Relational Approximations for Subspace Primitives

Xiang Liu
Department of Computer Science, University of Iowa, Iowa City, IA, USA

Kasturi Varadarajan
Department of Computer Science, University of Iowa, Iowa City, IA, USA

Abstract

We explore fundamental geometric computations on point sets that are given to the algorithm implicitly. In particular, we are given a database which is a collection of tables with numerical values, and the geometric computation is to be performed on the join of the tables. Explicitly computing this join takes time exponential in the size of the tables. We are therefore interested in geometric problems that can be solved by algorithms whose running time is a polynomial in the size of the tables. Such *relational algorithms* are typically not able to explicitly compute the join.

To avoid the NP-completeness bottleneck, researchers assume that the tables have a tractable combinatorial structure, like being acyclic. Even with this assumption, simple geometric computations turn out to be non-trivial and sometimes intractable. In this article, we study the problem of computing the maximum distance of a point in the join to a given subspace, and develop approximation algorithms for this NP-hard problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases relational algorithm, Euclidean distance, subspace approximation

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2025.12

Category APPROX

Acknowledgements The authors thank Kirk Pruhs for discussions that led to the problems studied here, and the organizers of the 2023 Workshop on Fined-Grained Complexity, Logic, and Query Evaluation at the Simons Institute for the Theory of Computing for facilitating these discussions.

1 Introduction

In this paper, we consider certain fundamental geometric computations that are trivially solvable in linear or quadratic time in the size of the input point set. In our setting, the input point set is not given explicitly, but rather in an implicit form as described below. An explicit representation can be exponentially larger than the implicit one, and the question we pursue is whether these geometric computations can be performed in time polynomial in the size of the implicit representation.

We are given a set $\{C_1, C_2, \ldots, C_m\}$ where each $C_j \subseteq \mathbb{Z}^+$ is a subset of the positive integers. We interpret each element of C_j as a coordinate axis or *feature*. Furthermore, for each $1 \leq j \leq m$, we are given a table (two-dimensional matrix) T_j with at least one row and exactly $|C_j|$ columns, where each column corresponds to a feature in C_j. The values in the table T_j are real numbers. Given a row r in T_j, we let $r[i] \in \mathbb{R}$ (or r_i) denote the value corresponding to feature $i \in C_j$. We assume tables don't have duplicate rows, and thus a table specifies a set of row vectors.

We define the join $T_1 \bowtie T_2 \cdots \bowtie T_m$ of $m \geq 1$ such tables as a table J whose columns correspond to the union $C = \bigcup_{j=1}^{m} C_j$ of the individual feature sets. Thus each row in the join is a vector with $|C|$ real components. Such a vector q belongs to the join J if in each table T_j, there is a row r^j such that q and r^j agree on the features in C_j – for each $i \in C_j$, $q[i] = r^j[i]$.

Thus, a row q in the join J is generated by picking rows $r^1 \in T_1, r^2 \in T_2, \ldots, r^m \in T_m$ that are pairwise compatible, that is, agreeing on the value on common features; and then "concatenating" these rows. That is, for each feature $i \in C$, we find an arbitrary C_j containing feature i, and let $q[i] := r^j[i]$. The join J consists of the set of all rows that can be generated in this way. An example of a join with feature set $\{a, b, c\}$ is shown below:

T_1	
a	b
1	1
1	2
2	1

T_2	
b	c
1	2
2	3
5	6

$J = T_1 \bowtie T_2$		
a	b	c
1	1	2
2	1	2
1	2	3

A relational database stores its data in the form of multiple tables as above. However, typical data analysis techniques such as clustering only work when the input data is in explicit form, as in the join J. Standard practice is thus to compute the join, and then run the data analysis algorithm on it. Let $d = |C|$ denote the total number of features and n the maximum number of rows in any given table. The space needed to store all the tables is easily upper bounded by $O(mnd)$, where we recall that m is the number of tables. However, the number of rows in the join $J = T_1 \bowtie T_2 \cdots \bowtie T_m$ is $\Theta(n^m)$ in the worst case, which can be exponentially larger. This naturally raises the following question: what properties of the join J can we compute using *relational algorithms*, defined as algorithms with running time polynomial in n, m, and d? See [2, 1, 9, 12] and the references therein. Notice that a relational algorithm typically is not able to explicitly construct the join.

Before describing prior work on relational algorithms, we point out the geometric nature of the join. The rows of each table T_j can be viewed as a set of points lying in the subspace spanned by the features/coordinate axes in C_j. The join $J = T_1 \bowtie T_2 \cdots \bowtie T_m$ is the set of *all* points p, in the subspace spanned by features in C, such that the projection of p onto the subspace spanned by C_j is an element of T_j, for each j.

1.1 Prior Work

Perhaps the most basic algorithmic question to ask about the join J of the tables $T_1, T_2, \ldots, T_m$ is whether it is non-empty – does it have at least one row/point in it? This turns out to be NP-Complete [7, 10, 9]. Indeed, a simple reduction from 3CNF-SAT has the property that the number of rows in the join is exactly the number of satisfying assignments to the input formula. (This is a good spot to emphasize that in this paper, the number d of features and the number m of tables are not viewed as constants.)

Given this state of affairs, the approach taken by papers in this area is to assume that the join has some additional structure that makes such basic algorithmic tasks tractable. The most common assumption is that the join is acyclic – this is a combinatorial assumption about how the features are distributed across the tables, and we define it in the next section. Assuming acyclic joins lets us use dynamic programming to solve certain algorithmic questions in polynomial time [2, 1]:

1. Compute the number of rows in the join J
2. Compute the maximum of the squared l_2 norm of rows in the join, that is, $\max_{q \in J} \sum_{i \in C} q_i^2$.

These algorithmic tasks are special cases of a *sum-product query*. This is a query $Q(J)$ over the join J of the form

$$Q(J) = \bigoplus_{q \in J} \bigotimes_{i \in C} F_i(q_i),$$

where $(R, \bigoplus, \bigotimes)$ is a commutative semiring over a set R. Corresponding to feature $i \in C$, $F_i : \mathbb{R} \to R$ is an easy to compute function with range R. For example, for the query asking for the number of rows in the join, R is the set of integers, F_i is the constant function that takes on the value 1, $\bigotimes$ is multiplication over the integers, and $\bigoplus$ is addition over the integers. For computing the maximum of the squared l_2 norm of the rows, R is the set of real numbers, $F_i(x) = x^2$, $\bigotimes$ is addition over the reals, and $\bigoplus$ is the max function.

The authors in [9] study sum-product queries under constraints on the rows of the join. They allow constraints of the form $\sum_{i \in C} g_i(q_i) \leq \beta$, where $g_i : \mathbb{R} \to \mathbb{R}$ is an easy-to-compute function, and β is a constant. They consider the problem of evaluating a sum-product query over those rows in the join that satisfy the given set of constraints. They show that with two constraints, such problems are not only NP-hard but also hard to approximate to within any multiplicative factor. Concretely, they show it is NP-hard to determine if there is a row q in the join J that satisfies the linear constraint $\sum_i q_i = 0$. This is via a reduction from the Partition problem, where we want to determine if a given set of positive integers can be partitioned into two sets whose sums are equal. Thus, it is a weak NP-hardness result [6]. The linear constraint $\sum_i q_i = 0$ is equivalent to the two linear inequality constraints $\sum_i q_i \geq 0$ and $\sum_i q_i \leq 0$.

On the other hand, the authors in [9] show that with just one inequality constraint, and some additional technical assumptions, there is a randomized approximation scheme for sum-product queries that guarantees a multiplicative $(1 + \epsilon)$-approximation, for any given parameter $\epsilon > 0$. For instance, they use this general result to get a polynomial-time $(1 + \epsilon)$-approximation for the following counting problems:

1. Count the number of rows in the join that lie in a given half space $\sum_i a_i q_i \leq \beta$.
2. Count the number of rows in the join that lie within a given ball $\sum_i (q_i - y_i)^2 \leq r^2$ (centered at point y and having radius $r \geq 0$).

The authors in [12] study the k-means problem, and show how the k-means++ algorithm [3] for constructing a coreset can be implemented in the relational setting, thus deriving an $O(1)$-approximation for the k-means problem. Their core technical result is a polynomial-time sampling algorithm that, given a set $c_1, c_2, \ldots, c_j$ of centers in $\mathbb{R}^{|C|}$, outputs a row $q \in J$ with probability proportional to the square of the Euclidean distance of q to the nearest center. This sampling result is surprising given the hardness of closely related problems. In particular, the problem of counting the number of rows in the join whose closest center is c_1 is NP-hard even for $j = 2$, and hard to approximate to within any multiplicative factor for $j = 3$. The technique of rejection sampling plays a key role in their sampling result.

Very recently, the authors in [5] present fast deterministic and randomized relational approximation algorithms for the k-median and k-means clustering problems assuming the dimension d is a constant. The authors in [4] present a method for constructing a coreset for certain optimization problems in machine learning. The work that we have reviewed builds on a body of theoretical and experimental research on relational algorithms. We refer the reader to [9, 12, 5, 4] for surveys of this prior work.

1.2 Our Contribution

From prior work, it is evident that in the relational setting, the study of the complexity of very fundamental geometric problems yields surprising answers. More of the terrain needs to be explored to form a better picture of the computational complexity. Motivated by this, we consider the following *Distance to k-Subspace* problem: Given a set $S = \{s_1, s_2, \ldots, s_k\}$ of orthogonal unit vectors, compute the maximum Euclidean distance of a row in the join to

$span(S)$, the subspace spanned by S. As a reminder, we are assuming an acyclic database. The Distance to Subspace problem is a basic primitive for other tasks including the subspace approximation problem [8]. We obtain the following results.

1. We show, in Section 4, that the Distance to k-Subspace problem is NP-hard even for $k = 1$.
2. We present a polynomial-time $\sqrt{d}$-approximation for the Distance to k-Subspace problem. To do so, we introduce a generalization of the join and compute measures on that generalization. This is likely to be an algorithmic tool of independent interest. As a consequence of our technique, we show that the row rank of the join J can be computed exactly in polynomial time. These results are presented in Section 3.
3. Next, we ask if for constant k, we can get a multiplicative $(1 + \epsilon)$-approximation for the Distance to k-Subspace problem, for any $\epsilon > 0$. The distance of a row q in the join from the given k-subspace can be written as $A(q) - B(q)$, where $A(q)$ and $B(q)$ are both non-negative polynomials. $A(q)$ and $B(q)$ can individually be maximized/minimized using prior work on sum-product queries, but obviously this does not, by itself, give us a way of maximizing the difference $A(q) - B(q)$.
   Instead, we keep track of all possible values of the pair $(A(q), B(q))$ in our dynamic program. However, there can be an exponential number of such pairs, as is to be expected for an NP-hard problem. We can keep track of suitably discretized values of $(A(q), B(q))$, but to get a polynomial-time algorithm using this approach, we need the instance to be well-conditioned in the following sense: the maximum 2-norm of rows in the join (that is, Distance to the 0-Subspace) is at most polynomially larger than the Distance to the given k-subspace.
   Our main technical contribution here is to show that the Distance to k-Subspace problem can be reduced to polynomially many instances of the same problem that are well-conditioned. To obtain such a reduction, one tool we develop is a proof that there exists a k-subspace spanned by k of the standard basis vectors that is "close" to any given k-subspace. Overall, our reduction gives us a $(1 + \epsilon)$-approximation (Section 4) for the Distance to k-Subspace problem.
4. We observe, in Section 4, that our $(1 + \epsilon)$-approximation to Distance to k-Subspace can be used as a primitive to obtain a $(2(1 + \epsilon))^k$-approximation for finding a k-subspace that minimizes the maximum Euclidean distance to the join. When the point set is given explicitly (as opposed to implicitly via the join), this is well known as the l_∞-subspace approximation problem [14]. The running time of our algorithm is polynomial for constant k. Our approximation factor for subspace approximation is large, and this result serves to illustrate the usefulness of the Distance to Subspace approximation primitive, which is the main focus of this work.

We emphasize that our work addresses the regime where the number d of features and the number m of tables are part of the input, and are not treated as constants. We are primarily focussed on the approximation guarantee that we can get with polynomial running time; we do not explore further optimizations of the running time.

2 Preliminaries

In this section, we present essential preliminaries. The database that is input to each of the problems considered here consists of a collection of tables $T_1, T_2, \ldots, T_m$. The columns of table T_j correspond to a subset $C_j \subseteq \mathbb{Z}^+$ of features, and thus each row in T_j is a vector

with $|C_j|$ real-valued components. We use n to denote the maximum number of rows in any table. Let $C = \bigcup_{j=1}^{m} C_j$ denote the set of all features. We will assume that $C = \{1, 2, \ldots, d\}$, where d denotes $|C|$. We assume that the columns in any table are ordered according to the natural ordering $\langle 1, 2, \ldots, d \rangle$ of the features in C. Thus, we can represent a row r in Table T_j unambiguously as a point in $\mathbb{R}^{|C_j|}$. Given such a row r and feature $i \in C_j$, we will use r_i or $r[i]$ to denote the component of r corresponding to feature i.

Given two row vectors r' and r'' over feature sets C' and C'', we say that r' and r'' are *compatible* if for each feature $i \in C' \cap C''$, $r'[i] = r''[i]$. We use the notation $r' \sim r''$ to denote that r' and r'' are compatible. The *concatenation* of compatible rows r' and r'' is the row vector over $C' \cup C''$ formed by combining the components of r' and r'' in the obvious way.

2.1 Acyclic Databases

We say that our database of m tables is *acyclic* if there is a tree τ with $O(m)$ nodes, each associated with one of our input tables, that has the following properties:

1. Each table in the database is associated with at least one node.
2. For each feature $i \in C$, the set of nodes whose tables contain i induces a connected subgraph of the tree.

Note that for technical reasons, we allow a table to be associated with more than one node of the tree τ. Given an acyclic database, such a tree τ can be computed in polynomial time; see for example [9].

Note that each row of the join $J = T_1 \bowtie T_2 \cdots \bowtie T_m$ is formed by picking one row from each table associated with each node of τ, such that the chosen rows are pairwise compatible, and then concatenating the chosen rows. For an acyclic database, it suffices to check compatibility between pairs of rows chosen from nodes that are neighbors in the tree τ. This follows from the properties of an acyclic database. This is the crucial property of acyclic databases that allows one to evaluate certain queries on the join efficiently via dynamic programming.

For the purpose of dynamic programming, it will be convenient to view the tree τ as rooted. Furthermore, we can assume that each internal node of τ has exactly two children. This is without loss of generality: If there is a node α with exactly one child α', we can make α' the left child of α and add a right child whose table is a copy of α's. If a node α has $j > 2$ children, then replace α with a complete binary tree with $j - 1$ internal nodes and j leaves, which will now correspond to the original j children of α. The table at each of the $j - 1$ internal nodes will be a copy of the table at α.

After applying these two operations, we have that each internal node of τ has exactly two children. The total number of nodes in τ is $O(m)$. The join is unchanged: we have merely added copies of some of the original tables. Finally, τ still witnesses the fact that the database is acyclic, as is readily checked. We will refer to τ as the *join tree* of the database.

For a node α in the tree τ, we use T_α to denote the table at node α, and J_α to denote the join of all the tables in the subtree rooted at α. We "assign" each feature $i \in C$ to the highest node in tree τ whose table contains feature i as a column. By highest, we mean the node closest to the root of τ. Let $\widehat{C}_\alpha$ denote the set of features assigned to node α, and C_α denote the set of features assigned to nodes in the subtree rooted at α. For a row r in table T_α, we denote the set of rows in the join J_α that are compatible with r by $J_\alpha \bowtie r$. The notation is reasonable as this set of rows is indeed the result of a join of J_α and a table containing the single row r.

3 Computing the Distance to a given k-Subspace

Assume we are given an acyclic database consisting of tables $T_1, T_2, \ldots, T_m$ and a set of k orthogonal unit vectors $S = \{s_i \mid 1 \leq i \leq k\} \subseteq \mathbb{R}^d$, where (as we recall) the set of features in the database is $C = \{1, 2, \ldots, d\}$. The Distance to k-Subspace problem is that of computing the maximum distance of a row in the join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ from the subspace spanned by S. That is, we want to compute

$$\max_{r \in J} ||r - \sum_{i=1}^{k} (r \cdot s_i) s_i||_2$$

The main result of this section is a $\sqrt{d}$-approximation to this problem. We begin with a scheme for implicitly representing functions $V : J \to \mathbb{R}^\mu$, and posing algorithmic problems concerning this representation.

3.1 An Implicit Representation

Suppose that for each feature $i \in C$, we are given a function $V_i : \mathbb{R} \to \mathbb{R}^\mu$, where $\mu \geq 1$ is identical across all features. We assume that these functions are easily computed. In fact, for the application here, we assume that $V_i(x)$ is given explicitly for each value x of feature i that occurs in any of the input tables. For any any row r in the join J, let $V(r) = \sum_{i \in C} V_i(r_i)$. Finally, let $V(J) = \{V(r) \mid r \in J\}$.

One illustrative example is to let $V_i(x) = (0, 0, ..., \underbrace{x}_{i-\text{th entry}}, ..., 0) \in \mathbb{R}^d$, where we recall that the feature set C is $\{1, 2, \ldots, d\}$. Then for any row r in the join J of the tables, $V(r) = \sum_{i \in C} V_i(r_i) = r$. Thus, $V(J) = J$. We will see more interesting examples shortly.

We are interested in computing measures of the point set $V(J)$. But this seems generally harder for $V(J)$ than for J. For example, there is a polynomial-time algorithm for computing $\max_{r \in J} ||r||_2^2$ using sum-product queries. However, computing $\max_{q \in V(J)} ||q||_2^2$ seems harder. Fortunately, we can show a positive result for the $|| \cdot ||_\infty$ norm.

▶ **Lemma 1.** *There is a polynomial-time algorithm for computing* $\max_{q \in V(J)} ||q||_\infty$.

Proof. For $1 \leq j \leq \mu$, let

$$\Psi_j^{max} = \max_{q \in V(J)} q_j = \max_{r \in J} \sum_{i \in C} V_i(r_i)[j],$$

and

$$\Psi_j^{min} = \min_{q \in V(J)} q_j = \min_{r \in J} \sum_{i \in C} V_i(r_i)[j].$$

That is, Ψ_j^{max} (resp. Ψ_j^{min}) is the maximum (minimum) value of the j-th coordinate over points in $V(J)$. As $\Psi_j^{max} = \max_{r \in J} \sum_{i \in C} V_i(r_i)[j]$, it is a sum-product query over the join J where the commutative semiring is the set of real numbers, the $\bigoplus$ operator is max, and the $\bigotimes$ is addition over the reals. It follows that Ψ_j^{max} and Ψ_j^{min} can be computed in polynomial time. Let $\Psi_j = \max\{|\Psi_j^{max}|, |\Psi_j^{min}|\}$.

Finally, we note that $\max_{q \in V(J)} ||q||_\infty$ is $\max_{j=1}^{\mu} \Psi_j$. ◀

3.2 A $\sqrt{d}$-approximation

We begin by showing that using the above implicit representation, we can efficiently maximize the l_∞ norm of the projection onto the orthogonal complement of the given k-subspace.

▶ **Lemma 2.** *Suppose that for $1 \leq k \leq d$ we are given a set of unit vectors $S = \{s_1, s_2, ..., s_k\} \subset \mathbb{R}^d$ that are pairwise orthogonal. We can compute*

$$\max_{r \in J} ||r - \sum_{i=1}^{k} (r \cdot s_i) s_i||_\infty$$

in polynomial time.

Proof. For $y \in \mathbb{R}^d$, let $y^\dagger$ denote the projection onto the orthogonal complement of the subspace spanned by S. Thus, $y^\dagger = y - \sum_{i=1}^{k} (y \cdot s_i) s_i$. Viewing $y^\dagger$ as a function of y, we see that $y^\dagger = \sum_{i=1}^{d} y_i b_i$, where each $b_i \in \mathbb{R}^d$ is a constant vector – it depends on S but not y. (In other words, $y^\dagger = By$, where the projection matrix B depends only on S.)

For each $1 \leq i \leq d$, define the function $V_i : \mathbb{R} \to \mathbb{R}^d$ by $V_i(x) = x b_i$. It follows that for any row r in the join J,

$$V(r) = \sum_{i=1}^{d} V_i(r_i) = \sum_{i=1}^{d} r_i b_i = r^\dagger.$$

Thus, $V(J) = \{r^\dagger \mid r \in J\}$. From Lemma 1, there is a polynomial-time algorithm to compute

$$\max_{q \in V(J)} ||q||_\infty = \max_{r \in J} ||r^\dagger||_\infty$$ ◀

For any $y \in \mathbb{R}^d$, we have $||y||_\infty \leq ||y||_2 \leq \sqrt{d}||y||_\infty$. Thus we obtain the main result of this Section:

▶ **Theorem 3.** *Suppose that for $1 \leq k \leq d$ we are given a set of unit vectors $S = \{s_1, s_2, ..., s_k\} \subset \mathbb{R}^d$ that are pairwise orthogonal. There is a polynomial-time algorithm to compute a $\sqrt{d}$-approximation to*

$$\max_{r \in J} ||r - \sum_{i=1}^{k} (r \cdot s_i) s_i||_2$$

Next, we consider the algorithmic task of computing the row rank of the join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ of the input acyclic database consisting of tables $T_1, T_2, \ldots, T_m$. That is, we want to compute the size of a maximal linearly independent set [13] of row vectors in J. An algorithm for this follows from Lemma 2.

▶ **Theorem 4.** *There is a polynomial time algorithms that, given an acyclic database as input, can compute the row rank of the join J.*

Proof. We build an orthonormal basis one vector at a time. Using prior work on sum-product queries, we compute a vector $s_1' \in J$ that maximizes the square of the Euclidean norm. If s_1' is the zero vector, the rank of J is 0. Otherwise, let s_1 be the unit vector corresponding to s_1', and we initialize $S = \{s_1\}$.

Suppose that we have computed a set $S = \{s_1, s_2, ..., s_j\}$ of unit vectors in the row space of J that are pairwise orthogonal. Using Lemma 2, we compute $s'_{j+1} \in J$ that achieves

$$\max_{r \in J} ||r - \sum_{i=1}^{j} (r \cdot s_i) s_i||_\infty$$

If this maximum is 0, then the rank of J is j and S is a basis for the row space of J. Otherwise, we add the unit vector s_{j+1} corresponding to s'_{j+1} to S and continue. ◀

4 Maximum Distance to a k-Subspace: A PTAS

Assume we are given an acyclic database consisting of tables $T_1, T_2, \ldots, T_m$ and an orthonormal set of k vectors $S = \{s_i \mid 1 \leq i \leq k\} \subseteq \mathbb{R}^d$, where (as we recall) the set of features in the database is $C = \{1, 2, \ldots, d\}$. We would like to compute the maximum distance of a row in the join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ from the subspace spanned by S. The distance of a point $q \in \mathbb{R}^d$ from the subspace spanned by S, is $\texttt{dist}(q, span(S)) := ||q - \sum_{i=1}^{k} (q \cdot s_i) s_i||_2$. Thus, our goal is to compute $\max_{q \in J} \texttt{dist}(q, span(S))$. Let $\hat{d}(q, span(S)) := ||q||_2^2 - \sum_{i=1}^{k} (q \cdot s_i)^2$. As $\hat{d}(q, span(S)) = \texttt{dist}^2(q, span(S))$, we reformulate this as computing $\texttt{D}(J, span(S)) := \max_{q \in J} \hat{d}(q, span(S))$. The main result of this section a polynomial-time approximation scheme (PTAS) for the problem for constant k.

We begin by showing that the Distance to k-Subspace problem is NP-hard even for $k = 1$.

▶ **Theorem 5.** *If there is a polynomial-time algorithm for computing $\texttt{D}(J, span(\{p\}))$ given the input acyclic database and vector $p \in \mathbb{R}^d$, then $P = NP$.*

Proof. The proof is by reduction from the Partition problem. Given a set of positive integers $F = \{f_1, f_2, ..., f_d\}$, the goal here is determine if F can be partitioned into two parts F' and F'' such that $diff(F', F'') := \sum_{f \in F'} f - \sum_{f \in F''} f = 0$. Given such an input, we compute an acyclic database with tables $T_1, T_2, \ldots, T_{2d}$ and a vector p as follows. The sequence of features is $\langle c_1, v_1, c_2, v_2, \ldots, c_d, v_d, c_{d+1} \rangle$. For $1 \leq i \leq d$, the columns of table T_{2i-1} are c_i and v_i, and the columns of table T_{2i} are v_i and c_{i+1}. Each table has two rows, whose values are determined as follows.

T_1	
c_1	v_1
0	f_1
0	$-f_1$

T_2	
v_1	c_2
f_1	0
$-f_1$	0

T_3	
c_2	v_2
0	f_2
0	$-f_2$

T_4	
v_2	c_3
f_2	0
$-f_2$	0

⋮

T_{2d-1}	
c_d	v_d
0	f_d
0	$-f_d$

T_{2d}	
v_d	c_{d+1}
f_d	0
$-f_d$	0

It is clear that the database is acyclic, and in fact, a tree that witnesses this is a path. We set $p = (1, 1, \ldots, 1) \in \mathbb{R}^{2d+1}$. Let $\ell(p) = span(\{p\})$ denote the line through p. Let J denote the join of the tables. If $\mathtt{D}(J, \ell(p)) = \sum_{f \in F} f^2$, we declare that the input Partition instance has a solution; otherwise, we declare that it doesn't.

The reduction runs in polynomial time, and we now argue that it is correct. For any partition (F', F'') of F, we define $\chi(F', F'')$ to be the vector $(c_1, v_1, \ldots, c_d, v_d, c_{d+1})$ where $c_i = 0$ for $1 \leq i \leq d+1$, and for each $1 \leq i \leq d$, $v_i = f_i$ if $f_i \in F'$ and $v_i = -f_i$ if $f_i \in F''$. Note that χ is a bijection from the set of partitions of F to the set of rows of the join J. Furthermore, $diff(F', F'') = \chi(F', F'') \cdot p$. Note that $||p||_2^2 = 2d + 1$, and for any row $q \in J$, we have $||q||_2^2 = \sum_{f \in F} f^2$; all rows in J have the same 2-norm. Thus, for any $q \in J$,

$$\hat{d}(q, \ell(p)) = ||q||_2^2 - \frac{(q \cdot p)^2}{||p||_2^2} = \sum_{f \in F} f^2 - \frac{(q \cdot p)^2}{2d+1}$$

We conclude that for any partition (F', F'') of F,

$$diff(F', F'') = 0 \iff \chi(F', F'') \cdot p = 0 \iff \hat{d}(\chi(F', F''), p) = \sum_{f \in F} f^2$$ ◀

Our PTAS for Distance to a k-Subspace has three major steps. Let $E = \{e_i \mid 1 \leq i \leq d\} \subseteq \mathbb{R}^d$ denote the unit vectors in the standard basis. We want to find a k-subset $E_k \subseteq E$ such that $span(E_k)$ is "close to" $span(S)$. This first step, along with the properties of $span(E_k)$, is described in Section 4.1. In the second step (Section 4.2), we use E_k to reduce the Distance to k-Subspace instance to polynomially many well-conditioned instances of the same problem. In the third step (Section 4.3), we use discretization and dynamic programming to solve a well-conditioned instance approximately in polynomial time. We conclude in Section 4.4 with an application of the PTAS to subspace approximation.

For a set $X \subset \mathbb{R}^d$, we denote the orthogonal complement of X by $X^\perp$. That is, $X^\perp = \{y \in \mathbb{R}^d \mid y \cdot x = 0 \text{ for all } x \in X\}$.

4.1 Finding a Close k-Subspace

Our algorithm for computing E_k is as follows. Let $proj(a, span(S))$ be the projection of a vector a on the subspace spanned by S.

Algorithm 1 Compute E_k.

Initialize $E_k = \varnothing, S' = \varnothing$
$e'_1 := \arg\max_{e_j \in E} |s_1 \cdot e_j|$. Add e'_1 to E_k
$p_1 :=$ a unit vector in the direction of $proj(e'_1, span(S))$
$S' \leftarrow S' \cup \{p_1\}$
for $i = 2, 3, \ldots, k$ **do**
  Compute a unit vector p'_i such that $p'_i \in span(S) \cap span(S')^\perp$
  $e'_i := \arg\max_{e_j \in E} |p'_i \cdot e_j|$. Add e'_i to E_k.
  $p_i :=$ a unit vector in the direction of $proj(e'_i, span(S) \cap [span(S')^\perp])$
  $S' \leftarrow S' \cup \{p_i\}$
end for

We state some useful properties of this algorithm:

▶ **Lemma 6.** *(a) The set $\{p_1, p_2, ..., p_k\}$ is an orthonormal basis for $span(S)$; (b) For any $1 \leq j < i \leq k$, we have $e'_j \cdot p_i = 0$; (c) We have $|E_k| = k$, that is, our algorithm never attempts to add a duplicate element to E_k; (d) For each $1 \leq i \leq k$, we have $e'_i \cdot p_i \geq \frac{1}{\sqrt{d}}$.*

Proof. Part (a) is evident. As for (b), fix $1 \leq j < i \leq k$. We have $p_i \cdot p_j = 0$. We have $p_j := proj(e'_j, span(S) \cap [span(p_1, p_2, ..., p_{j-1})^\perp])$, and $p_i \in span(S) \cap [span(p_1, p_2, ..., p_{j-1})^\perp]$. Given $p_i \cdot p_j = 0$, we have $p_i \cdot e'_j = 0$.

For (c), We pick $e'_i \in E$ such that $|e'_i \cdot p'_i|$ is maximum. So $|e'_i \cdot p'_i| \geq \frac{1}{\sqrt{d}}$. Given that $p'_i, p_i \in span(S) \cap [span(p_1, p_2, ..., p_{i-1})^\perp]$, and $p_i = proj(e'_i, span(S) \cap [span(p_1, p_2, ..., p_{i-1})^\perp])$, we have $e'_i \cdot p_i \geq |e'_i \cdot p'_i| \geq \frac{1}{\sqrt{d}}$. From part (b), for all $1 \leq j \leq i-1$, we have $e'_j \cdot p_i = 0$, so $e'_i \notin \{e'_j \mid 1 \leq j \leq i-1\}$. Thus $|E_k| = k$. We have already shown part (d) in the process. ◀

The key property of the set E_k computed above is that the k-subspaces $span(E_k)$ and $span(S)$ are close in the sense of the following theorem. For other measures of closeness of subspaces, see [11].

▶ **Theorem 7.** *For any unit vector $v \in span(S)$, there exists a unit vector $u \in span(E_k)$ such that $|v \cdot u| \geq \frac{1}{(\sqrt{d})^k 2^k}$. (We assume that the dimension $d \geq 36$ to get this simplified bound.)*

Proof. A high-level overview of the proof is that we construct a "best-response" unit vector $u \in span(E_k)$ for each unit vector $v \in span(S)$, depending on the relative magnitudes of components of v. Now we proceed to the formal proof.

We represent the unit vectors as $v = x_1 p_1 + x_2 p_2 + ... + x_k p_k$ and $u = y_1 e'_1 + y_2 e'_2 + ... + y_k e'_k$, where $\sum_{i=1}^k x_i^2 = \sum_{i=1}^k y_i^2 = 1$. Let $x = (x_1, x_2, ..., x_k)$ and $y = (y_1, y_2, ..., y_k)$ denote the corresponding unit vectors.

Let $P = \begin{bmatrix} p_1 & p_2 & ... & p_k \end{bmatrix} \in \mathbb{R}^{d \times k}$, $U = \begin{bmatrix} e'_1 & e'_2 & ... & e'_k \end{bmatrix} \in \mathbb{R}^{d \times k}$, and $A = P^\top U$. We have $v \cdot u = x^\top A y$. We have the following claim for A.

▷ Claim 8. A is an upper triangular matrix. Each diagonal element $a_{ii} \geq \frac{1}{\sqrt{d}}$. For other non-diagonal elements $a_{ij'}$, where $i < j'$, we have $|a_{ij'}| \leq 1$.

Proof. For $1 \leq j < i \leq k$, we have $a_{ij} = p_i \cdot e'_j = 0$ by part (b) of Lemma 6. For $1 \leq i \leq k$, we have $a_{ii} = p_i \cdot e'_i \geq \frac{1}{\sqrt{d}}$ by part (d). For other non-diagonal elements $a_{ij'}$, where $i < j'$: given p_i is a unit vector, $|a_{ij'}| = |p_i \cdot e'_j| \leq 1$. ◁

Let $z = x^\top A$. So $v \cdot u = z \cdot y$ and

$$z = (a_{11}x_1, a_{12}x_1 + a_{22}x_2, ..., a_{1k}x_1 + a_{2k}x_2 + ... + a_{kk}x_k)^\top$$

We want to pick a good vector y such that $|z \cdot y|$ is large enough. Let c be a small positive constant. We discuss different cases for x:

- Case 1: $x_1 \geq \frac{c}{(\sqrt{d})^{k-1}}$
- Case 2: $x_1 < \frac{c}{(\sqrt{d})^{k-1}}, x_2 \geq \frac{2c}{(\sqrt{d})^{k-2}}$

⋮

- Case j: $x_1 < \frac{c}{(\sqrt{d})^{k-1}}, x_2 < \frac{2c}{(\sqrt{d})^{k-2}}$, ..., $x_{j-1} < \frac{2^{j-2}c}{(\sqrt{d})^{k-j+1}}, x_j \geq \frac{2^{j-1}c}{(\sqrt{d})^{k-j}}$

⋮

- Case k: $x_1 < \frac{c}{(\sqrt{d})^{k-1}}, x_2 < \frac{2c}{(\sqrt{d})^{k-2}}$, ..., $x_{k-1} < \frac{2^{k-2}c}{(\sqrt{d})}, x_k \geq 2^{k-1}c$

In case 1, we pick $y_1 = 1$ and leave all other components 0. Then $|z \cdot y| = |a_{11}x_1| \geq \frac{1}{\sqrt{d}} \cdot \frac{c}{(\sqrt{d})^{k-1}} = \frac{c}{(\sqrt{d})^k}$.

In case 2, we pick $y_2 = 1$ and leave all other components 0. Then $|z \cdot y| = |a_{12}x_1 + a_{22}x_2| \geq a_{22}x_2 - |a_{12}x_1| \geq \frac{1}{\sqrt{d}} \cdot \frac{2c}{(\sqrt{d})^{k-2}} - \frac{c}{(\sqrt{d})^{k-1}} = \frac{c}{(\sqrt{d})^{k-1}}$.

Similarly, in the case j, we pick $y_j = 1$ and leave all other components 0. Then $|z \cdot y| = |a_{1j}x_1 + a_{2j}x_2 + ... + a_{jj}x_j| \geq a_{jj}x_j - \sum_{i=1}^{j-1} |a_{ij}x_i| \geq \frac{2^{j-1}c}{(\sqrt{d})^{k-j+1}} - \sum_{i=1}^{j-1} \frac{2^{i-1}c}{(\sqrt{d})^{k-i}} \geq \frac{2^{j-1}c}{(\sqrt{d})^{k-j+1}} - \frac{2^{j-2}c}{(\sqrt{d})^{k-j+1}} \cdot \frac{1}{1-\frac{2}{\sqrt{d}}} = \frac{2^{j-2}c}{(\sqrt{d})^{k-j+1}} \frac{\sqrt{d}-4}{\sqrt{d}-2}$.

In case k, we pick $y_k = 1$ and leave all other components 0. Then $|z \cdot y| = |a_{1j}x_1 + a_{2j}x_2 + ... + a_{kk}x_k| \geq a_{kk}x_k - \sum_{i=1}^{k-1} |a_{ik}x_i|$. Similar to the case j, the lower bound is $\frac{2^{k-2}c}{(\sqrt{d})} \frac{\sqrt{d}-4}{\sqrt{d}-2}$.

Therefore, given that $d \geq 36$, the smallest lower bound for $|z \cdot y|$ occurs in Case 1, where $|z \cdot y| \geq \frac{c}{(\sqrt{d})^k}$. We now argue that at least one of the cases must hold. It suffices that

$$\sum_{i=1}^{k} (\frac{2^{i-1}c}{(\sqrt{d})^{k-i}})^2 \leq \sum_{i=1}^{k} x_i^2 = 1$$

The inequality holds if $c = \frac{1}{2^k}$. Thus, $|v \cdot u| \geq \frac{c}{(\sqrt{d})^k} = \frac{1}{(\sqrt{d})^k 2^k}$. ◀

For each $1 \leq i \leq k$, let β_i denote the coordinate corresponding to e'_i, i.e., $e'_i = e_{\beta_i}$. Let $I = \{\beta_1, \beta_2, ..., \beta_k\} \subseteq [d]$ be the set of indices corresponding to the standard basis vectors in E_k. The following claim is shown by using observations made in the proof of Theorem 7.

▶ **Lemma 9.** *For any k-dimensional vector $a = (a_1, a_2, ..., a_k) \in \mathbb{R}^k$, we can efficiently compute a point $q \in span(S)$ such that $q_{\beta_i} = a_i$, for all $1 \leq i \leq k$.*

Proof. For any $1 \leq i \leq k$, let $w_i \in \mathbb{R}^k$ be p_i restricted to the indices in I. Similarly, let $e''_i \in \mathbb{R}^k$ be e'_i restricted to indices in I.

Let $M = \begin{bmatrix} w_1 & w_2 & ... & w_i & ... & w_k \end{bmatrix} \in \mathbb{R}^{k \times k}$, We want to show that there exists an vector $x \in \mathbb{R}^k$ such that $Mx = a$. We have the following claim for M.

▷ Claim 10. $rank(M) = k$.

Proof. Let $N = \begin{bmatrix} e''_1 & e''_2 & ... & e''_i & ... & e''_k \end{bmatrix} \in \mathbb{R}^{k \times k}$. We can observe that $p_i \cdot e'_j = w_i \cdot e''_j$ holds for $1 \leq i, j \leq k$. Thus, $M^\top N = P^\top U = A$. From claim 8, we know $rank(A) = k$. Thus $rank(M) = k$. ◁

Given M is a full rank square matrix, there exists an vector $x \in \mathbb{R}^k$ such that $Mx = a$. Then we can compute $q = Px$. ◀

We conclude with the following implication of the closeness in Theorem 7 of $span(S)$ and $span(E_k)$.

▶ **Lemma 11.** *Let w be an arbitrary unit vector in the $(d-k)$-subspace $span(E \setminus E_k)$. Let w' be a unit vector along the projection of w on $span(S)$. Then $|w \cdot w'| \leq \sqrt{1 - \frac{1}{R}}$, where $R = d^k 2^{2k}$.*

Proof. Let v be the projection of w' on the subspace spanned by E_k. Let $u = w' - v$. According to Theorem 2, $||v||_2 \geq \frac{1}{(\sqrt{d})^k 2^k}$, then $||u||_2 \leq \sqrt{1 - \frac{1}{d^k 2^{2k}}}$.

Then

$$|w \cdot w'| = |w \cdot (u + v)| = |w \cdot u| \leq ||w||_2 ||u||_2 \leq \sqrt{1 - \frac{1}{d^k 2^{2k}}} = \sqrt{1 - \frac{1}{R}}$$ ◀

4.2 Reduction to Well-Conditioned Instances

We will use the new orthonormal basis $\mathcal{P} = \{p_1, p_2, ..., p_k\}$ for $span(S)$. Therefore we will use the notation $span(\mathcal{P})$ for the subspace $span(S)$ in the following sections.

Recall that $I \subseteq [d]$ is the set of indices corresponding to the standard basis vectors in E_k. For each $i \in I$, let α_i^* be the node in the join tree such that $i \in \widehat{C}_{\alpha_i^*}$. See Section 2 for a reminder of these concepts. Let $\mathcal{A} = \{\alpha_i^* \mid i \in I\}$; we use $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{A}|}\}$ to denote the elements of the set. We can write the join J as

$$J = \bigcup_{r_1 \in T_{\alpha_1}, r_2 \in T_{\alpha_2}, ..., r_{|\mathcal{A}|} \in T_{\alpha_{|\mathcal{A}|}}} J \bowtie r_1 \bowtie r_2 \bowtie ... \bowtie r_{|\mathcal{A}|}$$

We therefore compute $\mathtt{D}(J \bowtie r_1 \bowtie r_2 \bowtie ... \bowtie r_{|\mathcal{A}|}, span(\mathcal{P}))$ for each row combination $\{r_1, r_2, ..., r_{|\mathcal{A}|}\}$, and return the maximum of these quantities. The number of row combinations is $O(n^k)$, which is polynomial as k is a constant. In the remainder of Section 4, we fix a single row combination $\{r_1, r_2, ..., r_{|\mathcal{A}|}\}$, and explain our approximation scheme for calculating $\mathtt{D}(J \bowtie r_1 \bowtie r_2 \bowtie ... \bowtie r_{|\mathcal{A}|}, span(\mathcal{P}))$. We will continue to use J to denote the new join. For each $1 \leq i \leq |\mathcal{A}|$, we delete all rows in the table T_{α_i} except r_i. Let $r^* = r_1 \bowtie r_2 \bowtie ... \bowtie r_{|\mathcal{A}|}$.

For any $q \in J$, we have $\hat{d}(q, span(\mathcal{P})) = ||q||_2^2 - \sum_{i=1}^k (q \cdot p_i)^2$. However, the individual terms $||q||_2^2$ and $\sum_{i=1}^k (q \cdot p_i)^2$ can be much larger than $\hat{d}(q, span(\mathcal{P}))$. We therefore apply a translation that preserves $\hat{d}(\cdot, span(\mathcal{P}))$ but makes these two terms small.

Let r^k be the vector formed from r^* by restricting to the indices in I. Using Lemma 9, we compute a vector $h \in span(\mathcal{P})$ such that $h_I = r^k$. We apply the translation $q' = q - h$. As we translate by a vector in $span(\mathcal{P})$, the distance to $span(\mathcal{P})$ is preserved:

▶ **Lemma 12.** *For any $q \in J$, $\hat{d}(q, span(\mathcal{P})) = \hat{d}(q', span(\mathcal{P}))$*

Also, we can observe that for any $q \in J$, $q'_I = q_I - h_I = r^k - r^k = \mathbf{0}$.

To effect the translation, we have to modify the tables. Thus, for each node α in the join tree, and for each row $r \in T_\alpha$ and each feature i of T_α, we set $r_i^{new} = r_i - h_i$. We henceforth refer to the r_i^{new} as just r_i and q' as q.

▶ **Lemma 13.** *After the modification of tables, for any $q \in J$, $q \in span(E \backslash E_k)$ as $q_I = \mathbf{0}$. Therefore by Lemma 11, $||q||_2^2 \leq R \cdot \hat{d}(q, span(\mathcal{P}))$.*

This bound on $||q||_2^2$ is what we mean by saying that the join J is *well-conditioned*.

4.3 Algorithm for a Well-Conditioned Instance

We can use dynamic programming over the join tree to explicitly compute the set $\{\hat{d}(q, span(\mathcal{P})) \mid q \in J\}$. However, this set can have exponential size. Our approximation algorithm will therefore work with a rounded version of $\hat{d}(q, span(\mathcal{P}))$. We observe that

$$\hat{d}(q, span(\mathcal{P})) = ||q||_2^2 - \sum_{i=1}^{k} (q \cdot p_i)^2 = \sum_{j=1}^{d} q_j^2 - \sum_{i=1}^{k} \left(\sum_{j=1}^{d} p_{ij} q_j \right)^2 \tag{1}$$

Where p_{ij} is the j-th element of p_i. By Lemma 13, we know that $||q||_2^2 \leq R \cdot \hat{d}(q, span(\mathcal{P}))$ for any $q \in J$; the join is well-conditioned.

Let $\beta = \max_{q \in J} ||q||_2^2$. Consider a node α of the join tree such that its table T_α contains a row r with an entry whose absolute value exceeds $\sqrt{\beta}$. Clearly, such a row r does not contribute to the eventual join J, so we delete such rows. Let $\eta = \frac{4R(k+1)d^2}{\epsilon}$, where $\epsilon > 0$ is the error parameter for the approximation scheme, and $\gamma = \frac{\sqrt{\beta}}{\eta}$. Notice from Equation 1 that $\hat{d}(q, span(\mathcal{P}))$ is built up from terms of the form $p_{ij}q_j$ and q_j. For $x \in \mathbb{R}$, let $[x] = sign(x)\lfloor |x|/\gamma \rfloor$. We will use $\gamma * [p_{ij}q_j]$ and $\gamma * [q_j]$ as our rounded version of $p_{ij}q_j$ and q_j accordingly – essentially we round to the nearest integer multiple of γ that makes the absolute value smaller. The rounded version we use for $\hat{d}(q, span(\mathcal{P}))$ will then be

$$\texttt{apxdist}(q, span(\mathcal{P})) := \gamma^2 \sum_{j=1}^{d} [q_j]^2 - \gamma^2 \sum_{i=1}^{k} \left(\sum_{j=1}^{d} [p_{ij}q_j] \right)^2 \tag{2}$$

We now describe our dynamic programming algorithm to use the $[p_{ij}q_j]$'s to compute an approximation to $\texttt{D}(J, span(\mathcal{P}))$.

4.3.1 Data Structures for the DP

For node α in the join tree, and row $q \in J_\alpha$, let $A_\alpha(q) = \sum_{j \in C_\alpha} [q_j]^2$, and $B_\alpha(q) = (B_\alpha(q)[1], B_\alpha(q)[2], ..., B_\alpha(q)[k])$, where $B_\alpha(q)[i] = \sum_{j \in C_\alpha} [p_{ij}q_j]$, for $1 \leq i \leq k$. Note that if α is the root of the join tree, then $\gamma^2 A_\alpha(q) - \gamma^2 \sum_{i=1}^{k} (B_\alpha(q)[i])^2$ is $\texttt{apxdist}(q, span(\mathcal{P}))$ (Equation 2).

For each node α in the join tree τ, our algorithm stores an array $\alpha.M[]$ indexed by the rows in table T_α. For a row $r \in T_\alpha$, the entry $\alpha.M[r]$ will eventually store the set

$$\{(A_\alpha(q), B_\alpha(q)) \mid q \in J_\alpha \bowtie r\}$$

That is, a signature $(A_\alpha(q), B_\alpha(q))$ is stored in $\alpha.M[r]$ for each row q in the join $J_\alpha \bowtie r$. Using the fact that the join is well-conditioned, we show later that the size of this set of signatures is bounded by a polynomial, even though the join itself can have an exponential number of rows.

4.3.2 The DP

To fill in the $\alpha.M[]$ arrays, the algorithm does a post-order traversal of the join tree τ, and solves for each node α of τ in that order.

- If α is a leaf node, then for each row r in T_α, we set $\alpha.M[r]$ to $\{(A_\alpha(r), B_\alpha(r))\}$. Note that $A_\alpha(r) = \sum_{j \in \widehat{C}_\alpha} [r_j]^2$, and $B_\alpha(r) = (B_\alpha(r)[1], B_\alpha(r)[2], ..., B_\alpha(r)[k])$, where $B_\alpha(r)[i] = \sum_{j \in \widehat{C}_\alpha} [p_{ij}r_j]$. If $\widehat{C}_\alpha = \emptyset$, then the above summations evaluate to 0, and $\alpha.M[r] = \{(0, [0, 0, ..., 0])\}$.
- If α is an internal node, let λ and ρ denote its left and right child. For each row $r \in T_\alpha$, we initialize $\alpha.M[r] = \emptyset$ and compute $\alpha.M[r]$ as follows:

Algorithm 2 Compute $\alpha.M[r]$.

$\Delta_A \leftarrow \sum_{j \in \widehat{C}_\alpha} [r_j]^2$
Initialize vector $\Delta_B := (\Delta_B[1], \Delta_B[2], ..., \Delta_B[k])$
for each $1 \leq i \leq k$ **do**
  $\Delta_B[i] \leftarrow \sum_{j \in \widehat{C}_\alpha} [p_{ij} r_j]$
end for
for each $r_\lambda \in T_\lambda$ and $r_\rho \in T_\rho$ **do**
  if $r_\lambda \sim r$ and $r_\rho \sim r$ **then**
    for each $(a_\lambda, b_\lambda) \in \lambda.M[r_\lambda]$ and $(a_\rho, b_\rho) \in \rho.M[r_\rho]$ **do**
      Add $(a_\lambda + a_\rho + \Delta_A, b_\lambda + b_\rho + \Delta_B)$ to $\alpha.M[r]$
    end for
  end if
end for

After the DP completes, we use the tables at the root α of the join tree to output this estimate for $\mathtt{D}(J, span(\mathcal{P}))$:

$$\max_{r \in T_\alpha} \max_{(a,b) \in \alpha.M[r]} \gamma^2 a - \gamma^2 \sum_{i=1}^{k} b[i]^2$$

4.3.3 Correctness of the DP

Let α be any node of the join tree, and r a row in the table T_α. We argue that for any row q in the join $J_\alpha \bowtie r$, $(A_\alpha(q), B_\alpha(q)) \in \alpha.M[r]$ after the DP completes processing node α. If α is a leaf, this is obvious. Assume that α is an internal node. Let λ and ρ be the left and right child of α, respectively. There is exactly one row $r_\lambda \in T_\lambda$ (resp. $r_\rho \in T_\rho$) such that $q \sim r_\lambda$ (resp. $q \sim r_\rho$). Clearly, $r \sim r_\lambda$ and $r \sim r_\rho$. Thus, q is a concatenation of 3 rows: (a) a row in $q_\lambda \in J_\lambda \bowtie r_\lambda$ (b) a row in $q_\rho \in J_\rho \bowtie r_\rho$; and (c) r itself.

Furthermore,

$$A_\alpha(q) = A_\lambda(q_\lambda) + A_\rho(q_\rho) + \Delta_A,$$

where $\Delta_A = \sum_{j \in \widehat{C}_\alpha} [r_j]^2$. And

$$B_\alpha(q)[i] = B_\lambda(q_\lambda)[i] + B_\rho(q_\rho)[i] + \Delta_B[i],$$

where $\Delta_B[i] = \sum_{j \in \widehat{C}_\alpha} [p_{ij} r_j]$. Thus the pair $(A_\alpha(q), B_\alpha(q))$ is added to $\alpha.M[r]$ by the DP when looking at rows $r_\lambda \in T_\lambda$ and $r_\rho \in T_\rho$.

Conversely, we can also argue that if the pair (a, b) is added by the DP to $\alpha.M[r]$, then $a = A_\alpha(q)$ and $b = B_\alpha(q)$ for some $q \in J_\alpha \bowtie r$.

4.3.4 Running Time

We now show that our algorithm runs in polynomial time. Consider the term $p_{ij} r_j$ for some $1 \leq i \leq k, 1 \leq j \leq d$ where r is some row of some table that contains j as a feature. Given that p_i is a unit vector, $|p_{ij} r_j| \leq |r_j| \leq \max_{q \in J} ||q||_2 = \sqrt{\beta}$. As $|[p_{ij} r_j]| = \lfloor |p_{ij} r_j| / \gamma \rfloor$, and $\gamma = \sqrt{\beta}/\eta$, we have

$$|[p_{ij} r_j]| \leq |p_{ij} r_j| / \gamma \leq \sqrt{\beta}/\gamma \leq \eta$$

Note that $[p_{ij}r_j]$ is an integer. Now fix a node α in the join tree and consider any row $q \in J_\alpha$. It follows that $\forall i \in [k]$, $B_\alpha(q)[i] = \sum_{j \in C_\alpha}[p_{ij}q_j]$ is an integer whose absolute value is at most $d\eta$. Similarly, $A_\alpha(q) = \sum_{j \in C_\alpha}[q_j]^2$ is an integer whose absolute value is at most $d^2\eta^2$. Similarly, $\forall i \in [k], B_\alpha(q)[i]$ is an integer whose absolute value is at most $d\eta$. The cardinality of the set $\{(A_\alpha(q), B_\alpha(q)) \mid q \in J_\alpha\}$ is therefore $O(d^{k+2}\eta^{k+2})$. Thus, the space used by the $\alpha.M[\cdot]$ array is $O(nd^{k+2}\eta^{k+2})$. Given that $\eta = 4R(k+1)d^2/\epsilon$, it is now easy to see that the overall running time is a polynomial.

4.3.5 Approximation Guarantee

Fix $q \in J_\alpha$. We note that for all $1 \le i, j \le d$, $|p_{ij}q_j - \gamma[p_{ij}q_j]| \le \gamma$ by design. Now for $1 \le i \le k, 1 \le j, j' \le d$, we have

$$\begin{aligned}
|(p_{ij}q_j)(p_{ij'}q_{j'}) - \gamma^2[p_{ij}q_j][p_{ij'}q_{j'}]| &\le |(p_{ij}q_j)(p_{ij'}q_{j'}) - \gamma[p_{ij}q_j](p_{ij'}q_{j'})| + \\
&\quad |\gamma[p_{ij}q_j](p_{ij'}q_{j'}) - \gamma^2[p_{ij}q_j][p_{ij'}q_{j'}]| \\
&\le \gamma|p_{ij'}q_{j'}| + \gamma|\gamma[p_{ij}q_j]| \\
&\le 2\gamma\sqrt{\beta} = 2\beta/\eta
\end{aligned}$$

Here, the last inequality is because we only round down the $p_{ij}q_j$. Now, $\hat{d}(q, span(\mathcal{P}))$ (Equation 1) and $\texttt{apxdist}(q, span(\mathcal{P}))$ (Equation 2) each have $(k+1)d^2$ corresponding terms, and the difference in absolute value between the corresponding terms is bounded by $2\beta/\eta$ above. Thus,

$$|\hat{d}(q, span(\mathcal{P})) - \texttt{apxdist}(q, span(\mathcal{P})| \le \frac{2(k+1)d^2}{\eta}\beta \le \frac{\epsilon}{2R}\beta \le \frac{\epsilon}{2}\texttt{D}(J, span(\mathcal{P})) \tag{3}$$

Here the last step is due to Lemma 13. Let $q^* \in J$ satisfy $\hat{d}(q^*, span(\mathcal{P})) = \texttt{D}(J, span(\mathcal{P}))$. The algorithm returns $\texttt{apxdist}(q, span(\mathcal{P}))$ for a row $q \in J$ such that

$$\begin{aligned}
\hat{d}(q, span(\mathcal{P})) &\ge \texttt{apxdist}(q, span(\mathcal{P})) - \frac{\epsilon}{2}\texttt{D}(J, span(\mathcal{P})) \\
&\ge \texttt{apxdist}(q^*, span(\mathcal{P})) - \frac{\epsilon}{2}\texttt{D}(J, span(\mathcal{P})) \\
&\ge \hat{d}(q^*, span(\mathcal{P})) - \epsilon\texttt{D}(J, span(\mathcal{P})) = (1-\epsilon)\texttt{D}(J, span(\mathcal{P}))
\end{aligned}$$

▶ **Theorem 14.** *There is a polynomial-time algorithm that, given an acyclic database with tables $T_1, T_2, \ldots, T_m$ over a total of d features, a set of orthogonal unit vectors $\mathcal{P} = \{p_1, p_2, ..., p_k\}$ (where k is a constant), and a parameter $0 < \epsilon < 1$, computes a $(1-\epsilon)$-approximation to the farthest distance of a row in the join from the subspace spanned by $\mathcal{P}$.*

4.4 Subspace Approximation

Theorem 14 can be used as a primitive to compute a k-subspace that approximately minimizes the maximum Euclidean distance of a row in the join to the k-subspace. This problem is well known in the literature as l_∞-*subspace approximation*; see [14] for references. Consider the following algorithm:

Let S be a set that consists of the origin in $\mathbb{R}^d$.
For $i \in [k]$:
    Compute using Theorem 14 a row $q \in J$ such that $\hat{d}(q, span(S)) \ge (1-\epsilon)\texttt{D}(J, span(S))$
    $S \leftarrow S \cup \{q\}$

It follows from (the proof of) Lemma 5.2 of [8] that for the set S computed by the algorithm and sufficiently small $0 < \epsilon < 1$, we have $\mathtt{D}(J, span(S)) \leq (2(1+\epsilon))^{2k}\mathtt{D}(J, \mathcal{F})$, for any k-subspace $\mathcal{F}$. That is, for constant k, we get an algorithm that runs in time polynomial in the size of the acyclic database and returns a $(2(1+\epsilon))^k$-approximation for the l_∞-subspace approximation problem.

References

1 Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. On functional aggregate queries with additive inequalities. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pages 414–431, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3294052.3319694.

2 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 13–28, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2902251.2902280.

3 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

4 Jiaxiang Chen, Qingyuan Yang, Ruomin Huang, and Hu Ding. Coresets for relational data and the applications. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

5 Aryan Esmailpour and Stavros Sintos. Improved approximation algorithms for relational clustering. *Proceedings of the ACM on Management of Data*, 2(5):1–27, 2024. doi:10.1145/3695831.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

7 Martin Grohe. The structure of tractable constraint satisfaction problems. In *Proceedings of the 31st International Conference on Mathematical Foundations of Computer Science*, MFCS'06, pages 58–72, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11821069_5.

8 Sariel Har-Peled and Kasturi R. Varadarajan. High-dimensional shape fitting in linear time. *Discret. Comput. Geom.*, 32(2):269–288, 2004. doi:10.1007/S00454-004-1118-2.

9 Mahmoud Abo Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. Approximate aggregate queries under additive inequalities. In Michael Schapira, editor, *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, pages 85–99. SIAM, 2021. doi:10.1137/1.9781611976489.7.

10 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6), November 2013. doi:10.1145/2535926.

11 Jianming Miao and Adi Ben-Israel. On principal angles between subspaces in rn. *Linear Algebra and its Applications*, 171:81–98, 1992. doi:10.1016/0024-3795(92)90251-5.

12 Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 97:1–97:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.97.

13 Gilbert Strang. *Linear Algebra and Its Applications, 3rd Edition*. Harcourt, Inc., 1988.

14 David P Woodruff and Taisuke Yasuda. New subset selection algorithms for low rank approximation: Offline and online. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1802–1813, 2023. doi:10.1145/3564246.3585100.