\mathcal{S}| = O(k \log^3 N)$, and 2) for every iteration i, $\mu_{X_i}(\hat{Q}_i) = O\left(\mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)\right)$. For every iteration i, let $Q_i^{\text{bad}} = \{p \in Q'_i \mid \phi(p, X_i) > 2\phi(p, \mathsf{OPT}(\boldsymbol{q}(\mathbf{D})))\}$. We show that the size of Q_i^{bad} is small with high probability.

Lemma 6.2. *For every iteration i, $|Q_i^{\text{bad}}| \leq \frac{\tau_i}{20 \log |\boldsymbol{q}(\mathbf{D})|}$, with probability at least $1 - \frac{1}{N^3}$.*

Since $|Q'_i| > 480 \cdot k \log^2 |\boldsymbol{q}(\mathbf{D})|$, $M + 2 = 2 \log |\boldsymbol{q}(\mathbf{D})| + 5$, and by Lemma 6.1, there exists at least one set Q'_{i,j^*} such that $c_{i,j^*} > \frac{\tau_i}{10 \log |\boldsymbol{q}(\mathbf{D})|}$, by the pigeonhole principle. So, the index β_i is well defined. In the proof of the next lemma, we also show that $\beta_i \neq \infty$.

Lemma 6.3. *For every iteration i, $|\hat{Q}_i| \geq \frac{\tau_i}{2}$, so the algorithm executes $O(\log |\boldsymbol{q}(\mathbf{D})|) = O(\log N)$ iterations, and $|\mathcal{S}| = O(k \log^3 N)$, with probability at least $1 - \frac{1}{N^2}$.*

The next lemma bounds the k-means cost of $\mathcal{S}$ and the running time of the algorithm.

Lemma 6.4. *For any arbitrary small constant $\varepsilon \in (0,1)$, it holds that $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq (84+\varepsilon) \cdot \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, with probability at least $1 - \frac{1}{N}$. The algorithm runs in $O(kN \log^5 N + k \log^7 N)$ time with probability at least $1 - \frac{1}{N}$.*

We argue that the same bounds hold for the relational k-median clustering. In fact, the constant approximation factor for the relational k-median problem is smaller since we do not raise the distances to the power of 2. We conclude with the next Theorem.

Theorem 6.5. *Given an acyclic join query $\boldsymbol{q}$ over d attributes, a database instance $\mathbf{D}$ of size N, a parameter k, and an arbitrarily small constant parameter $\varepsilon \in (0,1)$, there exists an algorithm that computes a set $\mathcal{S} \subseteq \boldsymbol{q}(\mathbf{D})$ of $O(k \log^3 N)$ points and a number $r_{\mathcal{S}}$ such that $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq (84+\varepsilon) \cdot \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, with probability at least $1 - \frac{1}{N}$. The running time of the algorithm is $O(kN \log^5 N + k \log^7 N)$ with probability at least $1 - \frac{1}{N}$. An algorithm with the same theoretical guarantees and running time exists for relational k-median clustering.*

6.2 Better approximation algorithm

Using our result from Theorem 6.5, along with the coreset construction from [31] we get the fastest known $(4+\varepsilon)\gamma$-approximation for the relational k-means clustering problem and the fastest known $(2+\varepsilon)\gamma$-approximation for the relational k-median clustering problem. The RBBD construction in the relational setting can also accelerate the coreset construction in [31].

First we execute the algorithm from Theorem 6.5 and we get $\mathcal{S}$ and $r_{\mathcal{S}}$. Given $\mathcal{S}, r_{\mathcal{S}}$, a coreset $C \subseteq \boldsymbol{q}(\mathbf{D})$ with $|C| = O(k\varepsilon^{-d} \log^4 N)$ for the relational k-means clustering can be constructed in $O(kN \log^5 N + k^2 \log^{11} N \cdot \min\{\log^{d+1}(k), k \log^5 N\})$ time [31]. Finally, a γ-approximation algorithm kMeansAlg is executed on the coreset C to derive a $(4+\varepsilon)\gamma$-approximation in $O(T_\gamma^{\mathsf{mean}}(k \log^4 N))$ additional time. In [32] we show how the relational RBBD tree construction can be used to improve the running time of the coreset construction in [31] to $O(kN \log^5 N + k \log^7 N)$ time.

Theorem 6.6. *Given an acyclic join query $\boldsymbol{q}$, a database instance $\mathbf{D}$ of size N, a parameter k, and an arbitrarily small constant parameter $\varepsilon \in (0,1)$, there exists an algorithm that computes a set $\mathcal{S} \subseteq \boldsymbol{q}(\mathbf{D})$ of k points such that $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq (4+\varepsilon)\gamma \mu_{\mathsf{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, with probability at least $1 - \frac{1}{N}$. The running time of the algorithm is $O(kN \log^5 N + k \log^7 N + T_\gamma^{\mathsf{mean}}(k \log^4 N))$ with probability at least $1 - \frac{1}{N}$. An algorithm with approximation factor $(2+\varepsilon)\gamma$ and the same running time exists for the relational k-median clustering problem.*

7 Extensions

Generality. Our method is general: Any algorithm in the standard computational setting that has access to the input data through σ queries to a BBD tree can be converted to a randomized algorithm in the relational setting with the same approximation guarantee and running time $\tilde{O}(\sigma \cdot N)$. As many geometric approximation algorithms fall into this category, our framework yields efficient relational counterparts for a variety of tasks. For example, in the full version of the paper [32], we show that our method can be used to implement an approximate version of the Gonzalez algorithm [37]. This leads to efficient algorithms for several relational diversity and fairness problems and improves the running time of the algorithms in [8, 45] under data complexity.

Cyclic queries. We use the notion of fractional hypertree width [38] and apply a standard procedure to extend our algorithms to every join query. For a cyclic query $\boldsymbol{q}$, we convert it to an equivalent acyclic query such that each relation is the result of a (possibly cyclic) join query with fractional edge cover at most $\mathsf{fhw}(\boldsymbol{q})$. We evaluate the (possibly cyclic) queries to get the new relations and then apply our algorithms to the acyclic query. All the guarantees are the same as in Theorems 5.2, 6.6 except for the running time, where we simply replace any N factor with N^{fhw}. Since it is a typical method in database theory, we give the details in [32].

8 Future work

This work opens several directions for future research. A central open question is whether our $\tilde{O}(kN)$-time algorithms for relational clustering are optimal. Although our methods outperform the current state-of-the-art, it is interesting to study whether other geometric techniques can be adapted to achieve $\tilde{O}(N)$ time algorithms in the relational setting. We also aim to investigate whether known lower bounds for k-center clustering in the standard computational setting [17, 34] can be extended to the relational setting, potentially establishing near-optimal results.

References

[1] https://db-engines.com/en/ranking_categories. [Accessed 2026-03-15].

[2] Yelp dataset challenge. https://www.yelp.com/dataset/challenge, 2017. Accessed 2026-03-15.

[3] Kaggle machine learning and data science survey. https://www.kaggle.com/datasets/kaggle/kaggle-survey-2018, 2018. [Accessed 2026-03-15].

[4] Mahmoud Abo Khamis, Ryan R Curtin, Benjamin Moseley, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Functional aggregate queries with additive inequalities. *ACM Transactions on Database Systems (TODS)*, 45(4):1–41, 2020.

[5] Mahmoud Abo Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. A relational gradient descent algorithm for support vector machine training. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 100–113. SIAM, 2021.

[6] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: in-database learning thunderstruck. In *Proceedings of the second workshop on data management for end-to-end machine learning*, pages 1–10, 2018.

[7] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 325–340, 2018.

[8] Pankaj K Agarwal, Aryan Esmailpour, Xiao Hu, Stavros Sintos, and Jun Yang. Computing a well-representative summary of conjunctive query results. *Proceedings of the ACM on Management of Data*, 2(5):1–27, 2024.

[9] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 15–28. Springer, 2009.

[10] Marcelo Arenas, Timo Camillo Merkl, Reinhard Pichler, and Cristian Riveros. Towards tractability of the diversity of query answers: Ultrametrics to the rescue. *Proceedings of the ACM on Management of Data*, 2(5):1–26, 2024.

[11] David Arthur and Sergei Vassilvitskii. k-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

[12] Sunil Arya and David M Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.

[13] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[14] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[15] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[16] MohammadH. Bateni et al. Extreme k-center clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3941–3949. AAAI Press, 2021.

[17] MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic k-center clustering for adaptive and oblivious adversaries. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2677–2727. SIAM, 2023.

[18] Chazelle Bernard. The discrepancy method: randomness and complexity, 2001.

[19] Paul B Callahan and S Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[20] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems*, 48(1):1–45, 2023.

[21] Timothy M Chan. Optimal partition trees. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 1–10, 2010.

[22] Jiaxiang Chen, Qingyuan Yang, Ruomin Huang, and Hu Ding. Coresets for relational data and the applications. *Advances in Neural Information Processing Systems*, 35:434–448, 2022.

[23] Zhaoyue Cheng and Nick Koudas. Nonlinear models over normalized data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1574–1577. IEEE, 2019.

[24] Zhaoyue Cheng, Nick Koudas, Zhe Zhang, and Xiaohui Yu. Efficient construction of nonlinear models over normalized data. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1140–1151. IEEE, 2021.

[25] Sam Coy, Artur Czumaj, and Gopinath Mishra. On parallel k-center clustering. In *Proceedings of the 35th ACM symposium on parallelism in algorithms and architectures*, pages 65–75, 2023.

[26] Ryan Curtin, Benjamin Moseley, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Rk-means: Fast clustering for relational data. In *International Conference on Artificial Intelligence and Statistics*, pages 2742–2752. PMLR, 2020.

[27] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, Berlin, Germany, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

[28] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Ranked enumeration of join queries with projections. *Proceedings of the VLDB Endowment*, 15(5):1024–1037, 2022.

[29] Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. In *24th International Conference on Database Theory*, 2021.

[30] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689, 2011.

[31] Aryan Esmailpour and Stavros Sintos. Improved approximation algorithms for relational clustering. *Proceedings of the ACM on Management of Data*, 2(5):1–27, 2024.

[32] Aryan Esmailpour and Stavros Sintos. Faster relational algorithms using geometric data structures. https://arxiv.org/abs/2603.11402, 2026. arXiv:2603.11402.

[33] Artur Czumaj et al. Fully-scalable mpc algorithms for clustering in high dimension. In *Proceedings of the 51st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 50:1–50:20. Leibniz International Proceedings in Informatics (LIPIcs), 2024.

[34] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.

[35] Arnold Filtser, Shaofeng H.-C. Jiang, Yi Li, Anurag Murty Naredla, Ioannis Psarros, Qiaoyuan Yang, and Qin Zhang. Faster approximation algorithms for k-center via data reduction. In *Proceedings of the 2025 International Conference on Machine Learning (ICML)*. PMLR, 2025.

[36] Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4:1–9, 1974.

[37] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[38] G. Gottlob, G. Greco, and F. Scarcello. Treewidth and hypertree width. *Tractability: Practical Approaches to Hard Problems*, 1, 2014.

[39] Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[40] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[41] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 150–158, 2005.

[42] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[43] Arun Kumar, Jeffrey Naughton, and Jignesh M Patel. Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1969–1984, 2015.

[44] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[45] Timo Camillo Merkl, Reinhard Pichler, and Sebastian Skritek. Diversity of answers to conjunctive queries. In *26th International Conference on Database Theory (ICDT 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[46] Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198, pages 97:1–97:21, 2021.

[47] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):1–40, 2018.

[48] Hung Q Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *Acm Sigmod Record*, 42(4):5–16, 2014.

[49] Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM*, 59(6):1–22, 2012.

[50] Steffen Rendle. Scaling factorization machines to relational data. *Proceedings of the VLDB Endowment*, 6(5):337–348, 2013.

[51] Report. Relational database management system market: Industry analysis and forecast 2021-2027: By type, deployment, end users, and region, 2022.

[52] Jeffrey S Salowe. L-infinity interdistance selection by parametric search. *Information processing letters*, 30(1):9–14, 1989.

[53] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q Ngo, and XuanLong Nguyen. Learning models over relational data: A brief tutorial. In *Scalable Uncertainty Management: 13th International Conference, SUM 2019, Compiègne, France, December 16–18, 2019, Proceedings 13*, pages 423–432. Springer, 2019.

[54] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18, 2016.

[55] Vaishali Surianarayanan, Neeraj Kumar, and Stavros Sintos. Clustering with set outliers and applications in relational clustering. *Proc. ACM Manag. Data*, 3(5), 2025. doi:10.1145/3767712.

[56] Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 13, page 1582. NIH Public Access, 2020.

[57] Nikolaos Tziavelis, Nofar Carmeli, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Efficient computation of quantiles over joins. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 303–315, 2023.

[58] Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. Towards factorized svm with gaussian kernels over normalized data. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1453–1464. IEEE, 2020.

[59] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, pages 82–94, 1981.

A Missing details from Section 4

A.1 Execution of centroid shrink when $\square_u$ is a box with a hole

First, we describe the procedure in the standard BBD tree [12, 13]. Initially $\rho = \square_u^+$. The authors, compute the smallest midpoint box that contains all points in $\rho \cap P_u$ along with the inner box $\square_u^-$. Let $\rho' \subseteq \rho$ be this box. They apply $O(d)$ fair splits on ρ' until they find a non-trivial fair split, i.e., a fair split that splits the points in $\rho' \cap P_u$ into two non-empty subsets. Let $\hat{\rho} \subset \rho$ be one of the two boxes created by the non-trivial fair split such that $|\hat{\rho} \cap P_u| \geq \frac{|\rho' \cap P_u|}{2}$. The key observation is that $|\hat{\rho} \cap P_u| \leq |\rho \cap P_u| - 1$. If $\hat{\rho}$ contains the inner box $\square_u^-$, then they check the following condition. If $|\hat{\rho} \cap P_u| > \frac{2}{3}|P_u|$, they set $\rho = \hat{\rho}$ and they continue recursively. If $|\hat{\rho} \cap P_u| \leq \frac{2}{3}|P_u|$, then they construct two children nodes w, v of u such that $\square_w = \rho' \setminus \hat{\rho}$ is a box (without a hole), and $\square_v$ is a box with a hole with $\square_v^+ = \hat{\rho}$ and $\square_v^- = \square_u^-$. Otherwise, let $\hat{\rho}$ be the first box found by the recursive procedure that does not contain the inner box $\square_u^-$. Then they create two children w, v of u such that $\square_w$ is a box with a hole with $\square_w^+ = \rho$ and $\square_w^- = \rho'$, and $\square_v$ is a box with a hole with $\square_v^+ = \rho'$ and $\square_v^- = \square_u^-$. Furthermore, they create two children v_1, v_2 of v. $\square_{v_2}$ is a box with a hole with $\square_{v_2}^+ = \rho' \setminus \hat{\rho}$ and $\square_{v_2}^- = \square_u^-$, while $\square_{v_1} = \hat{\rho}$ is a box (without a hole). The recursive procedure continues setting $\rho = \hat{\rho}$. Notice that $\rho = \hat{\rho}$ does not contain any hole. They continue the recursive approach by finding the smallest midpoint box ρ' that contains $P_u \cap \rho$ and applying a fair split on ρ' until they find $\hat{\rho}$ such that $\frac{1}{3}|P_u| \leq |\hat{\rho} \cap P_u| \leq \frac{2}{3}|P_u|$. In the end, they create two children nodes v_{1a}, v_{1b} of v_1 such that $\square_{v_{1b}}$ is a box with a hole, with $\square_{v_{1b}}^+ = \rho'$ and $\square_{v_{1b}}^- = \hat{\rho}$, while $\square_{1a} = \hat{\rho}$ is a box. Overall, the centroid shrink might create two new nodes v, w or six new nodes $w, v, v_1, v_{1a}, v_{1b}, v_2$. As shown in [13], the smallest midpoint box $\rho' \subseteq \rho$ that contains all points in $P_u \cap \rho$ can be found in $O(1)$ time if we know the corners of ρ and the corners of $\square_u^-$. Since in every recursive iteration $|\hat{\rho} \cap P_u| \leq |\rho' \cap P_u| - 1$, the centroid procedure is executed in $O(|P_u|)$ time.

For the RBBD tree, we execute the same operations on $\mathcal{N}_u$, instead of P_u. The first time we find a box $\hat{\rho}$ that does not contain $\square_u^-$, we construct the subtree with the six new nodes as shown above.

A.2 Proof of Theorem 4.1

We prove Theorem 4.1 by proving the next lemmas.

Lemma A.1. *The randomized centroid shrink procedure on a node u computes a midpoint box $\hat{\rho}$ such that $\frac{1}{3} - \varepsilon \leq \frac{|\hat{\rho} \cap P_u|}{|P_u|} \leq \frac{2}{3} + \varepsilon$, with probability at least $1 - \frac{1}{\Omega(n^\zeta)}$.*

Proof. The recursive approach in the randomized centroid shrink is similar to the standard centroid shrink, but instead of executing it on P_u it is executed in $\mathcal{N}_u$. Hence, the randomized centroid shrink procedure will finish after $O(\varepsilon^{-2} \log n^\zeta) = O(\varepsilon^{-2} \log n^\zeta)$ recursive calls. The algorithm stops the first time it finds a majority midpoint box $\hat{\rho}$ such that $|\hat{\rho} \cap \mathcal{N}_u| \leq \frac{2}{3}|\mathcal{N}_u|$. By definition, the set $\mathcal{N}_u$ is an ε-sample with probability at least $1 - \frac{1}{\Omega(n^\zeta)}$. Hence, $\frac{|\hat{\rho} \cap P_u|}{|P_u|} \leq \frac{|\hat{\rho} \cap \mathcal{N}_u|}{|\mathcal{N}_u|} + \varepsilon \leq \frac{2}{3} + \varepsilon$. It remains to show the other direction of the inequality. Let ρ' be the midpoint box that we executed the last fair split (to find $\hat{\rho}$). By definition, we had $\frac{|\mathcal{N}_u \cap \rho'|}{|\mathcal{N}_u|} > \frac{2}{3}$. Notice that by definition of $\hat{\rho}$, it holds that $|\mathcal{N}_u \cap \hat{\rho}| \geq \frac{|\mathcal{N}_u \cap \rho'|}{2}$. Hence, $\frac{|\mathcal{N}_u \cap \hat{\rho}|}{|\mathcal{N}_u|} \geq \frac{1}{2} \cdot \frac{|\mathcal{N}_u \cap \rho'|}{|\mathcal{N}_u|} \geq \frac{1}{3}$. Since $\mathcal{N}_u$ is an ε-sample with probability at least $1 - \frac{1}{\Omega(n^\zeta)}$, we conclude $\frac{|\hat{\rho} \cap P_u|}{|P_u|} \geq \frac{|\hat{\rho} \cap \mathcal{N}_u|}{|\mathcal{N}_u|} - \varepsilon \geq \frac{1}{3} - \varepsilon$. The lemma follows. □

Lemma A.2. *The RBBD tree has the same properties and the same asymptotic query complexity as the BBD tree, with high probability. The height of the RBBD tree is $O(\log n)$, the number of nodes is $O(n)$ and the query complexity is $O(\log n + \varepsilon^{-d+1})$, with probability at least $1 - \frac{1}{n^{\zeta-1}}$.*

Proof. The construction of the RBBD tree is similar to that of the BBD tree, except for the centroid shrink. Using the randomized centroid shrink, for a node u we find a box $\hat{\rho}$ such that $\frac{1}{3} - \varepsilon \leq \frac{|\hat{\rho} \cap P_u|}{|P_u|} \leq \frac{2}{3} + \varepsilon$, with probability at least $1 - \frac{1}{\Omega(n^\zeta)}$.

For now, let us assume that the inequality of Lemma A.1 holds for every execution of the randomized centroid shrink. Next, we show that under this assumption, asymptotically, there is no difference between the BBD tree and the RBBD tree for sufficiently small constant ε. In [13] the authors show that every 4 levels of descent in the BBD tree, the number of points associated with the nodes decreases by at least a factor $\frac{2}{3}$, so the height of the tree is $O(\log n)$ and the BBD tree has $O(n)$ nodes. Similarly, in our case, observe that each randomized centroid shrink introduces three new levels into the tree, and we alternate this with fair splits, so it follows that with each four levels the number of points decreases by at least a factor of $\frac{2}{3} - \varepsilon$. For a small ε, for example $\varepsilon < 0.1$, the height of the tree is $O(\log n)$ by the same argument. Equivalently we can argue that every 8 levels of descent in the RBBD tree, the number of points associated with the nodes decreases by at least a factor $\frac{2}{3}$, the height of the tree is $O(\log n)$ and the number of nodes is $O(n)$. Next, the authors in [13] argue that every $4d$ levels of descent in the BBD tree, the sizes of the associated cells decrease by at least a factor of $\frac{2}{3}$. Similarly, in our case, note that to decrease the size of a cell, we must decrease its size along each of d dimensions. Since fair splits are performed at least at every fourth level of the tree, and each such split decreases the longest side by at least a factor of $2/3$, it follows that after at most d splits (that is, at most $4d$ levels of the tree) the size decreases by this factor. All proofs of the BBD tree [13] also hold for the RBBD tree assuming that all randomized centroid shrink operators are executed correctly, i.e., assuming that Lemma A.1 holds with probability 1. Given that the inequality of Lemma A.1 holds with probability at least $1 - \frac{1}{\Omega(n^{\zeta})}$ and using the union bound over $O(n)$ nodes, we have that the probability that all randomized centroid shrink operators are executed correctly is at least $1 - \frac{1}{n^{\zeta-1}}$.

Finally, the query procedure between BBD tree and RBBD tree is identical. Given a query ball $\mathcal{B}(p, r)$ we start from the root of the tree. Let u be the node of the RBBD tree the query procedure processes. We check whether $\mathcal{B}(p, r) \cap \square_u = \emptyset$. If yes, we stop the execution towards that branch of the tree. If $\square_u \subseteq \mathcal{B}(p, (1+\varepsilon)r)$ we add u in the set of the canonical nodes $\mathcal{U}(p, r)$. In any other case, we continue the search in both children of u. In the end the RBBD tree always computes a set of canonical nodes $\mathcal{U}$ such that $\mathcal{B}(p, r) \cap P \subseteq P \cap \bigcup_{u \in \mathcal{U}} \square_u \subseteq \mathcal{B}(p, (1+\varepsilon)r) \cap P$. Since the height of the RBBD tree is $O(\log n)$ with high probability, following our discussion above, the query analysis from [12, 13] can also be used for the RBBD tree to show that $\mathcal{U}(p, r)$ can be computed in $O(\varepsilon^{-d+1} + \log n)$ time and $|\mathcal{U}(p, r)| = O(\varepsilon^{-d+1} + \log n)$, with probability at least $1 - \frac{1}{n^{\zeta-1}}$. □

Lemma A.3. *Assume that we can get a random sample set $\mathcal{N}_u$ from a box or a box with a hole, in $f(n, \varepsilon)$ time, where f is a positive real function that depends on n and ε, then the RBBD tree can be constructed in $O\left(n \log(n) + n \cdot \left(f(n, \varepsilon) + \varepsilon^{-2} \log^2 n\right)\right)$ time, with probability at least $1 - \frac{1}{n^{\zeta-1}}$.*

Proof. Since the fair split procedure is identical to the standard BBD tree, all fair split procedures are executed in $O(n \log n)$ time using the machinery from the standard BBD tree. Next, we focus on the randomized centroid shrink procedure. For a node u we get the set $\mathcal{N}_u$ in $f(n, \varepsilon)$ time. We find the smallest midpoint box ρ' that contains $\mathcal{N}_u$ along with the inner box $\square_u^-$ in $O(\varepsilon^{-2} \log n)$ time since we explicitly know all points in $\mathcal{N}_u$ and the corners of $\square_u^+$, using the highest and lowest values in each coordinate as described in [13]. After the computation of ρ', we execute $O(1)$ fair splits in order to find the first non-trivial fair split. The hyperplane in every fair split is computed straightforwardly in $O(1)$ time. Using a simple binary search tree over each coordinate of the points in $\mathcal{N}_u$, we can check whether a fair split is trivial in $O(\log(\varepsilon^{-2} \log n))$ time. After finding the first non-trivial fair split, we compute $\hat{\rho}$ in $O(\log(\varepsilon^{-2} \log n)) = O(\log n)$ time using the search binary tree that corresponds to the last non-trivial fair split by checking whether $|\hat{\rho} \cap \mathcal{N}_u| > |(\rho' \setminus \hat{\rho}) \cap \mathcal{N}_u|$. After each non-trivial split, the box $\hat{\rho}$ contains at most $|\rho' \cap \mathcal{N}_u| - 1$ samples from $\mathcal{N}_u$. Hence, this recursive approach runs for at most $O(|\mathcal{N}_u|) = O(\varepsilon^{-2} \log n)$ times. Thus, each randomized centroid shrink runs in $O(f(n, \varepsilon) + \varepsilon^{-2} \log^2 n)$ time. From Lemma A.2, the RBBD tree has $O(n)$ nodes with

probability at least $1 - \frac{1}{n^{\zeta-1}}$, so the running time of all randomized centroid shrink procedures is $O(n(f(n,\varepsilon) + \varepsilon^{-2}\log^2 n))$, with probability at least $1 - \frac{1}{n^{\zeta-1}}$. □

A.3 Sampling from a box with a hole

We consider the more involved case where u is a node such that $\square_u$ is a box with a hole. The goal is to sample uniformly at random $\Lambda = \Theta(\varepsilon^{-2}\log N)$ points from $\square_u \cap \boldsymbol{q}(\mathbf{D})$. Using Yannakakis algorithm [59], $|\boldsymbol{q}(\mathbf{D})|$ can be computed at the beginning of our approach in $O(N)$ time. From Lemma 2.1, it is known how to sample z tuples in a box efficiently using the procedure SampleRect. However, $\square_u$ is a box with a hole, so we should sample from the difference of two boxes, $\square_u^+ \setminus \square_u^-$. We partition $\square_u$ into $2d$ disjoint boxes using the corners of $\square_u^-$. See Figure 6 for an example in 2 dimensions. Let $\mathcal{R} = \{\rho_1, \ldots, \rho_{2d}\}$ be the set of $2d$ disjoin boxes that partition $\square_u$. First, we decide how many samples (out of the total Λ samples) to draw from each box in $\mathcal{R}$, and then we call the function SampleRect to obtain this number of samples from each box. We compute $C_i = \mathsf{CountRect}(\boldsymbol{q}, \mathbf{D}, \rho_i)$ for every $i \in [2d]$. Based on the values $C_1, \ldots, C_{2d}$, for every $j \in [\Lambda]$, we sample the box in $\mathcal{R}$ that a sample point should be taken from, i.e., a box $\rho_i \in \mathcal{R}$ is selected with probability $\frac{C_i}{\sum_{\ell\in[2d]} C_\ell}$. Let s_i be the number of samples we should get from ρ_i. Then, for every $i \in [2d]$, we run $S_i = \mathsf{SampleRect}(\boldsymbol{q}, \mathbf{D}, \rho_i, s_i)$. We set $\mathcal{N}_u = \bigcup_{i\in[2d]} S_i$. We claim that $\mathcal{N}_u$ is an ε-sample for the points in $\boldsymbol{q}(\mathbf{D}) \cap \square_u$. It is sufficient to show that $\mathcal{N}_u$ is a set of Λ uniform random samples (with replacement) from $\square_u \cap \boldsymbol{q}(\mathbf{D})$. Equivalently, we show that the probability of selecting a specific tuple $t \in \square_u \cap \boldsymbol{q}(\mathbf{D})$ taking one sample is $\frac{1}{|\boldsymbol{q}(\mathbf{D})\cap\square_u|}$. Let t be any tuple that belongs (without loss of generality) in ρ_i. The probability that t is selected taking one sample from $\square_u \cap \boldsymbol{q}(\mathbf{D})$ is $\frac{C_i}{\sum_{\ell\in[2d]C_\ell}} \cdot \frac{1}{C_i} = \frac{1}{\sum_{\ell\in[2d]C_\ell}} = \frac{1}{|\boldsymbol{q}(\mathbf{D})\cap\square_u|}$. Hence $\mathcal{N}_u$ is a set of Λ uniform samples (with replacement) from $\square_u \cap \boldsymbol{q}(\mathbf{D})$ and $\mathcal{N}_u$ is an ε-sample with high probability. After computing $\mathcal{N}_u$ we run the recursive approach of finding the minimum midpoint box that contains $\mathcal{N}_u$ and the inner box $\square_u^-$ and applying $O(d)$ fair splits until we find the first non-trivial fair split. This approach only uses $\mathcal{N}_u$, $\square_u^+$, and $\square_u^-$ to implement, so the implementation is the same as in the standard computational setting. Finally, we bound the running time. The CountRect procedure runs in $O(N)$ time so all numbers $C_1, \ldots, C_{2d}$ are computed in $O(2dN) = O(N)$ time. Then, each $\mathsf{SampleRect}(\boldsymbol{q}, \mathbf{D}, \rho_i, s_i)$ approach runs in $O(N + C_i \log N)$. Hence, the set $\mathcal{N}_u$ is computed in $O(N + \varepsilon^{-2}\log^2 N)$ time. After computing $\mathcal{N}_u$, the next steps of the randomized centroid shrink are executed in $O(\varepsilon^{-2}\log^2 N)$ time as shown in Lemma 4.2. Overall, the randomized centroid shrink in the relational setting runs in $O(N + \varepsilon^{-2}\log^2 N)$ time.

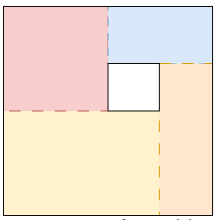


Fig. 6. Partition of a 2-d box with a hole into 4 boxes.

A.4 Count and representative point in a box with a hole

We assume that $\square_u$ is a box with a hole. We construct the partition $\mathsf{R} = \{\rho_1, \ldots, \rho_{2d}\}$ as we did in the randomized centroid shrink. For each $\rho_i \in \mathsf{R}$, we compute $C_i = \mathsf{CountRect}(\boldsymbol{q}, \mathbf{D}, \rho_i)$ and we set $u.c = \sum_{\ell\in[2d]} C_\ell$. Similarly, we compute a representative $u.rep$ by executing $\mathsf{ReprRect}(\boldsymbol{q}, \mathbf{D}, \rho_i)$ until we find a non-empty box (if any) in set R. Both $u.c$ and $u.rep$ for a node u are computed in $O(N)$ time using Lemma 2.1 and the fact that $2d = O(1)$.

A.5 Missing proofs from Section 4.4

Proof of Lemma 4.3. If all nodes of $\mathcal{T}$ were active then we would get a set of canonical nodes $\mathcal{U}'(x,r)$ such that $B(x,r) \cap \boldsymbol{q}(\mathbf{D}) \subseteq \bigcup_{u\in\mathcal{U}'(x,r)}(\square_u \cap \boldsymbol{q}(\mathbf{D})) \subseteq B(x,(1+\varepsilon)r) \cap \boldsymbol{q}(\mathbf{D})$. Since there are inactive nodes, the query procedure on $\mathcal{T}$ skips branches of the tree that are already inactive. Hence $\mathcal{U}(x,r) \subseteq \mathcal{U}'(x,r)$. Let p be an inactive point, i.e., $p \in \boldsymbol{q}(\mathbf{D}) \setminus Q'$. Point p will remain inactive because our procedure only makes new nodes inactive (it does not make any node active).

Then, assume an active point $p \in Q'$ such that $\phi(p,x) > (1+\varepsilon)r$. By definition, there is no node $u \in \mathcal{U}(x,r)$ such that $p \in \square_u$, so p will remain active. Finally, let $p \in Q'$ be an active point such that $\phi(p,x) < r$. Then there would be a node $u' \in \mathcal{U}'(x,r)$ such that $p \in \square_{u'}$. Since p is active there is no node $v \in \mathcal{T}$ such that $v.a = 0$ and $p \in \square_v$. Node u' becomes inactive, so p will be inactive. We conclude that $B(x,r) \cap Q' \subseteq \bigcup_{u \in \mathcal{U}(x,r)} (\square_u \cap Q') \subseteq B(x,(1+\varepsilon)r) \cap Q'$.

Then we show that the values $u.rep$ and $u.c$ are correct after the update operation. This is shown by induction. Let u be a node and v, w be its children. If both $v.a = w.a$ then our algorithm correctly sets $u.a = 0$ because $\square_u = \square_v \cup \square_w$. If $v.a = 1$, $w.a = 0$ and $v.rep, v.c$ are correct then our algorithm correctly sets $u.rep = v.rep$ and $u.c = v.c$ since $\square_u \cap Q' = \square_v \cap Q'$. If $v.a = 0$ and $w.a = 1$, we argue equivalently. Finally, if $v.a = w.a = 1$, then if the counts and representative points in v, w are correct then indeed $v.rep \in \square_u \cap Q'$, and by definition $|\square_u \cap Q'| = |\square_v \cap Q'| + |\square_w \cap Q'| = v.c + w.c$.

Every new node is constructed in $O(N + \varepsilon^{-2} \log^2 N)$ time as shown in Lemma 4.2. The oracle runs a query on the RBBD tree to compute the canonical nodes $\mathcal{U}(x,r)$. Since the number of samples we get in order to implement a randomized centroid shrink is $O(\varepsilon^{-2} \log N^{m+4})$, the query procedure in the RBBD tree has the same properties as the query procedure in the BBD tree with probability at least $1 - \frac{1}{N^{m+3}}$ (as shown in Lemma A.2). Hence $|\mathcal{U}(x,r)| = O(\varepsilon^{-d+1} + \log N)$ and $\mathcal{U}(x,r)$ is computed in $O(\varepsilon^{-d+1} + \log N)$ with probability at least $1 - \frac{1}{N^{m+3}}$. Thus, the running time is $O((\varepsilon^{-d+1} + \log N)((N + \varepsilon^{-2} \log^2 N)))$ with probability at least $1 - \frac{1}{N^{m+3}}$. The same holds for the proofs of the next lemmas in this section. □

Proof of Lemma 4.4. From the proof of Lemma 4.3, we have that the query procedure computes a set of nodes $\mathcal{U}(x,r)$ such that $B(x,r) \cap Q' \subseteq \bigcup_{u \in \mathcal{U}(x,r)} (\square_u \cap Q') \subseteq B(x,(1+\varepsilon)r) \cap Q'$. Since the counters on the active (constructed) nodes are correct, the lemma follows. □

Proof of Lemma 4.5. If $Q' = \emptyset$ then indeed it must be the case that $\texttt{root}.a = 0$. Otherwise, there would be a path of active nodes from the root of $\mathcal{T}$ to a node u (without constructed children) where $\square_u \cap \boldsymbol{q}(\mathbf{D}) \neq \emptyset$. If $Q' \neq \emptyset$, and given that the representative points in each node are correct (Lemma 4.3), we have that the returned point $\texttt{root}.rep$ is a point from Q'. In order to access $\texttt{root}.a$ and $\texttt{root}.rep$ we spend $O(1)$ time. □

Proof of Lemma 4.6. Let U be a subset of the nodes in $\mathcal{T}$ constructed as follows: a node u belongs in U if and only if i) $u \in \mathcal{T}$, ii) there is no inactive node from the root to u (including the roor and u), and iii) the children of u have not been constructed in $\mathcal{T}$ or u is a leaf node. Notice that the ReportRect$(\boldsymbol{q}, \mathbf{D}, \square_u)$ procedure is executed exactly for every $u \in U$. By definition of the active points Q' we have that $Q' = \bigcup_{u \in U} (\square_u \cap \boldsymbol{q}(\mathbf{D}))$. Indeed, for each point $p \in Q'$ there is no inactive node $w \in \mathcal{T}$ such that $p \in \square_w$. Hence the procedure ReportRect$(\boldsymbol{q}, \mathbf{D}, \square_u)$ for every $u \in U$ returns exactly Q'. For the running time, we note that $|U| \leq |\mathcal{T}|$ and each procedure ReportRect$(\boldsymbol{q}, \mathbf{D}, \square_u)$ runs in $O(N + |\square_u \cap \boldsymbol{q}(\mathbf{D})|)$ time. Hence the total running time is $O(|\mathcal{T}| \cdot N + |Q'|)$. □

Proof of Lemma 4.7. Our goal is to show that in each iteration, a point $p \in Q'$ is selected with probability $\frac{1}{|Q'|}$. Let $p \in Q'$ be a point stored in a leaf node u of a complete RBBD tree (such a node may not have been constructed in $\mathcal{T}$ yet). Recall that a point p belongs in Q' if and only if there is no inactive node in $\mathcal{T}$ from the root to u. Assume that v is the last node in the path from $\texttt{root}$ to u that was constructed in $\mathcal{T}$. Let $\texttt{root} \to v_1 \to \ldots \to v_\kappa \to v$. Let $w_1, \ldots, w_\kappa$ be the siblings of nodes $v_1, \ldots, v_\kappa$, respectively. From Section 4.3, we know that a point in a cell $\square_v$ is sampled with probability $\frac{1}{|\square_v \cap \boldsymbol{q}(\mathbf{D})|}$. The probability that p is selected is $\prod_{j \in [\kappa]} \frac{v_j.c}{v_j.c + w_j.c} \cdot \frac{1}{|\square_v \cap \boldsymbol{q}(\mathbf{D})|}$. Notice that v is active and no children of v have been constructed in the tree, so $\square_v \cap \boldsymbol{q}(\mathbf{D}) = \square_v \cap Q'$. Furthermore, by the correctness of the counts, $v_1.c + w_1.c = |Q'|$ and $v.c = |\square_v \cap Q'|$. Moreover, $v_j.c = v_{j+1}.c + w_{j+1}.c$, for every $j \in [\kappa]$. We conclude that the probability of selecting $p \in Q'$ is $\frac{1}{|Q'|}$.

For every sample, we spend $O(\log N)$ time to get a node v of $\mathcal{T}$ which is either a leaf node or its children have not been constructed, because the height of $\mathcal{T}$ is $O(\log N)$ with high probability. After we find v, the sampling procedure in $\square_v \cap Q' = \square_v \cap \boldsymbol{q}(\mathbf{D})$ is executed in $O(N)$ time as shown in Lemma 2.1. Since we get z samples, the total running time is $O(zN)$. □

B Missing proofs from Section 5

Proof of Theorem 5.1. We notice that for any pair $t, p \in \boldsymbol{q}(\mathbf{D})$, $||t-p||_\infty = \max_{j\in[d]} |\pi_{A_j}(t) - \pi_{A_j}(p)|$. Hence, the ℓ_∞ distance between two points can be found as the absolute difference between two of the input tuples' coordinates. Using this observation, we construct a set V of $O(N)$ points in $\mathbb{R}^1$ such that all pairwise ℓ_∞ distances in $\boldsymbol{q}(\mathbf{D})$ are included in the pairwise distances of points in V. Initially, $V = \emptyset$. For every $A_j \in \mathbf{A}$, and every $R_i \in \mathbf{R}$ such that $A_j \in \mathbf{A}_i$, we add in V the projections of tuples in $\mathcal{R}_i$ with respect to A_j, i.e., $V = V \cup \{\pi_{A_j}(R_i)\}$. After considering all attributes and relations, notice that for every ℓ_∞ pairwise distance between two tuples in $\mathbf{D}$, there exist two points in V with the same distance. After we constructed V, we run a binary search on the pairwise distances of the points in V. In order to do it, we use the result from [52] to return the z-th smallest pairwise distance of a set V of points in $\mathbb{R}^1$ in $O(|V| \cdot \log |V|) = O(N \log N)$ time. Hence, we run a binary search requesting the z-th smallest distance for the selected $O(\log N)$ values of z in the range $[0, 1, \ldots, |V \times V|]$. Let r' be the z-th smallest distance returned by [52]. We set $r = \sqrt{d} \cdot r'$ and we run the main algorithm described in Section 5. In the end, we return the last non-empty set $\mathcal{S}$ returned by the binary search.

Lemma B.1. *Using the binary search, the main algorithm for the relational k-center problem returns a set $\mathcal{S}$ of k points such that $\boldsymbol{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq 2\sqrt{d}(1+\varepsilon)\boldsymbol{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$.*

Proof. For any pair of points $t, p \in \mathbb{R}^d$, $||t-p||_\infty \leq ||t-p||_2 \leq \sqrt{d}||t-p||_\infty$. Let $r^* = \boldsymbol{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$ and let t^*, p^* be two tuples in $\boldsymbol{q}(\mathbf{D})$ such that that $||t^* - p^*||_2 = r^*$. Let $\hat{r} = ||t^* - p^*||_\infty$. By the definition of V there will be two points in V with distance exactly $\hat{r}$. By definition, we have $\hat{r} \leq r^* \leq \sqrt{d}\hat{r}$. For every $r' \geq \hat{r}$, we have $r = \sqrt{d}r' \geq \sqrt{d}\hat{r} \geq r^*$. Hence, from the analysis of Section 5, the main algorithm returns a non-empty set $\mathcal{S}'$ of (at most) k points such that $\mathbf{u}_{\mathcal{S}'}(\boldsymbol{q}(\mathbf{D})) \leq 2(1+\varepsilon)r$. Let $r' \leq \hat{r}$ be the distance in the last valid iteration of the binary search where a set $\mathcal{S} \neq \emptyset$ was returned. We have $r = \sqrt{d}r'$. In this case, after at most k iterations, $\mathcal{T}.\mathrm{rep}() = \emptyset$ (equivalently, all points in $\boldsymbol{q}(\mathbf{D})$ are inactive). Since in each iteration we make all points within distance $2r$ (and some points within distance at most $2(1+\varepsilon)r$) inactive around every $p_i \in \mathcal{S}$, it means that all points in $\boldsymbol{q}(\mathbf{D})$ lie within distance $2(1+\varepsilon)r$ from $\mathcal{S}$. Hence, we have $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq 2(1+\varepsilon)r = 2\sqrt{d}(1+\varepsilon)r' \leq 2\sqrt{d}(1+\varepsilon)\hat{r} \leq 2\sqrt{d}(1+\varepsilon)r^*$. The lemma follows. □

Finally, as an estimation of $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D}))$, we set $r_{\mathcal{S}} = 2(1+\varepsilon)r$, where r is the number that was given as input to the main algorithm that returned the final set $\mathcal{S}$. Notice that $\mathbf{u}_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq 2\sqrt{d}(1+\varepsilon)\mathbf{u}_{\mathrm{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$.

The running time of the algorithm using the binary search over the ℓ_∞ distances is only larger by a $\log N$ factor from the running time of the main algorithm. Indeed, the binary search explores $O(\log N)$ distances, and it spends $O(N \log N)$ time to compute each explored distance using the algorithm from [52]. The overall runtime is $O\left(N \log^2 N + k\left(\varepsilon^{-d+1} + \log N\right)\left(N + \varepsilon^{-2}\log^2 N\right)\log N\right) = O(k \cdot N \cdot \log^2 N + k \cdot \log^4 N) = \tilde{O}\left(k \cdot N\right)$, with probability at least $1 - \frac{1}{N}$. □

C Missing proofs from Section 6

Proof of Lemma 6.1. From Lemma 4.7, we note that the set X_i is a set of $\tilde{O}(k)$ samples selected from Q'_i uniformly at random. For every $j \in \{0, 1, 2, \ldots, M, \infty\}$, we execute $\mathcal{T}'.\mathrm{inactive}(x, b_j)$ over every $x \in X_i$. Let $p \in Q'_i$ be a point that was active at the moment that $c_{i,j}$ was defined

and became inactive after applying $\mathcal{T}'.\text{inactive}(x, b_j)$ for every $x \in X_i$. By the correctness of the $\mathcal{T}'.\text{inactive}(x, b_j)$ oracle (Lemma 4.3) $p \notin \bigcup_{\ell<j} Q'_{i,\ell}$ and $p \in Q'_{i,j}$. Furthermore, $p \notin \bigcup_{\ell>j} Q'_{i,\ell}$ because a point becomes inactive once. Hence for every pair $j_1, j_2 \in [M] \cup \{0, \infty\}$, $Q'_{i,j_1} \cap Q'_{i,j_2} = \emptyset$. Furthermore, when $j = \infty$, all remaining active points, by definition, will become inactive. Hence $\bigcup_{j\in[M]\cup\{0,\infty\}} Q'_{i,j} = Q'_i$. So property (1) holds.

Next, if we fix an index $j \in [M] \cup \{0, \infty\}$, for every $x \in X_i$, we execute $c_{i,j} = c_{i,j} + \mathcal{T}'.\text{count}(x, b_j)$ and then $\mathcal{T}'.\text{inactive}(x, b_j)$. By the proof of Lemma 4.3 and Lemma 4.4, the $\mathcal{T}'.\text{count}(x, b_j)$ oracle returns exactly the number of points in Q'_i that become inactive when we execute $\mathcal{T}'.\text{inactive}(x, b_j)$. Hence, it also holds that $c_{i,j} = |Q'_{i,j}|$, and property (3) holds.

Finally, we show property (2). Let $j = 0$, and let $x_1 \in X_i$ be the first sample we process. We execute $\mathcal{T}'.\text{inactive}(x_1, b_0)$. By Lemma 4.3, this oracle makes all points in Q'_i within distance b_0 from x_1 inactive (it might also make some points within distance $(1+\varepsilon)b_0$ inactive). Similarly, for any other $x_s \in X_i$, the oracle $\mathcal{T}'.\text{inactive}(x_s, b_0)$ makes inactive all currently active points within distance b_0 from x_s (it might also make some points within distance $(1+\varepsilon)b_0$ inactive). Hence, by the end of the process for $j = 0$, all points in $Q'_{i,0}$ within distance b_0 from a point in X_i become inactive with respect to $\mathcal{T}'$. Some points within distance $(1+\varepsilon)b_0$ from X_i also might have become inactive. Hence $Q'_{i,0}$ satisfies property (2). By strong induction assume that every set $Q'_{i,j}$ for each $j < j^*$ satisfies property (2), where $j^* \in [M] \cup \{\infty\}$. For every $x \in X_i$, the oracle $\mathcal{T}'.\text{inactive}(x, b_{j^*})$ makes inactive all currently active points in Q'_i that are within distance b_{j^*} from x (and might make some points inactive within distance $(1+\varepsilon)b_{j^*}$). Let $Q'_{i,j^*,x}$ be this set of points and hence we have $Q'_{i,j^*} = \bigcup_{x\in X} Q'_{i,j^*,x}$. From property (1), we have that $Q'_{i,j^*} \cap (\bigcup_{j<j^*} Q'_{i,j}) = \emptyset$. By the induction hypothesis the set $\bigcup_{j<j^*} Q'_{i,j}$ contains all points in Q'_i within distance b_{j^*-1} from X_i and might contain some points within distance $(1+\varepsilon)b_{j^*-1}$ from X_i. Hence, Q'_{i,j^*} contains all points with distance greater than $(1+\varepsilon)b_{j^*-1} = (1+\varepsilon)a_{j^*}$ and at most b_{j^*} from X_i, might contain some points within distance at most $(1+\varepsilon)b_{j^*}$ from X_i, and might contain some points with distance greater than $b_{j^*-1} = a_{j^*}$ from X_i. Hence, Q'_{i,j^*} satisfies property (2). □

Proof of Lemma 6.2. We fix the iteration i of the algorithm. For every $p_s \in \text{OPT}(\boldsymbol{q}(\mathbf{D}))$, we place the smallest ball B_s with center p_s that contains $\frac{\tau_i}{20\cdot k\cdot\log|\boldsymbol{q}(\mathbf{D})|}$ points from Q'_i. Since $X_i \subset Q'_i$ and $|X_i| = \Theta(k\log^2|\boldsymbol{q}(\mathbf{D})|)$, we have that there is at least one point of X_i inside every ball B_s, with probability at least $1 - \frac{1}{N^3}$ [39, 40]. Namely, $X_i \cap B_s \neq \emptyset$ for each $s \in [k]$. Let $x_s \in X_i$ denote the point that lies in the ball B_s, for each $s \in [k]$. Let $p \in Q'_i \setminus ((\bigcup_{\ell\in[k]} B_\ell) \cap Q'_i)$ be a point outside of the balls $\bigcup_{\ell\in[k]} B_\ell$. Let p_s be the closest center of $\text{OPT}(\boldsymbol{q}(\mathbf{D}))$ to p, i.e., $\phi(p, \text{OPT}(\boldsymbol{q}(\mathbf{D}))) = \phi(p, p_s)$. Since $x_s \in B_s \cap Q'_i$, notice that $\phi(p, x_s) \leq 2\phi(p, p_s)$, so $p \in Q'_i \setminus Q^{\text{bad}}_i$, i.e., p is a good point. Thus, with probability at least $1 - \frac{1}{N^3}$, all bad points in Q'_i lie inside the balls $B_1, \ldots, B_k$. Each of these balls contains at most $\frac{\tau_i}{20k\log|\boldsymbol{q}(\mathbf{D})|}$ points of Q'_i. It follows that $|Q^{\text{bad}}_i| \leq k \cdot \frac{\tau_i}{20k\log|\boldsymbol{q}(\mathbf{D})|} = \frac{\tau_i}{20\log|\boldsymbol{q}(\mathbf{D})|}$, with probability at least $1 - \frac{1}{N^3}$. □

Proof of Lemma 6.3. Clearly, $|\hat{Q}_i| \geq \tau_i - c_{i,\infty} - M \cdot \frac{\tau_i}{10\log|\boldsymbol{q}(\mathbf{D})|}$, since β_i is the largest index such that $Q'_{i,\beta_i} > \frac{\tau_i}{10\log|\boldsymbol{q}(\mathbf{D})|}$. Notice that for every point $p \in Q'_{i,\infty}$, it holds $\phi(p, X_i) \geq 2L|\boldsymbol{q}(\mathbf{D})|$. However $\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D})) \leq |\boldsymbol{q}(\mathbf{D})|L^2$ so $\phi(p, \text{OPT}(\boldsymbol{q}(\mathbf{D}))) \leq \sqrt{|\boldsymbol{q}(\mathbf{D})|}L$ and hence $p \in Q^{\text{bad}}_i$. Thus, $Q'_{i,\infty} \subseteq Q^{\text{bad}}_i$. Furthermore, from Lemma 6.2 we have $|Q^{\text{bad}}_i| \leq \frac{\tau_i}{20\log|\boldsymbol{q}(\mathbf{D})|}$ with probability at least $1 - \frac{1}{N^3}$. Hence, $|\hat{Q}_i| \geq \tau_i - \frac{\tau_i}{20\log|\boldsymbol{q}(\mathbf{D})|} - (2\log|\boldsymbol{q}(\mathbf{D})| + 3)\frac{\tau_i}{10\log|\boldsymbol{q}(\mathbf{D})|} \geq \frac{\tau_i}{2}$. So, in every iteration, at least half of the remaining points become inactive. We conclude that the algorithm runs for $O(\log|\boldsymbol{q}(\mathbf{D})|) = O(\log N)$ iterations with probability at least $1 - \frac{1}{N^2}$. In each iteration we add in $\mathcal{S}$ the set X_i and $|X_i| = O(k\log^2 N)$, so $|\mathcal{S}| = O(k\log^3 N)$, with probability at least $1 - \frac{1}{N^2}$. □

Proof of Lemma 6.4. We first prove the approximation factor. We fix the iteration i of the algorithm. If $\beta_i > 0$, then we argue as follows. Let $\mathcal{P} = Q'_{i,\beta_i} \setminus Q_i^{\text{bad}}$. For any point $p \in \hat{Q}_i \cap Q_i^{\text{bad}}$ and $q \in \mathcal{P}$, we have $\phi(p, X_i) \leq 2(1+\varepsilon)\phi(q, X_i)$. This inequality follows from the definition of the sets $Q'_{i,j}$ and the $\mathcal{T}'$.inactive$(\cdot,\cdot)$ oracle. Furthermore, $|\mathcal{P}| > \frac{\tau_i}{10\log|\boldsymbol{q}(\mathbf{D})|} - \frac{\tau_i}{20\log|\boldsymbol{q}(\mathbf{D})|} = \frac{\tau_i}{20\log|\boldsymbol{q}(\mathbf{D})|} \geq |\hat{Q}_i \cap Q_i^{\text{bad}}|$, with probability at least $1 - \frac{1}{N^2}$. By definition, every point in $\mathcal{P}$ is a good point with respect to X_i so $\mu_{X_i}(\mathcal{P}) \leq 4\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\mathcal{P})$. Furthermore, since $\mathcal{P} \subseteq \hat{Q}_i$ we have that $\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\mathcal{P}) \leq \mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)$. Thus, $\mu_{X_i}(\hat{Q}_i \cap Q_i^{\text{bad}}) \leq 4(1+\varepsilon)^2\mu_{X_i}(\mathcal{P}) \leq 16(1+\varepsilon)^2\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\mathcal{P}) \leq 16(1+\varepsilon)^2\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)$. So,

$$\begin{aligned}\mu_{X_i}(\hat{Q}_i) &= \mu_{X_i}(\hat{Q}_i \cap Q_i^{\text{bad}}) + \mu_{X_i}(\hat{Q}_i \setminus Q_i^{\text{bad}}) \leq 16(1+\varepsilon)^2\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i) + 4\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)\\ &\leq 20(1+\varepsilon)^2 \cdot \mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i).\end{aligned}$$

If $\beta_i = 0$, then for every point $q \in \hat{Q}_i$ we have $\phi(q, X_i) \leq (1+\varepsilon)\frac{L}{4|\boldsymbol{q}(\mathbf{D})|}$. Thus, for $\varepsilon \leq 0.1$,

$$\mu_{X_i}(\hat{Q}_i) = \sum_{q\in\hat{Q}_i} \phi^2(q, X_i) \leq (1+\varepsilon)^2\frac{L^2}{16|\boldsymbol{q}(\mathbf{D})|} < \frac{L^2}{9|\boldsymbol{q}(\mathbf{D})|} \leq \frac{1}{|\boldsymbol{q}(\mathbf{D})|}\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D})).$$

In any case, we have $\mu_{X_i}(\hat{Q}_i) \leq \max\{20(1+\varepsilon)^2 \cdot \mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i), \frac{1}{|\boldsymbol{q}(\mathbf{D})|}\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))\}$. By definition, recall that $\hat{Q}_{i_1} \cap \hat{Q}_{i_2} = \emptyset$ for every $i_1 \neq i_2$. Overall, $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq 21(1+\varepsilon)^2\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, with probability at least $1 - \frac{1}{N}$.

Finally, for every point $p \in \hat{Q}_i$ such that $p \in Q'_{i,j'}$ for an index $j' \leq \beta_i$ we charge $((1+\varepsilon)b_{j'})^2$ in $r_{\mathcal{S}}$. By definition, if $j' > 0$, $\phi(p, X_i) \leq (1+\varepsilon)b_{j'} \leq 2(1+\varepsilon)\phi(p, X_i)$. Hence the k-means cost of p with respect to X_i is charged at most $4(1+\varepsilon)^2\phi^2(p, X_i)$ and at least $\phi^2(p, X_i)$ in $r_{\mathcal{S}}$. If $j' = 0$, then by the bound of $\mu_{X_i}(\hat{Q}_i)$ when $\beta_i = 0$, we charge a value which is at most $\frac{1}{|\boldsymbol{q}(\mathbf{D})|}\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$ and at least $\mu_{\mathcal{S}}(\hat{Q})$ in $r_{\mathcal{S}}$. Generally, for all points in $\hat{Q}_i$, we charge a value at most $\sum_{p\in\hat{Q}_i} 4(1+\varepsilon)^2\phi^2(p, X_i) \leq 84(1+\varepsilon)^4\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\hat{Q}_i)$ and at least $\sum_{p\in\hat{Q}_i}\phi(p, X_i) \geq \mu_{\mathcal{S}}(\hat{Q}_i)$ in $r_{\mathcal{S}}$. Overall, $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq 84(1+\varepsilon)^4\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$. By setting $\varepsilon \leftarrow \varepsilon/400$, we get $\mu_{\mathcal{S}}(\boldsymbol{q}(\mathbf{D})) \leq r_{\mathcal{S}} \leq (84+\varepsilon)\mu_{\text{OPT}(\boldsymbol{q}(\mathbf{D}))}(\boldsymbol{q}(\mathbf{D}))$, wuit probability at least $1 - \frac{1}{N}$.

Next, we proceed with the running time. First, we run the k-center algorithm from Theorem 5.2 in $O\left(kN\log^2 N + k\log^4 N\right)$ time with probability at least $1 - \frac{1}{10N}$. Then the algorithm runs for $O(\log N)$ iterations with probability at least $\frac{1}{N^2}$. For every iteration i, we get the set X_i in $O(kN\log^2 N)$ time, with probability at least $1 - \frac{1}{N^{m+3}}$, using the result of Lemma 4.7. There are $O(\log N)$ different values of $\ell \in [M] \cup \{0, \infty\}$, and for each ℓ we compute $c_{i,\ell}$ in $O(kN\log^3 N + k\log^5 N)$ time with probability at least $1 - \frac{O(\log^2 N)}{N^3}$, executing the $\mathcal{T}'$.inactive() oracles $O(k\log^2 N)$ times (recall from Lemma 4.4 that $\mathcal{T}'$.inactive() runs in $O(N\log N + \log^3 N)$ time). Fixing an iteration i, all values $c_{i,\ell}$ are computed in $O(kN\log^4 N + k\log^6 N)$ time with probability at least $1 - \frac{O(\log^3 N)}{N^3} \geq 1 - \frac{1}{N^2}$. Overall, during the $O(\log N)$ iterations (with probability at least $1 - \frac{1}{N^2}$), the running time to compute all counts is $O(kN\log^5 N + k\log^7 N)$ with probability at least $1 - \frac{2}{N^2}$. Similarly, in the last iteration, denoted by i', notice that $|\mathcal{T}| = O(k\log^5 N)$ with probability at least $1 - \frac{2}{N^2}$. From Lemma 4.6, and the fact that $\tau_{i'} = O(k\log^2 N)$, the oracle $\mathcal{T}$.report() runs in $O(kN\log^5 N)$ time, with probability at least $1 - \frac{3}{N^2}$ Overall, our algorithm runs in $O(kN\log^5 N + k\log^7 N)$ time, with probability at least $1 - \frac{1}{N}$. □

Received December 2025; accepted February 2025