# Computing A Well-Representative Summary of Conjunctive Query Results*

PANKAJ K. AGARWAL, Department of Computer Science, Duke University, USA
ARYAN ESMAILPOUR, Department of Computer Science, University of Illinois Chicago, USA
XIAO HU, Cheriton School of Computer Science, University of Waterloo, Canada
STAVROS SINTOS, Department of Computer Science, University of Illinois Chicago, USA
JUN YANG, Department of Computer Science, Duke University, USA

Data summarization is a powerful approach to deal with large-scale data analytics, which has wide applications in web search, recommendation systems, approximate query processing, etc. It computes a small, compact *summary* that preserves vital properties of the original data. In this paper, we study the data summarization problem of *conjunctive query* results, i.e., computing a $k$-size subset of a conjunctive query output, for any given $k > 0$, that optimizes a certain objective. More specifically, we are interested in two commonly studied objectives: *cohesion*, which measures the maximum distance between a tuple in the query result tuples and its closest tuple in the summary ($k$-center clustering); and *diversity*, which measures the pairwise distances between the summary items. A simple approach that computes the entire query output and then applies existing algorithms on top of these materialized tuples suffers from high computational complexity because the query output can be large, e.g., for a relational database of $N$ tuples, the number of result tuples can be $N^{O(1)}$. We propose $O(1)$-approximation algorithms that compute well-representative summaries of size $k$ in time $\tilde{O}(N \cdot k^{O(1)})$, or even $\tilde{O}(N + k^{O(1)})$ in some cases,[1] without computing all result tuples. We also propose the first efficient $(2+\varepsilon)$-approximation algorithm for the $k$-center clustering problem over relational data. Our main idea is to formulate a few oracles that enable us to access specific query result tuples with certain properties, to show how these oracles can be implemented efficiently, and to compute desired summaries with few invocations of these oracles.

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

Additional Key Words and Phrases: relational data, conjunctive queries, diversity, coresets, oracles

**ACM Reference Format:**
Pankaj K. Agarwal, Aryan Esmailpour, Xiao Hu, Stavros Sintos, and Jun Yang. 2024. Computing A Well-Representative Summary of Conjunctive Query Results. *Proc. ACM Manag. Data* 2, 5 (PODS), Article 217 (November 2024), 27 pages. https://doi.org/10.1145/3695835

*This work was supported by NSF grants CCF-2223870, IIS-2402823, IIS-2348919, a US-Israel Binational Science Foundation Grant 2022131, and NSERC Discovery Grant.

[1]We use $\tilde{O}$ notation to hide $\log N$ or $\log^2 N$ factors.

Authors' Contact Information: Pankaj K. Agarwal, Department of Computer Science, Duke University, Durham, USA, pankaj@cs.duke.edu; Aryan Esmailpour, Department of Computer Science, University of Illinois Chicago, Chicago, USA, aesmai2@uic.edu; Xiao Hu, Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, xiaohu@uwaterloo.ca; Stavros Sintos, Department of Computer Science, University of Illinois Chicago, Chicago, USA, stavros@uic.edu; Jun Yang, Department of Computer Science, Duke University, Durham, USA, junyang@cs.duke.edu.


ACM 2836-6573/2024/11-ART217
https://doi.org/10.1145/3695835

## 1 Introduction

Data summarization is a potent strategy in large-scale data analytics, offering a means to compute a compact yet comprehensive dataset that preserves vital properties of the original data. A multitude of data summarization techniques have been developed for diverse applications, encompassing sampling [12, 23, 40, 53], histograms [47, 48], wavelet-based synopses [29, 39], sketching [5, 27, 28], coresets [6], and more [29]. The objectives on the *quality* of the data summaries strongly depend on the downstream application scenarios. Viewing query result tuples as points in a multi-dimensional space, two common objectives are *cohesion*, which measures the maximum distance between a result tuple and its closest tuple in the summary, and *diversity*, which measures the minimum or average pairwise distance between tuples in the summary.

Most past data-summarization algorithms assume they are directly given the data to summarize. However, one often desires to summarize the output of a query, especially when the query output is large. In such applications, the simple approach of computing all query result tuples and then applying a known algorithm is ineffective because of its high computational cost when the query output size is large (which is why we want to compute a summary in the first place). There is some recent work on computing summaries of query results [3, 11, 27, 57], but this line of work has focused on simple queries, such as range queries, where data resides in a single table. In real-world scenarios, more than 70% of the current data sets are *relational* [1], where data is stored in multiple tables, and the desired data is obtained by performing conjunctive queries – the combination of select, project, and join queries – on these tables. Note that the size of the output of a conjunctive query can be polynomially larger than the size of the tables. Furthermore, user queries may have very different local selection predicates. Hence, the challenging question is constructing a well-representative summary for the output of a given conjunctive query without computing and materializing its entire output. While there are some recent results on clustering in relational data [31, 54, 55], there is no result on efficiently computing a well-representative summary in relational data. Hence, in this paper, we take on this challenging question and investigate how to construct well-representative summaries for conjunctive query results efficiently.

### 1.1 Problem Definition

**Conjunctive query.** Let $\mathbf{R}$ denote a database schema and $\mathbf{A}$ the set of all attributes. $\mathbf{R}$ consists of a set of $m$ relations $\{R_1, \ldots, R_m\}$, where each relation $R_i$ has a subset of attributes $\mathbf{A}_i \subseteq \mathbf{A}$, satisfying $\bigcup_{i\in[m]} \mathbf{A}_i = \mathbf{A}$. Let $\mathtt{dom}(A)$ denote the domain of attribute $A \in \mathbf{A}$. For the simplicity of exposition, we assume that all attributes have the domain $\mathbb{R}$ of reals, though our results can be generalized to other domains. A database instance $\mathbf{I}$ consists of the set $\{R_i^{\mathbf{I}}\}$ of relational instances, where each $R_i^{\mathbf{I}}$ is a set of tuples over the domain $\mathbb{R}^{|\mathbf{A}_i|}$. Let $t \in \mathbb{R}^{|\mathbf{A}_i|}$ denote a tuple in $R_i^{\mathbf{I}}$: for each attribute $A \in \mathbf{A}_i$, we use $t.A$ to denote $t$'s value for attribute $A$; for each subset of attributes $X \subseteq \mathbf{A}_i$, we use $t.X$ to denote $t$'s projection onto attributes in $X$. When the context is clear, we will drop the superscript $\mathbf{I}$ and simply refer to relation instance $R_i^{\mathbf{I}}$ as relation $R_i$. We consider *conjunctive queries* (CQs):

$$Q := \pi_{\mathbf{y}} \left( (\sigma_{\mathbf{p}_1} R_1) \bowtie (\sigma_{\mathbf{p}_2} R_2) \bowtie \cdots \bowtie (\sigma_{\mathbf{p}_m} R_m) \right), \tag{1}$$

where $\mathbf{y} \subseteq \mathbf{A}$ defines the set of output attributes, each $\mathbf{p}_i$ is a Boolean predicate over $\mathbf{A}_i$, and $\pi, \sigma, \bowtie$ are relational projection, selection, and natural join operators. By renaming the attributes, we also allow self-joins in $Q$, i.e., the same relation can be joined multiple times in $Q$. Let $d = |\mathbf{y}|$. The output of $Q$ over database instance $\mathbf{I}$ is defined as

$$Q(\mathbf{I}) = \left\{ t' \in \mathbb{R}^d \;\middle|\; \exists t \in \mathbb{R}^{|\mathbf{A}|} : t.\mathbf{y} = t' \wedge (\forall i \in [m] : t.\mathbf{A}_i \in R_i^{\mathbf{I}} \wedge \mathbf{p}_i(t.\mathbf{A}_i)) \right\}.$$

Query $Q$ filters relation $R_i$ by the Boolean predicate $\mathbf{p}_i$, joins filtered tuples (one from each relation) sharing the same values on common attributes, and then projects the resulting tuples onto $\mathbf{y}$. Each tuple in $Q(\mathbf{I})$ is essentially a point in $\mathbb{R}^d$.

We say a CQ is *full* if $\mathbf{y} = \mathbf{A}$ (i.e., $\pi_{\mathbf{y}}$ is identity function). A full CQ is also called a *join query*. In this paper we mainly focus on acyclic CQs. Recall that a CQ is acyclic [16, 37] if there exists a tree $\mathcal{T}$, called a *join tree* of $Q$, where 1) the nodes of $\mathcal{T}$ are $R_1, \ldots, R_m$; 2) for each attribute $A \in \mathbf{A}$, the set of nodes whose attributes contain $A$ form an edge-connected subtree of $\mathcal{T}$. For simplicity, in all cases we assume all dangling tuples are removed from $\mathbf{I}$.[2]

**Well-representative summaries for CQs.** Given a CQ $Q$ of the form (1), a database $\mathbf{I}$, and a positive integer $k$, we refer to a subset $S \subseteq Q(\mathbf{I})$ of $k$ distinct tuples from $Q(\mathbf{I})$ as a *$k$-summary* of $Q(\mathbf{I})$. We use two different (though related) objectives to measure the *quality* of a summary. Let $\phi : \mathbb{R}^{|\mathbf{y}|} \times \mathbb{R}^{|\mathbf{y}|} \to \mathbb{R}_{\geq 0}$ be a distance function.

- The *cohesion* of $S$ is defined using both $Q(\mathbf{I})$ and $S$:

$$\rho(S, Q(\mathbf{I})) = \max_{t_2 \in Q(\mathbf{I})} \min_{t_1 \in S} \phi(t_1, t_2). \tag{2}$$

  Intuitively, cohesiveness ensures that every result tuple is close to some tuple in the summary. For any $k > 0$, let $\rho_k(Q(\mathbf{I}))$ denote the optimal cohesion of any $k$-summary of $Q(\mathbf{I})$, i.e., $\rho_k(Q(\mathbf{I})) = \min_{S \subseteq Q(\mathbf{I}), |S|=k} \rho(S, Q(\mathbf{I}))$. A $k$-summary $S^*$ is called an *optimally cohesive* $k$-summary if $\rho(S^*, Q(\mathbf{I})) = \rho_k(Q(\mathbf{I}))$. For a parameter $\alpha > 1$, a $k$-summary $S$ is *$\alpha$-cohesive* if $\rho(S) \leq \alpha \cdot \rho_k(Q(\mathbf{I}))$. We note that the definition of cohesion is identical to the definition of the cost in the $k$-center clustering problem. Hence, an $\alpha$-cohesive $k$-summary $S$ is also an $\alpha$-approximation of $k$-center clustering of $Q(\mathbf{I})$ (relational $k$-center problem).
- The *diversity* of a set $S \subset \mathbb{R}^{|\mathbf{y}|}$, has two variants:

$$\text{Sum-diversity:} \quad \delta(S) = \frac{1}{2} \sum_{x,y \in S \times S} \phi(x, y); \tag{3}$$

$$\text{Min-diversity:} \quad \delta(S) = \min_{x,y \in S \times S : x \neq y} \phi(x, y). \tag{4}$$

  Diversity seeks to ensure that summary tuples are far away from each other. For any $k > 0$, let $\mu_k(Q(\mathbf{I}))$ denote the optimal diversity of any $k$-summary of $Q(\mathbf{I})$, i.e., $\mu_k(Q(\mathbf{I})) = \min_{S \subseteq Q(\mathbf{I}), |S|=k} \delta(S)$. A $k$-summary $S^*$ is called an *optimally diverse* $k$-summary if $\delta(S^*) = \mu_k(Q(\mathbf{I}))$. Given a parameter $\alpha \in (0, 1)$, a $k$-summary $S$ is *$\alpha$-diverse* if $\delta(S) \geq \alpha \cdot \mu_k(Q(\mathbf{I}))$.

Our goal is to compute efficiently well-representative summaries for an input CQ $Q$ and database instance $\mathbf{I}$. We are interested in the *data complexity*: i.e., the query size $m$ is a constant, and the complexity of our algorithms is measured by the input size $N = \sum_{i \in [m]} |R_i^{\mathbf{I}}|$, and the output size $k$.

### 1.2 Related Work

**Summaries of a given data set.** Computing various summaries for a set $P$ of data points has been extensively studied in the literature under different objectives. A summary that maximizes the sum-diversity (3) is known as *remote-clique* or *max-sum* problem [59], and that maximizes the min-diversity (4) is known as *remote-edge* or *max-min* problem [64], which is NP-hard.[3] For the max-sum problem, there are efficient $\frac{1}{2}$-diverse algorithms that work for any distance function [17,

[2] Dangling tuples are those not participating in any result of the underlying join query, which can be done within $O(N)$ time for acyclic join queries. See Appendix A.

[3] The NP-hardness of max-min problem implies the NP-hardness of computing an optimally min-diverse summary. See Appendix A.

18, 59]. Better algorithms are proposed for the Euclidean distance, either with better approximation factors [19–21] or with better time complexity [11]. For the max-min problem, Tamir [64] showed that a greedy algorithm returns a $\frac{1}{2}$-approximation for any metric. A faster algorithm [11] is also known for the max-min problem the under Euclidean metric. Computing a summary of $P$ that minimize cohesion,[4] known as the *k-center* problem, is NP-hard.[5] The well-known Gonzalez's algorithm [41] returns a 2-cohesive $k$-summary of a set $P$ in $O(k \cdot |P|)$ time under any metric. Its running time was improved to $O(|P| \log k)$ in [38] (see also [44]). All the problems also have been studied under fairness constraints [2, 4, 18, 49, 51, 52, 56].

**Summaries of selection results.** There is some work on computing the summaries of range query outputs of a set of points $P$. A near-linear-size index exists [3, 57] that, given any query rectangle $\psi$, computes a $(2+\varepsilon)$-cohesive summary $k$-summary of $P \cap \psi$ in $O(k \cdot \text{polylog}(|P|))$ time. Subsequently, a similar index was shown to generate $O(1)$-diverse summaries for $P \cap \psi$. Additionally, summaries have been extensively studied for statistical queries, such as sampling, sketching, frequent moments, and embedding. We refer interested readers to [30] for details.

**Summaries of join results.** Computing summaries of the results for a join query is useful to answer analytical queries while providing provably accuracy guarantees, such as sampling [24, 26, 34, 50, 66, 67], factorization [58] and witness [46]. Recently, the *k-means and k-median clustering* problems over relational data have been studied [31, 36, 55]. The *coreset* for empirical risk minimization problems over relational data [25] has also been considered. However, the time complexity of their algorithm for constructing an $\varepsilon$-coreset depends on the *diameter* of the query results. Merkl et al. [54] studied the hardness of diversity problems over relational data under the Hamming metric. Under data complexity, they give an expensive algorithm (computing all query results) to construct an $O(k^m)$ (exact) coreset. Arenas et al. [13], also studied the hardness of diversity problems under the Hamming metric and *ultrametrics*. Furthermore, they propose polynomial time algorithms for some diversity problems under ultrametrics (Euclidean and Hamming metrics are not ultrametrics).

### 1.3 Our Contributions

For a CQ $Q$ with $d$ output attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $k$, we propose several $O(1)$-approximation algorithms that compute cohesive and diverse $k$-summaries for $Q(\mathbf{I})$ in $\tilde{O}(N \cdot k^{O(1)})$ or $\tilde{O}\left(N + k^{O(1)}\right)$ time, under the Euclidean or Hamming metric. We include all the $\log N$ factors in the analyses and theorems in the next sections. All our results for acyclic join queries are shown in Table 1, and the extended results to cyclic join queries and even join-project queries are discussed in Section 6. In the next sections we use the notation $\phi(\cdot,\cdot)$ for the Euclidean distance and $\phi_H(\cdot,\cdot)$ for the Hamming distance. For two tuples $p, q \in \mathbb{R}^d$, let $\phi(p,q) = (\sum_{A_j \in \mathbf{A}} (p.A_j - q.A_j)^2)^{1/2}$ and $\phi_H(p,q) = \sum_{A_j \in \mathbf{A}} \mathbb{1}(p.A_j \neq q.A_j)$, where $\mathbb{1}$ is the indicator function.

- **Cohesive summary (Section 3):**
  - **Euclidean metric (Section 3.1).** We design an algorithm to construct a $(2+\varepsilon)$-cohesive $k$-summary in $\tilde{O}(\min\{k^2 N, k^{\lceil d/2 \rceil + 1}\} + kN\varepsilon^{-d})$ time. This result also leads to the first efficient $(2+\varepsilon)$-approximation algorithm for the relational $k$-center clustering problem. The best previously known algorithm for the relational $k$-centering problem has either an additive approximation factor that depends on the diameter of $Q(\mathbf{I})$ or a relative approximation factor that depends exponentially on $d$ [25]. In order to derive this result, we first construct two $O(1)$-cohesive summaries: (i) a tree-based algorithm that runs in $O(Nk^2)$ time using $O(N+k^2)$ space,

[4]For a set $P$ of data points, the cohesion for a $k$-summary $S$ is similarly defined as $\rho(S) = \max_{t_2 \in P} \min_{t_1 \in S} \phi(t_1, t_2)$.

[5]The NP-hardness of the k-center problem implies the NP-hardness of computing an optimally cohesive summary. See Appendix A.

| Summaries | Metric | Time | Space | Approximation | Ref. |
|---|---|---|---|---|---|
| Cohesive | Euclidean | $k^2N + kN\varepsilon^{-d}$ | $N + k^2 + k\varepsilon^{-d}$ | $2+\varepsilon$ | §3.1 |
| | | $k^{\lceil d/2\rceil+1} + kN\varepsilon^{-d}$ | $N + k^{\lceil d/2\rceil+1} + k\varepsilon^{-d}$ | $2+\varepsilon$ | §3.1 |
| | Hamming | $Nk^d$ | $N + k^d$ | 2 | §3.2 |
| Min-diverse | Euclidean | $k^2N + kN\varepsilon^{-d}$ | $N + k^2 + k\varepsilon^{-d}$ | $\frac{1}{2}-\varepsilon$ | §4.1 |
| | | $k^{\lceil d/2\rceil+1} + kN\varepsilon^{-d}$ | $N + k^{\lceil d/2\rceil+1} + k\varepsilon^{-d}$ | $\frac{1}{2}-\varepsilon$ | §4.1 |
| | Hamming | $Nk^d$ | $N + k^d$ | $\frac{1}{2}$ | §4.2 |
| Sum-diverse | Euclidean | $(N+k)\varepsilon^{-(d-1)/2}$ | $N + k\varepsilon^{-(d-1)/2}$ | $\frac{1}{2}-\varepsilon$ | §5.1 |
| | Hamming | $Nk^2 + k^3$ | $N + k$ | $1-\frac{2}{k}$ | §5.2 |
| | | $Nk + k^2$ | $N + k$ | $\frac{1}{2}$ | §5.2 |

**Table 1.** Summary of our results for acyclic join queries. For simplicity, we hide the $O(\cdot)$ notation and $\log N$ or $\log^2 N$ factors. $N$ is the input size of the database, $k$ is the size of the summary, $\varepsilon \in (0,1)$ is an error parameter given as input, and $d$ is the number of attributes in the join query.

but whose approximation depends on the number of relations in the join query; (ii) a geometry-based 6-approximation algorithm that runs in $O\left(N + k^{\lceil d/2\rceil+1}\right)$ time using $O\left(N + k^{\lceil d/2\rceil+1}\right)$ space. We then combine the $O(1)$-approximation algorithms with a grid-based construction to derive a $\varepsilon$-coreset (formally defined in Section 3.1.3) of $O(\varepsilon^{-d}k)$ centers such that every result tuple from $\boldsymbol{Q}(\mathbf{I})$ is "near" enough to some center from the coreset. The $\varepsilon$-coreset leads to the construction of a $(2+\varepsilon)$-cohesive $k$-summary in $\tilde{O}\left(\min\{Nk^2, k^{\lceil d/2\rceil+1}\} + kN\varepsilon^{-d}\right)$ time.
  - **Hamming metric (Section 3.2).** Using an iterative approach that implicitly excludes tuples close to the selected tuples in the summary, we present an algorithm that constructs a 2-cohesive $k$-summary in $O\left(N \cdot k^d\right)$ time using $O(N + k^d)$ space.
- **Min-Diverse summary (Section 4)**:
  - **Euclidean metric (Section 4.1).** We first show that any cohesive coreset is also a min-diverse coreset. Hence, using the ideas for constructing cohesive summaries, we design an algorithm to construct a $(\frac{1}{2}-\varepsilon)$-min-diverse $k$-summary in $\tilde{O}\left(\min\left\{Nk^2, k^{\lceil d/2\rceil+1}\right\} + kN\varepsilon^{-d}\right)$ time.
  - **Hamming metric (Section 4.2).** Using ideas for constructing cohesive summaries, we design an algorithm to construct a $\frac{1}{2}$-min-diverse summary in $O\left(Nk^d\right)$ time using $O\left(N + k^d\right)$ space.
- **Sum-Diverse summary (Section 5)**:
  - **Euclidean metric (Section 5.1).** We present a geometric approach using the notion of *$\varepsilon$-net* that can construct an $(\frac{1}{2}-\varepsilon)$-diverse $k$-summary in $\tilde{O}\left((N+k)\varepsilon^{-(d-1)/2}\right)$ time using $O\left(N + k\varepsilon^{-(d-1)/2}\right)$ space.
  - **Hamming metric (Section 5.2).** We propose two algorithms to construct sum-diverse summaries. The first one, a local search algorithm, constructs a $(1-2/k)$-sum-diverse $k$-summary in $\tilde{O}\left(Nk^2 + k^3\right)$ time using $O(N+k)$ space. The second one, a greedy algorithm, constructs a $\frac{1}{2}$-sum-diverse $k$-summary in $\tilde{O}(Nk + k^2)$ time using $O(N+k)$ space.

**Main ideas.** As mentioned, to summarize the output of a CQ, one could compute all result tuples first and then directly apply existing algorithms developed for computing summaries for a given dataset [41, 59, 64]. However, materializing the entire query output is expensive. Instead, our approach is to formulate appropriate *oracles* that enable us to access some specific query

result tuples with certain properties—including nearest neighbor, farthest neighbor, top-$k$, and rectangular lookup—and show how to modify existing algorithms or design new algorithms using as few invocations of these oracles as possible. One key insight is that our computation can be modeled as *ranked enumeration of query answers*. Conceptually, given a CQ $Q$, a set of weight functions defined on attributes, and a database $\mathbf{I}$, ranked enumeration returns $Q(\mathbf{I})$ in ascending (or descending) order with respect to their weights, one at a time, with a bounded delay between consecutive answers. We carefully design weight functions using query and data to apply ranked enumeration in our settings. We also push selections down as far as possible. Specifically, given a set of selection predicates, each on an individual attribute, we simply push the selection predicates down to the base tables first and enumerate all result tuples in $Q(\mathbf{I})$ satisfying the selection predicates with a bounded delay between consecutive answers.

## 2 Relational Oracles

We show some relational oracles that will be commonly used throughout the paper.

**Ranked enumeration.** Let $w_i : \mathbb{R}^{|\mathbf{A}_i|} \to \mathbb{R}$ be a weight function, which takes as input a tuple $t \in R_i$ and outputs a real number. Let $\vec{w} = \langle w_1, w_2, \cdots, w_m \rangle$ be a set of weight functions. For a CQ $Q$, a database $\mathbf{I}$, and a pair of results $t_1, t_2 \in Q(\mathbf{I})$, we say $t_1 \preceq_{\vec{w}} t_2$ if $\sum_{j\in[m]} w_j(t_1.\mathbf{A}_j) \leq \sum_{j\in[m]} w_j(t_2.\mathbf{A}_j)$.

Lemma 2.1 ([33]). *For an acyclic join $Q$, a database $\mathbf{I}$, and a set of weight functions $\vec{w} = \langle w_1, w_2, \cdots, w_m \rangle$, an index of size $O(N)$ can be constructed in $O(N)$ time, such that given any value $k \in \mathbb{Z}^+$, the top-$k$ results of $Q(\mathbf{I})$ can be enumerated in ascending or descending order with respect to $\vec{w}$ within $O(\log N)$ delay using $O(k)$ additional space.*

By exploiting the variety of weight functions supported by ranked enumeration, we are able to define the Euclidean-based oracles and the top-$k$ oracle. For simplicity, let $\bar{\mathbf{A}}_i = \mathbf{A}_i - (\bigcup_{j<i} \mathbf{A}_j)$ be the set of *active attributes* for $R_i$, i.e., the attributes in $R_i$ that do not appear in a relation $R_j$ for $j < i$.

**Euclidean-based oracles.** Let $\theta \in \mathbb{R}^d$ be a tuple. The *nearest (resp. farthest) neighbor oracle* finds a tuple $t \in Q(\mathbf{I})$ that is closest to $\theta$ under the Euclidean metric. For each relation $R_i$ and for a tuple $p \in R_i$, we define $w_i(p) = \sum_{A_j \in \bar{\mathbf{A}}_i} (p.A_j - \theta.A_j)^2$. It is easy to see for any query result $t \in Q(\mathbf{I})$,

$$\sum_{i\in[m]} w_i(t.\mathbf{A}_i) = \sum_{i\in[m]} \sum_{A_j\in\bar{\mathbf{A}}_i} (t.A_j - \theta.A_j)^2 = \sum_{A\in\mathbf{A}} (t.A - \theta.A)^2 = \phi^2(\theta, t),$$

thanks to the *decomposability* of the squared Euclidean distance. The square (and square root) function is increasing for non-negative values, so the squared Euclidean distance preserves the ordering of Euclidean distance.

**Top-$k$ oracle.** Let $u = \langle u_1, u_2, \ldots, u_d \rangle \in \mathbb{R}^d$ be a vector. The *top-$k$* oracle finds the $k$ tuples from $Q(\mathbf{I})$ with the largest inner product with $u$. For each relation $R_i$ and for any $p \in R_i$, we define $w_i(p) = \sum_{A_j \in \bar{\mathbf{A}}_i} \big(p.A_j\big) \cdot u_j$. It is easy to see that for any query result $t \in Q(\mathbf{I})$,

$$\sum_{i\in[m]} w_i(t.\mathbf{A}_i) = \sum_{i\in[m]} \sum_{A_j\in\bar{\mathbf{A}}_i} \big(t.A_j\big) \cdot u_j = \langle t, u \rangle.$$

Lemma 2.2. *Given an acyclic join $Q$, a database instance $\mathbf{I}$ with input size $N$, and a tuple $\theta \in \mathbb{R}^d$, the nearest (resp. farthest) neighbor of $\theta$ in $Q(\mathbf{I})$, under the Euclidean metric, can be computed in $O(N)$ time. Each tuple in $Q(\mathbf{I})$ can be enumerated in ascending or descending order with respect to its*

*distance from $\theta$ within $O(\log N)$ delay. Furthermore, given a vector $u \in \mathbb{R}^d$, the $k$ tuples in $Q(\mathbf{I})$ with the highest inner product with $u$ can be computed in $O(N + k \log N)$ time.*

**Rectangular oracle.** Next, we focus on counting and enumerating the join results that lie in a rectangle. Let $\psi \subseteq \mathbb{R}^d$ be an axis-parallel rectangle, defined as the product of $d$ intervals, i.e., $\psi = \psi_1 \times \ldots \times \psi_d$, where $\psi_j = [a_j, b_j]$ and $a_j, b_j \in \mathbb{R}$. A tuple $t$ lies in $\psi$ if and only if $t.A_j \in \psi_j \Leftrightarrow a_j \leq t.A_j \leq b_j$, for every $A_j \in \mathbf{A}$. Hence, a rectangle $\psi$ defines a predicate $a_j \leq A_j \leq b_j$ for each attribute $A_j \in \mathbf{A}$. Given a rectangle $\psi$, we can find all tuples in $\mathbf{I}$ that pass the predicate in $O(N)$ time and then apply Yannakakis algorithm [65] to count the number of result tuples in $\psi \cap Q(\mathbf{I})$, or apply the index from [15] to enumerate the result tuples in $\psi \cap Q(\mathbf{I})$.

Lemma 2.3. *For an acyclic join $Q$, a database $\mathbf{I}$ of input size $N$, and a rectangle $\psi \in \mathbb{R}^d$, an index of size $O(N)$ can be constructed in $O(N)$ time such that i) the number of result tuples in $\psi \cap Q(\mathbf{I})$ can be returned in $O(N)$ time; and ii) all result tuples in $\psi \cap Q(\mathbf{I})$ can be enumerated with $O(1)$ delay.*

## 3 Cohesive Summaries

In this section, we present algorithms to construct cohesive $k$-summaries for the Euclidean and Hamming metrics. Recall that any algorithm presented in this section is also an algorithm for the relational $k$-center clustering problem.

### 3.1 Euclidean Metric

Our main algorithm for constructing a cohesive summary for Euclidean distance consists of three steps. In the first step, we compute a $\beta$-cohesive $k$-summary $S$ for some constant $\beta \geq 1$, along with a number $r$ such that $\rho_k(Q(\mathbf{I})) \leq \rho(S, Q(\mathbf{I})) \leq r \leq \beta \cdot \rho_k(Q(\mathbf{I}))$. Next, $S$ is used to construct a small set $P_\varepsilon \subseteq Q(\mathbf{I})$ (called *$\varepsilon$-coreset*) such that the optimally cohesive $k$-summary in $P_\varepsilon$ is an $(1+\varepsilon)$-cohesive $k$-summary in $Q(\mathbf{I})$. In the last step, we run a 2-approximation algorithm (Feder and Greene algorithm [38]) for the cohesive summary (in the non-relational setting) on $P_\varepsilon$, and we derive the final result. Throughout the paper, we use the term *non-relational setting* to denote the case where all input data is given in one relation.

**Algorithm 1:** CohesiveEuclidean$(Q, \mathbf{I}, k, \varepsilon)$

```
(S, r, β) ← ConstantApprox(Q, I, k);
P_ε ← Coreset(Q, I, S, r, β, ε);
S* ← FederGreene(P_ε, k) [38];
return S*;
```

The pseudocode of our main algorithm is shown in Algorithm 1. In the next subsections, we show all three steps in detail. In Subsections 3.1.1 and 3.1.2, we show two different constant approximations algorithms for the cohesive summary. We can invoke any of them as the ConstantApprox$(Q, \mathbf{I}, k)$ procedure in Algorithm 1. Even though they both return a constant approximation, we present both because neither dominates the other in terms of running time. The first one runs in roughly $O\left(k^2 N\right)$ time while the second one runs in $O\left(kN + k^{O(d)}\right)$ time. Next, in Subsection 3.1.3, we present the Coreset$(\cdot)$ procedure that gets as input the output of the previous constant approximation algorithm and constructs an $\varepsilon$ coreset $P_\varepsilon$ of size $O(\varepsilon^{-d}k)$. Finally, in Subsection 3.1.4, we run a 2-approximation algorithm for the cohesive summary (in the non-relational setting) on $P_\varepsilon$, and show the final result. All missing proofs in this section are given in Appendix B.1.

#### 3.1.1 *Constant cohesive summary: Tree-based approximation.*

In this part, we describe a hierarchical approach for constructing a cohesive summary.

**Main ideas.** Consider an acyclic join $Q$ and a database $\mathbf{I}$. Let $\mathbf{A}_x, \mathbf{A}_y \subseteq \mathbf{A}$ be two disjoint subsets of attributes. Let $S_x \subset \mathbb{R}^{|\mathbf{A}_x|}$ be a $r_x$-cohesive $k$-summary of $\pi_{\mathbf{A}_x} Q(\mathbf{I})$, and $S_y \subset \mathbb{R}^{|\mathbf{A}_y|}$ be a $r_y$-cohesive $k$-summary of $\pi_{\mathbf{A}_y} Q(\mathbf{I})$. We show that a set $S \subset \mathbb{R}^{|\mathbf{A}_x \cup \mathbf{A}_y|}$ of size $k$ can be computed efficiently with

small cohesion with respect to $\pi_{\mathbf{A}_x \cup \mathbf{A}_y} Q(\mathbf{I})$. Let

$$\bar{S} = \left\{ t \in S_x \times S_y : \exists t' \in \pi_{\mathbf{A}_x \cup \mathbf{A}_y} Q(\mathbf{I}), \phi(t, t') \leq \sqrt{2} \cdot \max\{r_x, r_y\} \right\},$$

be the set of tuples from the Cartesian product of two summaries that are "near" to some result tuple in $\pi_{\mathbf{A}_x \cup \mathbf{A}_y} Q(\mathbf{I})$. If $S$ is a 2-cohesive $k$-summary of $\bar{S}$, a key property we show is that $S$ is a $10\sqrt{2}\max\{r_x, r_y\}$-cohesive $k$-summary of $\pi_{\mathbf{A}_x \cup \mathbf{A}_y} Q(\mathbf{I})$. To construct a cohesive summary for $\pi_{\mathbf{A}_x \cup \mathbf{A}_y} Q(\mathbf{I})$, it suffices to construct a cohesive summary for $\pi_{\mathbf{A}_x} Q(\mathbf{I})$ and $\pi_{\mathbf{A}_y} Q(\mathbf{I})$ separately, further select a few representatives from the Cartesian product of these two summaries carefully, and finally return a $k$-cohesive summary for those representatives.

**Algorithm 2:** ConstantApprox_Tree($Q, \mathbf{I}, k$)

```
(S_v, r_v) ← ComputeRoot(Q, I, k);
S ← ∅;
foreach θ ∈ S_v do
    foreach i ∈ [m] do
        foreach p ∈ R_i do
            w_i(p) ← Σ_{A_j ∈ Ā_i} (p.A_j − θ.A_j)^2;
    w⃗ ← ⟨w_1, w_2, ··· , w_m⟩;
    Z ← index built for Q, I, w⃗ as Lemma 2.1;
    t_θ ← first result enumerated from Z;
    S ← S ∪ {t_θ};
while |S| < k do
    t ← the next result enumerated from Z;
    if t ∉ S then S ← S ∪ {t};
return S with radius r = 2 · r_v;
```

**Our algorithm.** Now, we are ready to describe our relational algorithm, with pseudocode given in Algorithm 2 and a running example in Figure 1. Algorithm 2 first calls Algorithm 3 as a primitive to return a set $S_v \subset \mathbb{R}^d$ of size $k$ and a value $r_v$ such that the cohesion $\rho(S_v, Q(\mathbf{I}))$ is small and the value $r_v$ is a sufficiently small upper bound of $\rho(S_v, Q(\mathbf{I}))$. Notice that $S_v$ will not necessarily be a subset of $Q(\mathbf{I})$. Then Algorithm 2 uses $S_v$ to construct a set $S \subseteq Q(\mathbf{I})$ that is a $O(1)$-cohesive $k$-summary and a value $r$ such that $\rho_k(Q(\mathbf{I})) \leq r \leq \alpha\rho_k(Q(\mathbf{I}))$, for a constant $\alpha$.

In Algorithm 3, we first construct a complete binary tree $\mathcal{T}$ with $m$ leaf nodes, where relation $R_i$ is stored at the $i$-th leaf node. For each node $u \in \mathcal{T}$, we denote $\bar{\mathbf{A}}_u = \bigcup_{R_i \text{is a descendant of } u} \bar{\mathbf{A}}_i$ and $Q_u = \pi_{\bar{\mathbf{A}}_u} Q(\mathbf{I})$. We visit all nodes in a bottom-up fashion, and for each node $u \in \mathcal{T}$, we compute i) a set $S_u$ of $k$ tuples in $\mathbb{R}^{|\bar{\mathbf{A}}_u|}$ with small cohesion $\rho(S_u, Q_u(\mathbf{I}))$, and ii) a sufficiently small upper bound $r_u$ of $\rho(S_u, Q_u(\mathbf{I}))$; see Lemma 3.4. Next, we show how to compute $S_u$ and $r_u$ for each node $u$.

If $u$ is a leaf node that corresponds to relation $R_i$, we compute an approximate $k$-summary $S_u$ for $\pi_{\bar{\mathbf{A}}_i}(R_i)$ by invoking the algorithm in [38]. The cohesion of $S_u$ is denoted as $r_u = \rho(S_u, \pi_{\bar{\mathbf{A}}_i}(R_i))$. If $u$ is an internal node, let $x, y$ be the two children of $u$ in $\mathcal{T}$. Let $S_x$, $S_y$ be the subsets of $k$ tuples we got from nodes $x, y$, respectively. Let $r^* = \max\{r_x, r_y\}$. We construct an approximate $k$-summary $S_u$ using the tuples in $S_x \times S_y$. To check for each tuple $\theta \in S_x \times S_y$, whether there exists any tuple in $Q_u(\mathbf{I})$ within distance $\sqrt{2} \cdot r^*$, we use a nearest-neighbor oracle. For relation $R_i$, we define the weight function $w_i(\cdot)$ as $w_i(p) = \sum_{A_j \in \bar{\mathbf{A}}_i \cap \bar{\mathbf{A}}_u} (p.A_j - \theta.A_j)^2$, where $p \in R_i$. We instantiate the index, defined by $Z$, for ranked enumeration from Lemma 2.1 with $\vec{w} = \langle w_1, \ldots, w_m \rangle$ as the vector of weight functions. Let $t_\theta$ be the first result tuple enumerated from $Z$ in ascending order ($t_\theta$ is the nearest neighbor of $\theta$ in $Q_u(\mathbf{I})$). If $\phi(\theta, t_\theta) \leq \sqrt{2}\max\{r_x, r_y\}$, then we keep $\theta$ in $\bar{S}_u$; otherwise, we skip $\theta$. Finally, we find a 2-cohesive $k$-summary of $\bar{S}_u$ by invoking the Feder-Greene algorithm [38], denoted as $S_u$, and we set the upper bound $r_u = \rho(S_u, \bar{S}_u) + \sqrt{2}\max\{r_x, r_y\}$. In the end, the Algorithm 3 returns the set $S_v$ and the upper bound $r_v$, for the root node $v$ of $\mathcal{T}$, to Algorithm 2.

Let $v$ be the root of $\mathcal{T}$. Recall that $S_v \subseteq Q(\mathbf{I})$ may not hold. To obtain a valid summary for $Q(\mathbf{I})$, in Algorithm 2, we visit every tuple $\theta \in S_v$ and find its nearest neighbor in $Q(\mathbf{I})$ using a similar

**Algorithm 3:** ComputeRoot($Q, \mathbf{I}, k$)

1 Let $\mathcal{T}$ be a complete binary tree with $m$ leaf nodes;
2 **foreach** *leaf node* $u \in \mathcal{T}$ **do**
3  Suppose $u$ corresponds to relation $R_i$;
4  $S_u \leftarrow k$-summary of $\pi_{\bar{\mathbf{A}}_i} R_i$ by algorithm in [38];
5  $r_u \leftarrow \rho(S_u, \pi_{\bar{\mathbf{A}}_i} R_i)$;
6 **foreach** *internal node* $u \in \mathcal{T}$ *in bottom-up way* **do**
7  $x, y \leftarrow$ two children of $u$ in $\mathcal{T}$, $\bar{S}_u \leftarrow \emptyset$;
8  **foreach** $\theta \in S_x \times S_y$ **do**
9  **foreach** $i \in [m]$ **do**
10  **foreach** $p \in R_i$ **do**
11  $w_i(p) \leftarrow \sum_{A_j \in \bar{\mathbf{A}}_i \cap \bar{\mathbf{A}}_u} \left(p.A_j - \theta.A_j\right)^2$
12  $\vec{w} \leftarrow \langle w_1, w_2, \cdots, w_m \rangle$;
13  $Z \leftarrow$ index built for $Q_u, \mathbf{I}, \vec{w}$ as Lemma 2.1;
14  $t_\theta \leftarrow$ the first result enumerated from $Z$;
15  **if** $\phi(t_\theta, \theta) \le \sqrt{2} \cdot \max\{r_x, r_y\}$ **then**
16  $\bar{S}_u \leftarrow \bar{S}_u \cup \{\theta\}$;
17  $S_u \leftarrow k$-summary of $\bar{S}_u$ by algorithm in [38];
18  $r_u \leftarrow \rho(S_u, \bar{S}_u) + \sqrt{2} \cdot \max\{r_x, r_y\}$;
19 **return** $(S_v, r_v)$;

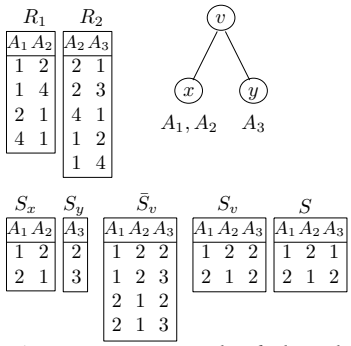


**Fig. 1.** A running example of Algorithm 3. Let $Q = R_1(A_1, A_2) \bowtie R_2(A_2, A_3)$ with a database $\mathbf{I}$ as shown. Let $k = 2$. The complete binary tree $\mathcal{T}$ is also shown with the root $v$ and two leaf nodes $x, y$, corresponding to $R_1, R_2$ respectively. In line 4, Algorithm 3 first computes a 2-summary of $R_1$ as $S_x$ and a 2-summary of $\pi_{A_3} R_2$ as $S_y$, with $r_x = 2$ and $r_y = 1$. It next investigates all tuples in $S_x \times S_y$, and checks if there exists some tuple in $Q(\mathbf{I})$ within distance $2\sqrt{2}$. It is easy to see $\bar{S}_v = S_x \times S_y$. In line 17, it computes a 2-summary of $\bar{S}_v$ as $S_v$. As $(2, 1, 2) \in Q(\mathbf{I})$, it adds $(2, 1, 2)$ to $S$. As $(1, 2, 2) \notin Q(\mathbf{I})$, it adds an arbitrary nearest neighbor $(1, 2, 1)$ to $S$. We observe that $\rho(S, Q(\mathbf{I})) = 2\sqrt{2}$.

nearest-neighbor oracle as we used before, constructing the index $Z$. Let $S$ be the set of nearest neighbors we compute. If $|S| < k$, we add arbitrary $k - |S|$ results from $Q(\mathbf{I})$ to $S$.

**Correctness.** By Lemma 3.2, and Lemma 3.1, the nearest neighbor queries with any $\theta$ can be answered correctly.

Lemma 3.1. *For $\vec{w}$ at line 7 of Algorithm 2, $\vec{w}(t) = \phi^2(t, \theta)$ for every $t \in Q(\mathbf{I})$.*

Lemma 3.2. *For $\vec{w}$ at line 12 of Algorithm 3, $\vec{w}(t) = \phi^2(t, \theta)$ for every $t \in Q_u(\mathbf{I})$.*

**Approximation.** We need the following lemma to show the approximation ratio:

Lemma 3.3. *For any internal node $u$ with children $x, y$ in $\mathcal{T}$, $\rho\left(\bar{S}_u, Q_u(\mathbf{I})\right) \le \sqrt{2} \cdot \max\{r_x, r_y\}$.*

Proof. For any node $x$ and its parent $u$ in $\mathcal{T}$ it holds that $\rho_k(Q_x(\mathbf{I})) \le \rho_k(Q_u(\mathbf{I}))$. For any tuple $p \in Q_x(\mathbf{I})$ there exists a tuple $p_t \in Q_u(\mathbf{I})$ such that $p_t.\bar{\mathbf{A}}_x = p$. By definition, it also holds that $\bar{\mathbf{A}}_x \subseteq \bar{\mathbf{A}}_u$. Hence, $\rho_k(Q_x(\mathbf{I})) \le \rho_k(Q_u(\mathbf{I}))$. Consider an arbitrary tuple $t \in Q_u(\mathbf{I})$. Let $\theta_x \in S_x$ be the nearest tuple to $t.\bar{\mathbf{A}}_x$, and $\theta_y \in S_y$ be the nearest tuple to $t.\bar{\mathbf{A}}_y$. By definition $\phi\left(t.\bar{\mathbf{A}}_x, \theta_x\right) \le r_x$ and $\phi\left(t.\bar{\mathbf{A}}_y, \theta_y\right) \le r_y$. We consider tuple $\theta$ as a concatenation of $\theta_x$ and $\theta_y$. Then, $\phi(t, \theta) = \sqrt{\phi^2\left(t.\bar{\mathbf{A}}_x, \theta_x\right) + \phi^2\left(t.\bar{\mathbf{A}}_y, \theta_y\right)} \le \sqrt{r_x^2 + r_y^2} \le \sqrt{2} \cdot \max\{r_x, r_y\}$. In line 8 of Algorithm 3, we consider the tuple $\theta$ in one of the iterations. Since $\phi(t, \theta) \le \sqrt{2} \cdot \max\{r_x, r_y\}$, the condition in line 15 holds, because $\phi(t_\theta, \theta) \le \phi(t, \theta) \le \sqrt{2} \cdot \max\{r_x, r_y\}$. Hence, $\theta$ is added in $\bar{S}_u$. Overall, for each tuple $t \in Q_u(\mathbf{I})$ there exists a tuple in $\bar{S}_u$ within distance $\sqrt{2} \cdot \max\{r_x, r_y\}$, so the result follows. □

Next, we point out the invariants that are preserved in the execution of our algorithm:

Lemma 3.4. *For node $u \in \mathcal{T}$ at level $\ell$, $\frac{1}{2} \cdot \rho_k(Q_u(\mathbf{I})) \le \rho(S_u, Q_u(\mathbf{I})) \le r_u \le (10\sqrt{2})^{\ell} \cdot \rho_k(Q_u(\mathbf{I}))$.*

Applying Lemma 3.4 to the root node $v$, we obtain $\rho(S_v, Q(\mathbf{I})) \le r_v \le (10\sqrt{2})^{\log m} \cdot \rho_k(Q(\mathbf{I}))$ and $\frac{1}{2} \cdot \rho_k(Q(\mathbf{I})) \le \rho(S_v, Q(\mathbf{I}))$.[6] From lines 3-10 in Algorithm 2, we have $\rho(S, Q(\mathbf{I})) \le 2\rho(S_v, Q(\mathbf{I}))$. Hence, we return a set $S \subseteq Q(\mathbf{I})$ and a value $r$ such that $\rho(S, Q(\mathbf{I})) \le r \le 2(10\sqrt{2})^{\log m} \cdot \rho_k(Q(\mathbf{I}))$.

**Complexity.** In Algorithm 3, it takes $O(N \log k)$ time to invoke the algorithm in [38] at line 4. At line 8, $|S_x \times S_y| = O(k^2)$. For each $\theta \in S_x \times S_y$, it takes $O(N)$ time to construct $Z$ at line 13 and get the first result tuple $t_\theta$ within $O(\log N)$ time. It takes $O(k^2 \log k)$ time to invoke the algorithm in [38] at line 17. The for-loop at lines 6-18 repeats $O(m)$ times, since there are $O(m)$ nodes in $\mathcal{T}$. In Algorithm 2, for each tuple $\theta \in S_v$ for the root node $v$, it takes $O(N)$ time to construct $Z$ at line 8 and get the first result tuple $t_\theta$ within $O(\log N)$ time. Overall, our algorithm runs in $O(Nk^2)$ time using $O(N + k^2)$ space.

Theorem 3.5. *For an acyclic join $Q$ of $m$ relations, a database $\mathbf{I}$ of input size $N$ and $\alpha = 2 \cdot (10\sqrt{2})^{\log m}$, an $\alpha$-cohesive $k$-summary of $Q(\mathbf{I})$ under Euclidean metric can be computed in $O\left(Nk^2\right)$ time using $O\left(N + k^2\right)$ space, with $r$ such that $\rho_k(Q(\mathbf{I})) \le r \le \alpha \cdot \rho_k(Q(\mathbf{I}))$.*

### 3.1.2 *Constant cohesive summary: Geometry-based approximation.*

Next, we exploit the properties of the Euclidean metric to obtain a different approximation algorithm.

**Main ideas.** Using the intuition from [43], we design an algorithm in the relational setting. We compute a summary $S'$ iteratively. Initially, let $\Gamma$ be a rectangle containing all tuples in $Q(\mathbf{I})$ and let $S' = \emptyset$. In each iteration, we solve a geometric problem computing the point $q_h \in \Gamma$ farthest from the set $S'$ (in the first iteration, $q_1$ is an arbitrary tuple in $\Gamma$). We next compute the tuple $s_h \in Q(\mathbf{I})$ closest to $q_h$ and add it in $S'$. Then, we implicitly remove from $\Gamma$ a ball around $q_i$ with radius $\phi(q_h, s_h)$ so that no other tuple could be selected close to $q_i$. We note that $\Gamma$ is not a rectangle after the first iteration. Instead, $\Gamma$ is the intersection of a rectangle with the complement of a set of balls. We use the nearest-neighbor oracle to compute the tuple in $Q(\mathbf{I})$ that is closest to $q_h$.

**Algorithm 4:** ConstantApprox_Geometry$(Q, \mathbf{I}, k)$

1 **foreach** $A_j \in \mathbf{A}$ **do**
2 $\quad a_j \leftarrow \min_{R_i} \min_{x \in \pi_{A_j} R_i} x$ and $b_j \leftarrow \max_{R_i} \max_{x \in \pi_{A_j} R_i} x$;
3 $\Gamma \leftarrow \times_{j: A_j \in \mathbf{A}} [a_j, b_j]$, $h \leftarrow 0$, $S' \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$;
4 **while** $h \le c^d \cdot k$ **do**
5 $\quad$ **if** $h = 0$ **then** $q_h \leftarrow$ an arbitrary tuple in $\Gamma$;
6 $\quad$ **else** $q_h \leftarrow$ the point in $\Gamma$ farthest from $S'$;
7 $\quad$ **foreach** $i \in [m]$ **do**
8 $\quad\quad$ **foreach** $p \in R_i$ **do**
9 $\quad\quad\quad w_i(p) \leftarrow \sum_{A_j \in \bar{\mathbf{A}}_i} (p.A_j - q_h.A_j)^2$;
10 $\quad \vec{w} \leftarrow \langle w_1, w_2, \cdots, w_m \rangle$;
11 $\quad Z \leftarrow$ an index built for $Q, \mathbf{I}, \vec{w}$ as Lemma 2.1;
12 $\quad s_h \leftarrow$ the first result enumerated from $Z$;
13 $\quad \mathcal{B}_h \leftarrow$ a ball centered at $q_i$ of radius $\phi(q_h, s_h)$;
14 $\quad S' \leftarrow S' \cup \{s_h\}$, $\Gamma \leftarrow \Gamma - \mathcal{B}_h$, $\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{B}_h\}$;
15 $\quad h \leftarrow h + 1$;
16 $S \leftarrow$ a $k$-summary of $S'$ by algorithm in [38];
17 **return** $S$;

**Our algorithm.** We next describe our algorithm in more detail with the pseudocode given in Algorithm 4 and a running example in Figure 2. We start with a rectangle $\Gamma \in \mathbb{R}^d$ that contains all tuples in $Q(\mathbf{I})$, and incrementally add points to a set $S'$. Initially $S' = \emptyset$. Let $\mathcal{B} = \emptyset$ be a set of balls in $\mathbb{R}^d$ (initially empty). We repeat the procedure $c^d \cdot k$ times for a constant $c$. In the first iteration, we choose an arbitrary point $q_1$ in $\Gamma$. In $h$-th iteration, for $h > 1$, we compute the point $q_h \in \Gamma$ with the farthest

[6] We use $\log(\cdot)$ for the logarithmic function with base 2.

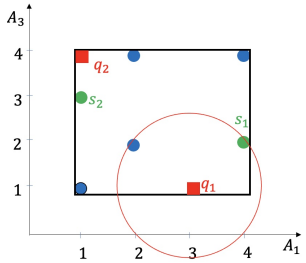


**Fig. 2.** The figure illustrates two iterations of Algorithm 4. Let $Q = \pi_{A_1,A_3} R_1(A_1, A_2) \bowtie R_2(A_2, A_3)$ with a database $\mathbf{I}$ as shown in Figure 1. Even though Algorithm 4 works for join queries, we consider the projection on $A_1$ and $A_3$ to show the main idea of the algorithm on the plane.

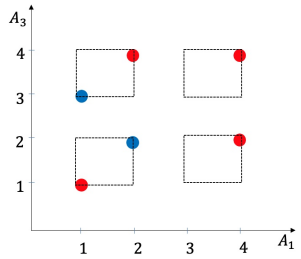


**Fig. 3.** Coreset construction shown in Algorithm 5. Let $Q = \pi_{A_1,A_3} R_1(A_1, A_2) \bowtie R_2(A_2, A_3)$ with a database $\mathbf{I}$ in Figure 1. Black dashed rectangles are non-empty grid cells and red points are selected tuples in the coreset.

distance from set $S'$. Next, by assigning weights $\vec{w}$ (similarly to Algorithm 2) we construct the ranked enumeration index $Z$ implementing a nearest-neighbor oracle. We find the nearest tuple in $Q(\mathbf{I})$ from point $q_h$, denoted as $s_h$, and add $s_h$ into $S'$. Let $\mathcal{B}_h$ be the ball of radius $\phi(q_h, s_h)$ centered at $q_h$. Set $\mathcal{B} = \mathcal{B} \cup \{\mathcal{B}_h\}$ and $\Gamma = \Gamma - \mathcal{B}_h$. We *implicitly* remove all result tuples in $Q(\mathbf{I})$ "covered" by $q_h$ within radius $\phi(q_h, s_h)$. After the procedure above, we get a set $S'$ of $c^d \cdot k$ candidate centers. At last, we compute a cohesive $k$-center $S$ of $S'$ by invoking the algorithm in [38].

**Correctness.** By Lemma 3.6, the nearest neighbor of every point $q_h \in Q(\mathbf{I})$ can be correctly found.

Lemma 3.6. *For $\vec{w}$ at line 10 of Algorithm 4, $\vec{w}(t) = \phi^2(t, q_h)$ for every tuple $t \in Q(\mathbf{I})$.*

The correctness of our algorithm follows from Lemma 3.6 and [43].

**Approximation.** From [43], we know $\rho(S', Q(\mathbf{I})) \le 4\rho_k(Q(\mathbf{I}))$. From [38], we know that $S$ is a 2-cohesive $k$-summary of $S'$, i.e., $\rho(S, S') \le 2\rho_k(S')$. Moreover, $\rho_k(S') \le \rho_k(Q(\mathbf{I}))$ since $S' \subseteq Q(\mathbf{I})$. Together, $\rho(S, Q(\mathbf{I})) \le \rho(S, S') + \rho(S', Q(\mathbf{I})) \le 2\rho_k(S') + 4\rho_k(Q(\mathbf{I})) \le 6\rho_k(Q(\mathbf{I}))$. Hence, Algorithm 4 returns a 6-cohesive $k$-summary.

**Complexity.** The initialization phase takes $O(N)$ time. In $h$-th iteration of the while-loop, it takes $O(N)$ time to construct $Z$ and get the first result $s_h$ within $O(\log N)$ time. Furthermore, there are $O(k)$ balls $\mathcal{B}_h$'s and $O(k)$ points in $S$. The point $q_h$ (farthest point from the current $S'$ in $\Gamma$) can be computed in $O(k^{\lceil d/2 \rceil + 1})$ time [8]. At last, invoking the algorithm in [38] takes $O(k \log k)$ time, since $|S'| = O(k)$. It computes the union of $k$ balls in $\mathbb{R}^d$, so the total space is $O(N + k^{\lceil d/2 \rceil})$.

Theorem 3.7. *For an acyclic join $Q$ with $d$ attributes and a database $\mathbf{I}$ of input size $N$, a 6-cohesive $k$-summary in Euclidean metric can be computed in $O\big(Nk + k^{\lceil d/2 \rceil + 1}\big)$ time using $O\big(N + k^{\lceil d/2 \rceil + 1}\big)$ space.*

### 3.1.3 *Coreset.*

Coresets for a set of points have been well studied [9], but these algorithms require direct access to $Q(\mathbf{I})$, which we do not have. Chen et al. [25] constructed coresets for risk minimization problems over relational data, but its complexity or utility depends on the diameter of the query results in $Q(\mathbf{I})$. Below, we show how to improve this result.

*Definition 3.8 (Coreset).* For an acyclic join $Q$ and a database $\mathbf{I}$, an integer $k \in \mathbb{N}^+$, and a parameter $\varepsilon > 0$, a subset $P_\varepsilon \subseteq Q(\mathbf{I})$ is an *$\varepsilon$-coreset* for the cohesive summary if $\rho(S^*_\varepsilon, Q(\mathbf{I})) \le (1+\varepsilon) \cdot \rho_k(Q(\mathbf{I}))$, where $S^*_\varepsilon \subseteq P_\varepsilon$ is the optimal cohesive $k$-summary for $P_\varepsilon$, i.e., $\rho(S^*_\varepsilon, P_\varepsilon) = \rho_k(P_\varepsilon)$.

**Our algorithm.** Let $S$ be an $O(1)$-cohesive $k$-summary for $Q(\mathbf{I})$ and $r$ be a small enough quantity such that $\rho(S, Q(\mathbf{I})) \le r$. We compute a grid over $\mathbb{R}^d$ with a sufficiently small diagonal (roughly $\varepsilon \cdot r$) and find the set of non-empty cells in the grid. For each such cell $\square$, we run the rectangular

oracle (Lemma 2.3) to get a tuple in $\square \cap Q(\mathbf{I})$. Finally, we return the set of tuples selected by the rectangular oracle in the non-empty cells.

We next describe our algorithm in more detail with the pseudocode given in Algorithm 5 and an example in Figure 3. Using Theorem 3.5 (or Theorem 3.7), we take as input a $\beta$-cohesive $k$-summary $S$ for $Q(\mathbf{I})$ for constant $\beta > 1$, and a value $r$ such that $\rho_k(Q(\mathbf{I})) \le \rho(S, Q(\mathbf{I})) \le r \le \beta \cdot \rho_k(Q(\mathbf{I}))$. We first construct a grid $G$ in $\mathbb{R}^d$ with cell diagonal length $\frac{\varepsilon \cdot r}{\beta}$ and find out all non-empty cells in $G$, i.e., those contain at least one tuple in $Q(\mathbf{I})$. Instead of visiting every tuple in $Q(\mathbf{I})$ to locate non-empty cells, which is too expensive, we resort to the cohesive summary $S$. Recall that for every tuple $x \in Q(\mathbf{I})$, there exists some $y \in S$ such that $\phi(x, y) \le r$. For every $y \in S$, let $\mathcal{B}_y$ be the ball centered at $y$ of radius $r$. Let $G_y \subseteq G$ be the set of cells covered or partially intersected by $\mathcal{B}_y$. At last, we visit every cell in $\bigcup_{y \in S} G_y$ and include an arbitrary tuple in the non-empty cells using the rectangular oracle as its *representative*. The set of all representatives is the coreset.

**Algorithm 5:** CORESET$(Q, \mathbf{I}, S, r, \beta, \varepsilon)$

```
P_ε ← ∅;
G ← a grid in ℝ^d with cell diagonal ε·r/β;
foreach y ∈ S do
    B_y ← ball of radius r centered at y;
    G_y ← {ψ ∈ G : ψ ∩ B_y ≠ ∅};
foreach ψ ∈ ⋃_{y∈S} G_y do
    if ψ ∩ Q(I) ≠ ∅ then
        p_ψ ← arbitrary tuple in ψ∩Q(I);
        P_ε ← P_ε ∪ {p_ψ};
return P_ε;
```

**Correctness.** Since $\rho(S, Q(\mathbf{I})) \le r$, $Q(\mathbf{I}) \subseteq \bigcup_{y \in S} \mathcal{B}_y$. Hence, every tuple in $Q(\mathbf{I})$ lies in one cell in $\bigcup_{y \in S} G_y$. As above, let $S^*_\varepsilon \subseteq P_\varepsilon$ be the optimally cohesive $k$-summary of $P_\varepsilon$. We have

$$\rho(S^*_\varepsilon, Q(\mathbf{I})) \le \rho(S^*_\varepsilon, P_\varepsilon) + \frac{\varepsilon r}{\beta} \le \rho_k(Q(\mathbf{I})) + \frac{\varepsilon r}{\beta} \le (1+\varepsilon) \cdot \rho_k(Q(\mathbf{I})),$$

where the last inequality follows from the fact that $r \le \beta \rho_k(Q(\mathbf{I}))$. Hence, $P_\varepsilon$ is an $\varepsilon$-coreset of $Q(\mathbf{I})$.

**Complexity.** As any ball of radius $r$ covers and partially intersects $O\left(\varepsilon^{-d}\right)$ grid cells with diagonal length $\varepsilon r/\beta$, we have $|P_\varepsilon| = O\left(k\varepsilon^{-d}\right)$. We execute rectangular oracles for $O(k\varepsilon^{-d})$ cells. Each query takes $O(N)$ time. Hence, Algorithm 5 runs in $O\left(kN\varepsilon^{-d}\right)$ time.

THEOREM 3.9. *For an acyclic join $Q$ with $d$ attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon > 0$, an $\varepsilon$-coreset for cohesive summaries under Euclidean distance of $O\left(\varepsilon^{-d}k\right)$ size can be constructed in $O\left(\min\left\{k^2N + kN\varepsilon^{-d}, N\log^2 N + kN\log(N)\varepsilon^{-d} + k^{\lceil d/2\rceil+1}\right\}\right)$ time.*

**Remark 1.** The min term in the time complexity in Theorem 3.9 depends on the algorithm used for computing cohesive summaries (Theorem 3.5 or Theorem 3.7).

**Remark 2.** We note that if the value $r$ is unknown (as in the algorithm of Theorem 3.7), it suffices to run a binary search on the $L_\infty$ distances of $Q(\mathbf{I})$. For each candidate $r'$, we check whether the set of balls $\{\mathcal{B}(y, \sqrt{d}r') \mid y \in S\}$ cover all tuples in $Q(\mathbf{I})$ by running rectangular oracles on the grids intersected by the balls. We repeat this procedure until we find the smallest value $r'$ that satisfies the condition. We describe the details in Appendix B.1.

#### 3.1.4 *Implications to cohesive summaries and relational $k$-center clustering.*

Let $P_\varepsilon$ be the $\varepsilon$-coreset obtained. We simply invoke the Feder-Greene algorithm [38] on $P_\varepsilon$. Let $S^*$ be the set of $k$ centers returned. By definition $\rho(S^*, Q(\mathbf{I})) \le (2+\varepsilon)\rho_k(Q(\mathbf{I}))$. We obtain:

THEOREM 3.10. *For an acyclic join $Q$ with $d$ attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon > 0$, a $(2+\varepsilon)$-cohesive $k$-summary for $Q(\mathbf{I})$ under Euclidean distance can be computed in $O\left(\min\left\{k^2N + kN\varepsilon^{-d}, N\log^2 N + kN\log(N)\varepsilon^{-d} + k^{\lceil d/2\rceil+1}\right\}\right)$ time. The same guarantees hold for the relational $k$-center clustering problem.*

**Remark**. The space needed by our algorithms in Theorems 3.9 and 3.10 depends on the $O(1)$-cohesive $k$-summary algorithm used for coreset construction. It is $O\left(N + k^2 + k\varepsilon^{-d}\right)$ (resp. $O(N + k^{\lceil d/2 \rceil + 1} + k\varepsilon^{-d})$ if the algorithm in Section 3.1.1 (resp. Section 3.1.2) is used.

### 3.2 Hamming Metric

First, we observe that it is trivial to obtain a $O(1)$-cohesive summary under the Hamming metric. The maximum Hamming distance between two tuples from $Q(\mathbf{I})$ is $d = O(1)$. Hence, an algorithm that chooses $k$ arbitrary tuples from $Q(\mathbf{I})$ returns a $d$-cohesive $k$-summary. However, this approximation ratio is rather unsatisfactory. The algorithm in Section 3.1 does not work for the Hamming metric mainly because the coreset construction only applies to the Euclidean metric. Hence, we need separate techniques for constructing a 2-cohesive $k$-summary under the Hamming metric.

**Algorithm 6:** CohesiveHamming$(Q, \mathbf{I}, k)$

```
γ ← (1/2) min_{A_j ∈ A}  min_{x≠y ∈ I(A_j)×I(A_j)} |x − y|;
for r_H ∈ {1, 2, ..., d} do
    R ← ∅, S ← ∅;
    M(R) ← decomposition of R;
    for i ∈ {1, ..., k} do
        C ← ∅;
        foreach c ∈ M(R) with density i do
            if |c ∩ Q(I)| ≥ 1 then C ← c and
              break;
        if C = ∅ then break;
        t ← arbitrary tuple in C ∩ Q(I);
        S ← S ∪ {t};
        R, M(R) ← UpdateR(Q, r_H, R, M(R), γ);
    if Σ_{c ∈ M(R) with density k} |c ∩ Q(I)| = 0 then break;
return S;
```

**Main ideas.** Suppose the value of $\rho_k(Q(\mathbf{I}))$ is known in advance. We repeat the following step for $k$ iterations: we choose an arbitrary tuple $t \in Q(\mathbf{I})$, add it to the $k$-summary $S$, and remove all items within distance $2\rho_k(Q(\mathbf{I}))$ from $Q(\mathbf{I})$. The resulting $S$ is a 2-cohesive summary for $Q(\mathbf{I})$. However, it is expensive to explicitly remove tuples in $Q(\mathbf{I})$, which requires materializing $Q(\mathbf{I})$. Instead, we compute a set of non-intersecting rectangles such that any point selected from these rectangles has a distance greater than $2\rho_k(Q(\mathbf{I}))$ (or equivalently at least $2\rho_k(Q(\mathbf{I})) + 1$) from the previously selected tuples $S$. For every new tuple $t$ we insert in $S$, we choose a set $\mathcal{R}^{(t)}$ of $O(2^d) = O(1)$ non-intersecting (open) rectangles around $t$ such that the union of these rectangles defines the points with distance at least $2\rho_k(Q(\mathbf{I})) + 1$ from $t$. Let $\mathcal{R} = \bigcup_{t \in S} \mathcal{R}^{(t)}$. In order to decide the next tuple to add in $S$, we only visit the regions in $\mathbb{R}^d$ with distance at least $2\rho_k(Q(\mathbf{I})) + 1$ from all tuples in $S$. One crucial observation is that these regions are covered by exactly $|S|$ rectangles in $\mathcal{R}$. We rely on the rectangular oracle in Section 2 to find tuples from $Q(\mathbf{I})$ that fall into these regions.

**Our algorithm.** We next describe our algorithm in more detail with the pseudocode given in Algorithm 6. For an attribute $A_j \in \mathbf{A}$, let $\mathbf{I}(A_j) = \{\pi_{A_j}(R_i) \mid R_i \in \mathbf{R}\}$ and let $\gamma$ be half of the minimum non-zero difference between two values in $\mathbf{I}(A_j)$. Let $\mathcal{R} = \emptyset$ be a set of rectangles that initially is empty. To identify the next tuple to insert in $S$, we construct a *rectangular decomposition* of the union of rectangles in $\mathcal{R}$. The rectangular decomposition [10] of $\mathcal{R}$, denoted $\mathcal{M}(\mathcal{R})$, is a partitioning of the union of the rectangles in $\mathcal{R}$ into rectangular contiguous regions, called *cells*, such that for each cell $\tau$, every point in $\tau$ lies in the same subset of $\mathcal{R}$.

Since, we do not know the optimum cohesion and $\rho_k(Q(\mathbf{I})) \in \{1, \ldots, d\}$, we try every value $r_H \in \{1, 2, \ldots, d\}$ as a guess for value $2\rho_k(Q(\mathbf{I})) + 1$. We repeat the following step for (at most) $k$ iterations. In the $i$-th iteration, we visit every cell in $\mathcal{M}(\mathcal{R})$ until we find one with density $i$ (a cell that is contained in exactly $i$ rectangles). Let $C$ be a cell with density $i$ such that $|C \cap Q(\mathbf{I})| \geq 1$. We

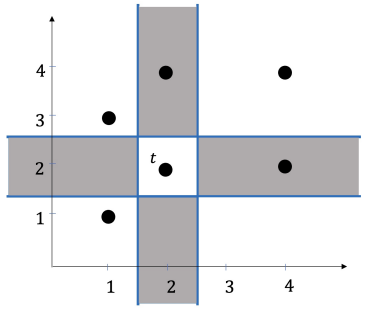


**Fig. 4.** All tuples have integer values and $\gamma = 0.5$. Set $\mathcal{R}_1^{(t)}$ contains the 4 gray (open) rectangles around $t$, $[1.5, 2.5]\times(-\infty, 1.5]$, $[2.5, \infty)\times[1.5, 2.5]$, $[1.5, 2.5] \times [2.5, \infty)$, $(-\infty, 1.5] \times [1.5, 2.5]$. All tuples/points in the rectangles $\mathcal{R}_1^{(t)}$ have Hamming distance 1 from $t$.

**Algorithm 7:** UPDATER$(\mathcal{Q}, r_H, \mathcal{R}, \mathcal{M}(\mathcal{R}), \gamma)$

```
1  for h ∈ {r_H, r_H + 1, ..., d} do
2      foreach X ⊆ A with |X| = h do
3          foreach A_j ∈ A do
4              if A_j ∉ X then
                   I_j ← [t.A_j − γ, t.A_j + γ];
5              else
6                  I_j^− ← (−∞, t.A_j − γ];
7                  I_j^+ ← [t.A_j + γ, +∞);
8          R ← R ∪ [ (×_{A_j ∉ X} I_j) × (×_{A_j ∈ X} {I_j^−, I_j^+}) ];
9          Update M(R);
10 return R, M(R);
```

get an arbitrary tuple $t \in C \cap \mathcal{Q}(\mathbf{I})$ using the rectangular oracle. If $i = 1$, $C$ is a rectangle containing all tuples in $\mathcal{Q}(\mathbf{I})$ and $t$ is any arbitrary tuple in $\mathcal{Q}(\mathbf{I})$. Next, we construct a set $\mathcal{R}^{(t)}$ of $O(1)$ rectangles that contain points with distance at least $r_H$ from $t$. This is described by Algorithm 7 as a primitive. For every $h = r_H, \ldots, d$, we construct the set of rectangles $\mathcal{R}_h^{(t)}$ such that if $p$ belongs to a rectangle in $\mathcal{R}_h^{(t)}$ then $\phi_H(p, t) = h$. In particular, for every subset $X \subseteq \mathbf{A}$ with $|X| = h$, we compute a set of intervals that will be used to create the rectangles in $\mathcal{R}_h^{(t)}$. If $A_j \notin X$, let $I_j = [t.A_j - \gamma, t.A_j + \gamma]$; otherwise, let $I_j^- = (-\infty, t.A_j - \gamma]$ and $I_j^+ = [t.A_j + \gamma, +\infty)$ and let $\mathcal{R}_X^t = \left(\times_{A_j \notin X} I_j\right) \times \left(\times_{A_j \in X} \{I_j^-, I_j^+\}\right)$. We define $\mathcal{R}_h^{(t)} = \bigcup_{X \subseteq \mathbf{A}, |X|=h} \mathcal{R}_X^t$, and let $\mathcal{R}^{(t)} = \bigcup_{h=r_H,\ldots,d} \mathcal{R}_h^{(t)}$. See an example in Figure 4. We add the set of rectangles $\mathcal{R}^{(t)}$ in $\mathcal{R}$, update the decomposition $\mathcal{M}(\mathcal{R})$, and proceed with the next iteration. At the end of Algorithm 6 (line 13), we check whether there is any uncovered point, i.e., if a tuple in $\mathcal{Q}(\mathbf{I})$ lies in a cell with density $k$. If no, we return $S$; otherwise, we proceed with the next value of $r_H$. Due to space limit, the analysis is shown in Appendix B.2.

THEOREM 3.11. *For an acyclic join $\mathcal{Q}$ with $d$ attributes and a database $\mathbf{I}$ of input size $N$, a 2-cohesive $k$-summary under Hamming metric can be computed in $O\left(Nk^d\right)$ time using $O(N + k^d)$ space.*

## 4 Min-diverse Summaries

In this section, we show how our ideas for constructing cohesive summaries can be used to design algorithms for constructing min-diverse summaries under the Euclidean and Hamming metrics.

### 4.1 Euclidean Metric

It is known that an $\varepsilon$-coreset for cohesive summaries is also an $\varepsilon$-coreset for min-diverse summaries [64].[7] After constructing an $\varepsilon$-coreset $P_\varepsilon$ for cohesive summaries using the algorithm described in Section 3.1.3, we run the algorithm from [11], in the non-relational setting, to derive a min-diverse $k$-summary over the set $P_\varepsilon$. This algorithm returns a $(\frac{1}{2} - \varepsilon)$-min-diverse $k$-summary of $P_\varepsilon$ in $O\left(|P_\varepsilon| \log(|P_\varepsilon|) + k(\log(|P_\varepsilon|) + \varepsilon^{-d})\right)$ time. Plugging Theorem 3.9 into this result, we obtain:

[7] A set $P_\varepsilon$ is an $\varepsilon$-coreset for the min-diverse summary problem if the optimal min-diverse $k$-summary in $P_\varepsilon$ is an $(1 - \varepsilon)$-min diverse $k$-summary in $\mathcal{Q}(\mathbf{I})$.

Theorem 4.1. *For an acyclic join $Q$ of $d$ attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon > 0$, a $\left(\frac{1}{2} - \varepsilon\right)$-min-diverse $k$-summary of $Q(\mathbf{I})$ under Euclidean metric can be computed in $O\left(\min\left\{k^2N + kN\varepsilon^{-d}, N\log^2 N + kN\log(N)\varepsilon^{-d} + k^{\lceil d/2\rceil+1}\right\}\right)$ time.*

As in Section 3.1.3, the space used by the algorithm in Theorem 4.1 depends on the algorithm for computing a $O(1)$-cohesive summary.

### 4.2 Hamming Metric

Again, our algorithm in Section 4.1 for the Euclidean metric cannot be applied to the Hamming metric because the coreset uses properties of the Euclidean metric. However, our algorithm in Section 3.2 for cohesive summaries under the Hamming metric can be extended to min-diverse summaries. Let $\sigma_k(Q(\mathbf{I}))$ be the minimum pairwise distance of the optimum min-diverse summary on $Q(\mathbf{I})$. For every new tuple $t$ we add to the returned set $S$, we compute a set of $O(1)$ non-intersecting rectangles that contain tuples with distance at least $\sigma_k(Q(\mathbf{I}))/2$ from $t$. There are only two minor differences with Algorithm 6. i) In line 2, we search for $r_h \in \{\lceil d/2\rceil+1, \lceil d/2\rceil+1-1, \ldots, 1\}$ to find the largest distance that separates the selected tuples. ii) In line 13, instead of checking whether there is no uncovered tuple, we check whether $S$ has size $k$. We obtain the next theorem.

Theorem 4.2. *For acyclic join $Q$ with $d$ attributes and a database $\mathbf{I}$ of input size $N$, a $\frac{1}{2}$-min-diverse $k$-summary under Hamming metric can be computed in $O\left(Nk^d\right)$ time using $O\left(N + k^d\right)$ space.*

## 5 Sum-diverse Summaries

We now describe the algorithms for constructing sum-diverse summaries under the Euclidean and Hamming metrics.

### 5.1 Euclidean Metric

**Main ideas.** In the non-relational setting, the following iterative algorithm described in [45] returns an $\frac{1}{2}$-sum-diverse $k$-summary over a set of points $P$. In each of the $k/2$ iterations, compute the farthest pair $(p_1, p_2)$ in $P$, add $\{p_1, p_2\}$ in the summary, remove them from $P$, and continue with the next iteration. To the best of our knowledge, there is no efficient algorithm to compute the farthest pair in $Q(\mathbf{I})$ (in the relational setting). Instead, we use the idea proposed in [11] to approximately compute the farthest pair among a set of points in the Euclidean setting using the notion of *$\varepsilon$-net*.

*Definition 5.1 ($\varepsilon$-net).* Let $\mathbb{S}^{d-1}$ be the unit sphere in $\mathbb{R}^d$. A centrally symmetric set $C \subseteq \mathbb{S}^{d-1}$ (i.e., if $u \in C$, then $-u \in C$) of $r = O\left(\varepsilon^{-(d-1)/2}\right)$ unit vectors in $\mathbb{R}^d$ is an *$\varepsilon$-net* if for every point $v \in \mathbb{S}^{d-1}$, there exists a point $u \in C$ with angle at most $\cos^{-1}\left(\frac{1}{1+\varepsilon}\right) = O\left(\sqrt{\varepsilon}\right)$.

As shown in [7, 22], for any pair of points $x, y \in \mathbb{R}^d$, it holds $(1-\varepsilon)\phi(x,y) \le \max_{u\in C}\langle u, x-y\rangle \le \phi(x,y)$. Hence, a $(1-\varepsilon)$-approximation of the maximum pairwise distance in a set of points can be found by only checking the top-1 points with respect to the vectors in the $\varepsilon$-net. Agarwal et al. [11], select the top $k$ points in each vector $u \in C$, and then run the iterative algorithm [45] on the union of the selected (top $k$) points to return a $(\frac{1}{2} - \varepsilon)$-sum-diverse $k$-summary.

**Our algorithm.** We next describe our algorithm in the relational setting with the pseudocode given in Algorithm 8 and a running example in Figure 5. Let $C$ be a centrally symmetric $\varepsilon$-net. We compute the top $k$ tuples in $Q(\mathbf{I})$ for every vector $u \in C$ using our relational top-$k$ oracle proposed in Section 2. More specifically, we define the weight function $w_i(\cdot)$ for tuples in $R_i$ as $w_i(p) = \sum_{A_j \in \bar{\mathbf{A}}_i} (p.A_j) \cdot u_j$, where $p \in R_i$. We construct the top-$k$ oracle by defining a ranked enumeration index $Z_u$ with the vector of weight functions $\vec{w} = \langle w_1, \ldots, w_m\rangle$. Let $P_u$ be the set of $k$ maximal tuples in $Q(\mathbf{I})$ in direction $u$, enumerated by $Z_u$ in descending order of their weights. At last, we invoke the iterative algorithm [45] to find a sum-diverse $k$-summary over $\bigcup_{u\in C} P_u$.

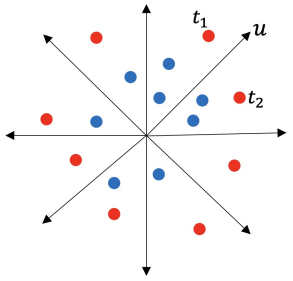


**Fig. 5.** A centrally symmetric set $C$ with 8 vectors. Each point represents a tuple in $Q(\mathbf{I})$. Assume $k = 2$. The top-2 tuples with respect to vector $u \in C$ are $t_1, t_2$. The set $\bigcup_{u \in C} P_u$ contains the red points/tuples.

**Algorithm 8:** GeometricSUMDiverse$(Q, \mathbf{I}, k)$

```
1  C ← a centrally symmetric ε-net;
2  foreach u ∈ C do
3      foreach i ∈ [m] do
4          foreach p ∈ R_i do
5              w_i(p) ← Σ_{A_j ∈ Ā_i} (p.A_j) · u_j
6      w⃗ ← ⟨w_1, w_2, ··· , w_m⟩;
7      Z_u ← an index built for Q, I, w⃗ as Lemma 2.1;
8      P_u ← the first k results enumerated from Z_u;
9  S ← sum-diverse k-summary over ⋃_{u∈C} P_u [45];
10 return S;
```

**Correctness.** By the definition of the top-$k$ oracle (Lemma 2.2), the next lemma holds. For completeness, we also show the straightforward proof in Appendix C.

Lemma 5.2. *For $\vec{w}$ at line 6 of Algorithm 8, $\vec{w}(t) = \langle u, t \rangle$ for every tuple $t \in Q(\mathbf{I})$.*

By Lemma 5.2, for any vector $u \in C$, all tuples in $Q(\mathbf{I})$ can be enumerated in a decreasing ordering of their inner product with $u$. The correctness follows from [11] and the discussion above.

**Complexity.** The $\varepsilon$-net $C$ can be computed in $O(r)$ time [6]. For each vector $u \in C$, we construct an index $Z_u$ in $O(N)$ time. In total, we can construct $\bigcup_{u \in C} P_u$ in $O(r(N + k \log N))$ time using $O(N + rk)$ space. Finally, the algorithm in [45] over the set $\bigcup_{u \in C} P_u$ (as implemented in [11]) runs in $O(rk \log N)$ time. The algorithm uses $O(rk)$ space to store the top $k$ tuples for every vector in $C$.

Theorem 5.3. *For an acyclic join $Q$ of $d$ attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon \in (0, \frac{1}{2})$, a $\left(\frac{1}{2} - \varepsilon\right)$-sum-diverse $k$-summary of $Q(\mathbf{I})$ under Euclidean metric can be computed in $O\left((N + k \cdot \log N)\varepsilon^{-(d-1)/2}\right)$ time using $O\left(N + k\varepsilon^{-(d-1)/2}\right)$ space.*

## 5.2 Hamming Metric

For the Hamming metric, we propose two algorithms that construct sum-diverse summaries. The first computes a better sum-diverse $k$-summary, while the second is faster by a factor of $k$.

**Main ideas.** In [21], the authors showed that if a distance is *of negative type* [62, 63], then a local search algorithm returns a $(1 - 2/k)$-sum-diverse $k$-summary in the non-relational setting.

*Definition 5.4 (Negative Type).* Let $\mathcal{D} \in \mathbb{R}^{n \times n}$ be the distance matrix of distance function $\sigma$. The function $\sigma$ is of negative type if for any vector $x = (x_1, \ldots, x_n)$ with $\sum_{i=1}^{n} x_i = 0$, $x^\top \mathcal{D} x \leq 0$.

In Appendix C, we prove that the Hamming distance is of negative type. Hence, we can obtain a relational version of the local search algorithm. Intuitively, our algorithm starts with a set $S$ of $k$ arbitrary tuples from $Q(\mathbf{I})$ and then repeats the following step for at most $O(k \log k)$ iterations: if there exists a pair of tuples $x \in S$ and $y \in Q(\mathbf{I}) \setminus S$ such that replacing $x$ with $y$ in $S$ can increase its diversity, i.e., $\delta(S \cup \{y\} \setminus \{x\}) > \delta(S)$, we update $S$ accordingly; Otherwise, we just return $S$.

**Our algorithm.** In the relational setting, it is challenging to find the tuples $x, y$ to update the set $S$. We next describe our algorithm in more detail with the pseudocode given in Algorithm 9. Initially, we add $k$ arbitrary tuples from $Q(\mathbf{I})$ to $S$. For every tuple $t \in S$, we define its *diversity* as $u(t) = \sum_{x \in S \setminus \{t\}} \phi_H(x, t)$, i.e., its sum of distances with remaining tuples in $S$. Note that the diversity

**Algorithm 9:** LocalSearch$(Q, \mathbf{I}, k)$

1 $S \leftarrow k$ arbitrary tuples from $Q(\mathbf{I})$;
2 **foreach** $t \in S$ **do**
3 $\quad u(t) \leftarrow \sum_{x \in S \setminus \{t\}} \phi_H(x, t)$;
4 $\Delta \leftarrow \frac{1}{2} \sum_{t \in S} u(t)$;
5 **for** $j \in \{1, 2, \cdots, O(k \log k)\}$ **do**
6 $\quad (p^-, p^+, M) \leftarrow$ Replace$(Q, \mathbf{I}, S)$;
7 $\quad$ **if** $\Delta > M$ **then return** $S$;
8 $\quad$ **foreach** $t \in S \setminus \{p^-\}$ **do**
9 $\quad\quad u(t) \leftarrow u(t) - \phi_H(t, p^-) + \phi_H(t, p^+)$;
10 $\quad u(p^+) \leftarrow \sum_{t \in S \setminus \{p^-\}} \phi_H(t, p^+)$;
11 $\quad S \leftarrow S \cup \{p^+\} \setminus \{p^-\}$;
12 $\quad \Delta \leftarrow \frac{1}{2} \sum_{t \in S} u(t)$;
13 **return** $S$;

**Algorithm 10:** Replace$(Q, \mathbf{I}, S)$

1 $M \leftarrow -\infty$, $p^- \leftarrow$ null, $p^+ \leftarrow$ null;
2 **foreach** $x \in S$ **do**
3 $\quad$ **foreach** $i \in [m]$ **do**
4 $\quad\quad$ **foreach** $p \in R_i$ **do**
5 $\quad\quad\quad w_i(p) \leftarrow \sum_{y \in S \setminus \{x\}} \phi_H\left(y.\bar{\mathbf{A}}_i, p.\bar{\mathbf{A}}_i\right)$;
6 $\quad \vec{w} \leftarrow \langle w_1, w_2, \cdots, w_m \rangle$;
7 $\quad Z \leftarrow$ index built for $Q, \mathbf{I}, \vec{w}$ as Lemma 2.1;
8 $\quad$ **while true do**
9 $\quad\quad y \leftarrow$ a result enumerated from $Z$;
10 $\quad\quad$ **if** $y \notin S$ **then break**;
11 $\quad \Delta_{x,y} \leftarrow \Delta - u(x) + \sum_{i \in [m]} w_i(y.\mathbf{A}_i)$ ;
12 $\quad$ **if** $\Delta_{x,y} > M$ **then**
13 $\quad\quad M \leftarrow \Delta_{x,y}$, $(p^-, p^+) \leftarrow (x, y)$;
14 **return** $(p^-, p^+, M)$;

of $S$ is essentially $\Delta = \frac{1}{2} \sum_{t \in S} u(t)$. We repeat the following step for at most $O(k \log k)$ iterations.[8] We call Algorithm 10 as a primitive to compute the pair of tuples $p^- \in S, p^+ \in Q(\mathbf{I}) \setminus S$ such that the diversity $M = \delta(S \cup \{p^+\} \setminus \{p^-\})$ is maximized. We do it as follows. For each tuple $x \in S$, we construct an index $Z$ from Lemma 2.1 with the following weight function $\vec{w}$. More specifically, for every tuple $p \in R_i$, we define the weight function $w_i(\cdot)$ as $w_i(p) = \sum_{y \in S \setminus \{x\}} \phi_H\left(y.\bar{\mathbf{A}}_i, p.\bar{\mathbf{A}}_i\right)$. All tuples in $Q(\mathbf{I})$ will be enumerated from $Z$ in the descending ordering to their sum of distances with tuples in $S \setminus \{x\}$, until we encounter some tuple $y \notin S$. In Algorithm 10 we use the notation $\Delta_{x,y}$ to maintain the diversity of the set $S \cup \{y\} \setminus \{x\}$. If replacing $p^-$ by $p^+$ in $S$ does not increase the diversity (line 7 of Algorithm 9) we stop and return $S$. Otherwise, we replace $p^-$ by $p^+$ in $S$ and update $u(t)$ for every $t \in S \setminus \{p^-\}$. Then, we enter into the next iteration.

Finally, in order to compute $w_i(\cdot)$ efficiently, we build a binary search tree $\mathcal{T}_j$, initially empty, for every attribute $A_j \in \bar{\mathbf{A}}_i$, as follows. For every $y \in S \setminus \{x\}$, we check whether the value $y.A_j$ exists in $\mathcal{T}_j$. If not, we add a node $u$ to $\mathcal{T}_j$ with value $u.\text{value} = y.A_j$ along with a counter $u.\text{count} = 1$. If yes, then let $u$ be the node with $u.\text{value} = y.A_j$. We increase the counter $u.\text{count}$ by 1. After constructing $T_j$, for every $A_j \in \bar{\mathbf{A}}_i$, we visit every tuple $p \in R_i$ and we search each $\mathcal{T}_j$ with key $p.A_j$. Let $u_j$ be the node such that $u_j.\text{value} = p.A_j$. We compute $w_i(p)$ as $\sum_{A_j \in \bar{\mathbf{A}}_i} (|S| - u_j.\text{count})$.

**Correctness.** We prove the next lemma.

Lemma 5.5. *For $\vec{w}$ at line 6 of Algorithm 10, $\vec{w}(t) = \sum_{y \in S \setminus \{x\}} \phi_H(t, y)$ for every tuple $t \in Q(\mathbf{I})$.*

Proof. For $t \in Q(\mathbf{I})$, $\sum_{i \in [m]} w_i(t.\mathbf{A}_i) = \sum_{i \in [m]} \sum_{y \in S \setminus \{x\}} \phi_H(t.\bar{\mathbf{A}}_i, y.\bar{\mathbf{A}}_i) = \sum_{y \in S \setminus \{x\}} \phi_H(t, y)$. □

For $x \in S$, by Lemma 5.5, tuple $y \in Q(\mathbf{I}) \setminus S$ that maximizes $\delta(S \cup \{y\} \setminus \{x\})$ can always be correctly found. Following [21] and the fact that Hamming distance is of negative type, our algorithm returns a $(1 - \frac{2}{k})$-sum-diverse $k$-summary for $Q(I)$.

**Complexity.** Initially, $k$ tuples from $Q(\mathbf{I})$ can be retrieved in $O(N + k)$ time. In every iteration, for every $x \in S$, it takes $O(N \log k) = O(N \log N)$ time to assign the weights $w_i$ using the binary

[8] The exact number of iterations is shown in the proof of Corollary 2 in [21].

search trees $\mathcal{T}_j$. The index $Z$ is constructed in $O(N)$ time. Each tuple $y$ can be enumerated with $O(\log N)$ delay. In the worst case, it may skip at most $k$ tuples before finding one that does not belong to $S$. Overall, for each tuple $x$, we spend $O(N \log N + k \log N)$ time.

Theorem 5.6. *For an acyclic join $Q$ and database $\mathbf{I}$ of input size $N$, a $(1-\frac{2}{k})$-sum-diverse $k$-summary under Hamming metric can be computed in $O\left(Nk^2 \log^2 N + k^3 \log^2 N\right)$ time using $O(N+k)$ space.*

**Remark 1.** The Euclidean metric is also of negative type. However, due to the square root operations in the computation of the Euclidean metric, we cannot use the ranked enumeration index to get the best tuple that improves the diversity in $S \setminus \{t\}$, as we did in the Hamming metric. So, this algorithm does not apply to the Euclidean metric.

**Remark 2.** The same high-level idea can be extended to another faster algorithm, but its quality is slightly worse than Theorem 5.6. Intuitively, we begin with a set $S$ containing one arbitrary tuple from $Q(\mathbf{I})$, and apply the following greedy strategy for $k$ iterations. In each iteration, we find the tuple $y \in Q(\mathbf{I}) \setminus S$ that maximizes the sum of all pairwise distances with tuples in $S$, i.e., $\arg\max_{t \in Q(\mathbf{I})} \sum_{p \in S} \phi_H(p, t)$, and add it to $S$. This greedy approach returns a $\frac{1}{2}$-sum-diverse $k$-summary for $Q(\mathbf{I})$. Note that the problem of finding the tuple from $Q(\mathbf{I}) \setminus S$ with the maximum sum of distances from tuples in $S$ is similar to finding the best tuple $y \in Q(\mathbf{I}) \setminus S$ to replace a tuple $x \in S$ as we had in the local search algorithm above. Due to the space limit, all details can be found in Appendix C. Hence, using the same machinery from our previous algorithm, we have:

Theorem 5.7. *For an acyclic join $Q$ and a database $\mathbf{I}$ of input size $N$, a $\frac{1}{2}$-sum-diverse $k$-summary under Hamming metric can be computed in $O\left((Nk + k^2) \log N\right)$ time using $O(N+k)$ space.*

## 6 Extensions

**From acyclic joins to cyclic joins.** All our algorithms can be extended to cyclic join queries by applying the generalized hypertree decomposition [42], as described in Appendix D. Each cyclic join is transformed into an acyclic one at the cost of increasing the input size from $N$ to $N^{\mathsf{fhtw}}$, where fhtw is defined as the *fractional hypertree width* of the input join query which roughly measures how close is the input query $Q$ from being acyclic (for example, for every acyclic query $\mathsf{fhtw} = 1$). All our approximation algorithms derived for acyclic joins can be applied without any modification, but time complexity increases by replacing $N$ with $N^{\mathsf{fhtw}}$.

**From joins to join-project queries.** All our algorithms can be extended to acyclic join-project queries using indexes for ranked enumeration over join-project queries [32]. Using the generalized hypertree decomposition [42], the results are also extended to cyclic join-project queries. All approximation ratios are preserved, but the time complexity for constructing sum-diverse summaries increases by a factor of $O(\min\{k, N\})$, since the index [32] for join-project queries can only support $O(N \log N)$-delay enumeration. The running time for constructing a cohesive or min-diverse summary remains the same for join-project queries. We show the details in Appendix E.

## 7 Conclusion

In this paper, we designed efficient algorithms for computing cohesive and diverse summaries for conjunctive query results under the Euclidean or Hamming metric. There are a few interesting questions left for future work. (1) *General metric*: In addition to Euclidean and Hamming metrics, it is unknown how to compute representative summaries for conjunctive query results under general metrics. (2) *Broader quality functions*: beyond cohesion and diversity, it remains to investigate a general framework of computing good summaries under various quality functions. (3) *Dynamic setting*: It is unknown how to maintain representative summaries for conjunctive query results in the dynamic settings, where input tuples can be inserted or deleted.

## References


[1] https://db-engines.com/en/ranking_categories.

[2] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–40, 2013.

[3] M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, and A. D. Mehrabi. Range-clustering queries. In *Proceedings of the 33rd International Symposium on Computational Geometry*, pages 5:1–5:16, 2017.

[4] R. Addanki, A. McGregor, A. Meliou, and Z. Moumoulidou. Improved approximation and scalability for fair max-min diversification. In *Range-clustering queries 25th International Conference on Database Theory*, pages 7:1–7:21, 2022.

[5] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *ACM Transactions on Database Systems*, 38(4):1–28, 2013.

[6] P. K. Agarwal, S. Har-Peled, and H. Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1-3):38–58, 2008.

[7] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry*, 1(4):189–201, 1992.

[8] P. K. Agarwal, J. Pach, and M. Sharir. State of the union (of geometric objects). pages 9–48. 2008.

[9] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33:201–226, 2002.

[10] P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier, 2000.

[11] P. K. Agarwal, S. Sintos, and A. Steiger. Efficient indexes for diverse top-k range queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 213–227, 2020.

[12] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42, 2013.

[13] M. Arenas, T. C. Merkl, R. Pichler, and C. Riveros. Towards tractability of the diversity of query answers: Ultrametrics to the rescue. *arXiv preprint arXiv:2408.01657*, 2024.

[14] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[15] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of the International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

[16] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[17] B. Birnbaum and K. J. Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing lps. *Algorithmica*, 55(1):42–59, 2009.

[18] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 155–166, 2012.

[19] A. Cevallos. Approximation algorithms for geometric dispersion. Technical report, EPFL, 2016.

[20] A. Cevallos, F. Eisenbrand, and R. Zenklusen. Local search for max-sum diversification. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 130–142. SIAM, 2017.

[21] A. Cevallos, F. Eisenbrand, and R. Zenklusen. An improved analysis of local search for max-sum diversification. *Mathematics of Operations Research*, 44(4):1494–1509, 2019.

[22] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 300–309, 2000.

[23] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems*, 32(2):9–es, 2007.

[24] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999.

[25] J. Chen, Q. Yang, R. Huang, and H. Ding. Coresets for relational data and the applications. *Advances in Neural Information Processing Systems*, 35:434–448, 2022.

[26] Y. Chen and K. Yi. Random sampling and size estimation over cyclic joins. In *Proceedings of the 23rd International Conference on Database Theory*, pages 7:1–7:18, 2020.

[27] G. Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, page 15, 2011.

[28] G. Cormode. Data sketching. *Communications of the ACM*, 60(9):48–55, 2017.

[29] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[30] G. Cormode and K. Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020.

[31] R. Curtin, B. Moseley, H. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Rk-means: Fast clustering for relational data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 2742–2752, 2020.
[32] S. Deep, X. Hu, and P. Koutris. Ranked enumeration of join queries with projections. *Proceedings of the VLDB Endowment*, 15(5):1024–1037, 2022.
[33] S. Deep and P. Koutris. Ranked enumeration of conjunctive query results. In *Proceedings of the 24th International Conference on Database Theory*, pages 5:1–5:19, 2021.
[34] S. Deng, S. Lu, and Y. Tao. On join sampling and hardness of combinatorial output-sensitive join algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 99–111, 2023.
[35] M. Deza and H. Maehara. Metric transforms and euclidean embeddings. *Transactions of the American Mathematical Society*, 317(2):661–671, 1990.
[36] A. Esmailpour and S. Sintos. Improved approximation algorithms for relational clustering. *Proceedings of the ACM on Management of Data*, 2(5), 2025.
[37] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.
[38] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 434–444, 1988.
[39] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 476–487, 2002.
[40] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 331–342, 1998.
[41] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
[42] G. Gottlob, G. Greco, and F. Scarcello. Treewidth and hypertree width. *Tractability: Practical Approaches to Hard Problems*, 1, 2014.
[43] S. Har-Peled, N. Kumar, D. M. Mount, and B. Raichel. Space exploration via proximity search. *Discrete & Computational Geometry*, 56:357–376, 2016.
[44] S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *Journal of the ACM*, 62(6):1–35, 2015.
[45] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133–137, 1997.
[46] X. Hu and S. Sintos. Finding smallest witnesses for conjunctive queries. In *Proceedings of the 27th International Conference on Database Theory*, pages 24:1–24:20, 2024.
[47] Y. Ioannidis. The history of histograms (abridged). In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 19–30, 2003.
[48] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 275–286, 1998.
[49] M. Jones, H. Nguyen, and T. Nguyen. Fair k-centers via maximum matching. In *Proceedings of the International Conference on Machine Learning*, pages 4940–4949, 2020.
[50] K. Kim, J. Ha, G. Fletcher, and W.-S. Han. Guaranteeing the o(agm/out) runtime for uniform sampling and size estimation over joins. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 113–125, 2023.
[51] M. Kleindessner, P. Awasthi, and J. Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pages 3448–3457. PMLR, 2019.
[52] Y. Kurkure, M. Shamo, J. Wiseman, S. Galhotra, and S. Sintos. Faster algorithms for fair max-min diversification in $R^d$. *Proceedings of the ACM on Management of Data*, 2(3):1–26, 2024.
[53] X. Liang, S. Sintos, Z. Shang, and S. Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1129–1141, 2021.
[54] T. C. Merkl, R. Pichler, and S. Skritek. Diversity of answers to conjunctive queries. In *Proceedings of the 26th International Conference on Database Theory*, pages 10:1–10:19, 2023.
[55] B. Moseley, K. Pruhs, A. Samadian, and Y. Wang. Relational algorithms for k-means clustering. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming*, pages 97:1–97:21, 2021.
[56] Z. Moumoulidou, A. McGregor, and A. Meliou. Diverse data selection under fairness constraints. In *Proceedings of the 24th International Conference on Database Theory*, pages 13:1–13:25, 2021.
[57] E. Oh and H.-K. Ahn. Approximate range queries for clustering. In *Proceedings of the 34th International Symposium on Computational Geometry*, pages 62:1–62:14, 2018.
[58] D. Olteanu and J. Závodnỳ. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298, 2012.

[59] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.
[60] J. S. Salowe. L-infinity interdistance selection by parametric search. *Information Processing Letters*, 30(1):9–14, 1989.
[61] I. J. Schoenberg. Remarks to Maurice Frechet's article"sur la definition axiomatique d'une classe d'espace distances vectoriellement applicable sur l'espace de hilbert. *Annals of Mathematics*, pages 724–732, 1935.
[62] I. J. Schoenberg. Metric spaces and completely monotone functions. *Annals of Mathematics*, pages 811–841, 1938.
[63] I. J. Schoenberg. Metric spaces and positive definite functions. *Transactions of the American Mathematical Society*, 44(3):522–536, 1938.
[64] A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4(4):550–567, 1991.
[65] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases*, volume 81, pages 82–94, 1981.
[66] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 1525–1539, 2018.
[67] Z. Zhao, F. Li, and Y. Liu. Efficient join synopsis maintenance for data warehouse. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2027–2042, 2020.

## A Missing details from Section 1

**Dangling Tuples.** Given an acyclic join $Q$ and a database $\mathbf{I}$ of input size $N$, we give the classic Yannakakis algorithm [65] that can remove dangling tuples that do not participate in any join result of $Q(I)$. This primitive runs in $O(N)$ time. For a CQ $Q$ and a database $\mathbf{I}$ of input size $N$, all dangling tuples can be removed in $O\left(N^{\text{fhtw}}\right)$ time (Appendix D).

**Algorithm 11:** RemoveDangling($Q, \mathbf{I}$)

```
Let 𝒯 be an arbitrary join tree of Q with root r;
while visit nodes a bottom-up way (excluding r) do
    foreach node u visited do
        R_{p_u} ← R_{p_u} ⋉ R_u for the parent node p_u of u;
while visit nodes a top-down way (excluding leaves) do
    foreach node u visited do
        R_{u'} ← R_{u'} ⋉ R_u for each child node u' of u;
return updated I;
```

**Reduction from plain data.** The hardness of the problems defined on plain data points can be carried to a relational data setting via the following reduction. Suppose we are given a set $P$ of $n$ points in the $d$-dimensional space, where $d = |\mathbf{A}|$, and each point $p \in P$ is associated with $d$ values (coordinates) $\langle p_1, p_2, \cdots, p_d \rangle$. Moreover, we give a distinct label $p_{\text{id}}$ to every point $p \in P$. We also label the attributes in $\mathbf{A}$, as $A_1, A_2, \cdots, A_d$. We construct database $\mathbf{I}$ as follows.For every point $p \in P$, we add a tuple $t_i^p$ to $R_i$ for every relation $R_i$, where $t_i^p.A_j = (p_{\text{id}}, p_j)$ for every $A_j \in \mathbf{A}_i$. It can be easily checked that there is a one-to-one mapping between the query results in the join $Q(\mathbf{I})$ and points in $P$. Let $t_p, t_{p'} \in Q(\mathbf{I})$ be the query results corresponding to $p, p' \in P$ respectively. The distance between $p$ and $p'$ is transformed to the distance between $t_p$ and $t_{p'}$. This reduction implies the NP-hardness of computing cohesive and min-diverse summaries.

**Optimality.** All lower bounds from non-relational settings hold in our relational setting. First, the dependency on $N$ in all our algorithms for acyclic join queries is near linear, which is optimal. Any algorithm for computing summaries needs to read the entire database at least. For Euclidean cohesive summary, we give $(2+\varepsilon)$-approximation algorithms in $\tilde{O}(Nk^2)$ or $\tilde{O}(Nk + k^d)$ time (assuming $\varepsilon$ as a small constant). For the non-relational setting, the best algorithm for 2-cohesive summary under any general metric runs in $O(Nk)$ time. Hence, the approximation factor and complexities of our algorithms are close (by, at most, a factor of $k$) to the optimum algorithms in the non-relational setting. Exactly the same results and lower bounds hold for min-diversity summaries. For Euclidean sum-diverse summaries, we give a $(\frac{1}{2} - \varepsilon)$-approximation algorithm in $\tilde{O}(N+k)$ time. For general distances in the non-relational setting, the best algorithm returns a $\frac{1}{2}$-approximation in $O(Nk)$ time, while in the Euclidean metric, the best algorithm returns a $(\frac{1}{2} - \varepsilon)$-approximation in $\tilde{O}(N+k)$ time. Our algorithm for sum-diverse summaries in the Euclidean metric is optimal.

## B Missing details from Section 3

### B.1 Missing details from Subsection 3.1

Proof of Lemma 3.1. $\sum_{i\in[m]} w_i(t.\mathbf{A}_i) = \sum_{i\in[m]} \sum_{A_j\in\bar{\mathbf{A}}_i} (t.A_j - \theta.A_j)^2 = \sum_{A_j\in\mathbf{A}} (t.A_j - \theta.A_j)^2 = \phi^2(t,\theta).$

Proof of Lemma 3.2. $\sum_{i\in[m]} w_i(t.\mathbf{A}_i) = \sum_{i\in[m]} \sum_{A_j\in\bar{\mathbf{A}}_i\cap\bar{\mathbf{A}}_u} (t.A_j - \theta.A_j)^2 = \sum_{A_j\in\bar{\mathbf{A}}_u} (t.A_j - \theta.A_j)^2 = \phi^2(t,\theta).$

Proof of Lemma 3.4. We will prove by induction on $\ell$. In the base case when $\ell = 1$, $u$ is a leaf node. Implied by [25] and the fact that we only keep the non-dangling tuples, $S_u$ is a 2-approximation of the $k$-center problem for $Q_u(\mathbf{I})$. Hence, $\rho(S_u, Q_u(\mathbf{I})) = r_u \le 2 \cdot \rho_k\left(Q_u(\mathbf{I})\right) \le 10\sqrt{2} \cdot \rho_k\left(Q_u(\mathbf{I})\right).$

When $\ell > 1$, $u$ is an internal node. Let $x, y$ be the two child nodes at level $\ell-1$. Let $r^* = \max\{r_x, r_y\}$. From Algorithm 3, for every tuple $\theta \in \bar{S}_u$ (after finishing the loop in lines 8-16), there exists a tuple $t \in Q_u(\mathbf{I})$ such that $\phi(t,\theta) \leq \sqrt{2} \cdot r^*$. On the other hand, from Lemma 3.3, for any tuple $t \in Q_u(\mathbf{I})$, there exists a tuple $\theta \in \bar{S}_u$ such that $\phi(t,\theta) \leq \sqrt{2} \cdot r^*$. Moreover,

$$r^* \leq (10\sqrt{2})^{\ell-1} \cdot \max\{\rho_k(Q_x(\mathbf{I})), \rho_k(Q_y(\mathbf{I}))\} \leq (10\sqrt{2})^{\ell-1} \cdot \rho_k(Q_u(\mathbf{I})),$$

where the first inequality holds by our hypothesis on $x, y$ and the second inequality follows from the observation in the proof of Lemma 3.3.

For an item $b$ and a set of items $A$, let $\mathsf{NN}(b, A)$ denote the nearest neighbor of $b$ in $A$, i.e., $\mathsf{NN}(b, A) = \arg\min_{a \in A} \phi(b, a)$. Let $O_1$ be the optimum cohesive $k$-summary of $Q_u(\mathbf{I})$, i.e., $\rho(O_1, Q_u(\mathbf{I})) = \rho_k(Q_u(\mathbf{I}))$. For each center $p \in O_1$, let $\bar{p} = \mathsf{NN}(p, \bar{S}_u)$. From Lemma 3.3, it holds that $\phi(p, \bar{p}) \leq \sqrt{2}r^*$. Consider the set of $k$ centers $O_2 = \{\bar{p} \mid p \in O_1\}$. For any tuple $x \in \bar{S}_u$, let $\hat{x} = \mathsf{NN}(x, Q_u(\mathbf{I}))$. From Algorithm 2, it holds that $\phi(x, \hat{x}) \leq \sqrt{2}r^*$. So, for an arbitrary tuple $s \in \bar{S}_u$, if $o = \mathsf{NN}(\hat{s}, O_1)$, it holds $\phi(s, \bar{o}) \leq \phi(s, \hat{s}) + \phi(\hat{s}, o) + \phi(o, \bar{o}) \leq \sqrt{2}r^* + \rho_k(Q_u(\mathbf{I})) + \sqrt{2}r^*$. Hence, we have $\rho_k(\bar{S}_u) \leq \rho_k(Q_u(\mathbf{I})) + 2 \cdot \sqrt{2}r^*$. Implied by [38], $\rho(S_u, \bar{S}_u) \leq 2\rho_k(\bar{S}_u)$. Finally, we come to

$$\rho(S_u, Q_u(\mathbf{I})) \leq \rho(S_u, \bar{S}_u) + \sqrt{2}r^* = r_u \leq (5\sqrt{2}(10\sqrt{2})^{\ell-1} + 2) \cdot \rho_k(Q_u(\mathbf{I})) < (10\sqrt{2})^{\ell} \rho_k(Q_u(\mathbf{I})).$$

To show $\rho(S_u, Q_u(\mathbf{I})) \geq \rho_k(Q_u(\mathbf{I}))/2$, we resort to the *generalized cohesive k-summary* problem by relaxing the condition that $S \subseteq Q_u(\mathbf{I})$. Let $\tilde{\rho}_k(Q_u(\mathbf{I}))$ be the optimal solution of the generalized cohesive $k$-summary problem of $Q_u(\mathbf{I})$. From the triangle inequality, $\frac{1}{2} \cdot \rho_k(Q_u(\mathbf{I})) \leq \tilde{\rho}_k(Q_u(\mathbf{I})) \leq \rho_k(Q_u(\mathbf{I}))$. By definition, $\rho(S_u, Q_u(\mathbf{I})) \geq \tilde{\rho}_k(Q_u(\mathbf{I})) \geq \frac{1}{2} \cdot \rho_k(Q_u(\mathbf{I}))$. □

Proof of Lemma 3.6. $\sum_{i \in [m]} w_i(t.\mathbf{A}_i) = \sum_{i \in [m]} \sum_{A_j \in \bar{\mathbf{A}}_i} \left(t.A_j - q_h.A_j\right)^2 = \phi^2(t, q_h)$. □

### B.1.1 *Coreset without having an upper bound on the clustering cost.*

Let $S$ be a $O(1)$-cohesive $k$-summary. We do not know the value of $r$, however, we know that the approximation ratio is $\beta$, where $\beta$ is a constant. The high-level idea of our algorithm is as follows: We run a binary search over all possible $L_\infty$ distances in $Q(\mathbf{I})$. For a distance $\ell$ we check whether all tuples in $Q(\mathbf{I})$ can be covered with balls having centers the points in $S$ and radius $\sqrt{d}\ell$. In the end, we find a number $r$ such that $\rho_k(Q(\mathbf{I})) \leq \rho(S, Q(\mathbf{I})) \leq r \leq c \cdot \beta\rho_k(Q(\mathbf{I}))$, for a constant $c$, and we execute the algorithm from Section 3.1.3. Let $A$ be a sorted array of the values among all the attributes in the database. We run a binary search on the pairwise distances of $A$. In [60] the authors show that the $j$-th smallest $L_\infty$ distance (or equivalently $L_1$ distance in $\mathbb{R}^1$) of $n$ points on a line can be computed in $O(n \log n)$ time. Let $\ell$ be a value we check in the binary search. Let $\hat{\ell} = \sqrt{d}\ell$. We define the grid $G_{\hat{\ell}}$ having grid cells with diagonal $\mu_{\hat{\ell}} = \varepsilon\hat{\ell}$. For each center $s \in S$, we define the ball $B_s$ of radius $\hat{\ell}$ and center $s$. Let $B_{\hat{\ell}} = \bigcup_{s \in S} B_s$. We check whether $B_{\hat{\ell}}$ covers all tuples in $Q(\mathbf{I})$. Unfortunately, we cannot visit all tuples in $Q(\mathbf{I})$ and the complexity of constructing the union of $k$ balls is large. Instead, we visit each grid cell $g \in G_{\hat{\ell}}$ such that $g$ is contained or partially intersected by $B_{\hat{\ell}}$. Let $\hat{G}_{\hat{\ell}}$ be the set of these grid cells. For each $g \in \hat{G}_{\hat{\ell}}$ we run a counting query using Lemma 2.3 to get $f_g = |g \cap Q(\mathbf{I})|$. We also run an additional counting query in a rectangle that contains all tuples to find $f = |Q(\mathbf{I})|$. If $\sum_{g \in \hat{G}_{\hat{\ell}}} f_g = f$ then we continue the binary search with smaller values of $\ell$. Otherwise, we continue with larger values of $\ell$. Let $\ell^*$ be the parameter in the last iteration of the binary search that $\hat{G}_{\hat{\ell}^*}$ covered all tuples in $Q(\mathbf{I})$. We set $r = (1+\varepsilon)\sqrt{d}\ell^*$ and then we follow the same procedure as in Subsection 3.1.3 to construct $P_\varepsilon$.

Lemma B.1. $\rho_k(Q(\mathbf{I})) \leq \rho(S, Q(\mathbf{I})) \leq r \leq (1+\varepsilon)\sqrt{d}\beta\rho_k(Q(\mathbf{I}))$.

Proof. For a vector $x \in \mathbb{R}^d$, let $||x||_\infty$ be its $L_\infty$ norm, and let $||x||_2$ be its $L_2$ norm. By definition, for two vectors $x, y \in \mathbb{R}^d$, we have $||x-y||_2 = \phi(x, y)$. It is known that for any vector $a \in \mathbb{R}^d$ it holds,

$||a||_\infty \le ||a||_2 \le \sqrt{d}||a||_\infty$. For every tuple $t \in Q(\mathbf{I})$, it is always true that there exists $s_t \in S$ with $||t - s_t||_2 \le \rho(S, Q(\mathbf{I})) \le \beta\rho_k(Q(\mathbf{I}))$. Let $t' \in Q(\mathbf{I})$ be the tuple such that $||s - s_{t'}||_2 = \rho(S, Q(\mathbf{I}))$. Let $\ell = ||s - s_{t'}||_\infty$. From above, it follows that $||s - s_{t'}||_\infty \ge \frac{\rho(S,Q(\mathbf{I}))}{\sqrt{d}}$. Hence, there exists an $L_\infty$ distance $\ell_a$ in $A$ satisfying $\ell_a \ge \frac{\rho(S,Q(\mathbf{I}))}{\sqrt{d}} \Leftrightarrow \sqrt{d}\ell_a \ge \rho(S, Q(\mathbf{I}))$. Hence, for any $\ell \ge \ell_a$, we have $\sum_{g\in\hat{G}_{\hat{\ell}}} f_g = f$. So $\ell^* \le \ell_a$. Next, we show that $\ell^* \ge \frac{\rho(S,Q(\mathbf{I}))}{(1+\varepsilon)\sqrt{d}}$ by contradiction. Assume $\ell^* < \frac{\rho(S,Q(\mathbf{I}))}{(1+\varepsilon)\sqrt{d}} \Leftrightarrow \sqrt{d}\ell^* < \frac{\rho(S,Q(\mathbf{I}))}{1+\varepsilon}$. Our grid-based algorithm for checking whether all points in $Q(\mathbf{I})$ lie in the cells $\hat{G}_{\hat{\ell}^*}$, counts all tuples within distance $\sqrt{d}\ell^*$ from centers $S$ and might count some tuples within distance $(1+\varepsilon)\sqrt{d}\ell^*$ from centers in $S$. Hence, if $(1+\varepsilon)\sqrt{d}\ell^* < \rho(S, Q(\mathbf{I}))$, our algorithm will return $\sum_{g\in\hat{G}_{\hat{\ell}}} f_g < f$. Overall, $\frac{1}{1+\varepsilon} \cdot \rho(S, Q(\mathbf{I})) \le \sqrt{d}\ell^* \le \sqrt{d}\ell_a \le \sqrt{d}\rho(S, Q(\mathbf{I})) \le \sqrt{d}\beta\rho_k(Q(\mathbf{I}))$. □

**Correctness** With Lemma B.1 and algorithm in Section 3.1.3, we can construct a desired coreset.

**Complexity.** For any $\ell$, there is at most $O(\varepsilon^{-d})$ grid cells with diagonal $\varepsilon\sqrt{d}\ell$ that intersect or fully contained in a ball of radius $\sqrt{d}\ell$. Hence, in each iteration of the binary search we spend $O(kN\varepsilon^{-d})$ to run the counting queries. Using [60] to get the $j$-th smallest $L_\infty$ distance among $A$, we need $O(N \log^2 N)$ additional time to execute the binary search. Overall, given $S$, we construct an $\varepsilon$-coreset in $O(N \log^2(N) + kN \log(N)\varepsilon^{-d})$ time using $O(N + k\varepsilon^{-d})$ space.

### B.2 Missing details from Subsection 3.2

**Correctness.** By definition, for a fixed $r_H$, all rectangles in $\mathcal{R}^{(t)}$ are disjoint and $p \in \mathcal{R}^{(t)}$ if and only if $\phi_H(t, p) \ge r_H$. Indeed, if $p \in \mathcal{R}_h^{(t)}$, for $h \ge r_H$, then the tuple $p$ has different values than $t$ in exactly $h$ attributes. Furthermore, for a fixed $r_H$, the condition in line 13 is satisfied if and only if $\rho(S, Q(\mathbf{I})) \ge r_H$. Indeed, a cell $\psi$ in $\mathcal{M}(\mathcal{R})$ has density $k$ if and only if every point $x \in \psi$ has distance at least $r_H$. Hence, if there is no tuple from $Q(\mathbf{I})$ in these cells, then every tuple in $Q(\mathbf{I})$ is within distance $r_H$ from $S$, i.e., $\rho(S, Q(\mathbf{I})) \le r_H - 1$. We show that $\rho(S, Q(\mathbf{I})) \le 2\rho_k(Q(\mathbf{I}))$. Equivalently, we show that for $r_H = 2\rho_k(Q(\mathbf{I})) + 1$ the condition in line 13 always holds. Let $S^*$ be the optimally cohesive $k$-summary of $Q(\mathbf{I})$. For every tuple $t_j \in S^*$, we define the ball $\mathcal{B}_j$ with center $t_j$ and radius $\rho_k(Q(\mathbf{I}))$. By definition, the union of all such $k$ balls covers all tuples in $Q(\mathbf{I})$. Let $t$ be the tuple that is selected in lines 10-11 of Algorithm 6 in an iteration $i$. Without loss of generality assume that $t$ belongs in the ball $\mathcal{B}_j$. The Hamming distance satisfies the triangle inequality, so for any $p \in \mathcal{B}_j \cap Q(\mathbf{I})$, it holds $\phi_H(t, p) \le \phi_H(t, t_j) + \phi_H(t_j, p) \le 2\rho_k(Q(\mathbf{I}))$. In other words, it holds that for any new tuple $t$ we add in $S$, the ball $\mathcal{B}'_t$ with center $t$ and radius $2\rho_k(Q(\mathbf{I}))$ completely covers a ball from the optimally cohesive $k$-summary. By definition, the union of the rectangles in $\mathcal{R}^{(t)}$ is the complement of ball $\mathcal{B}'_t$. After $k$ iterations all optimal $k$ balls are covered by the balls $\bigcup_{t\in S} \mathcal{B}'_t$, so there is no tuple in $Q(\mathbf{I})$ that lies in the complement of $\bigcup_{t\in S} \mathcal{B}'_t$. Equivalently, there is no tuple in $Q(\mathbf{I})$ that lies in a cell of density $k$ in line 13, so the condition is satisfied.

**Complexity.** The algorithm runs for $k$ iterations. In each iteration $i$, we add a tuple $t$ in $S$ and we construct $|\mathcal{R}^{(t)}| = O(1)$ rectangles. The decomposition $\mathcal{M}(\mathcal{R})$ is updated in $k^d$ time [10]. The cells with depth $i-1$ can also be found in $k^{O(d)}$ time after updating the decomposition. We run a rectangular query for every cell of depth $i-1$. From Lemma 2.3, each rectangular query takes $O(N)$ time. Overall, our algorithm runs in $O(Nk^d)$ time and uses $O(N + k^d)$ space.

## C Missing details from Section 5

Proof of Lemma 5.2. $\sum_{i\in[m]} w_i(t.\mathbf{A}_i) = \sum_{i\in[m]} \sum_{A_j\in\bar{\mathbf{A}}_i} (t.A_j) \cdot u_j = \sum_{A_j\in\mathbf{A}} (t.A_j) \cdot u_j = \langle u, t\rangle$. □

Lemma C.1. *The Hamming metric is of negative type.*

Proof. Using [35, 61], any distance function $\sigma$, over a set of $n$ items $P \in \mathbb{R}^d$, is of negative type if there exists a mapping from $P$ to $P' \in \mathbb{R}^{d'}$ for a positive integer $d'$, such that $\sigma(p,q) = \sigma'(p',q')$, where $p, q \in P$, $p'$ is the mapping of $p$ (similarly, $q'$ is the mapping of $q$), and $\sigma'$ is the *squared Euclidean distance*.

For the Hamming metric, we map all points in $P \in \mathbb{R}^d$ to points in $d' \leq n \cdot d^2$ dimensions as follows. Let $L$ be the ordered list of all distinct values (coordinates) over the points in $P$. Clearly, $|L| \leq nd$. We use $p_j$ to denote the $j$-th value of point $p \in P$. For every $p \in P$, we create $p' \in \mathbb{R}^{|L|d}$ as follows. For each $j \in [d]$, we create the zero vector $\vec{X}^{(j)} \in \mathbb{R}^{|L|}$, such that $\vec{X}^{(j)} = (0, \ldots, 0)$. Without loss of generality, assume that $p_j = L[i]$, i.e., the $j$-th value of $p$ is the $i$-th element in list $L$. We set $\vec{X}^{(j)}_i = 1/\sqrt{2}$, i.e., the $i$-th value of $\vec{X}^{(j)}$ is set to $1/\sqrt{2}$. Then $p'$ is the concatenation of all vectors $\vec{X}^{(j)}$, i.e., $p' = [\vec{X}^{(1)}, \ldots, \vec{X}^{(d)}]$. By definition, it holds that under the Hamming metric $\phi_H(p,q) = \sum_{1 \leq h \leq d|L|} (p'_h - q'_h)^2$. Intuitively, if two points have different $j$-th value then the squared Euclidean will sum up the terms $(1/\sqrt{2} - 0)^2 + (0 - 1/\sqrt{2})^2 = 1$. □

**Greedy algorithm for Sum-Diverse Summaries.** In Algorithm 12, we start with an arbitrary tuple $x_0 \in Q(\mathbf{I})$ and add it to $S$. We repeat the following step until the size of $S$ reaches $k$. We construct an index $Z$ that enumerates the tuples in $Q(\mathbf{I})$ in descending order to their sum of distances with the tuples in $S$. More specifically, for every tuple $p \in R_i$, we define the weight $w_i(p) = \sum_{y \in S} \phi_H\left(p.\bar{\mathbf{A}}_i, y.\bar{\mathbf{A}}_i\right)$. All tuples in $Q(\mathbf{I})$ are enumerated from $Z$ in the descending ordering, until we encounter some result $y \notin S$. We add $y$ to $S$ and continue in the next iteration.

**Algorithm 12:** GreedySUMDiverse$(Q, \mathbf{I}, k)$

```
1  S ← {x_0} for an arbitrary tuple x_0 ∈ Q(I);
2  while |S| ≤ k do
3      foreach i ∈ [m] do
4          foreach p ∈ R_i do
5              w_i(p) = Σ_{y∈S} φ_H(p.Ā_i, y.Ā_i);
6      w⃗ ← ⟨w_1, w_2, ··· , w_m⟩;
7      Z ← index built for Q, I, w⃗ as Lemma 2.1;
8      while true do
9          y ← a result enumerated from Z;
10         if y ∉ S then break;
11     S ← S ∪ {y};
12 return S;
```

**Correctness.** For $\vec{w}$ at line 6 of Algorithm 12, $\vec{w}(t) = \sum_{i \in [m]} w_i(t.\mathbf{A}_i) = \sum_{i \in [m]} \sum_{y \in S} \phi_H\left(t.\bar{\mathbf{A}}_i, y.\bar{\mathbf{A}}_i\right) = \sum_{y \in S} \phi_H(t, y)$ for every tuple $t \in Q(\mathbf{I})$. Ravi et al. [59], showed that the greedy algorithm returns a $\frac{1}{2}$-sum-diverse $k$-summary in the non-relational setting. Algorithm 12 implements the greedy algorithm in the relational setting so it also returns a $\frac{1}{2}$-sum-diverse $k$-summary for $Q(\mathbf{I})$.

**Complexity.** For every tuple $p \in R_i$, it takes $O(\log N)$ time to compute $w_i(p)$. It takes $O(N)$ time to construct the index $Z$. For every tuple $y$ returned by $Z$, it takes $O(\log k) = O(\log N)$ time to check if $y \in S$. All tuples can be enumerated from $Z$ within $O(\log N)$ delay. We need to enumerate at most $k-1$ results until we find one that does not belong in $S$. Our algorithm runs in $O(Nk \log N + k^2 \log N)$ time and uses $O(N + k)$ space.

**Remark.** This algorithm cannot be used for the Euclidean metric because the sum of distances from a tuple $t$ to set $S$ is a sum of square roots, so we cannot use the squared Euclidean metric as we did in the algorithms from Section 3 in the ranked enumeration index.

## D Extension to Cyclic Join

**Generalized Hypertree Decomposition [42].** We need the following notions to extend our results to cyclic joins. We also use a triple $(\mathbf{A}, \mathcal{E}, \mathbf{y})$ to represent a CQ $Q$, where $\mathcal{E} = \{\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_m\}$.

*Definition D.1 (Generalized Hypertree Decomposition).* Given a join $Q = (\mathbf{A}, \mathcal{E})$, a GHD of $Q$ is a pair $(\mathcal{T}, \lambda)$, where $\mathcal{T}$ is a tree as an ordered set of nodes and $\lambda : \mathcal{T} \to 2^{\mathbf{A}}$ is a labeling function which associates to each node $u \in \mathcal{T}$ a subset of attributes in $\mathbf{A}$, $\lambda_u$, such that the following conditions are satisfied: (1) For each relation $\mathbf{A}_i \in \mathcal{E}$, there is a node $u \in \mathcal{T}$ such that $\mathbf{A}_i \subseteq \lambda_u$; (2) For each attribute $A \in \mathbf{A}$, the set of nodes $\{u \in \mathcal{T} : A \in \lambda_u\}$ forms a connected subtree of $\mathcal{T}$.

*Definition D.2 (Fractional Edge Covering Number of CQ).* Given a CQ $Q = (\mathbf{A}, \mathcal{E}, \mathbf{y})$, a function $W : [m] \to [0, 1]$ is a fractional edge covering of $Q$ if $\sum_{i \in [m]: A \in \mathbf{A}_i} W(i) \geq 1$ holds for any attribute $A \in \mathbf{A}$. The weight of $W$ is defined as $\sum_{i \in [m]} W(i)$. The fractional edge covering number of $Q$ is the minimum weight of all possible fractional edge coverings of $Q$.

Given a GHD $(\mathcal{T}, \lambda)$ for a join $Q$, each node $u$ derives a subjoin over attributes $\lambda_u$ and relations $\mathcal{E}_u = \{e \cap u : e \in \mathcal{E}\}$. The width of each node $u \in \mathcal{T}$ is defined as the fractional edge covering number of $(\mathbf{A}_u, \mathcal{E}_u)$. The width of $(\mathcal{T}, \lambda)$ is defined as the maximum width over all nodes in $\mathcal{T}$. Then, the fractional hypertree width of a join follows:

*Definition D.3 (Fractional Hypertree Width [42]).* The fractional hypertree width of a join $Q$, denoted as $\mathsf{fhtw}(Q)$, is $\mathsf{fhtw}(Q) = \min_{(\mathcal{T}, \lambda)} \max_{u \in \mathcal{T}} \rho(\lambda_u, \mathcal{E}_u)$, i.e., the minimum width over all GHDs.

Basically, $O(N^{\mathsf{fhtw}})$ is an upper bound on the number of join results materialized for each node in $\mathcal{T}$, as well as the time complexity of computing the join results [14]. Thus, we can generalize all our results to cyclic joins. If the runtime of an algorithm for the acyclic join was $T(N, k, \varepsilon)$, it now becomes $T(N^{\mathsf{fhtw}}, k, \varepsilon)$.

## E Extension to join-project queries

As we did for join queries, we focus on acyclic join-project queries and then use the well-known GHD shown in Section 6 and Appendix D that maps any cyclic instance of input size $N$ to an acyclic instance of input size $N^{\mathsf{fhtw}}$. Hence, all our algorithms for acyclic join-project queries can be extended to cyclic join-project queries with the same approximation guarantees. The running time changes from $O(N \cdot f(k))$, where $f(\cdot)$ is a function of $k$, to $O(N^{\mathsf{fhtw}} \cdot f(k))$.

We first describe the high-level idea. Recall that $\mathbf{y}$ is the set of the output attributes. Recall that $d = |\mathbf{y}|$. A summary $S$ should be computed with respect to attributes only in $\mathbf{y}$, i.e., $S \subset \mathbb{R}^d$. For a relation $R_i$, let $R'_i = \pi_{\mathbf{y} \cap \mathbf{A}_i}(R_i)$, the projection of the tuples in $R_i$ on the output attributes $\mathbf{A}'_i = \mathbf{y} \cap \mathbf{A}_i$. All our algorithms run almost verbatim using $R'_i$ instead of $R_i$. Of course, the original relations $R_i$ are still used to identify the joined tuples. In fact, all our algorithms are straightforwardly extended to join-project queries if we use a nearest neighbor, farthest neighbor, top-$k$, and rectangular oracles that work on join-project queries. Next, we introduce such oracles.

**Ranked enumeration.** For simplicity, let $\bar{\mathbf{A}}'_i = \mathbf{A}'_i - (\bigcup_{j<i} \mathbf{A}'_j)$ be the set of active attributes for $R'_i$ i.e., the set of output attributes that do not appear in any relation before $R'_i$. Let $w_i : \mathbb{R}^{|\mathbf{A}_i|} \to \mathbb{R}$ be a weight function, which takes as input a tuple $t \in R_i$ and outputs a real number. Let $\vec{w} = \langle w_1, w_2, \cdots, w_m \rangle$ be a set of weight functions. For a CQ $Q$, a database $\mathbf{I}$, and a pair of results $t_1, t_2 \in Q(\mathbf{I})$, we say $t_1 \preceq_{\vec{w}} t_2$ if $\sum_{j \in [m]} w_j(t_1.\mathbf{A}'_j) \leq \sum_{j \in [m]} w_j(t_2.\mathbf{A}'_j)$. We use [32] instead of [33] to perform ranked enumeration of join-project queries.

Lemma E.1 ([32]). *For an acyclic join-project $Q$, a database $\mathbf{I}$, and a set of weight functions $\vec{w} = \langle w_1, w_2, \cdots, w_m \rangle$, an index of size $O(N)$ can be constructed in $O(N)$ time, such that given any value*

*$k \in \mathbb{Z}^+$, the top-$k$ results of $Q(\mathbf{I})$ can be enumerated in ascending or descending order with respect to $\vec{w}$ within $O(N \log N)$ delay.*

We note that the weights in [33] are defined on the attributes rather than on the tuples. However, from our construction, it is straightforward to also design weight functions on the attributes. Hence, for simplicity, we follow the definition of weight functions we had in all previous sections.

**Euclidean-based oracles.** Let $\theta \in \mathbb{R}^d$ be a tuple. The *nearest neighbor oracle* finds a tuple $t \in Q(\mathbf{I})$ such that $\phi(\theta, t)$ is minimized. The *farthest neighbor oracle* finds a tuple $t \in Q(\mathbf{I})$ such that $\phi(\theta, t)$ is maximized. For each relation $R_i$, we define $w_i(\cdot)$ as: $w_i(p.\mathbf{A}'_i) = \sum_{A_j \in \bar{\mathbf{A}}'_i} (p.A_j - \theta.A_j)^2$, where $p \in R_i$. If $\bar{\mathbf{A}}'_i = \emptyset$ then $w_i(p.\mathbf{A}'_i) = 0$. Thanks to the *decomposability* of squared Euclidean distance, for any query result $t \in Q(\mathbf{I})$, $\sum_{i\in[m]} w_i(t.\mathbf{A}'_i) = \sum_{i\in[m]} \sum_{A_j \in \bar{\mathbf{A}}'_i} (t.A_j - \theta.A_j)^2 = \sum_{A\in\mathbf{y}} (t.A - \theta.A)^2 = \phi^2(\theta, t)$, The square (and square root) function is increasing for non-negative values, so the order of the distances with respect to the squared Euclidean distance is the same as the order of the distances with respect to the Euclidean distance.

**Top-$k$ oracle.** Let $u = \langle u_1, u_2, \ldots, u_d \rangle$ be a vector in $\mathbb{R}^d$. The *top-$k$* oracle finds the $k$ tuples in $Q(\mathbf{I})$ with the largest inner product with respect to $u$. For each relation $R_i$, we define $w_i(\cdot)$ as: $w_i(p.\mathbf{A}'_i) = \sum_{A_j \in \bar{\mathbf{A}}'_i} (p.A_j) \cdot u_j$, where $p \in R_i$. It is easy to show that for any query result $t \in Q(\mathbf{I})$, $\sum_{i\in[m]} w_i(t.\mathbf{A}'_i) = \sum_{i\in[m]} \sum_{A_j \in \bar{\mathbf{A}}'_i} (t.A_j) \cdot u_j = \langle t, u \rangle$.

Lemma E.2. *Given an acyclic join-project $Q$ with $d$ output attributes, a database instance $\mathbf{I}$ with input size $N$, and a tuple $\theta \in \mathbb{R}^d$, a set of weight functions $\vec{w}$ can be constructed in $O(N)$ time, such that the nearest (resp. farthest) neighbor of $\theta$ in $Q(\mathbf{I})$, $\arg\min_{t\in Q(\mathbf{I})} \phi(\theta, t)$ (resp. $\arg\max_{t\in Q(\mathbf{I})} \phi(\theta, t)$), can be computed in $O(N \log N)$ time. Similarly, given a vector $u \in \mathbb{R}^d$, a set of weight functions $\vec{w}$ can be constructed in $O(N)$ time, such that the $k$ tuples in $Q(\mathbf{I})$ with the highest inner product with $u$ can be computed in $O(Nk \log N)$ time.*

**Rectangular oracle.** Similarly to the rectangular oracle in the join queries, we can find all tuples in $\mathbf{I}$ that pass the predicate, defined by the rectangle, and then apply the index from [32] to enumerate the query results on the surviving tuples.

Lemma E.3. *Given an acyclic join-project $Q$ with $d$ output attributes, a database instance $\mathbf{I}$ with input size $N$, and a rectangle $\psi \in \mathbb{R}^d$, an index of size $O(N)$ can be constructed in $O(N)$ time such that all results in $\psi \cap Q(\mathbf{I})$ can be enumerated with $O(N \log N)$ delay.*

Replacing the oracles we used in the main part with the oracles defined in this section, we conclude with the following results for cohesive and diverse summaries under the Euclidean metric.

Corollary E.4. *For an acyclic join-project $Q$ of $d$ output attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon > 0$, a $(2+\varepsilon)$-cohesive $k$-summary for $Q(\mathbf{I})$ under the Euclidean metric can be computed in $O\left(k^2 N \log N + kN \log(N)\varepsilon^{-d}\right)$ time.*

Corollary E.5. *For an acyclic join-project $Q$ of $d$ output attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon > 0$, a $\left(\frac{1}{2} - \varepsilon\right)$-min-diverse $k$-summary of $Q(\mathbf{I})$ under the Euclidean metric can be computed in $O\left(k^2 N \log N + kN \log(N)\varepsilon^{-d}\right)$ time.*

Corollary E.6. *For an acyclic join-project $Q$ of $d$ output attributes, a database $\mathbf{I}$ of input size $N$, and a parameter $\varepsilon \in (0, \frac{1}{2})$, a $\left(\frac{1}{2} - \varepsilon\right)$-sum-diverse $k$-summary of $Q(\mathbf{I})$ under the Euclidean metric can be computed in $O\left(kN \log(N)\varepsilon^{-(d-1)/2}\right)$ time.*

Equivalently, using the oracles defined in this section, we can derive the results for constructing cohesive and diverse summaries under the Hamming metric.

Received May 2024; revised August 2024; accepted September 2024