Rk-means: Fast Clustering for Relational Data

Ryan R. Curtin
RelationalAI
ryan@ratml.org

Benjamin Moseley
Tepper School of Business
Carnegie Mellon University
moseleyb@andrew.cmu.edu

Hung Q. Ngo
RelationalAI
hung.q.ngo@relational.ai

XuanLong Nguyen
Department of Statistics
University of Michigan
xuanlong@umich.edu

Dan Olteanu
University of Oxford
dan.olteanu@cs.ox.ac.uk

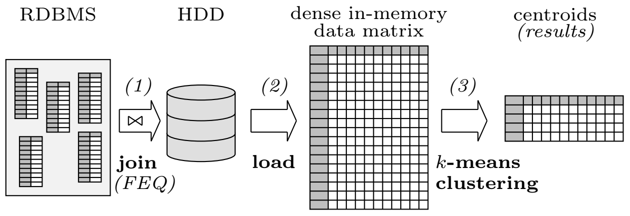
Maximilian Schleich
University of Oxford
max.schleich@cs.ox.ac.uk

Abstract

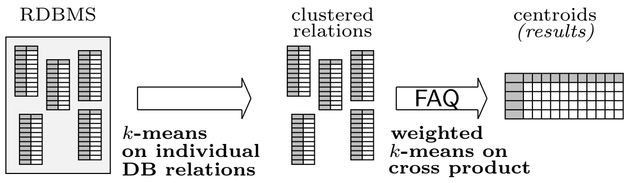
Conventional machine learning algorithms cannot be applied until a data matrix is available to process. When the data matrix needs to be obtained from a relational database via a feature extraction query, the computation cost can be prohibitive, as the data matrix may be (much) larger than the total input relation size. This paper introduces Rk-means, or relational k-means algorithm, for clustering relational data tuples without having to access the full data matrix. As such, we avoid having to run the expensive feature extraction query and storing its output. Our algorithm leverages the underlying structures in relational data. It involves construction of a small *grid coreset* of the data matrix for subsequent cluster construction. This gives a constant approximation for the k-means objective, while having asymptotic runtime improvements over standard approaches of first running the database query and then clustering. Empirical results show orders-of-magnitude speedup, and Rk-means can run faster on the database than even just computing the data matrix.

1 Introduction

Clustering is an ubiquitous technique for exploratory data analysis, whether applied to small samples or industrial-scale data. In the latter setting, two steps are typically performed: *(1) data preparation*, or extract-transform-load (ETL) operations, and *(2) clustering the extracted data*—often with a technique such as the popular k-means algorithm (Cady, 2017; Wu et al., 2008). In this setting, data often reside in a relational database, requiring a *feature extraction query* (FEQ) to be performed, *joining* involved relations together to form the data matrix: each row corresponds to a data tuple and each column a feature. Then, the data matrix is used as input to a clustering algorithm. This matrix can be expensive to compute, and may take up



(a) Typical k-means data science workflow. Alternate representations can be used for the data in step *(2)* for greater computational efficiency (e.g., streaming); and, approximation strategies are known for step *(3)*. However, the dataset often comes from an underlying database system, and in this case the expensive FEQ join *(1)* is unavoidable.


RDBMS
clustered relations
centroids (results)
k-means on individual DB relations
FAQ
weighted k-means on cross product

(b) The Rk-means data science workflow. We avoid ever computing the expensive FEQ by instead clustering each underlying relation (steps 1 and 2, Section 4); we then use FAQs for efficient weighted k-means of the cross-product of those relations (steps 3 and 4, Section 4). This gives significant empirical and theoretical accelerations, and bounded approximation—*without ever computing the full data matrix.*

Figure 1: Conventional k-means and Rk-means.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

space asymptotically larger than the database itself. Moreover, the join computation time may exceed the time it takes to obtain clusters. As a result it is not uncommon that the exploratory trip into the dataset may be stopped right at the gate.

As an example, consider a retailer database consisting of three tables: `product`, which contains data about p products, `store`, which contains data about s stores, and `transaction`, which contains the number of transactions for each (product, store) combination on a given day. A practitioner may want to cluster each (product, store) combination as part of an analysis to determine items with related sales patterns across different stores for a given week. To do this, she constructs a data matrix containing all (product, store) combinations (including those with zero sales) for a given week, and additional attributes for each product and store. This can be achieved, for instance, by the following feature extraction query, given in SQLite syntax:

```
SELECT P.id AS i, S.id AS s, P.type AS t, P.price AS p,
       S.yelp_rating AS y, sum(ifnull(T.count, 0)) AS c
    FROM product P, store S LEFT JOIN transactions T
    ON T.product_id == P.id AND T.store_id == S.id
       AND T.date BETWEEN '2019-05-13' AND '2019-05-20'
    GROUP BY P.id, S.id;
```

The result of this query is of size $\Theta(ps)$. But the `transaction` table can be significantly smaller than this, since many stores may have zero sales of a particular product in a given week. Thus, the size of the data matrix can be asymptotically greater than the total input relations' sizes. Real-world FEQs possess a similar explosion in both space and time complexity, only at a much larger scale, since they generally involve many more aggregations and tables. In Section 5, we present a real dataset from a large US retailer. The database has 6 tables of total size 1.5GB. The FEQ result, however, takes up 18GB, and constructing it takes longer than running a learning algorithm on it.

Stripping away the language of databases, a fundamental challenge is how to learn about the joint distribution of a data population given only marginal samples revealed by relational tables. This is possible when the objective function of an underlying model admits some factorization structure similar to conditional independence in graphical models (Koller and Friedman, 2009). This insight was exploited recently by database theorists to devise algorithms evaluating a generic class of relational queries called *functional aggregate queries*, or FAQs (Abo Khamis et al., 2016). The ability to answer FAQs quickly is a building block for a new class of efficient algorithms for training supervised learning models over relational data, *without* having to materialize (i.e. compute) the entire data matrix (Abo Khamis et al., 2018b,a; Schleich et al., 2016).

The goal of this paper is to devise a method for fast clustering of relational data, without having to materialize the full data matrix. The challenge of unsupervised learning tasks in general and the k-means algorithm in particular is that the learning objective is not decomposable across marginal samples in relational tables. To enable fast relational computation, we utilize the idea of constructing a *grid coreset*—a small set of points that provide a good summarization of the original (and unmaterialized) data tuples, based on which a provably good clustering can be obtained.

The resulting algorithm, which we call Rk-means, has several remarkable properties. First, Rk-means has a provable constant approximation guarantee relative to the k-means objective, despite the fact that the algorithm does not require access to the full data matrix. Our approximation analysis is established via a connection of Rk-means to the theory of optimal transport (Villani, 2009). Second, Rk-means is enhanced by leveraging structures prevalent in relational data: categorical variables, functional dependencies, and the topology of feature extraction queries. These structures lead to exponential reduction in coreset size without incurring loss in the coreset's approximation error. We show that Rk-means is provably more efficient both in time and space complexity when comparing against the application of the vanilla k-means to the full data matrix. Finally, experimental results show significant speedups with little loss in the k-means objective. We observe orders-of-magnitude improvement in the running time compared to traditional methods. Rk-means operates fine when other approaches would run out of memory, enabling clustering on truly massive datasets.

2 Discussion of related work

Coresets for clustering. From early work on k-means algorithm (Lloyd, 1982), ideas emerged for acceleration via coresets (Har-Peled and Mazumdar, 2004; Bachem et al., 2018). Coresets have become the cornerstone of modern streaming algorithms (Guha et al., 2003; Braverman et al., 2017), massively parallel (MPC) algorithms (Ene et al., 2011; Bahmani et al., 2012), and are used to speed up sequential methods (Meyerson et al., 2004; Sohler and Woodruff, 2018).

Unfortunately, existing algorithms for coreset construction do not readily lend themselves to the relational setting; there are several hurdles. First of all, coresets are formed by constructing a set S of data points (tuples) that represent the entire data set X well. Typically, S is a *weighted* representation of the data, where each point in the universe contributes one unit of weight to its closest point in X (Har-Peled and Mazumdar, 2018; Ene et al., 2011; Balcan et al., 2013). In our relational setting, X can *only* be formed by computing the FEQ, but our goal is to *avoid* materializing X.

Unfortunately, for our setting, most existing coreset algorithms construct S in phases by determining the farthest points from S (Thorup, 2001; Arthur and Vassilvitskii, 2007; Har-Peled and Mazumdar, 2018). This is difficult without X fully materialized. Another difficulty is that, even if the points in S are given, weighting the points in S is an open problem for relational algorithms (Khamis et al., 2019): consider that for a point $x \in S$, finding the number of closest points in (unmaterialized) X is non-trivial, as the points and their attributes are stored across several tables. No method, either deterministic or stochastic (e.g., sampling), is known that runs in time asymptotically faster than computing/materializing X. Our method avoids this by constructing a *grid coreset* S which can be decomposed over the tables in such a way that computing the weights of the points is a straightforward task.

Other Work. Our work draws inspiration from three lines of existing work and ideas: coresets for clustering (above), database algorithms, and optimal transport. As one example, database and disk hardware optimizations have been considered to improve clustering relational data (Ordonez, 2006; Ordonez and Omiecinski, 2004). Recent advances include the work of Abo Khamis et al. (2018b) and Schleich et al. (2016). k-means has also been connected to optimal transport, which goes back to at least (Pollard, 1982) (see also (Graf and Luschgy, 2000)). Recently this connection has received increased interest in the statistics and machine learning communities, resulting in fresh new clustering techniques (del Barrio et al., 2017; Ho et al., 2017; Ye et al., 2017). To our knowledge, these related lines of work have not been explored together. Motivated by clustering relational data, our attempt at solving a clustering problem formulated as optimal transport in the marginal (projected) spaces to scalably perform k-means clustering appears to be the first.

Finally, it is worth noting that despite its popularity, the basic k-means technique is not always a preferred choice in clustering categorical or high-dimensional data. One may either adopt other clustering techniques (Hartigan, 1975; Kaufman and Roussew, 1990; Everitt et al., 2011), or modify the basic k-means method, e.g., by suitably placing weights on different features of mixed data types and replacing metric ℓ_2 by ℓ_p (Huang, 1998), or incorporating a regularizer to combat high dimensionality (Witten and Tibshirani, 2010; Sun et al., 2012). As we shall see, the relational techniques and associated theory that we introduce for the basic k-means extend easily to such improvements.

3 The Rk-means algorithm

Although the Rk-means algorithm is motivated by application to relational databases, its basic idea is also of independent interest and can be easily described without the database language.

Algorithm 1 Rk-means: k-means via grid-coreset

```
 1: Input: query $Q$, number of clusters $k$
 2: Input: $[d] = S_1 \cup \cdots \cup S_m$, $\kappa \geq 2$
 3: Output: centroids $\boldsymbol{C} \in \mathcal{R}^{k \times d}$
 4: for $j = 1$ to $m$ do
 5:     $\boldsymbol{X}_j \leftarrow \{\boldsymbol{x}_{S_j} \mid \boldsymbol{x} \in \boldsymbol{X}\}$
 6:     $w_j \leftarrow$ weight function defined in (2)
 7:     $\boldsymbol{C}_j \leftarrow \mathsf{wkmeans}_1(\boldsymbol{X}_j, w_j, \kappa)$
 8: end for
 9: $\boldsymbol{G} \leftarrow \boldsymbol{C}_1 \times \ldots \times \boldsymbol{C}_m$ {the grid coreset}
10: $w_{\mathsf{grid}} \leftarrow$ weight function defined in (3)
11: $\boldsymbol{C} \leftarrow \mathsf{wkmeans}_2(\boldsymbol{G}, w_{\mathsf{grid}}, k)$
```

First we define the *weighted* k-means problem, which Rk-means solves (weights are also handy in combining mixed data types (Huang, 1998)). Let $\boldsymbol{X}$ be a set of points in $\mathbb{R}^d$, and $\boldsymbol{Y}$ be a non-empty set of points in the same space. Let $d(\boldsymbol{x}, \boldsymbol{Y}) := \min_{\boldsymbol{y} \in \boldsymbol{Y}} \|\boldsymbol{x} - \boldsymbol{y}\|$ denote the minimum distance from $\boldsymbol{x}$ to an element in $\boldsymbol{Y}$. In some cases, the ℓ_2 norm $\|\cdot\|$ may be replaced by the ℓ_p norm $\|\cdot\|_p$ for some $p \geq 1$. A *weighted k-means instance* is a pair $(\boldsymbol{X}, w)$, where $\boldsymbol{X}$ is a set of points in $\mathbb{R}^d$ and $w : \boldsymbol{X} \to \mathbb{R}^+$ is a weight function. Without loss of generality, assume $\sum_{\boldsymbol{x} \in \boldsymbol{X}} w(\boldsymbol{x}) = 1$. The task is to find a set $\boldsymbol{C} = \{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_k\}$ of k centroids to minimize the objective $L(\boldsymbol{X}, \boldsymbol{C}, w) = \sum_{x \in \boldsymbol{X}} w(\boldsymbol{x}) d(\boldsymbol{x}, \boldsymbol{C})^2$.

That is, we want to solve the problem $\mathsf{OPT}(\boldsymbol{X}, w) := \min_{\boldsymbol{C}} L(\boldsymbol{X}, \boldsymbol{C}, w)$. With Rk-means, we will do this by projecting $\boldsymbol{X}$ onto different sets of coordinates, and clustering each projection individually. To this end, let $[d] = S_1 \cup \cdots \cup S_m$ denote an arbitrary *partition* of the dimensions $[d]$ into non-empty subsets. For every $\boldsymbol{x} \in \mathbb{R}^d$, and $j \in [m]$, let $\boldsymbol{x}_{S_j}$ denote the projection of $\boldsymbol{x}$ onto the coordinates in S_j. Define the projection set $\boldsymbol{X}_j$ and corresponding weight function $w_j : \mathbb{R}^{S_j} \to R$ by

$$\boldsymbol{X}_j := \left\{\boldsymbol{x}_{S_j} \mid \boldsymbol{x} \in \boldsymbol{X}\right\}, \tag{1}$$

$$w_j(\boldsymbol{z}) := \sum_{\boldsymbol{x} \in \boldsymbol{X} \;:\; \boldsymbol{x}_{S_j} = \boldsymbol{z}} w(\boldsymbol{x}). \tag{2}$$

In words, the w_j are the *marginal measures* of w on the subspace of coordinates S_j. With these notations established, Algorithm 1 presents the high-level description of our algorithm, Rk-means.

For each $j \in [m]$, in line 7 we perform k-means to obtain κ individual clusters on each subspace S_j for some $\kappa \geq 2$. These are solved using some weighted k-means algorithm denoted by $\mathsf{wkmeans}_1$ with approximation ratio α.[1] Then, using the results of these clusterings, we

[1]That is, $\mathsf{wkmeans}_1$ finds clusterings $\hat{\boldsymbol{C}}$ where $L(\boldsymbol{X}, \hat{\boldsymbol{C}}, w)/L(\boldsymbol{X}, \boldsymbol{C}^*, w) \leq \alpha$, where $\boldsymbol{C}^* = \operatorname{argmin}_{\boldsymbol{C}} L(\boldsymbol{X}, \boldsymbol{C}, w)$.

assemble a cross-product weighted grid $\boldsymbol{G}$ of centroids, and then perform k-means clustering on these using the algorithm denoted $\mathsf{wkmeans}_2$ to reduce down to the desired result of k centroids. Typically, we take $\kappa = O(k)$. Note also that Rk-means could easily be modified to use a different κ value for different subspaces S_j.

Let $\boldsymbol{X} := \biguplus_{\boldsymbol{g}\in\boldsymbol{G}} \boldsymbol{X_g}$ denote a partition of $\boldsymbol{X}$ into $|\boldsymbol{G}|$ parts, where $\boldsymbol{X_g}$ denote the set of points in $\boldsymbol{X}$ closer to $\boldsymbol{g}$ than other grid points in $\boldsymbol{G}$ (breaking ties arbitrarily). Then, the weight function $w_{\mathsf{grid}} : \boldsymbol{G} \to \mathbb{R}^+$ (line 11) is

$$w_{\mathsf{grid}}(\boldsymbol{g}) := \sum_{\boldsymbol{x}\in\boldsymbol{X_g}} w(\boldsymbol{x}). \quad (3)$$

Weighted k-means and optimal transport. We will analyze Rk-means in the language of optimal transport. The connection of k-means in general, and of our algorithm to optimal transport in particular, provides another interesting insight into our algorithm.

The *optimal transport distance* characterizes the distance between two probability measures, by measuring the optimal cost of transporting mass from one to another (Villani, 2009). Although this is defined more generally for any two probability measures in abstract spaces, for our purpose it is convenient to consider two discrete probability measures P and P' on $\mathbb{R}^d$.

Let $\boldsymbol{Z}$ and $\boldsymbol{Z}'$ be two finite point sets in $\mathbb{R}^d$. Let δ denote the Dirac measure. Let $P := \sum_{\boldsymbol{z}\in\boldsymbol{Z}} p(\boldsymbol{z})\delta_{\boldsymbol{z}}$ and $P' := \sum_{\boldsymbol{z}'\in\boldsymbol{Z}'} p'(\boldsymbol{z}')\delta_{\boldsymbol{z}'}$ be two measures with supports $\boldsymbol{Z}$ and $\boldsymbol{Z}'$, respectively. The mass transportation plan is formalized by a *coupling*: a joint distribution $\boldsymbol{Q} = (q(\boldsymbol{z},\boldsymbol{z}'))_{(\boldsymbol{z},\boldsymbol{z}')\in\boldsymbol{Z}\times\boldsymbol{Z}}$, where the marginal constraints $\sum_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{z},\boldsymbol{z}'} = p'(\boldsymbol{z}')$ and $\sum_{\boldsymbol{z}'\in\boldsymbol{Z}'} q_{\boldsymbol{z},\boldsymbol{z}'} = p(\boldsymbol{z})$ hold.

Definition 3.1. For any $p \geq 1$, the *Wasserstein distance* of order p is defined by the minimization of $\boldsymbol{Q}$ over all possible couplings: $\mathsf{W}_p(P,P') = \min_{\boldsymbol{Q}}(\{\sum_{\boldsymbol{z},\boldsymbol{z}'} q(\boldsymbol{z},\boldsymbol{z}')\,\|\boldsymbol{z}-\boldsymbol{z}'\|_p^p)\}^{1/p}$.

Let $P^{\text{in}} = \sum_{\boldsymbol{x}\in\boldsymbol{X}} w(\boldsymbol{x})\delta_{\boldsymbol{x}}$ be the discrete measure associated with the input instance of our weighted k-means problem; then, this can be expressed precisely as an optimal transport problem: $M^* = \arg\min\ \mathsf{W}_2^2(M, P^{\text{in}})$, where the optimization is over the space of discrete measures M that have k support points (the set $\boldsymbol{C}$ of k centroids). Note that $\mathsf{OPT}(\boldsymbol{X}, w) = \mathsf{W}_2^2(M^*, P^{\text{in}})$. Replacing ℓ_2 by say ℓ_1, we obtain the k-median problem, for which the objective becomes $W_1(M^*, P^{\text{in}})$.

Approximation Analysis. We next analyze the approximation ratio of Rk-means working with the W_2^2 objective, provided that $\mathsf{wkmeans}_1$ has approximation ratio α and $\mathsf{wkmeans}_2$ has approximation ratio γ.[2] The reason we might want to invoke different algorithms to solve these sub-problems is because, as we shall show in the next section, we may want to exploit the (relational) structures of the FEQ to construct a "nice" partition $S_1 \cup \cdots \cup S_m$. We show that the overall approximation ratio of Rk-means is $(\sqrt{\alpha} + \sqrt{\gamma} + \sqrt{\alpha\gamma})^2$. In many common cases, the database has structure that allows $\alpha = 1$, yielding an overall ratio of $(1 + 2\sqrt{\gamma})^2$.

For our analysis it is useful to understand Algorithm 1 in the language of optimal transport. For any finite point set $\boldsymbol{Y} \subset \mathbb{R}^d$ and a measure $M = \sum_{\boldsymbol{y}\in\boldsymbol{Y}} p(\boldsymbol{y})\delta_{\boldsymbol{y}}$ with support $\boldsymbol{Y}$, define the marginal measures M_j on coordinates S_j *induced* by M in the natural way, i.e. $M_j := \sum_{\boldsymbol{z}\in\boldsymbol{Z}} p_j(\boldsymbol{z})\delta_{\boldsymbol{z}}$ where p_j is defined analogous to w_j in (2). Under this notation, P^{in} induces the marginal measures $P_j^{\text{in}} := \sum_{\boldsymbol{z}\in\boldsymbol{X}_j} w_j(\boldsymbol{z})\delta_{\boldsymbol{z}}$. Then, Algorithm 1 can be described by the following steps:

(1) For each $j \in [m]$, pick M_j to be the (α-approximate) minimizer of $\mathsf{W}_2^2(M_j, P_j^{\text{in}})$, where $\boldsymbol{C}_j = \text{supp}(M_j)$ is the support of M_j and $|\boldsymbol{C}_j| = \kappa$ (line 7).

(2) Collect the κ^d grid points $\boldsymbol{G}$ and let probability measure Q be the one with support in $\boldsymbol{G}$ such that Q minimizes $\mathsf{W}_2^2(Q, P^{\text{in}})$. (We solve this problem exactly!)

(3) Finally, return P which is the measure with exactly k support points in $\mathbb{R}^d$ that (γ-approximately) minimizes $\mathsf{W}_2^2(P, Q)$ (line 11).

This is precisely the solution obtained by Algorithm 1. We present next some useful facts.

Lemma 3.2. *For any discrete measure M on $\mathbb{R}^d$, $\mathsf{W}_2^2(M, P^{in}) \geq \sum_{j=1}^m \mathsf{W}_2^2(M_j, P_j^{in})$.*

Proof. Immediate: a coupling of two measures induces valid marginal couplings of marginal measures. □

Proposition 3.3. *(a) If $\kappa \geq |supp(M_j^*)|\ \forall\ j \in [m]$, then $\mathsf{W}_2(P^{in}, P) \leq (\sqrt{\gamma}+\sqrt{\alpha}+\sqrt{\gamma\alpha})\mathsf{W}_2(P^{in}, M^*)$. (b) For any $\kappa \geq 1$, there exists a distribution P^{in} such that $\frac{\mathsf{W}_2(P^{in},P)}{\mathsf{W}_2(P^{in},M^*)} \geq \sqrt{1 - e^{-m/(2\kappa)}}\frac{\sqrt{3}k^{3/(2m)}}{2\kappa m^{1/2}}$.*

The condition of part (a) is satisfied, for instance, by setting $\kappa = k$. In practice, $\kappa < k$ may suffice. Moreover, part (b) dictates that κ must grow with k appropriately for our algorithm to maintain a constant approximation guarantee. Since solution $\boldsymbol{C}$ has cost $L(\boldsymbol{X}, \boldsymbol{C}, w) = \mathsf{W}_2^2(P, P^{\text{in}})$, and $\mathsf{OPT}(\boldsymbol{X}, w) = \mathsf{W}_2^2(M^*, P^{\text{in}})$, the following theorem is immediate from Prop. 3.3(a).

Theorem 3.4. *Suppose $\mathsf{wkmeans}_1$ and $\mathsf{wkmeans}_2$ have approximation ratios α and γ. Then by choosing $\kappa = k$, the solution $\boldsymbol{C}$ given by Rk-means has the following guarantee: $L(\boldsymbol{X}, \boldsymbol{C}, w) \leq (\sqrt{\gamma} + \sqrt{\alpha} + \sqrt{\gamma\alpha})^2\mathsf{OPT}(\boldsymbol{X}, w)$.*

Specifically, if both sub-problems are solved optimally ($\alpha = \gamma = 1$), Rk-means is a 9-approximation.

[2]The best known approximation ratio is 6.357 for data in Euclidean space (Ahmadian et al., 2017).

Proof of Prop. 3.3. (a) By the definition of Q, the optimal transport plan from P^{in} to Q is such that each support point $s \in S$ is received by all $x \in \boldsymbol{X}$ nearest to s compared to other points in S. So,

$$\begin{aligned}
&\mathsf{W}_2^2(P^{\text{in}}, Q)\\
&= \sum_{\boldsymbol{x}\in\boldsymbol{X}} w(\boldsymbol{x}) d(\boldsymbol{x}, \boldsymbol{G})^2 = \sum_{\boldsymbol{x}\in\boldsymbol{X}} w(\boldsymbol{x}) \sum_{j=1}^{m} d(\pi_{S_j}\boldsymbol{x}, \boldsymbol{C}_j)^2\\
&= \sum_{j=1}^{m} \sum_{\boldsymbol{z}\in\boldsymbol{X}_j} w_j(\boldsymbol{z}) d(\boldsymbol{z}, \boldsymbol{C}_j)^2 = \sum_{j=1}^{m} \mathsf{W}_2^2(M_j, P_j^{\text{in}})\\
&\le \alpha \sum_{j=1}^{m} \mathsf{W}_2^2(M_j^*, P_j^{\text{in}}) \le \alpha \cdot \mathsf{W}_2^2(M^*, P^{\text{in}}).
\end{aligned}$$

The second to last inequality is due to the α-approximation of $\mathsf{wkmeans}_1$, and condition that $|\text{supp}(M_j)| \ge |\text{supp}(M_j^*)|$. The last inequality follows from Lemma 3.2. Apply triangle inequality of W_2 (cf. Villani (2009), pg. 106).

$$\begin{aligned}
&\mathsf{W}_2(P^{\text{in}}, P) && (4)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \mathsf{W}_2(Q, P) && (5)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \sqrt{\gamma} \cdot \mathsf{W}_2(Q, M^*) && (6)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \sqrt{\gamma}(\mathsf{W}_2(Q, P^{\text{in}}) + \mathsf{W}_2(P^{\text{in}}, M^*)) && (7)\\
&= (1+\sqrt{\gamma})\mathsf{W}_2(P^{\text{in}}, Q) + \sqrt{\gamma} \cdot \mathsf{W}_2(P^{\text{in}}, M^*) && (8)\\
&\le (1+\sqrt{\gamma})\sqrt{\alpha}\mathsf{W}_2(P^{\text{in}}, M^*) + \sqrt{\gamma} \cdot \mathsf{W}_2(P^{\text{in}}, M^*) && (9)\\
&= \left(\sqrt{\alpha} + \sqrt{\gamma} + \sqrt{\alpha\gamma}\right) \cdot \mathsf{W}_2(M^*, P^{\text{in}}). && (10)
\end{aligned}$$

The second inequality is due to the fact that $\mathsf{wkmeans}_2$ has approximation ratio γ; the first and third are again triangle inequalities. We conclude the proof.

(b) We need only construct an example of P^{in} for the case $d = m$. Although P^{in} as an input to the algorithm is a discrete measure, for our purposes it suffices to take P^{in} to be the uniform distribution on $[0, 1]^m$ (which can be approximated arbitrarily well by a discrete measure). It is simple to verify that if $k_0 = k^{1/m}$ is a natural number, then M^* is a uniform distribution on the regular grid of size k_0 in each dimension. It follows that $\mathsf{W}_2^2(P^{\text{in}}, M^*) \le \frac{m}{12k_0^3} = \frac{m}{12k^{3/m}}$. The grid points $\boldsymbol{G}$ range over the set $S := [1/(2\kappa), 1 - 1/(2\kappa)]^m$. Moreover, Q is a uniform distribution on $\boldsymbol{G}$. Now P is the outcome of line (11) so the support of P must lie in the convex hull S of $\boldsymbol{G}$. The cost of each unit mass transfer from an atom in the complement of set $[1/(4\kappa), 1 - 1/(4\kappa)]^m$ to one in S is at least $(1/4\kappa)^2$, so $\mathsf{W}_2^2(P^{\text{in}}, P) \ge (1/4\kappa)^2 \cdot [1 - (1 - 1/(2\kappa))^m]$. Lastly, note that $(1 - 1/(2\kappa))^m < e^{-m/2\kappa}$. □

Regularized Rk-means It is possible to extend our approach to accommodate regularization techniques. This can be useful when the data are very high dimensional (Sun et al., 2012; Witten and Tibshirani, 2010). Thus, the clustering formulation can be expressed as a regularized optimal transport problem: $M^* = \arg\min\ \mathsf{W}_2^2(M, P^{\text{in}}) + \Omega(M)$ where the optimization is over the space of discrete measures M that have k support points (the set $\boldsymbol{C}$ of k centroids), and the regularizer $\Omega(M) \ge 0$ typically decomposes over the m-partition of variables: $\Omega(M) = \sum_{j=1}^m \Omega_j(M_j)$. For instance, $\Omega_j(M_j)$ may be taken to be a multiple of the ℓ_1 norm of M_j's supporting atoms (e.g., group lasso penalty). The algorithm has the same three steps as before, with some modification in (1') and (3'):

(1') For each $j \in [m]$, pick M_j to be the (α-approximate) minimizer of $\mathsf{W}_2^2(M_j, P_j^{\text{in}}) + \Omega_j(M_j)$, where $\boldsymbol{C}_j = \text{supp}(M_j)$ is the support of M_j and $|C_j| = \kappa$ (line 7).

(3') Finally, return P which is the measure with exactly k support points in $\mathbb{R}^d$ that (γ-approximately) minimizes $\mathsf{W}_2^2(P, Q) + \Omega(P)$ (line 11).

Proposition 3.5. *If $\kappa \ge |supp(M_j^*)|\ \forall\ j \in [m]$, then* $\frac{\mathsf{W}_2^2(P^{in}, P) + \Omega(P)}{\mathsf{W}_2^2(P^{in}, M^*) + \Omega(M^*)} \le 2\alpha + 4\gamma + 4\alpha\gamma$.

If both subproblems for regularized k-means can be solved optimally, our method yields a 10-approximation on the penalized W_2^2 objective. We conclude by noting that our technique extends easily to the W_p^p objective for any $p \ge 1$, but the approximation ratio will be changed according to p.

4 Leveraging relational data

We now explain the "relational" part of the Rk-means algorithm, where we exploit relational structures in the data and the FEQ to achieve significant computational savings. Three classes of relational structures prevalent in RDBMSs are (a) *categorical variables*, (b) *functional dependencies* (FDs), and (c) the topology of the FEQ. We exploit these structures to carefully select the partition $S_1 \cup \cdots \cup S_m$ to use for Rk-means, to compute the marginal sub-problems $(\boldsymbol{X}_j, w_j)$, the components $\boldsymbol{C}_j$ of the coreset $\boldsymbol{G}$, and the grid weight w_{grid} *without* materializing the entire coreset $\boldsymbol{G}$. When selecting partitions, there are two competing criteria: first, we need a partition so that the approximation ratio α for $\mathsf{wkmeans}_1$ is as small as possible. For example, if $|S_j| = 1\ \forall\ j$, so $m = d$, then we can apply the well-known optimal solution for k-means in 1 dimension using dynamic programming in $O(n^2k)$ time (Wang and Song, 2011); this then provides $\alpha = 1$. On the other hand, we want the remainder of algorithm to be fast by keeping the size of the grid $\boldsymbol{G}$, namely $|\boldsymbol{G}| \le \kappa^m$, small.

Categorical variables. Real-world relational database queries typically involve many categorical variables (e.g., color, month, or city). In practice, practitioners may endow non-uniform weights for different categorical variables, or categories (Huang, 1998). In terms of representation, a common way to deal with categorical variables is to one-hot encode them, whereby a categorical feature such as city is represented by an indicator vector $\boldsymbol{x}_{\mathsf{city}} = [\mathbf{1}_{\mathsf{city}=c_1}, \mathbf{1}_{\mathsf{city}=c_2}, \cdots, \mathbf{1}_{city=c_L}]$ where $\{c_1, \ldots, c_L\}$ is the set of cities occurring in the data. The subspace associated with these indicator vectors is known as the *categorical subspace* of a categorical variable. One-hot encoding substantially increases the data matrix size via an increase in the dimensionality of the data. Fortunately, this is not a problem—by treating each categorical variable as a subset of the partition, we can solve the *weighted* k-means subproblem within a categorical subspace efficiently and optimally.

Theorem 4.1. *Given a categorical weighted k-means instance, an optimal solution is to put each of the first $k-1$ highest weight indicator vectors in its own cluster, and the remaining vectors in the same cluster.*

This means that for a categorical variable with L categories, we can compute the optimal clustering for the sub-problem in only $O(nL \log L)$ time. See Appendix C.1 for more details.

Functional dependencies. Next, we address the second call to $\mathsf{wkmeans}_2$: its runtime is dependent on the size of the grid $\boldsymbol{G}$, which can be up to $O(k^m)$, where m is the number of features from the input. Databases often contain *functional dependencies* (FDs), which we can exploit to reduce the size of $\boldsymbol{G}$. An FD is a dimension whose value depends entirely on the value of another dimension. For example, suppose a dataset has such as storeID, zip, city, state, and country. Here, storeID functionally determines zip, which determines city, which in turns determines state, leading to country. This common structure is known as an *FD-chain*, and appears often in real-world FEQs. If we were to apply Rk-means naively, these five features would contribute a factor of k^5 to the grid size. However, by using the FD structure of the database, we show that only a factor of $5k$ is contributed to the grid size, because most of the k^5 grid points $\boldsymbol{g}$ have $w_{\mathsf{grid}}(\boldsymbol{g}) = 0$ (see (3)). More generally, when there is an *FD-chain* of p features, their overall contribution to the grid size is a factor of $O(kp)$, not $O(k^p)$, and the grid points with non-zero weights can be computed efficiently in time $O(kp)$.

Theorem 4.2. *Suppose all d input features can be partitioned into m FD-chains of size $d_1, \cdots, d_m$, respectively. Then, the number of grid points $\boldsymbol{g} \in \boldsymbol{G}$ with non-zero w_{grid} weight is bounded by $\prod_{i=1}^{m}(1+d_i(k-1))$. Furthermore, the set of non-zero weight grid points can be computed in time $\tilde{O}(\prod_{i=1}^{m}(1+d_i(k-1)))$.*

Note that in the above theorem, if there was *no* FD, then d features each form their own chain of size 1, in which case $\prod_{i=1}^{m}(1+d_i(k-1)) = k^m$; thus, the theorem strictly generalizes the no-FD case.

Query structure. Finally, we explain how the FEQ's structure can be exploited to speed up the computation of subproblems, the grid, and grid weights. In particular, we make use of recent advances in relational query evaluation algorithms (Abo Khamis et al., 2017, 2016; Ngo, 2018; Olteanu and Schleich, 2016). The InsideOut algorithm from the FAQs framework in particular (Abo Khamis et al., 2017) allows us to compute the grid weights without explicitly the grid points.

For concreteness, we describe the steps of Rk-means as implemented in the database, noting the additional speedups we can get over the description in Alg. 1.

Step 1 (lines 5 and 6). *Project $\boldsymbol{X}$ into each subspace S_j and compute the weight w of each point.*

In a relational database, the projected sets $\boldsymbol{X}_j$ already exist in normalized form (Abiteboul et al., 1995). In fact, in sixth normal form (6NF) databases Date et al. (2002), each relation in the database will generally correspond to one variable. So, the sets $\boldsymbol{X}_j$ and their marginal weights can be computed efficiently. This step perfectly aligns with our strategy of picking the partition $S_1 \cup \cdots \cup S_m$ to match the database schema!

Step 2 (line 7). *Find κ centroids in each subspace S_j.*

If the subspace S_j corresponds to a single continuous variable, we can solve the one-dimensional k-means problem quickly and optimally (Wang and Song, 2011); and if S_j corresponds to a categorical feature, then it is solved trivially (and optimally) using Theorem 4.1.

Step 3 (lines 9 and 10). *Construct the coreset $\boldsymbol{G}$ and the associated weights w_{grid}.*

When constructing $\boldsymbol{G}$, it is unnecessary to represent any points in $\boldsymbol{G}$ that have zero weight. We can use the InsideOut database algorithm (Abo Khamis et al., 2016) to efficiently compute nonzero weights, and then extract only those grid points in $\boldsymbol{G}$ with nonzero weight.

Step 4 (line 11). *Cluster the weighted coreset $\boldsymbol{G}$.*

We use a modified version of Lloyd's weighted k-means that exploits the structure of $\boldsymbol{G}$ and sparse representation of categorical values to speed up computation. See Appendix C.3 for details.

Runtime analysis. We compare Rk-means to the standard setting of first extracting the matrix $\boldsymbol{X}$ from the database and then perform clustering on $\boldsymbol{X}$ directly. The precise runtime statement requires defining a few parameters such as "fractional hypertree width" and "fractional edge cover number" of the FEQ. Details are relegated to Appendix A, and we simply state the main thrust of the runtime result:

Theorem 4.3. *There are classes of feature extraction queries (FEQs) for which the runtime of Rk-means is*

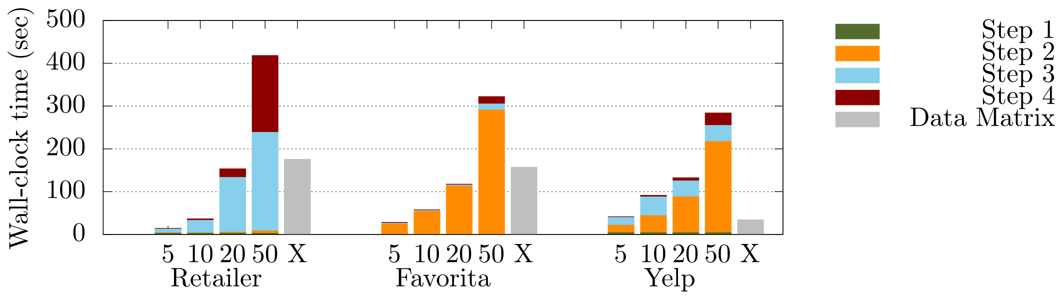


Figure 2: Breakdown of the compute time of Rk-means for each step of the algorithm with $\kappa = k \in \{5, 10, 20, 50\}$. The time to compute $\boldsymbol{X}$ is provided as reference.

asymptotically less than $|\boldsymbol{X}|$, and the ratio between $|\boldsymbol{X}|$ and the runtime of Rk-means can be a polynomial in N, the size of the largest input relation.

The key insight to read from this theorem is that Rk-means can, in principle, run faster than simply exporting the data matrix, without even running *any* clustering algorithm (be it sampling-based, streaming, etc.). Of course, the result only concerns a class of FEQs "on paper". Section 5 examines real FEQs, which also demonstrate Rk-means's runtime superiority.

5 Experimental results

We empirically evaluate the performance of Rk-means on three real datasets for three sets of experiments: (1) we break down and analyze the performance of each step in Rk-means; (2) we benchmark the performance and approximation of Rk-means against mlpack (Curtin et al., 2018) (v. 3.1.0), a fast C++ machine learning library; and (3) we evaluate the performance and approximation of Rk-means for setting $\kappa < k$; i.e., different number of clusters for Steps 2 and 4.

The experiments show that the coresets of Rk-means are often significantly smaller than the data matrix. As a result, Rk-means can scale easily to large datasets, and can compute the clusters with a much lower memory footprint than mlpack. When $\kappa = k$, Rk-means is orders-of-magnitude faster than the end-to-end computation for mlpack—up to 115×. Typically, the approximation level is very minor. Also, setting $\kappa < k$ can lead to further performance speedups (sometimes exceeding 200×!) with only moderate increase in approximation.

Setup. We prototyped Rk-means on top of the LMFAO engine **?**. Rk-means is implemented in multi-threaded C++11; this makes mlpack a comparable implementation. All experiments were performed on an AWS `x1e.8xlarge` instance, which has 1 TiB of RAM and 32 vCPUs. All Relations are given sorted by their join attributes.

To construct the data matrix that forms the input to mlpack, we use PostgreSQL 10.6 (`psql`) to evaluate the FEQ. The seminal k-means++ algorithm (Arthur and Vassilvitskii, 2007) is used for initializing the k-means cluster. We run Rk-means and mlpack + `psql` five times and report the average approximation and runtime. The timeout for all experiments was set to six hours (21,600 seconds) per trial. Our runtime results omit data loading/saving times. For mlpack + `psql`, `psql` must export $\boldsymbol{X}$ to disk, and then mlpack must then read it from disk! Rk-means has no need to do this, and thus the runtime numbers are skewed in mlpack's favor. This skew may be significant: loading and saving a large CSV file may take hours.

	Retailer	Favorita	Yelp
Relations	5	6	6
Attributes	39	15	25
One-hot Enc.	95	1470	1617
# Rows in $\boldsymbol{D}$	84M	125M	8.7M
Size of $\boldsymbol{D}$	1.5GB	2.5GB	0.2GB
# Rows in $\boldsymbol{X}$	84M	127M	22M
Size of $\boldsymbol{X}$	18GB	7GB	2.4GB
# Rows in Coreset $\boldsymbol{G}$			
$\kappa = 5$	1.43M	14.94K	2.69M
$\kappa = 10$	9.58M	85.88K	11.71M
$\kappa = 20$	38.16M	632.5K	11.89M
$\kappa = 50$	73.75M	7.87M	12.46M

Table 1: Statistics for the input database $\boldsymbol{D}$, data matrix $\boldsymbol{X}$, and coresets $\boldsymbol{G}$ for the three dataset.

Datasets. We use three real datasets: (1) *Retailer* is used by a large US retailer for sales forecasting; (2) *Favorita* (Favorita Corp., 2017) is a public dataset for retail forecasting; and (3) *Yelp* is from the public Yelp Dataset Challenge (Yelp, 2017) and used to predict users' ratings of businesses. Table 1 presents key statistics for the three datasets, including the size of data matrix $\boldsymbol{X}$ and the coreset $\boldsymbol{G}$ for each dataset and different κ-values. $|\boldsymbol{G}|$ is highly data dependent. For *Favorita*, $\boldsymbol{G}$ is orders-of-magnitude smaller than the data matrix. For *Retailer*, when $\kappa = 20$ and $\kappa = 50$, $|\boldsymbol{G}|$ approaches $|\boldsymbol{X}|$, but Rk-means still provides a speedup. Additional dataset details are given in Appendix D.

Breakdown of Rk-means. Figure 2 shows the time it takes Rk-means to cluster the three datasets for different values of k with $\kappa = k$. The total time is broken down into the four steps of the algorithm from Section 4. We provide the time it takes `psql` to compute $\boldsymbol{X}$ as ref-

Retailer	k = 5	k = 10	k = 20	k = 50	k=20, κ = 10	k = 50, κ = 20
Compute $\boldsymbol{X}$ (psql)	175.47	175.47	175.47	175.47	175.47	175.47
Clustering (mlpack)	65.41	158.81	385.67	1,453.88	385.67	1,453.88
Rk-means	15.66	54.59	230.17	650.20	63.51	344.31
Relative Speedup	15.38×	6.12×	2.44×	2.51×	8.84×	4.73×
Relative Approx.	0.20	0.08	0.03	0.00	0.03	0.02

Favorita	k = 5	k = 10	k = 20	k = 50	k=20, κ = 10	k = 50, κ = 20
Compute $\boldsymbol{X}$ (psql)	156.86	156.86	156.86	156.86	156.86	156.86
Clustering (mlpack)	1,002.54	6,449.32	11,794.49	>21,600.00	11,794.49	>21,600
Rk-means	27.95	57.72	118.36	334.65	57.65	120.77
Relative Speedup	41.49×	114.59×	100.98×	>64.55×	207.30×	>178.86×
Relative Approx.	2.99	0.35	0.12	–	1.93	–

Yelp	k = 5	k = 10	k = 20	k = 50	k=20, κ = 10	k = 50, κ = 20
Compute $\boldsymbol{X}$ (psql)	33.83	33.83	33.83	33.83	33.83	33.83
Clustering (mlpack)	210.59	640.43	2,107.83	11,474.24	2,107.83	11,474.24
Rk-means	43.37	107.71	195.22	405.11	114.34	241.34
Relative Speedup	5.64×	6.26×	10.97×	28.41×	18.73×	47.68×
Relative Approx.	0.37	0.26	0.13	0.05	0.27	0.20

Table 2: End-to-end runtime and approximation comparison of Rk-means and mlpack on each dataset. The first four columns use different $\kappa = k$ values; the last two show results for setting $\kappa < k$.

erence (gray bar). In many cases, Rk-means can cluster *Retailer* and *Favorita* faster than it takes `psql` to even compute the data matrix! The relative performance of the four steps is data dependent. For *Retailer*, most of the time is spent on constructing $\boldsymbol{G}$ in Step 3, which is relatively large. For *Favorita*, however, Step 2 takes the longest, as there is one continuous variable with many distinct values, and the DP algorithm for clustering runs in time quadratic in the number of distinct values. Here, performance could be improved by clustering this dimension with a different k-means algorithm; this would increase the approximation somewhat.

Comparison with mlpack. The left columns of Table 2 compares the runtime and approximation of Rk-means against mlpack on the three datasets for different k values with $\kappa = k$. The approximation is given relative to the objective value obtained by mlpack. Speedup is given by comparing the end-to-end performance of Rk-means and mlpack (ignoring disk I/O time), which for mlpack includes the time needed by `psql` to materialize $\boldsymbol{X}$. Overall, Rk-means often outperforms even just the clustering step from mlpack, and when end-to-end computation is considered, Rk-means gives up to 115× speedup. mlpack timed out after six hours for *Favorita* with $k = 50$. In addition, Rk-means has a much smaller memory footprint than mlpack: for instance, on *Favorita* with $k = 20$, mlpack uses over 900GiB of RAM to cluster the dataset, whereas Rk-means only requires 18Gib. To try and reduce RAM usage, we benchmarked against mlpack with sparse matrices from the Armadillo library (Sanderson and Curtin, 2018); this did reduce RAM usage, but the overhead of working with sparse data structures meant an overall slowdown. Overall, in our simulations, the approximation level is moderate, and consistently well below the 9-approximation bound from Theorem 3.4.

Our simulations show a high level of agreement in the clusterings obtained by the algorithms: for *Retailer* with $k = 20$, the average normalized mutual information is 0.743 between five mlpack k-means clusterings, and 0.711 between Rk-means and mlpack clusterings.

Setting $\kappa < k$ for Step 2. We next evaluate the effect of setting κ to a smaller value than the number of clusters k. This exploits the speed/approximation tradeoff: smaller κ helps reduce the size of $\boldsymbol{G}$, at the cost of more approximation. Table 2 presents for each dataset the results for setting $k = 20, \kappa = 10$ and $k = 50, \kappa = 20$, and compares them to the relative performance and approximation over computing k clusters in mlpack.

By setting $\kappa < k$, Rk-means can compute the k clusters up to 208× faster than mlpack and 3.6× faster than when $\kappa = k$, while the approximation remains moderate. Our results are data dependent—but as the database scales, our speedups will be even more significant.

6 Conclusion

We introduce Rk-means, a method to construct k-means clustering coresets on relational data directly from the database. Rk-means gives a provably good clustering of the entire dataset, without ever materializing the data set; this also yields asymptotic improvements in running time. Experimentally, we observe that the coreset has size up to 180x smaller than the size of the data matrix and this results in orders-of-magnitude improvements in runtime, while still providing empirically good clusterings. Although our work here primarily focuses on k-means clustering, we believe our construction of grid coresets and the accompanying theory is useful for other unsupervised learning tasks and plan to explore such possibilities in future work.

7 Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588. B. Moseley was supported in part by NSF grants CCF-1725543, 1733873, 1845146, a Google Research Award, a Bosch junior faculty chair and an Infor faculty award. X. Nguyen was supported in part by grant NSF CAREER DMS-1351362 and a grant from Toyota Research Institute.

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0. URL `http://webdam.inria.fr/Alice/`.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28, 2016.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Juggling functions inside a database. *SIGMOD Rec.*, 46(1):6–13, 2017.

Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: In-database learning thunderstruck. In *2nd Workshop on Data Mgt for End-To-End ML*, DEEM'18, pages 8:1–8:10, 2018a.

Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *PODS*, pages 325–340, 2018b.

Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *FOCS*, pages 61–72, 2017.

D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *SODA*, page 1027–1035, 2007.

Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k -means clustering via lightweight coresets. In *SIGKDD*, pages 1119–1127, 2018.

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

Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1995–2003, 2013.

Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 576–585, 2017.

Field Cady. *The Data Science Handbook*. John Wiley & Sons, 2017.

Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong Zhang. mlpack 3: a fast, flexible machine learning library. *J. Open Source Software*, 3:726, 2018.

C.J. Date, H. Darwen, and N. Lorentzos. *Temporal Data & The Relational Model*. Elsevier, 2002.

E. del Barrio, J. A. Cuesta-Albertos, C. Matran, and A. Mayo-Iscar. Robust clustering tools based on optimal transportation. *Statistics and Computing*, pages 1–22, 2017.

Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689, 2011.

B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley & Sons, 2011.

Favorita Corp. Corporacion Favorita Grocery Sales Forecasting: Can you accurately predict sales for a large grocery chain? `https://www.kaggle.com/c/favorita-grocery-sales-forecasting/`, 2017.

S. Graf and H. Luschgy. *Foundations of quantization for probability distributions*. Springer-Verlag, New York, 2000.

Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11 (1):4:1–4:20, 2014.

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

S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *SODA*, 2004.

Sariel Har-Peled and Soham Mazumdar. Coresets for k-means and k-median clustering and their applications. *CoRR*, abs/1810.12826, 2018. URL `http://arxiv.org/abs/1810.12826`.

J. A. Hartigan. *Clustering algorithms*. Wiley, New York, 1975.

N. Ho, X. Nguyen, M. Yurochkin, H. H. Bui, V. Huynh, and D. Phung. Multilevel clustering via Wasserstein means. In *ICML*, pages 1501–1509, 2017.

Z. Huang. Extensions to the k-means algorithm for clustering large data sets of categorical values. *Data mining and Knowledge discovery*, 2:283–304, 1998.

L. Kaufman and P. J. Roussew. *Finding Groups in Data - An Introduction to Cluster Analysis.* Wiley & Sons, 1990.

Mahmoud Abo Khamis, Ryan Curtin, Benjamin Moseley, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. On functional aggregate queries with additive inequalities. In *PODS*, 2019.

Daphne Koller and Nir Friedman. *Probabilistic graphical models.* Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2009. Principles and techniques.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

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.

Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*, pages 111–124, 2018.

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

Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

Carlos Ordonez. Integrating k-means clustering with a relational DBMS using SQL. *IEEE Trans. Knowl. Data Eng.*, 18(2):188–201, 2006.

Carlos Ordonez and Edward Omiecinski. Efficient disk-based k-means clustering for relational databases. *IEEE Trans. Knowl. Data Eng.*, 16(8):909–921, 2004.

D. Pollard. Quantization and the method of k-means. *IEEE Trans. Inf. Theory*, 28(2):199–204, 1982.

C. Sanderson and R.R. Curtin. A user-friendly hybrid sparse matrix class in C++. In *Proceedings of the 2018 International Congrees on Mathematical Software (ICMS)*, pages 422–430. Springer, 2018. ISBN 978-3-319-96418-8.

Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, pages 3–18, 2016.

Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813, 2018.

W. Sun, J. Wang, and Y. Fang. Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 9: 148–167, 2012.

Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, pages 249–260, 2001.

Todd L. Veldhuizen. Leapfrog Triejoin: A simple, worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481*, 2012.

Cédric Villani. *Optimal transport*, volume 338 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 2009.

Haizhou Wang and Mingzhou Song. Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29, 2011.

D. Witten and R. Tibshirani. A framework for feature selection in clustering. *Journal of the American Statistical Association*, 105:713–726, 2010.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008.

J. Ye, P. Wu, J. Wang, and J. Li. Fast discrete distribution clustering using barycenter with sparse support. *IEEE Trans. Signal Proc.*, 65(9):2317–2332, 2017.

Yelp. Yelp dataset challenge, `https://www.yelp.com/dataset/challenge/`, 2017.

Rk-means: Fast Clustering for Relational Data: Supplementary Material

Ryan R. Curtin
RelationalAI
ryan@ratml.org

Benjamin Moseley
Tepper School of Business
Carnegie Mellon University
moseleyb@andrew.cmu.edu

Hung Q. Ngo
RelationalAI
hung.q.ngo@relational.ai

XuanLong Nguyen
Department of Statistics
University of Michigan
xuanlong@umich.edu

Dan Olteanu
University of Oxford
dan.olteanu@cs.ox.ac.uk

Maximilian Schleich
University of Oxford
max.schleich@cs.ox.ac.uk

Abstract

This supplementary material contains details from Rk-means omitted from the main paper due to space constraints.

A Background on Database Queries and FAQs

Recent advancements in the database community have produced new classes of query plans and join algorithms Abo Khamis et al. (2017, 2016); Ngo (2018); Olteanu and Schleich (2016) for the efficient evaluation of general database queries. These general algorithms hinge on the expression of a database query as a *functional aggregate query*, or FAQ Abo Khamis et al. (2016).

Loosely speaking, an FAQ is a collection of *aggregations* (be they sum, max, min, etc.) over a number of functions known as *factors*[3], in the same sense as that used in graphical models. In particular, if there was only one aggregation (such as sum), then an FAQ is just a sum-product form typically used to compute the partition function. An FAQ is more general as it can involve many marginalization operators, one for each variable, and they can interleave in arbitrary way. Every relational database query can be expressed in this way. Consider the example query of Section 1: for this, the task of the database query evaluator is to compute `max(transactions.count)` for every tuple (i, s, t, p, y) that exists in the output. We can express this as a function:

$$\phi(i,s,t,p,y) = \max_c \max_i \max_s \psi_P(i,t,p)\psi_T(i,s,c)\psi_S(s,y). \quad (11)$$

In this we have three *factors* $\psi_P(\cdot)$, $\psi_T(\cdot)$, and $\psi_S(\cdot)$, which correspond to the `product`, `transactions`, and `store` tables, respectively. We define $\psi_P(i,t,p) = 1$ if the tuple (i,t,p) exists in the `product` table and 0 otherwise; we define $\psi_S(s,y)$ similarly. We define $\psi_T(i,s,c) = c$ if the tuple (i,s,c) exists in the `transactions` table and 0 otherwise. Thus, given any tuple (i,s,t,p,y), we can compute `max(transactions.count)`$= \phi(i,s,t,p,y)$.

In order to efficiently solve an FAQ (of which Equation (11) is but one example), the InsideOut algorithm of Abo Khamis et al. (2016) may be used; InsideOut is a variable elimination algorithm, inspired from variable elimination in graphical model, with several new twists. One twist is to adapt worst-case optimal join algorithms Ngo et al. (2018); Veldhuizen (2012) to speed up computations by exploiting sparsity in the data. Another twist is that the algorithm has to carefully pick a variable order to minimize the runtime, while at the same time respect the correctness and semantic of the query. Unlike in the case of computing a sum-product where the summation opeartors are commutative, in a FAQ the operators may not be commutative.

To characterize the runtime of this algorithm, we must first observe that each database query and thus FAQ corresponds to a hypergraph $\mathcal{H} = \{\mathcal{V}, \mathcal{E}\}$. The vertices $\mathcal{V}$ of this hypergraph correspond to the vari-

[3]A full formal definition of FAQs can be found in Abo Khamis et al. (2016), but is not required for our work here so we omit it.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

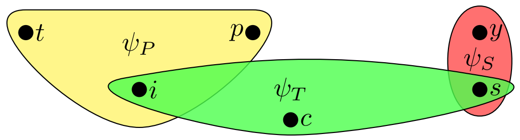


Figure 3: Example hypergraph $\mathcal{H}$ for the example query and FAQ in Equation 11.

ables of the FAQ expression; in our example, we have $\mathcal{V} = \{i, s, t, p, y, c\}$. The hyperedges $\mathcal{E}$, then, correspond to each factor $\psi_P(\cdot)$, $\psi_T(\cdot)$, and $\psi_S(\cdot)$—which in turn correspond to the tables in the database. This hypergraph $\mathcal{H}$ is shown in Figure 3.

Roughly, InsideOut proceeds by first selecting a variable ordering σ, reordering the FAQ accordingly, and then solving the inner subproblems repeatedly, in much the same way variable elimination works for inference in graphcal models Koller and Friedman (2009). The runtime of InsideOut is dependent on a notion of width of $\mathcal{H}$ called FAQ-width, or faqw$(\cdot)$. Fully describing this width is beyond the scope of this paper and we encourage readers to refer to Abo Khamis et al. (2016) for full details. The FAQ-width is a generalized version of *fractional hypertree width* of Grohe and Marx (2014) (denoted by fhtw). When the FAQ query does not have free variables, faqw = fhtw. Given some FAQ with hypergraph $\mathcal{H}$, via Section 4.3.4 of Abo Khamis et al. (2017), InsideOut runs in time $\tilde{O}(N^{\text{faqw}_{\mathcal{H}}(\sigma)} + Z)$, where we assume that the support of each factor[4] is no more than $O(N)$, and Z is the number of tuples in the output. As an example, the hypergraph of Figure 3 has $\text{faqw}_{\mathcal{H}}(\sigma) = 1$. Overall, InsideOut gives us the most efficient known way to evaluate problems that can be formulated as FAQs.

B Missing details from Section 3

B.1 Proposition 3.5

Proof of Prop. 3.5. As before the optimal transport plan from P^{in} to Q is such that each support point $s \in S$ is received by all $x \in \boldsymbol{X}$ nearest to s compared to other points in S. So,

$$
\begin{aligned}
&\mathsf{W}_2^2(P^{\text{in}}, Q) + \Omega(M) \\
&= \sum_{j=1}^{m} \mathsf{W}_2^2(M_j, P_j^{\text{in}}) + \Omega(M) \qquad (12)\\
&\le \alpha \sum_{j=1}^{m} (\mathsf{W}_2^2(M_j^*, P_j^{\text{in}}) + \Omega_j(M_j^*)) \qquad (13)\\
&\le \alpha(\mathsf{W}_2^2(M^*, P^{\text{in}}) + \Omega(M^*)). \qquad (14)
\end{aligned}
$$

[4]Or in our case, the number of tuples in the table corresponding to that factor.

The second to last inequality is due to the α-approximation of (regularized) $\mathsf{wkmeans}_1$, and condition that $|\text{supp}(M_j)| \ge |\text{supp}(M_j^*)|$. The last inequality follows from Proposition 3.2 and the definition of Ω. By the triangle inequality of W_2, as before

$$
\begin{aligned}
&\mathsf{W}_2(P^{\text{in}}, P) \qquad (15)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \mathsf{W}_2(Q, P) \qquad (16)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \sqrt{\gamma \mathsf{W}_2^2(Q, M^*) + \gamma\Omega(M^*) - \Omega(P)} \qquad (17)\\
&\le \mathsf{W}_2(P^{\text{in}}, Q) + \big(2\gamma \mathsf{W}_2^2(P^{\text{in}}, Q) + 2\gamma \mathsf{W}_2^2(P^{\text{in}}, M^*) \\
&\qquad + \gamma\Omega(M^*) - \Omega(P)\big)^{\frac{1}{2}} . \qquad (18)
\end{aligned}
$$

Hence, by Cauchy-Schwarz and combining with (14) we obtain

$$
\begin{aligned}
&\mathsf{W}_2^2(P^{\text{in}}, P) \\
&\le 2\Big\{(1 + 2\gamma)\mathsf{W}_2^2(P^{\text{in}}, Q) \qquad (19)\\
&\qquad + 2\gamma \mathsf{W}_2^2(P^{\text{in}}, M^*) + \gamma\Omega(M^*) - \Omega(P)\Big\} \qquad (20)\\
&\le (2\alpha + 4\gamma + 4\alpha\gamma)\mathsf{W}_2^2(P^{\text{in}}, M^*) \\
&\qquad + (2\alpha + 2\gamma + 4\alpha\gamma)\Omega(M^*) \\
&\qquad - (2 + 4\gamma)\Omega(M) - 2\Omega(P). \qquad (21)
\end{aligned}
$$

The conclusion is immediate by noting that Ω is a non-negative function. □

C Missing details from Section 4

C.1 Categorical variables

As we have mentioned, real-world relational database queries often involve a significant number of categorical variables, such as color, month, or city. The most common way to deal with categorical variables in practical settings is to one-hot encode them, whereby a categorical feature such as city is represented by an indicator vector

$$
\boldsymbol{x}_{\mathsf{city}} = \left[\mathbf{1}_{\mathsf{city}=c_1} \quad \mathbf{1}_{\mathsf{city}=c_2} \quad \cdots \mathbf{1}_{\mathsf{city}=c_L}\right] \qquad (22)
$$

where $\{c_1, \ldots, c_L\}$ is the set of cities occuring in the data. The subspace associated with these indicator vectors is known as the *categorical subspace* of a categorical variable. This one-hot representation substantially increases the data matrix size via an increase in the *dimensionality* of the data. For example, a dataset of about 30 mostly categorical features with hundreds or thousands of categories for each feature will have its dimensionality exploded to the order of thousands with one-hot encoding.

The k-means subproblem within a categorical subspace is solvable efficiently *and* optimally, without one-hot encoding. This optimal solution can be computed in the same time it takes to find the number of points in each category, which is a vast improvement on either an optimal dynamic program or Lloyd's algorithm. Furthermore, it helps keep m as low as the number of database attributes in the query.

Consider a weighted k-means subproblem solved by $\mathsf{wkmeans}_1$ defined on a categorical subspace induced by a categorical feature K that has L categories. Then, the instance is of the form $(\boldsymbol{I}, v)$, where $\boldsymbol{I}$ is the collection of L indicator vectors $\mathbf{1}_e$, one for each element $e \in \mathsf{Dom}(K)$. (One can think of $\boldsymbol{I}$ as the identity matrix of order L.) Define the weight function v as

$$v(\mathbf{1}_e) = \sum_{\boldsymbol{x}\in\boldsymbol{X}, \boldsymbol{x}_K = e} w(\boldsymbol{x}). \tag{23}$$

For any set $F \subseteq \mathsf{Dom}(K)$, let $\boldsymbol{v}_F$ denote the vector $(v(\mathbf{1}_e))_{e\in F}$. Also, $\|\boldsymbol{v}_F\|_1$ and $\|\boldsymbol{v}_F\|_2$ denote the ℓ_1 and ℓ_2 norm, respectively. It is useful to rewrite the categorical weighted k-means problem:

Proposition C.1. *The categorical weighted k-means instance $(\boldsymbol{I}, v)$ admits the following optimization objective:*

$$\mathsf{OPT}(\boldsymbol{I}, v) = \|\boldsymbol{v}\|_1 - \max_{\mathcal{F}} \sum_{F\in\mathcal{F}} \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1}, \tag{24}$$

where $\mathcal{F}$ ranges over all partitions of $\mathsf{Dom}(K)$ *into k parts.*

Proof. First, consider a subset $F \subseteq \mathsf{Dom}(K)$ of the categories; the centroid $\boldsymbol{\mu}$ of (weighted) indicator vectors $\mathbf{1}_e$, $e \in F$, can be written down explicitly:

$$\mu_e = \begin{cases} 0 & e \notin F \\ \frac{v_e}{\|\boldsymbol{v}_F\|_1} & v \in F, \end{cases} \tag{25}$$

The weighted sum of squared distances between $\mathbf{1}_e$ for all $e \in F$ to $\boldsymbol{\mu}$ is

$$\begin{aligned} &\sum_{e\in F} (\|\boldsymbol{\mu}\|_2^2 - \mu_e^2 + (\mu_e - 1)^2) v_e \\ &= \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1} + \sum_{e\in F} ((\mu_e - 1)^2 - \mu_e^2) v_e \\ &= \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1} + \sum_{e\in F} (-2\mu_e + 1) v_e \\ &= \|\boldsymbol{v}_F\|_1 - \|\boldsymbol{v}_F\|_2^2 / \|\boldsymbol{v}_F\|_1 . \end{aligned}$$

Thus, the weighted k-means objective takes the form

$$\min_{\mathcal{F}} \sum_{F\in\mathcal{F}} \left(\|\boldsymbol{v}_F\|_1 - \|\boldsymbol{v}_F\|_2^2 / \|\boldsymbol{v}_F\|_1 \right) \tag{26}$$

$$= \|\boldsymbol{v}\|_1 - \max_{\mathcal{F}} \sum_{F\in\mathcal{F}} \|\boldsymbol{v}_F\|_2^2 / \|\boldsymbol{v}_F\|_1 , \tag{27}$$

which concludes the proof. □

In (24), note that $\|\boldsymbol{v}\|_1$ is the total weight of input points; hence, we can equivalently solve the inner maximization problem. With the categorical weighted k-means objective in place, we can derive the optimal clustering. To do so, We next need the following elementary lemma.

Lemma C.2. *Suppose that $x, a_1, a_2, b_1, b_2 > 0$, $b_1^2 \geq a_1$, $b_2^2 \geq a_2$ and $x \geq \max\{a_1/b_1, a_2/b_2\}$. Then $x + \frac{a_1+a_2}{b_1+b_2} \geq \max\left\{\frac{x^2+a_1}{x+b_1} + \frac{a_2}{b_2}, \frac{x^2+a_2}{x+b_2} + \frac{a_1}{b_1}\right\}$.*

Proof. It suffices to establish $x + \frac{a_1+a_2}{b_1+b_2} \geq \frac{x^2+a_1}{x+b_1} + \frac{a_2}{b_2}$, or equivalently

$$x - \frac{x^2 + a_1}{x + b_1} \geq \frac{a_2}{b_2} - \frac{a_1 + a_2}{b_1 + b_2},$$

which can be simplified as

$$x(b_1 + b_2 + a_1/b_1 - a_2/b_2) \geq a_1 b_2/b_1 + a_2 b_1/b_2. \tag{28}$$

To verify this inequality, consider two cases. If $a_1/b_1 \geq a_2/b_2$, then $LHS \geq x(b_1 + b_2) \geq (a_2/b_2)b_1 + (a_1/b_1)b_2$. On the other hand, if $a_2/b_2 > a_1/b_1$. Since $b_2 - a_2/b_2 \geq 0$,

$$\begin{aligned} LHS &\geq (a_2/b_2)(b_1 + b_2 + a_1/b_1 - a_2/b_2) \\ &= a_2 b_1/b_2 + a_2 + a_1 a_2/(b_1 b_2) - a_2^2/b_2^2 \\ &= a_2 b_1/b_2 + a_1 b_2/b_1 \\ &\quad + (b_2 - a_2/b_2)(a_2/b_2 - a_1/b_1) \\ &\geq a_2 b_1/b_2 + a_1 b_2/b_1. \end{aligned}$$

Thus the proof is complete. □

Then, the optimal solution to the categorical k-means instance is an immediate consequence:

Corollary C.3. *Let $(e_1, \ldots, e_L)$ be a permutation of* $\mathsf{Dom}(K)$ *such that $v_{e_1} \geq v_{e_2} \geq \ldots \geq v_{e_L}$. Then for any $k \geq 2$ and any k-partition $\mathcal{F}$ of* $\mathsf{Dom}(K)$*, there holds*

$$v_{e_1} + \ldots + v_{e_{k-1}} + \frac{\sum_{i=k}^L v_i^2}{\sum_{i=k}^L v_i} \geq \sum_{F\in\mathcal{F}} \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1}.$$

Proof. We prove the claim by induction on k. Let $F \in \mathcal{F}$ be the set containing the element $\{e_1\}$. If there is only one element in F then we apply the induction hypothesis on the remaining terms. Otherwise, F contains at least two elements. Let $G \in \mathcal{F}$ be an arbitrary element of $\mathcal{F}$ where $G \neq F$. Define $\mathcal{F}'$ to be the partition obtained from $\mathcal{F}$ by replacing (F, G) with $(\{e_1\}, F \cup G - \{e_1\})$. Then, Lemma C.2 can be applied to get

$$\sum_{F \in \mathcal{F}} \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1} \leq \sum_{F \in \mathcal{F}'} \frac{\|\boldsymbol{v}_F\|_2^2}{\|\boldsymbol{v}_F\|_1}.$$

Induction on the tail $k-1$ terms completes the proof. □

Theorem 4.1 follows trivially from the above corollary. Corollary C.3 and the objective for k-means on a single attribute in the equation of Proposition C.1 establishes precisely the structure of the optimal solution for data consisting of a single categorical variable.

C.2 Reducing the coreset size with FDs

Next, we address the second call to $\mathsf{wkmeans}_2$: its runtime is dependent on the size of the grid $\boldsymbol{G}$, which can be up to $O(k^m)$, where m is the number of features from the input. Databases often contain *functional dependencies* (FDs), which we can exploit to reduce the size of $\boldsymbol{G}$. An FD is a dimension whose value depends entirely on the value of another dimension. For example, for a retailer dataset that includes geographic information, one might encounter features such as storeID, zip, city, state, and country. Here, storeID functionally determines zip, which determines city, which in turns determines state, leading to country. This common structure is known as an *FD-chain*, and appears often in real-world FEQs.

If we were to apply Rk-means without exploiting the FDs, the features storeID, zip, city, state, and country would contribute a factor of k^5 to the grid size. However, by using the functional dependency structure of the database, we show that only a factor of $5k$ is contributed to the grid size, because most of the k^5 grid points $\boldsymbol{g}$ have $w_{\mathsf{grid}}(\boldsymbol{g}) = 0$ as defined in (3). More generally, whenever there is an *FD chain* of (simple) functional dependencies including p features, their overall contribution to the grid size is a factor of $O(kp)$ instead of $O(k^p)$, and the grid points with non-zero weights can be computed efficiently in time $O(kp)$.

Lemma C.4. *Suppose all d input features are categorical and form an FD-chain. Then, the total number of grid points $\boldsymbol{g} \in \boldsymbol{G}$ with non-zero w_{grid} weight is at most $d(k-1)+1$.*

Proof. Suppose the features are $K_1, \ldots, K_d$, where K_i functionally determine K_{i+1}, and $\mathsf{Dom}(K_i) = \{e_1^i, e_2^i, \cdots, e_{n_i}^i\}$. Without loss of generality, we also assume that the elements in $\mathsf{Dom}(K_i)$ are sorted in descending order of weights:

$$w(\mathbf{1}_{e_1^i}) \geq w(\mathbf{1}_{e_2^i}) \geq \cdots \geq w(\mathbf{1}_{e_{n_i}^i}). \tag{29}$$

From Corollary C.3, we know the set $\boldsymbol{C}_i$ of k centroids of each of the categorical subspace for K_i: there is a centroid $\boldsymbol{\mu}_j^i = \mathbf{1}_{e_j^i}$ for each $j \in [k-1]$, and then a centroid $\boldsymbol{\mu}_k^i$ of the rest of the indicator vectors. The elements e_j^i for $j \in [k-1]$ shall be called "heavy" elements, and the rest are "light" elements.

Now, consider an input vector $\boldsymbol{x} = (x_1, \ldots, x_d)$ where $x_i \in \mathsf{Dom}(K_i)$. Under one-hot encoding, this vector is mapped to a vector of indicator vectors $\mathbf{1}_{\boldsymbol{x}} := (\mathbf{1}_{x_1}, \cdots, \mathbf{1}_{x_d})$. We need to answer the question: which grid point in $\boldsymbol{G} = \boldsymbol{C}_1 \times \cdots \times \boldsymbol{C}_d$ is $\mathbf{1}_{\boldsymbol{x}}$ closest to? Since the ℓ_2^2-distance is decomposable into component sum, we can determine the closest grid point by looking at the closest centroid in $\boldsymbol{C}_i$ for $\mathbf{1}_{x_i}$, for each $i \in [d]$.

If $x_i \in \{e_1^i, \ldots, e_{k-1}^i\}$, then the corresponding one-hot-encoded version $\mathbf{1}_{x_i}$ *is* itself one of the centroids in $\boldsymbol{C}_i$, and thus it is its own closest centroid. Otherwise, the closest centroid to $\mathbf{1}_{x_i}$ is $\boldsymbol{\mu}_k^i$, because $\left\|\mathbf{1}_{x_i} - \boldsymbol{\mu}_k^i\right\|^2 < 2$, and $\left\|\mathbf{1}_{x_i} - \boldsymbol{\mu}_j^i\right\|^2 = 2$ for every $j \in [k-1]$.

Let $\boldsymbol{\mu}^i(x_i) \in \boldsymbol{C}_i$ denote the closest centroid in $\boldsymbol{C}_i$ to $\mathbf{1}_{\boldsymbol{x}_i}$. The closest grid point to $\mathbf{1}_{\boldsymbol{x}}$ is completely determined: $\boldsymbol{g} = (\boldsymbol{\mu}^1(x_1), \cdots, \boldsymbol{\mu}^d(x_d))$. Furthermore, let $i \in [d]$ denote the smallest index such that x_i is heavy. Then, we can write $\boldsymbol{g}$ as

$$\boldsymbol{g} = (\boldsymbol{\mu}_k^1, \cdots, \boldsymbol{\mu}_k^{i-1}, \mathbf{1}_{x_i}, \boldsymbol{\mu}^{i+1}(x_{i+1}), \cdots, \boldsymbol{\mu}^d(x_d)) \tag{30}$$

Note that once x_i is fixed, due to the FD-chain the *entire* suffix $(\mathbf{1}_{x_i}, \boldsymbol{\mu}^{i+1}(x_{i+1}), \cdots, \boldsymbol{\mu}^d(x_d)$ of $\boldsymbol{g}$ is determined. Hence, the number of different $\boldsymbol{g}$s can only be at most $d(k-1)+1$: there are $d+1$ choices for i (from 0 to d), and $k-1$ choices for x_i if $i > 0$. □

Theorem 4.2 follows trivially from the above lemma, because the ℓ_2^2-distance is the sum over the ℓ_2^2-distances of the subspaces.

C.3 Analysis of Step 4 of Rk-means

Here we discuss the optimization and acceleration of Step 4 of the Rk-means implementation as described in Section 4. Recall that the categorical subspace k-means problem is solved trivially using Theorem 4.1, where we sort all the weights, and the heaviest $k-1$ elements form their own centroid, while the remaining vectors are clustered together (the "light cluster").

If S_j is a categorical subspace corresponding to a categorical variable K where $\mathsf{Dom}(K) = \{e_1, \ldots, e_L\}$. Without loss of generality, assume $w(\mathbf{1}_{e_1}) \geq \cdots \geq w(\mathbf{1}_{e_L})$, then the centroid of the light cluster is an

L-dimensional vector $\boldsymbol{c} = (s_e)_{e \in \mathsf{Dom}(K)}$

$$s_{e_i} := \begin{cases} 0 & i \in [k-1] \\ \frac{w(\mathbf{1}_{e_i})}{\sum_{j=k}^{L} w(\mathbf{1}_{e_j})} & i \geq k \end{cases} \tag{31}$$

This encoding is sound and space-inefficient.

Remember also that Step 4 clusters the coreset $\boldsymbol{G}$ using a modified version of Lloyd's weighted k-means that exploits the structure of $\boldsymbol{G}$ and sparse representation of categorical values. We show how to improve the distance computation $\|\boldsymbol{c}_j - \boldsymbol{\mu}_j\|^2$ for sub-space S_j, where $\boldsymbol{c}_j$ and $\boldsymbol{\mu}_j$ are the j-th components of a grid point and respectively of a centroid for $\boldsymbol{G}$. Since this subspace corresponds to a categorical variable K with, say, L_j categories, it is mapped into L_j sub-dimensions. Let $\boldsymbol{c}_j = [s_1, \dots, s_{L_j}]$ and $\boldsymbol{\mu}_j = (t_1, \dots, t_{L_j})$. Using the explicit one-hot encoding of its categories, we would need $O(L_j)$ time to compute $\|\boldsymbol{c}_j - \boldsymbol{\mu}_j\|^2 = \sum_{\ell \in [L_j]} (s_\ell - t_\ell)^2$. We can instead achieve $O(1)$ time as shown next. There are k distinct values for $\boldsymbol{c}_j$ by our coreset construction, each represented by a vector of size L_j with one non-zero entry for $k-1$ of them and $L_j - k + 1$ non-zero entries for one of them.

If $\boldsymbol{c}_j = \mathbf{1}_e$ is an indicator vector for some element $e \in K$ (e is one of the $k-1$ heavy categories), then

$$\|\boldsymbol{c}_j - \boldsymbol{\mu}_j\|^2 = \|\mathbf{1}_e - \boldsymbol{\mu}_j\|^2 = 1 - 2t_e + \|\boldsymbol{\mu}_j\|^2 . \tag{32}$$

If $\boldsymbol{c}_j$ is a light cluster centroid,

$$\|\boldsymbol{c}_j - \boldsymbol{\mu}_j\|^2 = \|\boldsymbol{c}_j\|^2 + \|\boldsymbol{\mu}_j\|^2 - 2\langle \boldsymbol{c}_j, \boldsymbol{\mu}_j \rangle . \tag{33}$$

In (32), by pre-computing $\|\boldsymbol{\mu}_j\|^2$ we only spend $O(1)$-time per heavy element e. In (33), by also pre-computing $\|\boldsymbol{c}_j\|^2$ and $\langle \boldsymbol{c}_j, \boldsymbol{\mu}_j \rangle$, and by noticing that $\boldsymbol{c}_j$ is $(L_j - k + 1)$-sparse, we spend $O(L_j - k)$-time here. Overall, we spend time $O(L_j)$ for computing $\|\boldsymbol{c}_j - \boldsymbol{\mu}_j\|^2$ per categorical dimension, modulo the pre-computation time.

Step 4 thus requires $O(|\boldsymbol{G}|mk + \sum_{j \in [m]} L_j k) = O(|\boldsymbol{G}|mk + Dkm)$ per iteration, whereas a generic approach would take time $O(\sum_{j \in [m]} |\boldsymbol{G}| k L_j) = O(|\boldsymbol{G}|Dkm)$. Our modified weighted k-means algorithm thus saves a factor proportional to the total domain sizes of the categorical variables, which may be as large as D.

C.4 Theorem 4.3

Proof of Theorem 4.3. Let N denote the maximum number of tuples in any input relation of the FEQ, $|\boldsymbol{X}|$ the number of tuples in the data matrix, fhtw the *fractional hypertree width* of the FEQ t the number of iterations of Lloyd's algorithm, d denote the number of features pre-one-hot encoding, r number of input relations to the FEQ, D the real dimensionality of the problem after one-hot-encoding.

We analyze the time complexity for each of the four steps of the Rk-means algorithm.

Step 1 projects $\boldsymbol{X}$ into each subspace S_j and compute the total weight of each projected point:

$$\forall j \in [d] : w_j(\boldsymbol{x}_{S_j}) := \sum_{\boldsymbol{x}_{[d] \setminus \{S_j\}}} \prod_{F \in \mathcal{E}} R_F(\boldsymbol{x}_F) \tag{34}$$

Each of the d FAQs (34) in Step 1 can be computed in time $\tilde{O}(rd^2 N^{\mathsf{fhtw}})$ using $\mathsf{InsideOut}$, as we have reviewed in Section A.

In Step 2, the optimal clustering in each dimension takes time $\tilde{O}(L_j)$ for each categorical variable j (whose domain size is L_j, and $O(kN^2)$ for each continuous variable, with an overall runtime of $O(kdN^2)$.

Step 3 constructs $\boldsymbol{G}$, whose size is bounded by $|\boldsymbol{X}|$ and by the FD result of Theorem 4.2. In practice, this number can be much smaller since we skip the data points whose weights are zero. To perform this step we construct a tree decomposition of FEQ with with equal fhtw (this step is data-independent, only dependent on the size of FEQ). Then, from each value x_j of an input variable X_j, we determine its centroid $c(x_j)$ which was computed in step 2. By conditioning on combinations of $(c_1, \dots, c_j)$, we can compute w_{grid} one for each combiation in $\tilde{O}(dN^{\mathsf{fhtw}})$-time, for a total run time of $\tilde{O}(rd|\boldsymbol{G}|N^{\mathsf{fhtw}})$.

Step 4 – as analyzed in Section C.3 – clusters $\boldsymbol{G}$ in time $O((|\boldsymbol{G}| + D)kmt)$, where t is the number of iterations of k-means used in Step 4. The most expensive computation is due to the one-dimensional clustering for the continuous variables and the computation of the coreset.

To compare the total runtime with $|\boldsymbol{X}|$, we only need to note that $|\boldsymbol{X}|$ can be as large as N^{ρ^*}, where $\rho*$ is the *fractional edge covering number* of the FEQ's hypergraph Ngo (2018). Depending on the query, ρ^* is always at least 1, and can be as large as the number of features d. Furthermore, there are classes of queries where fhtw is bounded by a constant, yet ρ^* is unbounded Marx (2013). This means, for classes of FEQs where $\rho^* > \max\{\mathsf{fhtw}, 2\}$ the ratio between $|\boldsymbol{X}|$ and Rk-means's runtime will be $\tilde{O}mega(N^{\rho^* - \max\{\mathsf{fhtw}, 2\}}/t)$, which is unbounded. $\square$

For reference, we compare the asymptotic runtime of Rk-means to the standard implementation of Lloyd's algorithm. The standard implementation contains two steps: (1) compute the one-hot-encoded data matrix

$\boldsymbol{X}$, and (2) run Lloyd's algorithm on $\boldsymbol{X}$. The first step, materializing $\boldsymbol{X}$, takes time $\tilde{O}(rd^2 N^{\mathsf{fhtw}} + D|\boldsymbol{X}|)$. The second step, running Lloyd algorithm, takes time $\tilde{O}(tkD|\boldsymbol{X}|)$, as is well known. Thus, the standard approach takes time $\tilde{O}(rd^2 N^{\mathsf{fhtw}} + tkD|\boldsymbol{X}|)$.

D Missing details from Section 5

We provide a more detailed description of the three datasets introduced in Section 5.

Retailer has five relations: *Inventory* stores the number of inventory units for each date, location, and stock keeping unit (sku); *Location* keeps for each store: its zipcode, the distance to the closest competitors, and the type of the store; *Census* provides 14 attributes that describe the demographics of a given zipcode, including population size or average household income; *Weather* stores statistics about the weather condition for each date and store, including the temperature and whether it rained; *Items* keeps track of the price, category, subcategory, and category cluster of each sku.

Favorita has six relations: *Sales* stores the number of units sold for items for a given date and store, and an indicator whether or not the unit was on promotion at this time; *Items* provides additional information about the skus, such as the item class and price; *Stores* keeps additional information on stores, like the city they are located it; *Transactions* stores the number of transaction for each date and store; *Oil* provides the oil price for each date; and *Holiday* indicates whether a given day is a holiday. The original dataset gave the `units_sold` attribute with a precision of three decimals places. This resulted in a very many distinct values for this attribute, which has a significant impact on the Step 2 of the Rk-means algorithm. We decreased the precision for this attribute to two decimal places, which decreases the number of distinct values by a factor of four. This modification has no effect on the final clusters or their accuracy.

Yelp has five relations: *Review* gives the review rating that a user gave to a business and the date of the review; *User* provides information about the users, including how many reviews the made, when they join, and how many fans they have; *Business* provides information about the businesses that are reviewed, such as their location and average rating; *Category* provide information about the categories, i.e. Restaurant, and respectively attributes of the business, *Attributes* is an aggregated relation, which stores the number of attributes (i.e., open late) that have been assigned to a business. A business can be categorized in many ways, which is the main reason why the size of the join is significantly larger than the underlying relations.

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0. URL `http://webdam.inria.fr/Alice/`.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28, 2016.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Juggling functions inside a database. *SIGMOD Rec.*, 46(1):6–13, 2017.

Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: In-database learning thunderstruck. In *2nd Workshop on Data Mgt for End-To-End ML*, DEEM'18, pages 8:1–8:10, 2018a.

Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *PODS*, pages 325–340, 2018b.

Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *FOCS*, pages 61–72, 2017.

D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *SODA*, page 1027–1035, 2007.

Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k -means clustering via lightweight coresets. In *SIGKDD*, pages 1119–1127, 2018.

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

Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1995–2003, 2013.

Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 576–585, 2017.

Field Cady. *The Data Science Handbook*. John Wiley & Sons, 2017.

Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong

Zhang. mlpack 3: a fast, flexible machine learning library. *J. Open Source Software*, 3:726, 2018.

C.J. Date, H. Darwen, and N. Lorentzos. *Temporal Data & The Relational Model*. Elsevier, 2002.

E. del Barrio, J. A. Cuesta-Albertos, C. Matran, and A. Mayo-Iscar. Robust clustering tools based on optimal transportation. *Statistics and Computing*, pages 1–22, 2017.

Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689, 2011.

B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley & Sons, 2011.

Favorita Corp. Corporacion Favorita Grocery Sales Forecasting: Can you accurately predict sales for a large grocery chain? `https://www.kaggle.com/c/favorita-grocery-sales-forecasting/`, 2017.

S. Graf and H. Luschgy. *Foundations of quantization for probability distributions*. Springer-Verlag, New York, 2000.

Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11 (1):4:1–4:20, 2014.

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

S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *SODA*, 2004.

Sariel Har-Peled and Soham Mazumdar. Coresets for k-means and k-median clustering and their applications. *CoRR*, abs/1810.12826, 2018. URL `http://arxiv.org/abs/1810.12826`.

J. A. Hartigan. *Clustering algorithms*. Wiley, New York, 1975.

N. Ho, X. Nguyen, M. Yurochkin, H. H. Bui, V. Huynh, and D. Phung. Multilevel clustering via Wasserstein means. In *ICML*, pages 1501–1509, 2017.

Z. Huang. Extensions to the k-means algorithm for clustering large data sets of categorical values. *Data mining and Knowledge discovery*, 2:283–304, 1998.

L. Kaufman and P. J. Roussew. *Finding Groups in Data - An Introduction to Cluster Analysis*. Wiley & Sons, 1990.

Mahmoud Abo Khamis, Ryan Curtin, Benjamin Moseley, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. On functional aggregate queries with additive inequalities. In *PODS*, 2019.

Daphne Koller and Nir Friedman. *Probabilistic graphical models*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2009. Principles and techniques.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

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.

Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*, pages 111–124, 2018.

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

Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

Carlos Ordonez. Integrating k-means clustering with a relational DBMS using SQL. *IEEE Trans. Knowl. Data Eng.*, 18(2):188–201, 2006.

Carlos Ordonez and Edward Omiecinski. Efficient disk-based k-means clustering for relational databases. *IEEE Trans. Knowl. Data Eng.*, 16(8):909–921, 2004.

D. Pollard. Quantization and the method of k-means. *IEEE Trans. Inf. Theory*, 28(2):199–204, 1982.

C. Sanderson and R.R. Curtin. A user-friendly hybrid sparse matrix class in C++. In *Proceedings of the 2018 International Congrees on Mathematical Software (ICMS)*, pages 422–430. Springer, 2018. ISBN 978-3-319-96418-8.

Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, pages 3–18, 2016.

Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813, 2018.

W. Sun, J. Wang, and Y. Fang. Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 9: 148–167, 2012.

Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, pages 249–260, 2001.

Todd L. Veldhuizen. Leapfrog Triejoin: A simple, worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481*, 2012.

Cédric Villani. *Optimal transport*, volume 338 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 2009.

Haizhou Wang and Mingzhou Song. Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29, 2011.

D. Witten and R. Tibshirani. A framework for feature selection in clustering. *Journal of the American Statistical Association*, 105:713–726, 2010.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008.

J. Ye, P. Wu, J. Wang, and J. Li. Fast discrete distribution clustering using barycenter with sparse support. *IEEE Trans. Signal Proc.*, 65(9):2317–2332, 2017.

Yelp. Yelp dataset challenge, `https://www.yelp.com/dataset/challenge/`, 2017.