Relational Algorithms for k-Means Clustering

Benjamin Moseley
Carnegie Mellon University, Pittsburgh, PA, USA

Kirk Pruhs
University of Pittsburgh, PA, USA

Alireza Samadian
University of Pittsburgh, PA, USA

Yuyan Wang
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

This paper gives a k-means approximation algorithm that is efficient in the *relational algorithms model*. This is an algorithm that operates directly on a relational database without performing a join to convert it to a matrix whose rows represent the data points. The running time is potentially exponentially smaller than N, the number of data points to be clustered that the relational database represents.

Few relational algorithms are known and this paper offers techniques for designing relational algorithms as well as characterizing their limitations. We show that given two data points as cluster centers, if we cluster points according to their closest centers, it is NP-Hard to approximate the number of points in the clusters on a general relational input. This is trivial for conventional data inputs and this result exemplifies that standard algorithmic techniques may not be directly applied when designing an efficient relational algorithm. This paper then introduces a new method that leverages rejection sampling and the k-means++ algorithm to construct a $O(1)$-approximate k-means solution.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases k-means, clustering, approximation, big-data, databases

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.97

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2008.00358`

Funding *Benjamin Moseley*: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
Kirk Pruhs: Supported in part by NSF grants CCF-1907673, CCF-2036077 and an IBM Faculty Award.

1 Introduction

Kaggle surveys [2] show that the majority of learning tasks faced by data scientists involve *relational data*. Conventional formats usually represent data with multi-dimensional points where each dimension corresponds to a feature of the data. In contrast, a **relational database** consists of tables $T_1, T_2, \ldots, T_m$ where the features could be stored partially in the tables. The columns in each table are a subset of features[1] and the rows are data records for

[1] In relational database context the columns are also referred to as *attributes* but here we call them features per the tradition of broader communities.

these features. The underlying data is represented by the **design matrix** $J = T_1 \bowtie \cdots \bowtie T_m$ where each row in J can be interpreted as a data point. Here the **join** ($\bowtie$) is a binary operator on two tables T_i and T_j. The result of the join is the set of all possible concatenations of two rows from T_i and T_j such that they are equal in their common columns/features. If T_i and T_j have no common columns their join is the cross product of all rows. See Table 1 for an example of join operation on two tables.

Table 1 A join of tables T_1 and T_2. Each has 5 rows and 2 features, sharing f_2. The join has all features from both tables. The rows with $f_2 = x$ in the join is the cross product of all rows with $f_2 = x$ from T_1 and T_2. For example, for $f_2 = 1$, the four rows in $T_1 \bowtie T_2$ has (f_1, f_3) values $\{(1,1),(1,2),(2,1),(2,2)\}$, this is the cross product of $f_1 \in \{1,2\}$ from T_1 and $f_3 \in \{1,2\}$ from T_2.

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3

Almost all learning tasks are designed for data in matrix format. The current standard practice for a data scientist is the following.

Standard Practice:

1. Extract the data points from the relational database by taking the join of all tables to find the design matrix $J = T_1 \bowtie \cdots \bowtie T_m$.
2. Then interpret each row of J as a point in a Euclidean space and the columns as the dimensions, corresponding to the features of data.
3. Import this design matrix J into a standard algorithm.

A relational database is a highly compact data representation format. The size of J can be exponentially larger than the input size of the relational database [10]. Extracting J makes the standard practice inefficient. Theoretically, there is a potential for exponential speed-up by running algorithms *directly* on the tables in relational data. We call such algorithms **relational algorithms** if their running time is polynomial in the size of tables when the database is *acyclic*. Acyclic databases will be defined shortly. This leads to the following exciting algorithmic question.

The Relational Algorithm Question:

A. Which standard algorithms can be implemented as relational algorithms?
B. For standard algorithms that are *not* implementable by relational algorithms, is there an alternative efficient relational algorithm that has similar performance?

This question has recently been of interest to the community. However, few algorithmic techniques are known. Moreover, we do not have a good understanding of which problems can be solved on relational data and which cannot. Relational algorithm design has a interesting combinatorial structure that requires a deeper understanding.

We design a relational algorithm for k-means. It has a polynomial time complexity for **acyclic** relational databases. The relational database is acyclic if there exists a tree with the following properties. There is exactly one node in the tree for each table. Moreover, for any feature (i.e. column) f, let $V(f)$ be the set of nodes whose corresponding tables contain feature f. The subgraph induced on $V(f)$ must be a connected component. Acyclicity can be easily checked, as the tree can be found in polynomial time if it exists [27].

Luckily, most of the natural database schema are acyclic or nearly acyclic. Answering seemingly simple questions on general (cyclic) databases, such as if the join is empty or not is NP-Hard. For general databases, efficiency is measured in terms of the **fractional hypertree width** of the database (denoted by "fhtw"). The parameter measures how close the database structure is to being acyclic. It is 1 for acyclic databases and larger as the database is farther from being acyclic.

State-of-the-art algorithms for queries as simple as counting the number of rows in the design matrix have linear dependency on n^{fhtw} where n is the *maximum* number of rows in all input tables [7]. Running in time linear in n^{fhtw} is the goal, as fundamental barriers need to be broken to be faster. Notice that this is polynomial time when fhtw is a fixed constant (i.e. nearly acyclic). Our algorithm has linear dependency on n^{fhtw}, matching the state-of-the-art.

Relational Algorithm for k-means. k-means is perhaps the most widely used data mining algorithm (e.g. k-means is one of the few models in Google's BigQuery ML package [1]). The input to the k-means problem consists of a collection S of points in a Euclidean space and a positive integer k. A feasible output is k points $c_1, \ldots, c_k$, which we call **centers**. The objective is to choose the centers to minimize the aggregate squared distance from each original point to its nearest center.

Recall extracting all data points could take time exponential in the size of a relational database. Thus, the problem is to find the cluster centers without fully realizing all of the data points the relational data represents.

[15] was the first paper to give a non-trivial k-means algorithm that works on relational inputs. The paper gives an $O(1)$-approximation. The algorithm's running time has superlinear dependency on k^d when the tables are acyclic and thus is not polynomial. Here k is the number of cluster centers and d is the dimension (a.k.a number of features) of the points. This is equivalently the number of distinct columns in the relational database. For a small number of dimensions, this algorithm is a large improvement over the standard practice and they showed the algorithm gives up to 350x speed up on real data versus performing the query to extract the data points (not even including the time to cluster the output points).

Several questions remain. Is there a relational algorithm for k-means? What algorithmic techniques can we use as building blocks to design relational algorithms? Moreover, how can we show some problems are hard to solve using a relational algorithm?

Overview of Results. The main result of the paper is the following.

▶ **Theorem 1.** *Given an acyclic relational database with tables $T_1, T_2, \ldots T_m$ where the design matrix J has N rows and d columns. Let n be the maximum number of rows in any table. Then there is a randomized algorithm running in time polynomial in d, n and k that computes an $O(1)$ approximate k-means clustering solution with high probability.*

The discussion about the algorithm's time complexity for cyclic databases is left out due to space limits. To illustrate the challenges for finding such an algorithm as described in the prior theorem, even when the database is acyclic, consider the following theorem.

▶ **Theorem 2.** *Given an acyclic relational database with tables $T_1, T_2, \ldots T_m$ where the design matrix J has N rows and d columns. Given k centers $c_1, \ldots, c_k$, let J_i be the set of points in J that are closest to c_i for $i \in [k]$. It is #P-Hard to compute $|J_i|$ for $k \geq 2$ and NP-Hard to approximate $|J_i|$ to any factor for $k \geq 3$.*

We show the proof in Section 2.1. We prove it by reducing a NP-Hard problem to the problem of determining if J_i is empty or not. Counting points closest to a center is a fundamental building block in almost all k-means algorithms. Moreover, we note that performing one iteration of the classic Lloyd's algorithm, that is, to re-compute the centroids of all J_i's, is also $\#P$-Hard. The proof is omitted here.

Together this necessitates the design of new techniques to address the main theorem, shows that seemingly trivial algorithms are difficult relationally, and suggests computing a coreset is the right approach for the problem as it is difficult to cluster the data directly.

Overview of Techniques. We first compute a **coreset** of all points in J. That is, a collection of points with weights such that if we run an $O(1)$ approximation algorithm on this weighted set, we will get a $O(1)$ approximate solution for all of J. To do so, we sample points according to the principle in k-means++ algorithm and assign weights to the points sampled. The number of points chosen will be $\Theta(k \log N)$. Any $O(1)$-approximate weighted k-means algorithm can be used on the coreset to give Theorem 1.

k-means++. k-means++ is a well-known k-means algorithm [9, 8]. The algorithm iteratively chooses centers $c_1, c_2, \ldots$. The first center c_1 is picked uniformly from J. Given that $c_1, \ldots, c_{i-1}$ are picked, a point x is picked as c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]}(\|x - c_j\|_2^2)$ and $Y = \sum_{x \in J} L(x)$. Here $[i-1]$ denotes $\{1, 2, \ldots, i-1\}$.

Say we sample $\Theta(k \log N)$ centers according to this distribution, which we call the ***k*-means++ distribution**. It was shown in [8] that if we cluster the points by assigning them to their closest centers, the total squared distance between points and their cluster centers is at most $O(1)$ times the optimal k-means cost with high probability. Note that this is not a feasible k-means solution because more than k centers are used. However, leveraging this, the work showed that we can construct a coreset by weighting these centers according to the number of points in their corresponding clusters.

We seek to mimic this approach with a relational algorithm. Let's focus on one iteration where we want to sample the center c_i given $c_1, \ldots, c_{i-1}$ according to the k-means++ distribution. Consider the assignment of every point to its closest center in $c_1, \ldots, c_{i-1}$. Notice that the k-means++ probability is determined by this assignment. Indeed, the probability of a point being sampled is the cost of assigning this point to its closest center ($\min_{j \in [i-1]} \|x - c_j\|_2^2$) normalized by Y. Y is the summation of this cost over all points.

The relational format makes this distribution difficult to compute without the design matrix J. It is hard to efficiently characterize which points are closest to which centers. The assignment *partitions* the data points according to their closest centers, where each partition may not be easily represented by a compact relational database (unlike J).

A Relational k-means++ Implementation. Our approach will sample every point according to the k-means++ distribution without computing this distribution directly. Instead, we use **rejection sampling** [13], which allows one to sample from a "hard" distribution P using an "easy" distribution Q. Rejection sampling works by sampling from Q first, then reject the sample with another probability used to bridge the gap between Q and P. The process is repeated until a sample is accepted. In our setting, P is the k-means++ distribution, and we need to find a Q which could be sampled from efficiently with a relational algorithm (without computing J). Rejection sampling theory shows that for the sampling to be efficient, Q should be close to P point-wise to avoid high rejection frequency. In the end, we will *perfectly simulate* the k-means++ algorithm.

We now describe the intuition for designing such a Q. Recall that P is determined by the assignment of points to their closest centers. We will approximate this assignment up to a factor of $O(i^2d)$ when sampling the i^{th} center c_i, where d is the number of columns in J. Intuitively, the approximate assignment makes things easier since for any center we can easily find the points assigned to it using an efficient relational algorithm. Then Q is found by normalizing the squared distance between each point and its assigned center.

The approximate assignment is designed as follows. Consider the d-dimensional Euclidean space where the data points in J are located. The algorithm divides space into a **laminar** collection of **hyper-rectangles**[2] (i.e., $\{x \in \mathcal{R}^d : v_j \leq x_j \leq w_j, j = 1, \ldots, d\}$, here x_j is the value for feature f_j). We assign each hyper-rectangle to a center. A point assigns itself to the center that corresponds to the *smallest* hyper-rectangle containing the point.

The key property of hyper-rectangles that benefits our relational algorithm is: we can efficiently represent all points from J inside any hyper-rectangle by removing some entries in each table from the original database and taking the join of all tables. For example, if a hyper-rectangle has constraint $v_j \leq x_j \leq w_j$, we just remove all the rows with value outside of range $[v_j, w_j]$ for column f_j from the tables containing column f_j. The set of points assigned to a given center can be found by adding and subtracting a laminar set of hyper-rectangles, where each hyper-rectangle can be represented by a relational database.

Weighting the Centers. We have sampled a good set of cluster centers. In order to get a coreset we need to assign weights to them. As we have already mentioned, assuming $P \neq \#P$, the weights cannot be computed relationally. In fact, they cannot be approximated up to any factor in polynomial time unless $P = NP$. Rather, we design an alternative relational algorithm for computing the weights. Each weight will not be an approximate individually, but we prove that the weighted centers form an $O(1)$-approximate coreset in aggregate.

The main algorithmic idea is that for each center c_i we generate a collection of hyperspheres around c_i containing geometrically increasing numbers of points. The space is then partitioned using these hyperspheres where each partition contains a portion of points in J. Using the algorithm from [3], we then sample a poly-log sized collection of points from each partition, and use this subsample to estimate the fraction of the points in this partition which are closer to c_i than any other center. The estimated weight of c_i is aggregated accordingly.

Paper Organization. As relational algorithms are relatively new, we begin with some special cases which help the reader build intuition. In Section 2 we give a warm-up by showing how to implement 1-means++ and 2-means++ (i.e. initialization steps of k-means++). In this section, we also prove Theorem 2 as an example of the limits of relational algorithms. In Section 3 we go over background on relational algorithms that our overall algorithm will leverage. In Section 4 we give the k-means++ algorithm via rejection sampling. Section 5 shows an algorithm to construct the weights and then analyze this algorithm.

2 Warm-up: Efficiently Implementing 1-means++ and 2-means++

This section is a warm-up to understand the combinatorial structure of relational data. We will show how to do k-means++ for $k \in \{1, 2\}$ (referred to as 1- and 2-means++) on a simple join structure. We will also show the proof of Theorem 2 which states that counting the number of points in a cluster is a hard problem on relational data.

[2] A laminar set of hyper-rectangles means any two hyper-rectangles from the set either have no intersection, or one of them contains the other.

First, let us consider relationally implementing 1-means++ and 2-means++. For better illustration, we consider a special type of acyclic table structure named **path join**. The relational algorithm used will be generalized to work on more general join structures when we move to the full algorithm in Section 4.

In a path join each table T_i has two features/columns f_i, and f_{i+1}. Table T_i and T_{i+1} then share a common column f_{i+1}. Assume for simplicity that each table T_i contains n rows. The design matrix $J = T_1 \bowtie T_2 \bowtie \ldots \bowtie T_m$ has $d = m + 1$ features, one for each feature (i.e. column) in the tables.

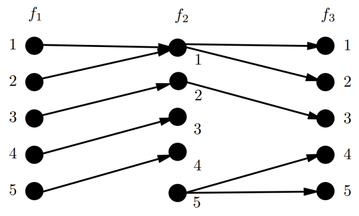
Even with this simple structure, the size of the design matrix J could still be exponential in the size of database - J could contain up to $n^{m/2}$ rows , and $dn^{m/2}$ entries. Thus the standard practice could require time and space $\Omega(mn^{m/2})$ in the worst case.

Table 2 A path join instance where the two tables T_1 and T_2 have $m = 2$ and $n = 5$. This shows T_1, T_2, the design matrix J, and the resulting layered directed graph G. *Every* path from the left most layer to the right most layer of this graph G corresponds to one data point for the clustering problem (i.e. a row of the design matrix).

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$J = T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3



Graph Illustration of the Design Matrix. Conceptually consider a directed acyclic graph G, where there is one layer of nodes corresponding to each feature $f_i(i = 1, \ldots, d)$, and edges only point from nodes in layer f_i to layer f_{i+1}.

The nodes in G correspond to feature values, and edges in G correspond to rows in tables. There is one vertex v in layer f_i for each value that appears in column f_i in table T_{i-1} or T_i, and one edge pointing from u in layer f_i to v in layer f_{i+1}, if (u, v) is a row in table T_i. Then, there is a one-to-one correspondence between **full paths** in G (paths from layer f_1 to layer f_d) and rows in the design matrix.

A Relational Implementation of 1-means++. Implementing the 1-means++ algorithm is equivalent to *generating a full path uniformly at random from* G. We generate this path by iteratively picking a row from table $T_1, \ldots, T_m$, corresponding to picking an arc pointing from layer f_1 to f_2, f_2 to f_3, ..., such that concatenating all picked rows (arcs) will give a point in J (full path in G).

To sample a row from T_1, for every row $r \in T_1$, consider $r \bowtie J$, which is all rows in J whose values in columns (f_1, f_2) are equivalent to r. Let the function $F_1(r)$ denote the total number of rows in $r \bowtie J$. This is also the number of full paths passing arc r. Then, every $r \in T_1$ is sampled with probability $\frac{F_1(r)}{\sum_{r' \in T_1} F_1(r')}$, notice $\sum_{r' \in T_1} F_1(r')$ is the total number of full paths. Let the picked row be r_1.

After sampling r_1, we can conceptually throw away all other rows in T_1 and focus only on the rows in J that uses r_1 to concatenate with rows from other tables (i.e., $r_1 \bowtie J$). For any row $r \in T_2$, let the function $F_2(r)$ denote the number of rows in $r \bowtie r_1 \bowtie J$, also equivalent to the total number of full paths passing arc r_1 and r. We sample every r with probability

$\frac{F_2(r)}{\sum_{r' \in T_2} F_2(r')}$. Notice that $\sum_{r' \in T_2} F_2(r') = F_1(r_1)$, the number of full paths passing arc r_1. Repeat this procedure until we have sampled a row in the last table T_m: for table T_i and $r \in T_i$, assuming we have sampled $r_1, \dots, r_{i-1}$ from $T_1, \dots, T_{i-1}$ respectively, throw away all the other rows in previous tables and focus on $r_1 \bowtie \dots \bowtie r_{i-1} \bowtie J$. $F_i(r)$ is the number of rows in $r \bowtie r_1 \bowtie \dots \bowtie r_{i-1} \bowtie J$ and r is sampled with probability proportional to $F_i(r)$. It is easy to verify that every full path is sampled uniformly.

For every table T_i we need to find the function $F_i(\cdot)$ which is defined on all its rows. There are m such functions. For each $F_i(\cdot)$, we can find all $F_i(r)$ values for $r \in T_i$ using a one-pass dynamic programming and then sample according to the values. Repeating this procedure m rounds completes the sampling process. This gives a polynomial time algorithm.

A Relational Implementation for 2-means++. Assume $x = (x_1, \dots, x_d)$ is the first center sampled and now we want to sample the second center. By k-means++ principles, any row $r \in J$ is sampled with probability $\frac{\|r-x\|^2}{\sum_{r' \in J} \|r'-x\|^2}$. For a full path in G corresponding to a row $r \in J$ we refer to $\|r - x\|^2$ as the **aggregated cost** over all d nodes/features.

Similar to 1-means++, we pick one row in each table from T_1 to T_m and putting all the rows together gives us the sampled point. Assume we have sampled the rows $r_1, r_2, \dots, r_{i-1}$ from the first $i-1$ tables and we focus on all full paths passing $r_1, \dots, r_{i-1}$ (i.e., the new design matrix $r_1 \bowtie \dots \bowtie r_{i-1} \bowtie J$). In 1-means++, we compute $F_i(r)$ which is the total number of full paths passing arc $r_1, \dots, r_{i-1}, r$ (i.e., $r \bowtie r_1 \bowtie \dots \bowtie r_{i-1} \bowtie J$.) and sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values. In 2-means++, we define $F_i(r)$ to be the summation of aggregated costs over all full paths which pass arcs $r_1, \dots, r_{i-1}, r$. We sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values.

It is easy to verify the correctness. Again, each $F_i(\cdot)$ could be computed using a one-pass dynamic programming which gives the values for all rows in T_i when we sample from T_i. This would involve m rounds of such computations and give a polynomial relational algorithm.

2.1 Hardness of Relationally Computing the Weights

Here we prove Theorem 2. We first show that given a set of centers, counting the number of points in J that is closest to any of them is $\#P$-hard. We prove $\#P$-Hardness by a reduction from the well known $\#P$-hard Knapsack Counting problem. The input to the Knapsack Counting problem consists of a set $W = \{w_1, \dots, w_h\}$ of nonnegative integer weights, and a nonnegative integer L. The output is the number of subsets of W with aggregate weight at most L. To construct the relational instance, for each $i \in [h]$, we define the tables T_{2i-1} and T_{2i} as follows:

T_{2i-1}	
f_{2i-1}	f_{2i}
0	0
0	w_i

T_{2i}	
f_{2i}	f_{2i+1}
0	0
w_i	0

Let centers c_1 and c_2 be arbitrary points such that points closer to c_1 than c_2 are those points p for which $\sum_{i=1}^{d} p_i \le L$. Then there are 2^h rows in J, since w_i can either be selected or not selected in feature $2i$. The weight of c_1 is the number of points in J closer to c_1 than c_2, which is in turn exactly the number of subsets of W with total weight at most L.

Now we prove the second part of Theorem 2: given an acyclic database and a set of centers $c_1, \dots, c_k$, it is NP-Hard to approximate the number of points assigned to each center when $k \ge 3$. We prove it by reduction from Subset Sum. In Subset Sum problem, the input is a set of integers $A = w_1, \ldots, w_m$ and an integer L, the output is true if there is a subset of A such that its summation is L. We create the following acyclic schema. There are m tables. Each table T_i has a single unique column x_i with two rows $w_i, 0$. Then the join of the tables has 2^m rows, and it is a cross product of the rows in different tables in which each row represents one subset of A.

Then consider the following three centers: $c_1 = (\frac{L-1}{m}, \frac{L-1}{m}, \ldots, \frac{L-1}{m})$, $c_2 = (\frac{L}{m}, \ldots, \frac{L}{m})$, and $c_1 = (\frac{L+1}{m}, \frac{L+1}{m}, \ldots, \frac{L+1}{m})$. The Voronoi diagram that separates the points assigned to each of these centers consists of two parallel hyperplanes: $\sum_i x_i = L-1/2$ and $\sum_i x_i = L+1/2$ where the points between the two hyperplanes are the points assigned to c_2. Since all the points in the design matrix have integer coordinates, the only points that are between these two hyperplanes are those points for which $\sum_i x_i = L$. Therefore, the approximation for the number of points assigned to c_2 is non-zero if and only if the answer to Subset Sum is True.

3 Related Work and Background

Related Work on K-means. Constant approximations are known for the k-means problem in the standard computational setting [20, 18]. Although the most commonly used algorithm in practice is a local search algorithm called Lloyd's algorithm, or sometimes confusingly just called "the k-means algorithm". The k-means++ algorithm from [9] is a $\Theta(\log k)$ approximation algorithm, and is commonly used in practice to seed Lloyd's algorithm. Some coreset construction methods have been used before to design algorithms for the k-means problem in other restricted access computational models, including steaming [17, 12], and the MPC model [16, 11], as well as speeding up sequential methods [21, 25].

Relational Algorithms for Learning Problem. Training different machine learning models on relational data has been studied; however, many of the proposed algorithms are not efficient under our definition of a relational algorithm. It has been shown that using repeated patterns in the design matrix, linear regression, and factorization machines can be implemented [23] more efficiently. [19, 24, 5] has improved the relational linear regression and factorization machines for different scenarios. A unified relational algorithm for problems such as linear regression, singular value decomposition and factorization machines proposed in [6]. Algorithms for training support vector machine is studied in [26, 4]. In [14], a relational algorithm is introduced for Independent Gaussian Mixture Models, and they have shown experimentally that this method will be faster than materializing the design matrix.

Relational Algorithm Building Blocks. In the path join scenario, the 1- and 2-means++ sampling methods introduced in subsection 2 have similar procedures: starting with the first table T_1, iteratively evaluate some general function $F_i(\cdot)$ defined on all rows in the table T_i, sample one row r_i according to the distribution normalized from $F_i(\cdot)$. The function $F_i(\cdot)$ for table T_i is defined on the matrix $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$ where J is the design matrix. This matrix is also the design matrix of a new relational database, constructed by throwing away all rows in previous tables apart from the sampled $r_1, \ldots, r_{i-1}$.

We can generalize the computation of $F_i(\cdot)$ functions into a broader class of queries that we know could be implemented efficiently on *any* acyclic relational databases, namely **SumProd queries**. See [7] for more details. In the following lemmas assume the relational database has tables $T_1, \ldots, T_m$ and their design matrix is J, let n be the maximum number of rows in each table T_i, m be the number of tables and d be the number of columns in J.

▶ **Definition 3.** *For the j^{th} feature ($j \in [d]$) let $q_j : \mathbb{R} \to S$ be an efficiently computable function that maps feature values to some set S. Let the binary operations $\oplus$ and $\otimes$ be any operators such that $(S, \oplus, \otimes)$ forms a commutative semiring. The value of $\bigoplus_{x \in J} \bigotimes_{j \in [d]} q_j(x_j)$ is a SumProd query.*

▶ **Lemma 4** ([7])**.** *Any SumProd query can be computed efficiently in time $O(md^2 n^{fhtw} \log(n))$ where fhtw is the fractional hypertree width of the database. For acyclic databases fhtw=1 so the running time is polynomial.*

Despite the cumbersome formal definition of SumProd queries, below we list their key applications used in this paper. With a little abuse of notation, throughout this paper we use $\Psi(n, d, m)$ to denote the worst-case time bound on any SumProd queries.

▶ **Lemma 5.** *Given a point $y \in \mathcal{R}^d$ and a hyper-rectangle $b = \{x \in \mathcal{R}^d : v_i \le x_i \le w_i, i = 1, \dots, d\}$ where v and w are constant vectors, we let $J \cap b$ denote the data points represented by rows of J that also fall into b. Pick any table T_j. Using one single SumProd query we can compute for all $r \in T_j$ the value $\sum_{p \in r \bowtie J \cap b} \|p - y\|_2^2$. The time required is at most that required by one SumProd query, $\Psi(n, d, m)$,*

Lemma 5 is intuitively based on the fact that we can efficiently represent all points in $J \cup b$ by a new relational database, which is constructed by removing some entries in each table from the original database. The following lemma follows by an application of the main result in [3].

▶ **Lemma 6** ([3])**.** *Given a hypersphere $\{x \in \mathcal{R}^d : \|x - y_0\|^2 \le z_0^2\}$ where y_0 is a given point and z_0 is the radius, a $(1 + \epsilon)$-approximation of the number of points in J that lie inside this hypersphere could be computed in $O\left(\frac{m^6 \log^4 n}{\epsilon^2} \Psi(n, d, m)\right)$ time.*

Notice that a SumProd query could be used to output either a scalar (similar to Lemma 6) or a vector whose entries are function values for every row r in a chosen table T_j (in Lemma 5). We say the SumProd query is **grouped by** T_j in the latter case.

4 The k-means++ Algorithm

In this section, we describe a relational implementation of the k-means++ algorithm. It is sufficient to explain how center c_i is picked given the previous centers $c_1, \dots, c_{i-1}$. Recall that the k-means++ algorithm picks a point x to be c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]} \|x - c_j\|_2^2$ and $Y = \sum_{x \in J} L(x)$ is a normalizing constant.

The implementation consists of two parts. The first part, described in Section 4.1, shows how to partition the d-dimensional Euclidean space into a laminar set of hyper-rectangles (referred to as **boxes** hereafter) that are generated around the previous centers. The second part, described in Section 4.2, samples according to the "hard" distribution P using rejection sampling and an "easy" distribution Q.

Conceptually, we assign every point in the design matrix J to an *approximately* nearest center among $c_1, \dots, c_{i-1}$. This is done by assigning every point in J to one of the centers contained in the *smallest* box this point belongs to. Then Q is derived using the squared distance between the points in J and their assigned centers.

4.1 Box Construction

Here we explain the algorithm for constructing a set of laminar boxes given the centers sampled previously. The construction is completely combinatorial. It only uses the given centers and we don't need any relational operation for the construction.

Algorithm Description. Assume we want to sample the i^{th} point in k-means++. The algorithm maintains two collections $\mathcal{G}_i$ and $\mathcal{B}_i$ of tuples. Each tuple consists of a box and a point in that box, called the **representative** of the box. This point is one of the previously sampled centers. One can think of the tuples in $\mathcal{G}_i$ as "active" ones that are subject to changes and those in $\mathcal{B}_i$ as "frozen" ones that are finalized, thus removed from $\mathcal{G}_i$ and added to $\mathcal{B}_i$. When the algorithm terminates, $\mathcal{G}_i$ will be empty, and the boxes in $\mathcal{B}_i$ will be a laminar collection of boxes that we use to define the "easy" probability distribution Q.

The initial tuples in $\mathcal{G}_i$ consist of one *unit hyper-cube* (side length is 1) centered at each previous center c_j, $j \in [i-1]$, with its representative point c_j. Up to scaling of initial unit hyper-cubes, we can assume that initially no pair of boxes in $\mathcal{G}_i$ intersect. This property of $\mathcal{G}_i$ is maintained throughout the process. Initially $\mathcal{B}_i$ is empty. Over time, the implementation keeps growing the boxes in $\mathcal{G}_i$ in size and moves tuples from $\mathcal{G}_i$ to $\mathcal{B}_i$.

The algorithm repeats the following steps in rounds. At the beginning of each round, there is no intersection between any two boxes in $\mathcal{G}_i$. The algorithm performs a doubling step where it **doubles** every box in $\mathcal{G}_i$. Doubling a box means each of its $d-1$ dimensional face is moved twice as far away from its representative. Mathematically, a box whose representative point is $y \in \mathcal{R}^d$ may be written as $\{x \in \mathcal{R}^d : y_i - v_i \leq x_i \leq y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$. This box becomes $\{x \in \mathcal{R}^d : y_i - 2v_i \leq x_i \leq y_i + 2w_i, i = 1, \ldots, d\}$ after doubling.

After doubling, the algorithm performs the following operations on intersecting boxes until there are none. The algorithm iteratively picks two arbitrary intersecting boxes from $\mathcal{G}_i$. Say the boxes are b_1 with representative y_1 and b_2 with representative y_2. The algorithm executes a **melding** step on (b_1, y_1) and (b_2, y_2), which has the following procedures:

- Compute the smallest box b_3 in the Euclidean space that contains both b_1 and b_2.
- Add (b_3, y_1) to $\mathcal{G}_i$ and delete (b_1, y_1) and (b_2, y_2) from $\mathcal{G}_i$.
- Check if b_1 (or b_2) is a box created by the doubling step at the beginning of the current round and hasn't been melded with other boxes ever since. If so, the algorithm computes a box b_1' (resp. b_2') from b_1 (resp. b_2) by **halving** it. That is, each $d-1$ dimensional face is moved so that its distance to the box's representative is halved. Mathematically, a box $\{x \in \mathcal{R}^d : y_i - v_i \leq x_i \leq y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$, where vector y is its representative, becomes $\{x \in \mathcal{R}^d : y_i - \frac{1}{2}v_i \leq x_i \leq y_i + \frac{1}{2}w_i, i = 1, \ldots, d\}$ after halving. Then (b_1', y_1) (or (b_2', y_2)) is added to $\mathcal{B}_i$. Otherwise do nothing.

Notice that melding decreases the size of $\mathcal{G}_i$.

The algorithm terminates when there is one tuple (b_0, y_0) left in $\mathcal{G}_i$, at which point the algorithm adds a box that contains the whole space with representative y_0 to $\mathcal{B}_i$. Note that during each round of the doubling and melding, the boxes which are added to $\mathcal{B}_i$ are the ones that after doubling were melded with other boxes, and they are added at their shapes before the doubling step.

▶ **Lemma 7.** *The collection of boxes in $\mathcal{B}_i$ constructed by the above algorithm is laminar.*

Proof. Note that right before each doubling step, the boxes in $\mathcal{G}_i$ are disjoint and that is because the algorithm in the previous iteration melds all the boxes that have intersection with each other. We prove by induction that at all time, for every box b in $\mathcal{B}_i$ there exist a box b' in $\mathcal{G}_i$ such that $b \subseteq b'$. Since the boxes added to $\mathcal{B}_i$ in each iteration are a subset of the boxes in $\mathcal{G}_i$ before the doubling step and they do not intersect each other, laminarity of $\mathcal{B}_i$ is a straight-forward consequence.

Initially $\mathcal{B}_i$ is empty and therefore the claim holds. Assume in some arbitrary iteration ℓ this claim holds right before the doubling step, then after the doubling step since every box in $\mathcal{G}_i$ still covers all of the area it was covering before getting doubled, the claim holds.

Furthermore, in the melding step every box b_3 that is resulted from melding of two boxes b_1 and b_2 covers both b_1 and b_2; therefore, b_3 will cover b_1 and b_2 if they are added to $\mathcal{B}_i$, and if a box in $\mathcal{B}_i$ was covered by either of b_1 or b_2, it will be still covered by b_3. ◀

The collection of boxes in $\mathcal{B}_i$ can be thought of as a tree where every node corresponds to a box. The root node is the entire space. In this tree, for any box b', among all boxes included by b', we pick the inclusion-wise *maximal* boxes and let them be the **children** of b'. Thus the number of boxes in $\mathcal{B}_i$ is $O(i)$ since the tree has i leaves, one for each center.

4.2 Sampling

To define our easy distribution Q, for any point $x \in J$, let $b(x)$ be the minimal box in $\mathcal{B}_i$ that contains x and $y(x)$ be the representative of $b(x)$. Define $R(x) = \|x - y(x)\|_2^2$, and $Q(x) = \frac{R(X)}{Z}$ where $Z = \sum_{x \in J} R(x)$ normalizes the distribution. We call $R(x)$ the **assignment cost** for x. We will show how to sample from target distribution $P(\cdot)$ using $Q(\cdot)$ and rejection sampling, and how to implement the this designed sampling step relationally.

Rejection Sampling. The algorithm repeatedly samples a point x with probability $Q(x)$, then either (A) rejects x and resamples, or (B) accepts x as the next center c_i and finishes the sampling process. After sampling x, the probability of accepting x is $\frac{L(x)}{R(x)}$, and that of rejecting x is $1 - \frac{L(x)}{R(x)}$. Notice that here $\frac{L(x)}{R(x)} \le 1$ since $R(x) = \|x - y(x)\|_2^2 \ge \min_{j \in [i-1]} \|x - c_j\|_2^2$.

If $S(x)$ is the the event of initially sampling x from distribution Q, and $A(x)$ is the event of subsequently accepting x, the probability of choosing x to be c_i in one given round is:

$$\Pr[S(x) \text{ and } A(x)] = \Pr[A(x) \mid S(x)] \Pr[S(x)] = \frac{L(x)}{R(x)} Q(x) = \frac{L(x)}{Z}$$

Thus the probability of x being the accepted sample is proportional to $L(x)$, as desired.

We would like $Q(\cdot)$ to be close to $P(\cdot)$ point-wise so that the algorithm is efficient. Otherwise, the acceptance probability $\frac{L(x)}{R(x)}$ is low and it might keep rejecting samples.

Relational Implementation of Sampling. We now explain how to relationally sample a point x with probability $Q(x)$. The implementation heavily leverages Lemma 5, which states for given box b^* with representative y^*, the cost of assigning all points in $r \bowtie J \cap b^*$ to y^* for each row $r \in T_i$ can be computed in polynomial time using a SumProd query grouped by T_i. Recall that we assign all points in J to the representative of the smallest box they belong to. We show that the total assignment cost is computed by evaluating SumProd queries on the boxes and then adding/subtracting the query values for different boxes.

Following the intuition provided in Section 2, the implementation generates a single row from table $T_1, T_2, \ldots, T_m$ sequentially. The concatenation of these rows (or the join of them) gives the sampled point x. It is sufficient to explain assuming we have sampled $r_1, \ldots, r_{\ell-1}$ from the first $\ell - 1$ tables, how to implement the generation of a row from the next table T_ℓ. Just like 1- and 2-means++ in subsection 2, the algorithm evaluates a function $F_\ell(\cdot)$ defined on rows in T_ℓ using SumProd queries, and samples r with probability $\frac{F_\ell(r)}{\sum_{r' \in T_\ell} F_\ell(r')}$. Again, we focus on $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, denoting the points in J that uses the previously sampled rows. The value of $F_\ell(r)$ is determined by points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$.

To ensure we generate a row according to the correct distribution Q, we define the function $F_\ell(\cdot)$ as follows. Let $F_\ell(r)$ be the total assignment cost of all points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$. That is, $F_\ell(r) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J} R(x)$. Notice that the definition of function $F_\ell(\cdot)$ is very similar to 2-means++ apart from that each point is no longer assigned to a given center, but the representative of the smallest box containing it.

Let $G(r, b^*, y^*)$ denote the cost of assigning all points from $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$ that lies in box b^* to a center y^*. By replacing the J in Lemma 5 by $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, we can compute all $G(r, b^*, y^*)$ values in polynomial time using one SumProd query grouped by T_ℓ. The value $F_\ell(r)$ can be expanded into subtraction and addition of $G(r, b^*, y^*)$ terms. The expansion is recursive. For a box b_0, let $H(r, b_0) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J \cap b_0} R(x)$. Notice that $F_\ell(r) = H(r, b_0)$ if b_0 is the entire Euclidean space. Pick any row $r \in T_\ell$. Assume we want to compute $H(r, b_0)$ for some tuple $(b_0, y_0) \in \mathcal{B}_i$.

Recall that the set of boxes in $\mathcal{B}_i$ forms a tree structure. If b_0 has no children this is the base case - $H(r, b_0) = G(r, b_0, y_0)$ by definition since all points in b_0 must be assigned to y_0. Otherwise, let $(b_1, y_1), \ldots, (b_q, y_q)$ be the tuples in $\mathcal{B}_i$ where $b_1, \ldots, b_q$ are children of b_0. Notice that, by definition all points in $b_0 \setminus (\bigcup_{j \in [q]} b_j)$ is assigned to y_0. Then, one can check that the following equation holds for any r:

$$H(r, b_0) = G(r, b_0, y_0) - \sum_{j \in [q]} G(r, b_j, y_0) + \sum_{j \in [q]} H(r, b_j)$$

Starting with setting b_0 as the entire Euclidean space, the equation above could be used to recursively expand $H(\cdot, b_0) = F_\ell(\cdot)$ into addition and subtraction of $O(|\mathcal{B}_i|)$ number of $G(\cdot, \cdot, \cdot)$ terms, where each term could be computed with one SumProd query by Lemma 5.

Runtime Analysis of the Sampling. We now discuss the running time of the sampling algorithm simulating k-means++. These lemmas show how close the probability distribution we compute is as compared to the k-means++ distribution. This will help bound the running time.

▶ **Lemma 8.** *Consider the box construction algorithm when sampling the i^{th} point in the k-means++ simulation. Consider the end of the j^{th} round where all melding is finished but the boxes have not been doubled yet. Let b be an arbitrary box in $\mathcal{G}_i$ and $h(b)$ be the number of centers in b at this time. Let c_a be an arbitrary one of these $h(b)$ centers. Then:*

A. *The distance from c_a to any $d-1$ dimensional face of b is at least 2^j.*

B. *The length of each side of b is at most $h(b) \cdot 2^{j+1}$.*

Proof. The first statement is a direct consequence of the definition of doubling and melding since at any point of time the distance of all the centers in a box is at least 2^j. To prove the second statement, we define the assignment of the centers to the boxes as following. Consider the centers inside each box b right before the doubling step. We call these centers, the centers assigned to b and denote the number of them by $h'(b)$. When two boxes b_1 and b_2 are melding into box b_3, we assign their assigned centers to b_3.

We prove each side length of b is at most $h'(b)2^{j+1}$ by induction on the number j of executed doubling steps. Since $h'(b) = h(b)$ right before each doubling, this will prove the second statement. The statement is obvious in the base case, $j = 0$. The statement also obviously holds by induction after a doubling step as j is incremented and the side lengths double and the number of assigned boxes don't change. It also holds during every meld step because each side length of the newly created larger box is at most the aggregate maximum side lengths of the smaller boxes that are moved to $\mathcal{B}_i$, and the number of assigned centers in the newly created larger box is the aggregate of the assigned centers in the two smaller boxes that are moved to $\mathcal{B}_i$. Note that since for any box b all the assigned centers to b are inside b at all times, $h'(b)$ is the number of centers inside b before the next doubling. ◀

This lemma bounds the difference of the two probability distributions.

▶ **Lemma 9.** *Consider the box generation algorithm when sampling the ith point in the k-means++ simulation. For all points x, $R(x) \leq O(i^2 d) \cdot L(x)$.*

Proof. Consider an arbitrary point x. Let c_ℓ, $\ell \in [i-1]$, be the center that is closest to x under the 2-norm distance. Assume j is minimal such that just before the $(j+1)$-th doubling round, x is contained in a box b in $\mathcal{G}_i$. We argue about the state of the algorithm at two times, the time s just before doubling round j and the time t just before doubling round $j+1$. Let b be a minimal box in $\mathcal{G}_i$ that contains x at time t, and let y be the representative for box b. Notice that we assign x to the representative of the smallest box in $\mathcal{B}_i$ that contains it, so x will be assigned to y. Indeed, none of the boxes added into $\mathcal{B}_i$ before time t contains x by the minimality of j, and when box b gets added into $\mathcal{B}_i$ (potentially after a few more doubling rounds) it still has the same representative y. By Lemma 8 the squared distance from from x to r is at most $(i-1)^2 d 2^{2j+2}$. So it is sufficient to show that the squared distance from x to c_ℓ is $\Omega(2^j)$.

Let b' be the box in $\mathcal{G}_i$ that contains c_ℓ at time s. Note that x could not have been inside b' at time s by the definition of t and s. Then by Lemma 8 the distance from c_ℓ to the edge of b' at time t is at least 2^{2j-2}, and hence the distance from c_ℓ to x is also at least 2^{2j-2} as x is outside of b'. ◀

The following theorem bounds the running time.

▶ **Theorem 10.** *The expected time complexity for running k' iterations of this implementation of k-means++ is $O(k'^4 dm\Psi(n,d,m))$.*

Proof. When picking center c_i, a point x can be sampled with probability $Q(x)$ in time $O(mi\Psi(n,m,d))$. This is because the implementation samples one row from each of the m tables. To sample one row we evaluate $O(|\mathcal{B}_i|)$ SumProd queries, each in $O(\Psi(n,m,d))$ time. As mentioned earlier $\mathcal{B}_i$ can be thought of as a tree of boxes with $i-1$ leaves, so $|\mathcal{B}_i| = O(i)$.

By Lemma 9, the probability of accepting any sampled x is $\frac{L(x)}{R(x)} = \frac{1}{O(i^2 d)}$. The expected number of sampling from Q until getting accepted is $O(i^2 d)$. Thus the expected time of finding c_i is $O(i^3 dm\Psi(n,m,d))$. Summing over $i \in [k']$, we get $O(k'^4 dm\Psi(n,m,d))$. ◀

5 Weighting the Centers

Our algorithm samples a collection C of $k' = \Theta(k \log N)$ centers using the k-means++ sampling described in the prior section. We give weights to the centers to get a coreset.

Ideally, we would compute the weights in the standard way. That is, let w_i denote the number of points that are closest to point c_i among all centers in C. These pairs of centers and weights (c_i, w_i) are known to form a coreset. Unfortunately, as stated in Theorem 2, computing such w_i's even approximately is NP hard. Instead, we will find a different set of weights which still form a coreset and are computable.

Next we describe a relational algorithm to compute a collection W' of weights, one weight $w'_i \in W'$ for each center $c_i \in C$. The proof that the centers with these alternative weights (c_i, w'_i) also form a coreset is postponed until Section 6.

Algorithm for Computing Alternative Weights. Initialize the weight w'_i for each center $c_i \in C$ to zero. In the d-dimensional Euclidean space, for each center $c_i \in C$, we generate a collection of hyperspheres (also named **balls**) $\{B_{i,j}\}_{j\in[\lg N]}$, where $B_{i,j}$ contains approximately 2^j points from J. The space is then partitioned into $\{B_{i,0}, B_{i,1} - B_{i,0}, B_{i,2} - B_{i,1}, \ldots\}$. For each partition, we will sample a small number of points and use this sample to estimate the number of points in this partition that are closer to c_i than any other centers, and thus aggregating w'_i by adding up the numbers. Fix small constants $\epsilon, \delta > 0$. The following steps are repeated for $j \in [\lg N]$:

- Let $B_{i,j}$ be a ball of radius $r_{i,j}$ centered at c_i. Find a $r_{i,j}$ such that the number of points in $J \cap B_{i,j}$ lies in the range $[(1-\delta)2^j, (1+\delta)2^j]$. This is an application of Lemma 6.
- Let τ be a constant that is at least 30. A collection $T_{i,j}$ of $\frac{\tau}{\epsilon^2}k'^2 \log^2 N$ "test" points are independently sampled following the same **approximately uniform** distribution with replacement from every ball $B_{i,j}$. Here an "approximately uniform" distribution means one where every point p in $B_{i,j}$ is sampled with a probability $\gamma_{p,i,j} \in [(1-\delta)/|B_{i,j}|, (1+\delta)/|B_{i,j}|]$ on each draw. This can be accomplished efficiently similar to the techniques used in Lemma 6 from [3]. We leave out the details due to space limit.
- Among all sampled points $T_{i,j}$, find $S_{i,j}$, the set of points that lie in the **"donut"** $D_{i,j} = B_{i,j} - B_{i,j-1}$. Then the cardinality $s_{i,j} = |S_{i,j}|$ is computed.
- Find $t_{i,j}$, the number of points in $S_{i,j}$ that are closer to c_i than any other center in C.
- Compute the ratio $f'_{i,j} = \frac{t_{i,j}}{s_{i,j}}$ (if $s_{i,j} = t_{i,j} = 0$ then $f'_{i,j} = 0$).
- If $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$ then w'_i is incremented by $f'_{i,j} \cdot 2^{j-1}$, else w'_i stays the same.

At first glance the algorithm appears naive: w'_i can be significantly underestimated if in some donuts only a small portion of points are closest to c_i, making the estimation inaccurate based on sampling. However, we prove the following theorem which shows that the alternative weights computed by our algorithm actually form a coreset.

▶ **Theorem 11.** *The centers C, along with the computed weights W', form an $O(1)$-approximate coreset with high probability.*

The running time of a naive implementation of this algorithm would be dominated by sampling of the test points. Sampling a single test point can be accomplished with m applications of the algorithm from [3] and setting the approximation error to $\delta = \epsilon/m$. Recall the running time of the algorithm from [3] is $O\left(\frac{m^6 \log^4 n}{\delta^2}\Psi(n,d,m)\right)$. Thus, the time to sample all test points is $O\left(\frac{k'^2 m^9 \log^6 n}{\epsilon^4}\Psi(n,d,m)\right)$. Substituting for k', and noting that $N \leq n^m$, we obtain a total time for a naive implementation of $O\left(\frac{k^2 m^{11} \log^8 n}{\epsilon^4}\Psi(n,d,m)\right)$.

6 Analysis of the Weighting Algorithm

The goal in this subsection is to prove Theorem 11 which states that the alternative weights form an $O(1)$-approximate coreset with high probability. Throughout our analysis, "with high probability" means that for any constant $\rho > 0$ the probability of the statement not being true can be made less than $\frac{1}{N^\rho}$ asymptotically by appropriately setting the constants in the algorithm.

Intuitively, if a decent fraction of the points in each donut are closer to center c_i than any other center, then Theorem 11 can be proven by using a straight-forward application of Chernoff bounds to show that each alternate weight w'_i is likely close to the true weight w_i. The conceptual difficultly is if only a very small portion of points in a donut $D_{i,j}$ are closer to c_i than any other points, in which case the estimated $f'_{i,j} < \frac{1}{2k'^2 \log N}$ and thus the "uncounted" points in $D_{i,j}$ would contribute no weight to the computed weight w'_i. We call this the **undersampled** case. If many donuts around a center i are undersampled, the computed weight w'_i may well poorly approximate the actual weight w_i.

To address this, we need to prove that omitting the weight from these uncounted points does not have a significant impact on the objective value. We break our proof into four parts. The first part, described in subsubsection 6.1, involves conceptually defining a fractional weight w_i^f for each center $c_i \in C$. Each point has a weight of 1, and instead of giving all this weight to its closest center, we allow fractionally assigning the weight to various "near" centers. w_i^f is then the aggregated weight over all points for c_i. The second part, described in subsubsection 6.2, establishes various properties of the fractional weight that we will need. The third part, described in subsubsection 6.3, shows that each fractional weight w_i^f is likely to be closely approximated the computed weight w_i'. The fourth part, described in subsubsection 6.4, shows that the fractional weights of the centers in C form a $O(1)$-approximate coreset. Subsubsection 6.4 also contains the proof of Theorem 11.

6.1 Defining the Fractional Weights

To define the fractional weights we first define an auxiliary directed acyclic graph $G = (S, E)$ where there is one node in S corresponding to each row in J. For the rest of this section, with a little abuse of notation, we use S to denote both the nodes in graph G, and the set of d-dimensional data points in the design matrix. Let p be an arbitrary point in $S - C$. Let $\alpha(p)$ denote the subscript of the center closest to p, i.e., if $c_i \in C$ is closest to p then $\alpha(p) = i$. Let $D_{i,j}$ be the donut around c_i that contains p. If $D_{i,j}$ is not undersampled then p will have one outgoing edge (p, c_i). Therefore, let us now assume that $D_{i,j}$ is undersampled. Defining the outgoing edges from p in this case is a bit more complicated.

Let $A_{i,j}$ be the points $q \in D_{i,j}$ that are closer to c_i than any other center in C (i.e., $\alpha(q) = i$). If $j = 1$ then $D_{i,1}$ contains only the point p, and the only outgoing edge from p goes to c_i. Therefore, let us now assume $j > 1$. Let c_h the center that is closest to the most points in $D_{i,j-1}$, the next donut in toward c_i from $D_{i,j}$. That is $c_h = \arg\max_{c_j \in C} \sum_{q \in D_{i,j-1}} \mathbb{1}_{\alpha(q)=c_j}$. Let $M_{i,j-1}$ be points in $D_{i,j-1}$ that are closer to c_h than any other center. That is, $M_{i,j-1}$ is the collection of $q \in D_{i,j-1}$ such that $\alpha(q) = h$. Then there is a directed edge from p to each point in $M_{i,j-1}$. Before defining how to derive the fractional weights from G, let us take a detour to note that G is acyclic.

▶ **Lemma 12.** *G is acyclic.*

Proof. Consider a directed edge $(p, q) \in E$, and c_i be the center in C that p is closest to, and $D_{i,j}$ the donut around c_i that contains p. Then, since $p \in D_{i,j}$ it must be the case that $\|p - c_i\|_2^2 > r_{i,j-1}$. Since $q \in B_{i,j-1}$ it must be the case that $\|q - c_i\|_2^2 \le r_{i,j-1}$. Thus $\|p - c_i\|_2^2 > \|q - c_i\|_2^2$. Thus, the closest center to q must be closer to q than the closest center to p is to p. Thus as one travels along a directed path in G, although identify of the closest center can change, the distance to the closest center must be monotonically decreasing. Thus, G must be acyclic. ◀

We explain how to compute a fractional weight w_p^f for each point $p \in S$ using the network G. Initially, each w_p^f is set to 1. Then conceptually these weights flow toward the sinks in G, splitting evenly over all outgoing edges at each vertex. More formally, the following flow step is repeated until is no longer possible to do so:

Flow Step. Let $p \in S$ be an arbitrary point that currently has positive fractional weight and that has positive outdegree h in G. Then for each directed edge (p, q) in G increment w_q^f by w_p^f/h. Finally, set w_p^f to zero.

As the sinks in G are exactly the centers in C, the centers in C will be the only points that end up with positive fractional weight. Thus, we use w_i^f to refer to the resulting fractional weight on center $c_i \in C$.

6.2 Properties of the Fractional Weights

Let $f_{i,j}$ be the fraction of points that are closest to c_i among all centers in C in this donut $D_{i,j} = B_{i,j} - B_{i,j-1}$. We show in Lemma 13 and 14 that with high probability, either the estimated ratio is a good approximation of $f_{i,j}$, or the real ratio $f_{i,j}$ is very small.

We show in Lemma 16 that the maximum flow through any node is bounded by $1+\epsilon$ when N is big enough. This follows by induction because each point has $\Omega(k' \log N)$ neighbors and every point can have in degree from one set of nodes per center. We further know every point that is not uncounted actually contributes to their centers' weight.

▶ **Lemma 13.** *With high probability, either $|f_{i,j} - f'_{i,j}| \le \epsilon f_{i,j}$ or $f'_{i,j} \le \frac{1}{2k'^2 \log N}$.*

▶ **Lemma 14.** *If $f_{i,j} > \frac{1+\epsilon}{2k'^2 \log N}$ then with high probability $f'_{i,j} \ge \frac{1}{2k'^2 \log N}$.*

The proofs of Lemmas 13 and 14 are omitted; see [22] for the full version of this work.

We now seek to bound the fractional weights computed by the algorithm. Let $\Delta_i(p)$ denote the total weight received by a point $p \in S \setminus C$ from other nodes (including the initial weight one on p). Furthermore, let $\Delta_o(p)$ denote the total weight sent by p to all other nodes. Notice that in the flow step $\Delta_o(p) = \Delta_i(p)$ for all p in $S \setminus C$.

▶ **Lemma 15.** *Let $\Delta_i(p)$ denote the total weight received by a point $p \in S \setminus C$ from other nodes (including the initial weight one on p). Furthermore, let $\Delta_o(p)$ denote the total weight sent by p to all other nodes. With high probability, for all $q \in S$, $\Delta_i(q) \le 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p)$.*

Proof. Fix the point q that redirects its weight (has outgoing arcs in G). Consider its direct predecessors: $P(q) = \{p : (p,q) \in E\}$. Partition $P(q)$ as follows: $P(q) = \bigcup_{i=1,\ldots,k'} P_{c_i}(q)$, where $P_{c_i}(q)$ is the set of points that have flowed their weights into q, but c_i is actually their closest center in C. Observe the following. The point q can only belong to one donut around c_i. Due to this, $P_{c_i}(q)$ is either empty or contains a set of points in a single donut around c_i that redirect weight to q.

Fix $P_{c_i}(q)$ for some c_i. If this set is non-empty suppose this set is in the j-th donut around c_i. Conditioned on the events stated in Lemma 13 and 14, since the points in $P_{c_i}(q)$ are undersampled, we have $|P_{c_i}(q)| \le \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$. Consider any $p \in P_{c_i}(q)$. Let β_i be the number of points that p charges its weight to (this is the same for all such points p). It is the case that β_i is at least $\frac{(1-\delta)2^{j-1}}{2k'}$ since p flows its weights to the points that are assigned to the center that has the most number of points assigned to it from c_i's $(j-1)$th donut.

Thus, q receives weight from $|P_{c_i}(q)| \le \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$ points and each such point gives its weight to at least $\frac{(1-\delta)2^{j-1}}{2k'}$ points with equal split. The total weight that q receives from points in $P_{c_i}(q)$ is at most the following.

$$\frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \Delta_o(p)$$

$$\le \frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \max_{p \in P_{c_i}(q)} \Delta_o(p)$$

$$\leq \frac{2k'}{(1-\delta)2^{j-1}} \cdot \frac{(1+\epsilon)\cdot 2^{j-1}}{2k'^2 \log N} \max_{p \in P_{c_i}(q)} \Delta_o(p) \qquad [|P_{c_i}(q)| \leq \tfrac{(1+2\epsilon)2^{j-1}}{2k'^2 \log N}]$$
$$\leq \frac{1+2\epsilon}{k' \log N} \max_{p \in P_{c_i}(q)} \Delta_o(p) \qquad [\delta \leq \tfrac{\epsilon}{2} \leq \tfrac{1}{10}]$$

Switching the max to $\max_{p:(p,q)\in E} \Delta_o(p)$, summing over all centers $c_i \in C$ and adding the original unit weight on q gives the lemma. ◀

The following crucial lemma bounds the maximum weight that a point can receive.

▶ **Lemma 16.** *Fix η to be a constant smaller than $\frac{\log(N)}{10}$ and $\epsilon < 1$. Say that for all $q \in S \setminus C$ it is the case that $\Delta_o(q) = \eta\Delta_i(q)$. Then, with high probability for any $p \in S \setminus C$ it is the case that $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$.*

Proof. We can easily prove this by induction on nodes. The lemma is true for all nodes that have no incoming edges in G. Now assume it is true for all nodes whose longest path that reaches them in G has length $t-1$. Now we prove it for nodes whose longest path that reaches then in G is t. Fix such a node q. For any node p such that $(p,q) \in E$, by induction we have $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$, so $\Delta_o(p) \leq 2(1 + \frac{2\eta}{\log N})$. By Lemma 15, $\Delta_i(q) \leq 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p) \leq 1 + \left(\frac{\eta(1+2\epsilon)}{\log N}\right)\left(1 + \frac{2\eta}{\log N}\right) = 1 + \frac{\eta}{\log N} + \frac{\eta}{\log N} \cdot \frac{2(1+2\epsilon)\eta + 2\epsilon}{\log N} \leq 1 + \frac{2\eta}{\log N}$. ◀

6.3 Comparing Alternative Weights to Fractional Weights

It only remains to bound the cost of mapping points to the centers they contribute weight to. This can be done by iteratively charging the total cost of reassigning each node with the flow. In particular, each point will only pass its weight to nodes that are closer to their center. We can charge the flow through each node to the assignment cost of that node to its closest center, and argue that the cumulative reassignment cost bounds the real fractional assignment cost. Further, each node only has $1+\epsilon$ flow going through it. This will be sufficient to bound the overall cost in Lemma 18.

▶ **Lemma 17.** *With high probability, for every center c_i, it is the case that the estimated weight w'_i computed by the weighting algorithm is $(1 \pm 2\epsilon)w^f_i$ where w^f_i is the fractional weight of i.*

The proofs of Lemmas 17 is omitted; see [22] for the full version of this work.

6.4 Comparing Fractional Weights to Optimal

Next, we bound the total cost of the fractional assignment defined by the flow. According to the graph G, any point $p \in S$ and $c_i \in C$, we let $\omega(p, c_i)$ be the fraction of weights that got transferred from p to c_i. Naturally we have $\sum_{c_i \in C} \omega(p, c_i) = 1$ for any $p \in S$ and the fractional weights $w^f_i = \sum_{p \in S} \omega(p, c_i)$ for any $c_i \in C$.

▶ **Lemma 18.** *Let ϕ_{opt} be the optimal k-means cost on the original set S. With high probability, it is the case that:*

$$\sum_{p \in S} \sum_{c_i \in C} \omega(p, c_i) \|p - c_i\|^2 \leq 160(1+\epsilon)\phi_{opt}$$

Proof. Let $\phi^* = \sum_{p \in S} \|p - c_{\alpha(p)}\|^2$. Consider any $p \in S$ and center c_i such that $\omega(p, c_i) > 0$. Let P be any path from p to c_i in G. If node p's only outgoing arc is to its closest center $c_{\alpha(p)} = c_i$, then $P = p \to c_i$, we have $\sum_{c \in C} \omega(p, c)\|p - c\|^2 = \|p - c_{\alpha(p)}\|^2$. Otherwise assume $P = p \to q_1 \to q_2 \to \ldots \to q_\ell \to c_i$. Note that the closest center to q_ℓ is c_i. Let $\Delta(P)$ be the fraction of the original weight of 1 on p that is given to c_i along this path according to the flow of weights. As we observed in the proof of Lemma 12, we have $\|p - c_{\alpha(p)}\| > \|q_1 - c_{\alpha(p)}\| \geq \|q_1 - c_{\alpha(q_1)}\| > \|q_2 - c_{\alpha(q_1)}\| \geq \|q_2 - c_{\alpha(q_2)}\| > \ldots > \|q_\ell - c_{\alpha(q_\ell)}\|$. This follows because for any arc (u, v) in the graph, v is in a donut closer to $c_{\alpha(u)}$ than the donut u is in, and v is closer to $c_{\alpha(v)}$ than $c_{\alpha(u)}$.

We use the relaxed triangle inequality for squared ℓ_2 norms. For any three points x, y, z, we have $\|x - z\|^2 \leq 2(\|x - y\|^2 + \|y - z\|^2)$. Thus, we bound $\|p - c_i\|^2$ by

$$\begin{aligned}
\|p - c_i\|^2 &= \|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1 + q_1 - c_i\|^2 \\
&\leq 2\|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1\|^2 + 2\|q_1 - c_i\|^2 && \text{[relaxed triangle inequality]} \\
&\leq 2(\|p - c_{\alpha(p)}\| + \|c_{\alpha(p)} - q_1\|)^2 + 2\|q_1 - c_i\|^2 && \text{[triangle inequality]} \\
&\leq 8\|p - c_{\alpha(p)}\|^2 + 2\|q_1 - c_i\|^2 && [\|p - c_{\alpha(p)}\| \geq \|c_{\alpha(p)} - q_1\|].
\end{aligned}$$

Applying the prior steps to each q_i gives the following.

$$\|p - c_i\|^2 \leq 8(\|p - c_{\alpha(p)}\|^2 + \sum_{j=1}^{\ell} 2^j \|q_j - c_{\alpha(q_j)}\|^2)$$

Let $\mathcal{P}_q(j)$ be the set of all paths P that reach point q using j edges. If $j = 0$, it means P starts with point q. We seek to bound $\sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)\|q - c_{\alpha(q_j)}\|^2$. This will bound the charge on point q above over all path P that contains it.

Define a weight function $\Delta'(p)$ for each node $p \in S \setminus C$. This will be a new flow of weights like Δ, except now the weight increases at each node. In particular, give each node initially a weight of 1. Let $\Delta'_o(p)$ be the total weight leaving p. This will be evenly divided among the nodes that have outgoing edges from p. Define $\Delta'_i(p)$ to be the weight incoming to p from all other nodes plus one, the initial weight of p. Set $\Delta'_o(p)$ to be $2\Delta'_i(p)$, twice the incoming weight.

Lemma 16 implies that the maximum weight of any point p is $\Delta'_i(p) \leq 1 + \frac{4}{\log N}$. Further notice that for any q it is the case that $\Delta'_i(q) = \sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)$. Letting $\mathcal{P}(p, c_i)$ be the set of all paths that start at p to center c_i. Notice such paths correspond to how p's unit weight goes to c_i. We have $\omega(p, c_i) = \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)$. Let $\mathcal{P}$ denote the set of all paths, $\ell(P)$ denote the length of path P (number of edges on P) , and let $P(j)$ denote the jth node on path P. Thus we have the following.

$$\begin{aligned}
&\sum_{p \in S} \sum_{c_i \in C} \omega(p, c_i)\|p - c_i\|^2 = \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)\|p - c_i\|^2 \\
\leq\; & 8 \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p, c_i)} \Delta(P)(\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2) \\
=\; & 8 \sum_{P \in \mathcal{P}} \Delta(P)(\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2) \\
=\; & 8 \sum_{q \in S} \sum_{j=0}^{+\infty} \sum_{P \in \mathcal{P}_q(j)} 2^j \Delta(P)\|q - c_{\alpha(q)}\|^2 = 8 \sum_{q \in S} \Delta'_i(q)\|q - c_{\alpha(q)}\|^2 \\
\leq\; & \sum_{q \in S} 8(1 + \frac{4}{\log N})\|q - c_{\alpha(q)}\|^2 = 8(1 + \frac{4}{\log N})\phi^*
\end{aligned}$$

Lemma 18 follows because if $k' \geq 1067k \log N$, $\phi^* \leq 20\phi_{opt}$ with high probability by Theorem 1 in [8]. ◀

Finally, we prove that finding any $O(1)$-approximation solution for optimal weighted k-means on the set (C, W') gives a constant approximation for optimal k-means for the original set S. Let $W^f = \{w_1^f, \dots, w_{k'}^f\}$ be the fractional weights for centers in C. Let $\phi^*_{W^f}$ denote the optimal weighted k-means cost on (C, W^f), and $\phi^*_{W'}$ denote the optimal weighted k-means cost on (C, W'). We first prove that $\phi^*_{W^f} = O(1)\phi_{\mathrm{OPT}}$, where ϕ_{OPT} denote the optimal k-means cost on set S.

▶ **Lemma 19.** *Let (C, W^f) be the set of points sampled and the weights collected by fractional assignment ω. With high probability, we have $\phi^*_{W^f} = O(1)\phi_{\mathrm{OPT}}$.*

Proof. Consider the cost of the fractional assignment we've designed. For $c_i \in C$, the weight is $w_i^f = \sum_{p \in S} \omega(p, c_i)$. Denote the k-means cost of ω by $\phi_\omega = \sum_{p\in S}\sum_{c \in C} \omega(p, c)\|p - c\|^2$. By Lemma 18, we have that $\phi_\omega \leq 160(1+\epsilon)\phi_{\mathrm{OPT}}$.

Intuitively, in the following we show $\phi^*_{W^f}$ is close to ϕ_ω. As always, we let C_{OPT} denote the optimal centers for k-means on set S. For a set of points X with weights $Y : X \to \mathbb{R}^+$ and a set of centers Z, we let $\phi_{(X,Y)}(Z) = \sum_{x \in X} Y(x) \min_{z \in Z} \|x - z\|^2$ denote the cost of assigning the weighted points in X to their closest centers in Z. Note that $\phi^*_{W^f} \leq \phi_{(C,W^f)}(C_{\mathrm{OPT}})$ since C_{OPT} is chosen with respect to S.

$$\begin{aligned}
\phi^*_{W^f} \leq \phi_{(C,W^f)}(C_{\mathrm{OPT}}) &= \sum_{c_i \in C}\sum_{p \in S} \min_{c \in C_{\mathrm{OPT}}} \omega(p, c_i)\|c_i - c\|^2 && [w_i^f = \textstyle\sum_{p\in S} \omega(p, c_i)] \\
&\leq \sum_{c_i \in C}\sum_{p \in S} \min_{c \in C_{\mathrm{OPT}}} \omega(p, c_i) \cdot 2(\|p - c_i\|^2 + \|p - c\|^2) && [\text{relaxed triangle inequality}] \\
&= 2\phi_\omega + 2\phi_{\mathrm{OPT}} \leq 322(1+\epsilon)\phi_{\mathrm{OPT}}
\end{aligned}$$
◀

Using the mentioned lemmas, we can prove the final approximation guarantee.

Proof of Theorem 11. Using Lemma 17, we know $w_i' = (1 \pm 2\epsilon)w_i^f$ for any center c_i. Let C_k' be k centers for (C, W') that is a γ-approximate for optimal weighted k-means. Let C^f_{OPT} be the *optimal* k centers for (C, W^f), and C'_{OPT} optimal for (C, W'). We have $\phi_{(C,W^f)}(C_k') \leq (1+2\epsilon)\phi_{(C,W')}(C_k')$ for the reason that the contribution of each point grows by at most $(1+2\epsilon)$ due to weight approximation. Using the same analysis, $\phi_{(C,W')}(C^f_{\mathrm{OPT}}) \leq (1+2\epsilon)\phi^*_{W^f}$. Combining the two inequalities, we have

$$\begin{aligned}
\phi_{(C,W^f)}(C_k') &\leq (1+2\epsilon)^2\phi_{(C,W')}(C_k') \leq (1+2\epsilon)^2\gamma\phi^*_{W'} \\
&\leq (1+2\epsilon)^2\gamma\phi_{(C,W')}(C^f_{\mathrm{OPT}}) && [\text{by optimality of } \phi^*_{W'}] \qquad (1) \\
&\leq (1+2\epsilon)^3\gamma\phi^*_{W^f} \leq 322\gamma(1+2\epsilon)^4\phi_{\mathrm{OPT}} && [\text{using Lemma 19}]
\end{aligned}$$

Let $\phi_S(C_k') = \sum_{p \in S} \min_{c \in C_k'} \|p - c\|^2$. For every point $p \in S$, to bound its cost $\min_{c \in C_k'} \|p - c\|^2$, we use multiple relaxed triangle inequalities for every center $c_i \in C$, and take the weighted average of them using $\omega(p, c_i)$.

$$\begin{aligned}
\phi_S(C_k') &= \sum_{p \in S} \min_{c \in C_k'} \|p - c\|^2 = \sum_{p \in S}\sum_{c_i \in C} \omega(p, c_i) \min_{c \in C_k'} \|p - c\|^2 && [\textstyle\sum_{c_i \in C} \omega(p, c_i) = 1] \\
&\leq \sum_{p \in S}\sum_{c_i \in C} \omega(p, c_i) \min_{c \in C_k'} 2(\|p - c_i\|^2 + \|c_i - c\|^2) && [\text{relaxed triangle inequality}] \\
&= 2\phi_\omega + 2\phi_{(C,W^f)}(C_k') && [\textstyle\sum_{p \in S} \omega(p, c_i) = w_i^f] \\
&\leq 2\phi_\omega + 2 \cdot 322\gamma(1+2\epsilon)^4\phi_{\mathrm{OPT}} && [\text{inequality (1)}] \\
&\leq 2 \cdot 160(1+\epsilon)\phi_{\mathrm{OPT}} + 2 \cdot 322\gamma(1+2\epsilon)^4\phi_{\mathrm{OPT}} && [\text{Lemma 18}] \\
&= O(\gamma)\phi_{\mathrm{OPT}}
\end{aligned}$$
◀

References

1 https://cloud.google.com/bigquery-ml/docs/bigqueryml-intro.

2 Kaggle machine learning and data science survey. https://www.kaggle.com/kaggle/kaggle-survey-2018, 2018.

3 Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. Approximate aggregate queries under additive inequalities. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 85–99. SIAM, 2021.

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

5 Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: in-database learning thunderstruck. In *Second Workshop on Data Management for End-To-End Machine Learning*, page 8. ACM, 2018.

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

7 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, page 13–28, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2902251.2902280.

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

9 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

10 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *IEEE Symposium on Foundations of Computer Science*, pages 739–748, 2008.

11 Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

12 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *International Conference on Machine Learning*, pages 576–585, 2017.

13 George Casella, Christian P Robert, Martin T Wells, et al. Generalized accept-reject sampling schemes. In *A Festschrift for Herman Rubin*, pages 342–347. Institute of Mathematical Statistics, 2004.

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

15 Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Rk-means: Fast clustering for relational data. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2742–2752. PMLR, 2020.

16 Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 681–689, 2011.

17 Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

18 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.

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

20 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016. `doi:10.1137/130938645`.

21 Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1-3):61–87, 2004.

22 Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering, 2020. `arXiv:2008.00358`.

23 Steffen Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6(5), pages 337–348. VLDB Endowment, 2013.

24 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18. ACM, 2016.

25 Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *Symposium on Foundations of Computer Science*, pages 802–813, 2018.

26 Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. Towards factorized svm with gaussian kernels over normalized data.

27 Clement Tak Yu and Meral Z Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *Computer Software and The IEEE Computer Society's Third International Applications Conference*, pages 306–312. IEEE, 1979.