On Functional Aggregate Queries with Additive Inequalities

Mahmoud Abo Khamis
RelationalAI

Ryan R. Curtin
RelationalAI

Benjamin Moseley
Carnegie Mellon University

Hung Q. Ngo
RelationalAI

XuanLong Nguyen
University of Michigan

Dan Olteanu
University of Oxford

Maximilian Schleich
University of Oxford

ABSTRACT

Motivated by fundamental applications in databases and relational machine learning, we formulate and study the problem of answering *functional aggregate queries* (FAQ) in which some of the input factors are defined by a collection of additive inequalities between variables. We refer to these queries as FAQ-AI for short.

To answer FAQ-AI in the Boolean semiring, we define *relaxed* tree decompositions and *relaxed* submodular and fractional hypertree width parameters. We show that an extension of the InsideOut algorithm using Chazelle's geometric data structure for solving the semigroup range search problem can answer Boolean FAQ-AI in time given by these new width parameters. This new algorithm achieves lower complexity than known solutions for FAQ-AI. It also recovers some known results in database query answering.

Our second contribution is a relaxation of the set of polymatroids that gives rise to the *counting* version of the submodular width, denoted by #subw. This new width is sandwiched between the submodular and the fractional hypertree widths. Any FAQ and FAQ-AI over one semiring can be answered in time proportional to #subw and respectively to the relaxed version of #subw.

We present three applications of our FAQ-AI framework to relational machine learning: k-means clustering, training linear support vector machines, and training models using non-polynomial loss. These optimization problems can be solved over a database asymptotically faster than computing the join of the database relations.

1 INTRODUCTION

We consider the problem of computing functional aggregate queries with inequality joins, or FAQ-AI queries for short. This is a fundamental computational problem that goes beyond databases: core computation for supervised and unsupervised machine learning can be formulated in FAQ-AI.

Inequalities occur naturally in scenarios involving temporal and spatial relationships between objects in databases. In a retail scenario (e.g., TPC-H), we would like to compute the revenue generated by a customer's orders whose dates closely precede the ship dates of their lineitems. In streaming scenarios, we would like to detect patterns of events whose time stamps follow a particular order [12]. In spatial data management scenarios, we would like to retrieve objects whose coordinates are within a multi-dimensional range or in close proximity of other objects [25]. The evaluation of Core XPath queries over XML documents amounts to the evaluation of a special class of conjunctive queries with inequalities expressing tree relationships in the pre/post plane [16].

1.1 Motivating examples

A key insight of this paper is that the efficient computation of inequality joins can reduce the computational complexity of supervised and unsupervised machine learning.

Example 1.1. The k-means algorithm divides the input dataset $\boldsymbol{G}$ into k clusters of similar data points [20]. Each cluster $\boldsymbol{G}_i$ has a mean $\boldsymbol{\mu}_i \in \mathbb{R}^n$, which is chosen according to the following optimization (similarity is defined here with respect to the ℓ_2 norm):

$$\min_{(G_1,\dots,G_k)} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \boldsymbol{G}_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2 . \tag{1}$$

Let $\mu_{i,\ell}$ be the ℓ'th component of mean vector $\boldsymbol{\mu}_i$. For a data point $\boldsymbol{x} \in \boldsymbol{G}$, the function c_{ij} computes the difference between the squares of the ℓ_2-distances from $\boldsymbol{x}$ to $\boldsymbol{\mu}_i$ and from $\boldsymbol{x}$ to $\boldsymbol{\mu}_j$:

$$c_{ij}(\boldsymbol{x}) = \sum_{\ell \in [n]} [\mu_{i,\ell}^2 - 2x_\ell(\mu_{i,\ell} + \mu_{j,\ell}) - \mu_{j,\ell}^2].$$

A data point $\boldsymbol{x} \in \boldsymbol{G}$ is closest to mean $\boldsymbol{\mu}_i$ from the set of k means iff $\forall j \in [k] : c_{ij}(\boldsymbol{x}) \leq 0$.

To compute the mean vector $\boldsymbol{\mu}_i$, we need to compute the sum of values for each dimension $\ell \in [n]$ over $\boldsymbol{G}_i$: $\sum_{\boldsymbol{x} \in \boldsymbol{G}_i} x_\ell$. If the dataset $\boldsymbol{G}$ is the join of database relations $(R_p)_{p \in [m]}$ over schemas $S_p \subseteq [n], \forall p \in [m]$, we can formulate this sum computation as a datalog-like query with aggregates [17]:

$$Q_1^{(i,\ell)}\left(\sum x_\ell\right) \leftarrow \left(\bigwedge_{p \in [m]} R_p(\boldsymbol{x}_{S_p})\right) \wedge \left(\bigwedge_{j \in [k]} c_{ij}(\boldsymbol{x}) \leq 0\right).$$

Section 4 gives further queries necessary to compute the means. As we show in this paper, such queries with aggregates and inequalities can be computed asymptotically faster than the join defining $\boldsymbol{G}$. □

Simple queries with inequalities can already show the limitations of current evaluation techniques, as highlighted next.

Example 1.2. State-of-the-art techniques take time $O(N^2)$ to compute the following query over relations of size at most N:

$$Q_2() \leftarrow R(a, b) \wedge S(b, c) \wedge T(c, d) \wedge a \leq d,$$

Example 3.9 (3.19) shows how to compute Q_2 (its counting version) in time $O(N^{1.5})$ using the techniques introduced in this paper. □

1.2 The FAQ-AI problem

One way to answer the above queries is to view them as *functional aggregate queries* (FAQ) [4] formulated in sum-product form over (potentially many) semirings. We therefore briefly introduce FAQ over a single semiring.

First we establish notation. For any positive integer n, let $\mathcal{V} = [n]$. For $i \in \mathcal{V}$, let X_i denote a variable/attribute, and x_i denote a value in the discrete domain $\mathsf{Dom}(X_i)$ of X_i. For any $K \subseteq \mathcal{V}$, define $\boldsymbol{X}_K = (X_i)_{i \in K}$, $\boldsymbol{x}_K = (x_i)_{i\in K} \in \prod_{i \in K} \mathsf{Dom}(X_i)$. That is, $\boldsymbol{X}_K$ is a tuple of variables and $\boldsymbol{x}_K$ is a tuple of values for these variables.

Let a semiring $(\boldsymbol{D}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1})$ and a **multi**[1]-hypergraph $\mathcal{H} = (\mathcal{V} = [n], \mathcal{E})$. To each edge $K \in \mathcal{E}$ we associate a function $R_K : \prod_{v\in K} \mathsf{Dom}(X_i) \to \boldsymbol{D}$ called *factor*.[2] A (single-semiring) FAQ query with free variables $F \subseteq \mathcal{V}$ has the form:

$$Q(\boldsymbol{x}_F) = \bigoplus_{\boldsymbol{x}_{\mathcal{V}\setminus F} \in \prod_{i\in\mathcal{V}\setminus F} \mathsf{Dom}(X_i)} \bigotimes_{K\in\mathcal{E}} R_K(\boldsymbol{x}_K). \tag{2}$$

Under the Boolean semiring $(\{\mathsf{true}, \mathsf{false}\}, \vee, \wedge, \mathsf{false}, \mathsf{true})$, the query Q in (2) becomes a conjunctive query: The factors R_K represent input relations, where $R_K(\boldsymbol{x}_K) = \mathsf{true}$ iff $\boldsymbol{x}_K \in R_K$, with some notational overloading. For counting the number of tuples in the result of a join query, we can use instead the sum-product semiring and define an indicator function $R_K(\boldsymbol{x}_K) = \boldsymbol{1}_{\boldsymbol{x}_K \in R_K}$ for every input relation R_K. To aggregate over some input variable, say X_k, we can designate an identity factor $R_k(x_k) = x_k$.

It is known [4] that over an arbitrary semiring, the query (2) can be answered in time $O(N^{\mathsf{fhtw}(Q)} \cdot \log N)$, where fhtw denotes the *fractional hypertree width* of the query and Q has no free variables [15]. If Q does have free variables, fhtw-width becomes FAQ-width instead [4]. Here N is the size of the largest factor R_K. Over the Boolean semiring, the time can be lowered to $\tilde{O}(N^{\mathsf{subw}(Q)})$ [6], where subw is the *submodular width* [26] and $\tilde{O}$ hides a polylogarithmic factor in N.

Motivated by the examples in Section 1.1, we formulate a class of FAQ queries called FAQ-AI: the hyperedge multiset $\mathcal{E}$ is partitioned into two multisets $\mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_\ell$, where s stands for "skeleton" and ℓ stands for "ligament". The input to our class of queries consists of the following: (1) to each hyperedge $K \in \mathcal{E}_s$, there corresponds a function $R_K : \prod_{i\in K} \mathsf{Dom}(X_i) \to \boldsymbol{D}$, as in the FAQ case; (2) to each hyperedge $S \in \mathcal{E}_\ell$, there corresponds $|S|$ functions $\theta^S_v : \mathsf{Dom}(X_v) \to \mathbb{R}$, one for every variable $v \in S$. The query we want to compute is the following:

$$Q(\boldsymbol{x}_F) = \bigoplus_{\boldsymbol{x}_{\mathcal{V}\setminus F}} \left(\bigotimes_{K\in\mathcal{E}_s} R_K(\boldsymbol{x}_K) \right) \otimes \left(\bigotimes_{S \in \mathcal{E}_\ell} \boldsymbol{1}_{\sum_{v\in S} \theta^S_v(x_v) \le 0} \right). \tag{3}$$

The summation $\bigoplus$ is over tuples $\boldsymbol{x}_{\mathcal{V}\setminus F} \in \prod_{i\in\mathcal{V}\setminus F} \mathsf{Dom}(X_i)$. The notation $\boldsymbol{1}_A$ denotes the indicator function of the event A in the semiring $(\boldsymbol{D}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1})$: $\boldsymbol{1}_A = \boldsymbol{1}$ if A holds, and $\boldsymbol{0}$ otherwise. The (uni-variate) functions θ^S_v can be user-defined functions, e.g., $\theta^S_1(x_1) = x_1^2/2$, or binary predicates with one key in $\mathsf{Dom}(X_v)$ and a numeric value. The only requirement we impose is that, given x, the value $\theta^S_v(x)$ can be accessed/computed in $\tilde{O}(1)$-time.

[1] This means that $\mathcal{E}$ is a multiset.
[2] The naming is borrowed from graphical models literature, where FAQ has its root.

Note that if $\mathcal{E}_\ell = \emptyset$, then we get back the FAQ formulation (2). Thus, FAQ-AI can also be considered a *super-class* of FAQ queries, i.e., FAQ and FAQ-AI are the same language.

Example 1.3. The queries from Section 1.1 are instances of (3):

$$Q_1^{(i,\ell)}() = \bigoplus_{\boldsymbol{x}_{[n]}} x_\ell \otimes \left(\bigotimes_{p\in[m]} R_p(\boldsymbol{x}_{S_p}) \right) \otimes \left(\bigotimes_{j\in[k]} \boldsymbol{1}_{c_{ij}(\boldsymbol{x}) \le 0} \right), \tag{4}$$

$$Q_2() = \bigoplus_{\boldsymbol{x}_{[4]}} R(x_1, x_2) \otimes S(x_2, x_3) \otimes T(x_3, x_4) \otimes \boldsymbol{1}_{x_1 - x_4 \le 0}. \tag{5}$$

Q_1 is on the sum-product semiring. Q_2 can be on any semiring: Example 3.9 discusses the case of the Boolean semiring while Example 3.19 discusses the sum-product semiring. □

1.3 Our contributions

To answer FAQ queries of the form (2), currently there are two dominant width parameters: fractional hypertree width (fhtw [15]) and submodular width (subw [26]).[3] It is known that subw ≤ fhtw for any query, and in the Boolean semiring we can answer (2) in $\tilde{O}(N^{\mathsf{subw}})$-time [6, 26]. For non-Boolean semirings, the best known algorithm, called InsideOut [4, 5], evaluates (2) in time $O(N^{\mathsf{fhtw}} \log N)$. For queries with free variables, fhtw is replaced by the more general notion of FAQ-width (faqw) [4]; however, for brevity we discuss the non-free variable case here.

Following [5], both width parameters subw and fhtw can be defined via two constraint sets: the first is the set TD of all tree decompositions of the query hypergraph $\mathcal{H}$, and the second is the set of polymatroids Γ_n on n vertices of $\mathcal{H}$. The widths subw and fhtw are then defined as maximin or minimax optimization problems on the domain pair TD and Γ_n, subject to "edge domination" constraints for Γ_n. Section 2 presents these notions and other related preliminary concepts in detail.

Our contributions include the following:

Answering FAQ-AI *over Boolean semiring.* On the Boolean semiring, one way to answer query (3) is to apply the PANDA algorithm [26], using edge domination constraints on $\mathcal{E}_s$ and the set TD of *all* tree decompositions of $\mathcal{H} = (\mathcal{V}, \mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_\ell)$. However, this is sub-optimal. Therefore, in Section 3.2 we define a new notion of tree decomposition: *relaxed tree decomposition*, in which the hyperedges in $\mathcal{E}_\ell$ only have to be covered by adjacent TD bags. Then, we present a variant of the InsideOut algorithm running on these relaxed TDs, exploiting Chazelle's classic geometric data structure [9] for solving the semigroup range search problem. We show that our InsideOut variant meets the "relaxed fhtw" runtime, which is the analog of fhtw on relaxed TD. The PANDA algorithm can use the InsideOut variant as a blackbox to meet the "relaxed subw" runtime. The relaxed widths are smaller than the non-relaxed counterparts, and are strictly smaller for some classes of queries, which means our algorithms yield asymptotic improvements over existing ones.

Answering FAQ *over other semirings.* Next, to prepare the stage for answering FAQ-AI over non-Boolean semirings, in Section 3.3 we revisit FAQ over non-Boolean semirings, where **no** known algorithm can achieve the subw-runtime. Here, we relax the set Γ_n

[3] Section 2.1 overviews other notions of widths.

of polymatroids to a superset Γ'_n of *relaxed polymatroids*. Then, by relaxing the subw definition over relaxed polymatroids, we obtain a new width parameter called "sharp submodular width" (#subw). We show how a variant of PANDA, called #PANDA, can achieve a runtime of $\tilde{O}(N^{\#\mathsf{subw}})$ for evaluating FAQ over non-Boolean semirings. We prove that subw ≤ #subw ≤ fhtw, and that there are classes of queries where #subw is unboundedly smaller than fhtw.

Answering FAQ-AI *over other semirings.* Getting back to FAQ-AI, we apply the #subw result under both relaxations: relaxed TD and relaxed polymatroids, to obtain a new width parameter called the relaxed #subw. We show that the new variants of PANDA and InsideOut can achieve the relaxed #subw runtime. We also show that there are queries for which relaxed #subw is essentially the best we can hope for, modulo k-sum-hardness.

Applications in relational Machine Learning. Equipped with the algorithms for answering FAQ-AI, in Section 4 we return to relational machine learning applications over datasets defined by feature extraction queries over relational databases. We show how one can train linear SVM, k-means, and ML models over Huber/hinge loss functions without completely materializing the output of the feature extraction queries. In particular, this shows that for these important classes of ML models, one can sometimes train models in time sub-linear in the training dataset size.

1.4 Related work

Appendix C revisits two prior results on the evaluation of queries with inequalities through FAQ-AI lenses: Core XPath queries over XML documents and inequality joins over tuple-independent probabilistic databases [30]. Throughout the paper, we contrast our new width notions with fhtw and subw and our new algorithm #PANDA with the state-of-the-art algorithms PANDA and InsideOut for FAQ and FAQ-AI queries. A seminal work considers the containment and minimization problem for queries with inequalities [22]. There is a bulk of work on queries with *dis*equalities (not-equal), e.g., [3] and references therein, which are at times referred to as inequalities.

Section 4 sets the context for our results on machine learning.

2 PRELIMINARIES

Throughout the paper, we use the following convention. For any Boolean event/variable A and a given semiring $(\boldsymbol{D}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, let $\mathbf{1}_A$ denote the indicator variable, which takes the value $\mathbf{1}$ if A holds (or is true), and $\mathbf{0}$ othewise. We assume without loss of generality in the paper that semiring operations $\oplus$ and $\otimes$ can be performed in $O(1)$-time. (When the assumption does not hold, for the set semiring for instance, we can multiply the claimed runtime with the real operation's runtime.)

2.1 Tree decompositions and polymatroids

We briefly define tree decompositions, fhtw and subw parameters. We refer the reader to the recent survey by Gottlob et al. [13] for more details and historical contexts. In what follows, the hypergraph $\mathcal{H}$ should be thought of as the hypergraph of the input query, although the notions of tree decomposition and width parameters are defined independently of queries.

A *tree decomposition* of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a pair (T, χ), where T is a tree and $\chi : V(T) \to 2^{\mathcal{V}}$ maps each node t of the tree to a subset $\chi(t)$ of vertices such that

(1) every hyperedge $F \in \mathcal{E}$ is a subset of some $\chi(t)$, $t \in V(T)$ (i.e. every edge is covered by some bag),
(2) for every vertex $v \in \mathcal{V}$, the set $\{t \mid v \in \chi(t)\}$ is a non-empty (connected) sub-tree of T. This is called the *running intersection property*.

The sets $\chi(t)$ are often called the *bags* of the tree decomposition.

Let $\mathsf{TD}(\mathcal{H})$ denote the set of all tree decompositions of $\mathcal{H}$. When $\mathcal{H}$ is clear from context, we use TD for brevity.

To define width parameters, we use the polymatroid characterization from Abo Khamis et al. [6]. A function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ is called a (non-negative) *set function* on $\mathcal{V}$. A set function f on $\mathcal{V}$ is *modular* if $f(S) = \sum_{v \in S} f(\{v\})$ for all $S \subseteq \mathcal{V}$, is *monotone* if $f(X) \leq f(Y)$ whenever $X \subseteq Y$, and is *submodular* if $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ for all $X, Y \subseteq \mathcal{V}$. A monotone, submodular set function $h : 2^{\mathcal{V}} \to \mathbb{R}_+$ with $h(\emptyset) = 0$ is called a *polymatroid*. Let Γ_n denote the set of all polymatroids on $\mathcal{V}$ with $|\mathcal{V}| = n$.

Given some $\mathcal{H}$, define the set of *edge dominated* set functions:

$$\mathsf{ED} := \{h \mid h : 2^{\mathcal{V}} \to \mathbb{R}_+, h(F) \leq 1, \forall F \in \mathcal{E}\}. \tag{6}$$

With this, we define the submodular width and fractional hypertree width of a given hypergraph $\mathcal{H}$:

$$\mathsf{fhtw}(\mathcal{H}) := \min_{(T,\chi)\in\mathsf{TD}} \max_{h\in\mathsf{ED}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)), \tag{7}$$

$$\mathsf{subw}(\mathcal{H}) := \max_{h\in\mathsf{ED}\cap\Gamma_n} \min_{(T,\chi)\in\mathsf{TD}} \max_{t\in V(T)} h(\chi(t)). \tag{8}$$

It is known [26] that $\mathsf{subw}(\mathcal{H}) \leq \mathsf{fhtw}(\mathcal{H})$, and there are classes of hypergraphs with bounded subw and unbounded fhtw. Furthermore, fhtw is strictly less than other width notions such as (generalized) hypertree width and tree width.

Remark 2.1. Prior to Abo Khamis et al. [6], the commonly used definition of $\mathsf{fhtw}(\mathcal{H})$ is $\mathsf{fhtw}(\mathcal{H}) := \min_{(T,\chi)\in\mathsf{TD}} \max_{t\in V(T)} \rho^*_{\mathcal{E}}(\chi(t))$, where $\rho^*_{\mathcal{E}}(B)$ is the fractional edge cover number of a vertex set B using the hyperedge set $\mathcal{E}$. It is straightforward to show, using linear programming duality [6], that

$$\max_{t\in V(T)} \max_{h\in\mathsf{ED}\cap\Gamma_n} h(\chi(t)) = \max_{t\in V(T)} \rho^*_{\mathcal{E}}(\chi(t)), \tag{9}$$

proving the equivalence of the two definitions. However, the characterization (7) has two primary advantages: (i) it exposes the minimax / maximin duality between fhtw and subw, and more importantly (ii) it makes it completely straightforward to relax the definitions by replacing the $\mathsf{ED} \cap \Gamma_n$ constraints by other applicable constraints, as shall be shown in later sections. □

Definition 2.2 (F-connex tree decomposition [7, 32]). Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and a set $F \subseteq \mathcal{V}$, a tree decomposition (T, χ) of $\mathcal{H}$ is *F-connex* if $F = \emptyset$ or the following holds: There is a nonempty subset $V' \subseteq V(T)$ that forms a connected subtree of T and satisfies $\bigcup_{t\in V'} \chi(t) = F$.

We use TD_F to denote the set of all F-connex tree decompositions of $\mathcal{H}$. (Note that when $F = \emptyset$, $\mathsf{TD}_F = \mathsf{TD}$.)

2.2 InsideOut and PANDA

To answer the FAQ query (2), we need a model for the representation of the input factors R_K. The support of the function R_K is the set of tuples $\boldsymbol{x}_K$ such that $R(\boldsymbol{x}_K) \neq \mathbf{0}$. We use $|R_K|$ to denote the size of its support. For example, if R_K represents an input relation, then $|R_K|$ is the number of tuples in R_K. In practice, there often are factors with infinite support, e.g., R_K represents a built-in function in a database, an arithmetic operator, or a comparison operator as in (3). To deal with this more general setting, the edge set $\mathcal{E}$ is partitioned into two sets $\mathcal{E} = \mathcal{E}_{\not\infty} \cup \mathcal{E}_\infty$, where $|R_K|$ is finite for all $K \in \mathcal{E}_{\not\infty}$ and $|R_K| = \infty$ for all $K \in \mathcal{E}_\infty$. For simplicity, we often state runtimes of algorithms in terms of the "input size" $N := \max_{K \in \mathcal{E}_{\not\infty}} |R_K|$. Moreover, we use $|Q|$ to denote the output size of Q.

InsideOut *[4, 5].* To answer (2), the InsideOut algorithm works by eliminating variables, along with an idea called the "indicator projection". Its runtime is described by the FAQ-*width* of the query, a slight generalization of fhtw. In the context of one semiring, we can define faqw(Q) by applying Definition (7) over a restricted set of tree decompositions and edge dominated polymatroids. In particular, let $F \subseteq \mathcal{V}$ denote the set of free variables in (2), and recall TD_F from Definition 2.2. Then,

$$\mathsf{ED}_{\not\infty} := \{h \mid h : 2^{\mathcal{V}} \to \mathbb{R}_+, h(F) \leq 1, \forall F \in \mathcal{E}_{\not\infty}\}, \quad (10)$$

$$\mathsf{faqw}(Q) := \min_{(T,\chi)\in\mathsf{TD}_F} \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)) \quad (11)$$

$$(\text{by Remark 2.1}) = \min_{(T,\chi)\in\mathsf{TD}_F} \max_{t\in V(T)} \rho^*_{\mathcal{E}_{\not\infty}}(\chi(t)) \quad (12)$$

Note that $\mathsf{faqw}(Q) = \mathsf{fhtw}(\mathcal{H})$ when $F = \emptyset$ and $\mathcal{E}_\infty = \emptyset$ (i.e. $\mathcal{E} = \mathcal{E}_{\not\infty}$). A simple result from Abo Khamis et al. [4] is the following:

Proposition 2.3. InsideOut *answers query* (2) *in time* $O(N^{\mathsf{faqw}(Q)} \cdot \log N + |Q|)$.

To solve the FAQ-AI (3), we can apply Proposition 2.3 with $\mathcal{E}_\infty \supseteq \mathcal{E}_\ell$ (because all ligament factors are infinite). But this is suboptimal—later, we show a new InsideOut variant that is polynomially better.

PANDA *[6].* In case of the Boolean semiring, i.e., when the FAQ query (2) is of the form

$$Q(\boldsymbol{x}_F) = \bigvee_{\boldsymbol{x}_{\mathcal{V}\setminus F}\in\prod_{i\in\mathcal{V}\setminus F}\mathsf{Dom}(X_i)} \bigwedge_{K\in\mathcal{E}} R_K(\boldsymbol{x}_K), \quad (13)$$

we can do much better than Proposition 2.3. When $F = \emptyset$, Marx [26] showed that (13) can be answered in time $\tilde{O}(N^{O(\mathsf{subw}(Q))})$. The PANDA algorithm [6] generalizes Marx's result to deal with general degree constraints, and to meet precisely the $\tilde{O}(N^{\mathsf{subw}(Q)})$-runtime. In fact, PANDA works with queries such as (13) with free variables as well. In the context of this paper, we can define the following notion of *submodular* FAQ-*width* in a very natural way:

$$\mathsf{smfw}(Q) := \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_n} \min_{(T,\chi)\in\mathsf{TD}_F} \max_{t\in V(T)} h(\chi(t)). \quad (14)$$

Then, the results from Abo Khamis et al. [6] imply:

Proposition 2.4. PANDA *answers query* (13) *in time* $\tilde{O}(N^{\mathsf{smfw}(Q)} + |Q|)$.

These results only work for the Boolean semiring. Section 3 introduces a variant of PANDA, called #PANDA, that also works for non-Boolean semirings.

2.3 Semigroup range searching

Orthogonal range counting (and searching) is a classic and ubiquitous problem in computational geometry [11]: given a set S of N points in a d-dimensional space, build a data structure that, given any d-dimensional rectangle, can efficiently return the number of enclosed points. More generally, there is the semigroup range searching problem [9], where each point $\boldsymbol{p} \in S$ of the N input points also has a weight $w(\boldsymbol{p}) \in G$, where $(G, \oplus)$ is a *semigroup*.[4] The problem is: given a d-dimensional rectangle R, compute $\bigoplus_{\boldsymbol{p}\in S\cap R} w(\boldsymbol{p})$.

Classic results by Chazelle [9] show that there are data structures for semigroup range searching which can be constructed in time $O(N \log^{d-1} N)$, and answer rectangular queries in $O(\log^{d-1} N)$-time. Also, this is almost the best we can hope for [10]. There are more recent improvements to Chazelle's result (see, e.g., Chan et al. [8]), but they are minor (at most a log factor), as the original results were already very close to matching the lower bound.

Most of these range search/counting problems can be reduced to the dominance range searching problem (on semigroups), where the query is represented by a point $\boldsymbol{q}$, and the objective is to return $\bigoplus_{\boldsymbol{q}\leq\boldsymbol{p},\boldsymbol{p}\in S} w(\boldsymbol{p})$. Here, $\leq$ denotes the "dominance" relation (coordinate-wise $\leq$). We can think of $\boldsymbol{q}$ as the lower-corner of an infinite rectangle query.

3 RELAXED TREE DECOMPOSITIONS AND RELAXED POLYMATROIDS

3.1 Connection to a geometric data structure

We start with a special case of (3) in which the skeleton part $\mathcal{E}_s$ contains only *two* hyperedges U and L. Formally, consider the aggregate query of the form

$$Q(\boldsymbol{x}_F) = \bigoplus_{\boldsymbol{x}_{\mathcal{V}\setminus F}} \Phi_1(\boldsymbol{x}_U) \otimes \Phi_2(\boldsymbol{x}_L) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\sum_{v\in S}\theta^S_v(x_v)\leq 0}\right), \quad (15)$$

where Φ_1 and Φ_2 are two input functions/relations over variable sets U and L, respectively. We prove the following very simple but important lemma:

Lemma 3.1. *Let* $N = \max\{|\Phi_1|, |\Phi_2|\}$*, and* $k = |\mathcal{E}_\ell|$*, then when* $F \subseteq U$*, query* (15) *can be answered in time* $O(N \cdot (\log N)^{\max(k-1,1)})$.

Proof. If there is a hyperedge $S \in \mathcal{E}_\ell$ for which $S \subseteq U$, then in a $O(N \log N)$-time pre-processing step we can "absorb" the factor $\mathbf{1}_{\sum_{v\in S}\theta^S_v(x_v)\leq 0}$ into the factor Φ_1, by replacing $\Phi_1(\boldsymbol{x}_U)$ with $\Phi_1(\boldsymbol{x}_U) \otimes \mathbf{1}_{\sum_{v\in S}\theta^S_v(x_v)\leq 0}$. A similar absorption can be done with $S \subseteq L$. Hence, without loss of generality we can assume that $S \not\subseteq L$ and $S \not\subseteq U$ for all $S \in \mathcal{E}_\ell$. Furthermore, we only need to show that we can compute (15) for $F = U$, because after $Q(\boldsymbol{x}_U)$ is computed, we can marginalize away variables $\boldsymbol{x}_{U\setminus F}$ in $O(N \log N)$-time.

Abusing notation somewhat, for each $S \in \mathcal{E}_\ell$ and each $T \subseteq S$, define the function $\theta^S_T : \prod_{v\in T} \mathsf{Dom}(X_v) \to \mathbb{R}$ by

$$\theta^S_T(\boldsymbol{x}_T) := \sum_{v\in T} \theta^S_v(x_v). \quad (16)$$

[4]In a semigroup we can add two elements using $\oplus$, but there is no additive inverse.

Fix a tuple $\boldsymbol{x}_U$ such that $\Phi_1(\boldsymbol{x}_U) \neq 0$. A tuple $\boldsymbol{x}_L$ is said to be $\boldsymbol{x}_U$-*adjacent* if $\pi_{U\cap L}\boldsymbol{x}_U = \pi_{U\cap L}\boldsymbol{x}_L$. We show how to compute the following sum in poly-logarithmic time:

$$\bigoplus_{\boldsymbol{x}_{L\setminus U}} \Phi_1(\boldsymbol{x}_U) \otimes \Phi_2(\boldsymbol{x}_L) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\sum_{v\in S}\theta_v^S(x_v)\leq 0} \right) = \tag{17}$$

$$\Phi_1(\boldsymbol{x}_U) \otimes \bigoplus_{\boldsymbol{x}_{L\setminus U}} \Phi_2(\boldsymbol{x}_L) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\theta^S_{S\cap U}(\boldsymbol{x}_{S\cap U})\leq -\theta^S_{S\setminus U}(\boldsymbol{x}_{S\setminus U})} \right). \tag{18}$$

where the inner sum ranges over only tuples $\boldsymbol{x}_L$ which are $\boldsymbol{x}_U$-adjacent; non-adjacent tuples contribute 0.

Now, for each $\boldsymbol{x}_U$ define two k-dimensional points:

$$\boldsymbol{q}(\boldsymbol{x}_U) = (q_S(\boldsymbol{x}_U))_{S\in\mathcal{E}_\ell} \quad \text{where} \quad q_S(\boldsymbol{x}_U) := \theta^S_{S\cap U}(\boldsymbol{x}_{S\cap U}), \tag{19}$$

$$\boldsymbol{p}(\boldsymbol{x}_L) = (p_S(\boldsymbol{x}_L))_{S\in\mathcal{E}_\ell} \quad \text{where} \quad p_S(\boldsymbol{x}_L) := -\theta^S_{S\setminus U}(\boldsymbol{x}_{S\setminus U}). \tag{20}$$

We write $\boldsymbol{q}(\boldsymbol{x}_U) \leq \boldsymbol{p}(\boldsymbol{x}_L)$ to say that $\boldsymbol{q}(\boldsymbol{x}_U)$ is dominated by $\boldsymbol{p}(\boldsymbol{x}_L)$ *coordinate-wise*: $q_S(\boldsymbol{x}_U) \leq p_S(\boldsymbol{x}_L)\ \forall\ S \in \mathcal{E}_\ell$. Assign to each point $\boldsymbol{p}(\boldsymbol{x}_L)$ a "weight" of $\Phi_2(\boldsymbol{x}_L)$. Now, taking (18),

$$\bigoplus_{\boldsymbol{x}_{L\setminus U}} \Phi_2(\boldsymbol{x}_L) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\theta^S_{S\cap U}(\boldsymbol{x}_{S\cap U})\leq -\theta^S_{S\setminus U}(\boldsymbol{x}_{S\setminus U})} \right)$$

$$= \bigoplus_{\boldsymbol{x}_{L\setminus U}} \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{q_S(\boldsymbol{x}_U)\leq p_S(\boldsymbol{x}_L)} \right) \otimes \Phi_2(\boldsymbol{x}_L) \tag{21}$$

$$= \bigoplus_{\boldsymbol{x}_{L\setminus U}} \mathbf{1}_{\boldsymbol{q}(\boldsymbol{x}_U)\leq \boldsymbol{p}(\boldsymbol{x}_L)} \otimes \Phi_2(\boldsymbol{x}_L). \tag{22}$$

The expression thus computes, for a given "query point" $\boldsymbol{q}(\boldsymbol{x}_U)$, the weighted sum over all points $\boldsymbol{p}(\boldsymbol{x}_L)$ that dominate the query point. This is precisely the dominance range counting problem, which—modulo a $O(N(\log N)^{\max(k-1,1)})$-preprocessing step—can be solved in time $O((\log N)^{\max(k-1,1)})$ [9], as reviewed in Section 2.3.

To conclude the proof, note that (15) can be written as (assuming $F \subseteq U$ as is the case in Lemma 3.1)

$$Q(\boldsymbol{x}_F) = \bigoplus_{\boldsymbol{x}_{U\setminus F}} \underbrace{\bigoplus_{\boldsymbol{x}_{L\setminus U}} \Phi_1(\boldsymbol{x}_U) \otimes \Phi_2(\boldsymbol{x}_L) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\sum_{v\in S}\theta_v^S(x_v)\leq 0} \right)}_{\text{same as (17)}},$$

where the outer sum ranges over N tuples $\boldsymbol{x}_U$ in Φ_1. ◻

Example 3.2. Let R be a binary relation. Suppose we want to count the number of tuples satisfying $R(a, b) \wedge R(b, c) \wedge a < c$, then by setting $F = \emptyset$, $U = \{a, b\}$, $L = \{b, c\}$, it is easy to see that the problem can be reduced to the form (15) with $k = 1$, $\mathcal{E}_\ell = \{\{a, c\}\}$. We can thus compute this count in time $O(N \log N)$. ◻

3.2 Relaxed tree decompositions

Equipped with this basic case, we can now proceed to solve the general setting of (3). To this end, we define a new width parameter.

Definition 3.3 (Relaxed tree decomposition)**.** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_\ell)$ denote a multi-hypergraph whose edge multiset is partitioned into $\mathcal{E}_s$ and $\mathcal{E}_\ell$. A *relaxed* tree decomposition of $\mathcal{H}$ (with respect to the partition $\mathcal{E}_s \cup \mathcal{E}_\ell$) is a pair (T, χ), where $T = (V(T), E(T))$ is a tree, and $\chi : V(T) \to 2^{\mathcal{V}}$ satisfies the following properties:

- The running intersection property holds: for each node $v \in \mathcal{V}$ the set $\{t \in V(T) \mid v \in \chi(t)\}$ is a connected subtree in T.
- Every "skeleton" edge $F \in \mathcal{E}_s$ is covered by some bag $\chi(t)$, $t \in V(T)$.
- Every "ligament" edge $F \in \mathcal{E}_\ell$ is covered by the union of two *adjacent* bags: $F \subseteq \chi(s) \cup \chi(t)$, where $\{s, t\} \in E(T)$.

Let $\mathsf{TD}^\ell(\mathcal{H})$ denote the set of all relaxed tree decompositions of $\mathcal{H}$ (with respect to the skeleton-ligament partition). When $\mathcal{H}$ is clear from context we use TD^ℓ for the sake of brevity. Let TD^ℓ_F denote the set of all relaxed F-connex tree decompositions of $\mathcal{H}$.

3.2.1 FAQ-AI *on a general semiring.* We use relaxed TDs in conjunction with Lemma 3.1 to answer FAQ-AI with a relaxed notion of faqw. In particular, the *relaxed* width parameters of $\mathcal{H}$ are defined in exactly the same way as the usual width parameters defined in Section 2, except we allow the TDs to range over relaxed ones.

Definition 3.4 (Relaxed faqw)**.** Let Q be an FAQ-AI query (3), and $\mathcal{H} = (\mathcal{V}, \mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_\ell)$ be its hypergraph. Furthermore, let $\mathcal{E}_{\not\infty} \subseteq \mathcal{E}_s$ denote the set of hyperedges $K \in \mathcal{E}$ for which $|R_K| < \infty$. Then, the *relaxed* FAQ-*width* of Q is defined by

$$\mathsf{faqw}_\ell(Q) := \min_{(T,\chi)\in\mathsf{TD}^\ell_F} \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)) \tag{23}$$

When $F = \emptyset$, faqw_ℓ collapses back to fhtw_ℓ, in which case we define the *relaxed* fhtw for FAQ-AI Q *without* free variables:

$$\mathsf{fhtw}_\ell(Q) := \min_{(T,\chi)\in\mathsf{TD}^\ell_\emptyset} \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t)) \tag{24}$$

A relaxed tree decomposition (T, χ) of Q is *optimal* if its width is equal to faqw_ℓ, i.e., $\mathsf{faqw}_\ell(Q) = \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_n} \max_{t\in V(T)} h(\chi(t))$.

Theorem 3.5. *Any* FAQ-AI *query Q of the form* (3) *on any semiring can be answered in time $O(N^{\mathsf{faqw}_\ell(Q)} \cdot (\log N)^{\max(k-1,1)} + |Q|)$, where k is the maximum number of additive inequalities covered by a pair of adjacent bags in an optimal relaxed tree decomposition.*[5]

Proof. We first consider the case of no free variables (i.e. $F = \emptyset$), because this case captures the key idea. Fix an optimal relaxed TD (T, χ). We first compute, for each bag $t \in V(T)$ of the tree decomposition, a factor $\Phi_t : \prod_{i\in\chi(t)} \mathsf{Dom}(X_i) \to \boldsymbol{D}$ such that

$$Q() = \bigoplus_{\boldsymbol{x}_{\mathcal{V}}} \left(\bigotimes_{K\in\mathcal{E}_s} R_K(\boldsymbol{x}_K) \right) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\sum_{v\in S}\theta_v^S(x_v)\leq 0} \right) \tag{25}$$

$$= \bigoplus_{\boldsymbol{x}_{\mathcal{V}}} \left(\bigotimes_{t\in V(T)} \Phi_t(\boldsymbol{x}_{\chi(t)}) \right) \otimes \left(\bigotimes_{S\in\mathcal{E}_\ell} \mathbf{1}_{\sum_{v\in S}\theta_v^S(x_v)\leq 0} \right). \tag{26}$$

To define the factors Φ_t, we need the notion of the *indicator projection* [4, 5]. For a given $K \in \mathcal{E}_s$ and $t \in V(T)$ such that $J := K \cap \chi(t) \neq \emptyset$, the indicator projection of R_K onto the bag $\chi(t)$ is a function $\pi_{t,K} : \prod_{v\in J} \mathsf{Dom}(X_v) \to \{\mathbf{0}, \mathbf{1}\}$ defined by

$$\pi_{t,K}(\boldsymbol{x}_J) := \begin{cases} \mathbf{1} & \exists \boldsymbol{x}_{K\setminus J} \text{ s.t. } R_K((\boldsymbol{x}_J, \boldsymbol{x}_{K\setminus J})) \neq \mathbf{0}, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{27}$$

[5]Note that k can be a lot smaller than $|\mathcal{E}_\ell|$.

Recall from Definition 3.3 that every $K \in \mathcal{E}_s$ is covered by at least one bag $\chi(t)$ for $t \in V(T)$. Fix an arbitrary coverage assignment $\alpha : \mathcal{E}_s \to V(T)$, where K is covered by the bag $\chi(\alpha(K))$. Then, the factors Φ_t are defined by:

$$\Phi_t(\boldsymbol{x}_{\chi(t)}) := \bigotimes_{K \in \alpha^{-1}(t)} R_K(\boldsymbol{x}_K) \otimes \bigotimes_{\substack{K \in \mathcal{E}_s \\ K \cap \chi(t) \neq \emptyset}} \pi_{t,K}(\boldsymbol{x}_{K \cap \chi(t)}). \quad (28)$$

It is straightforward to verify that (26) holds. Using any worst-case optimal join algorithm [28, 29, 36] we can compute (28) in time

$$\tilde{O}(N^{\rho^*_{\mathcal{E}_{\not\infty}}(\chi(t))}) = O(N^{\max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_n} h(\chi(t))}). \quad (29)$$

Over all $t \in V(T)$, our runtime is bounded by $O(N^w)$, where

$$w = \max_{t \in V(T)} \max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_n} h(\chi(t)). \quad (30)$$

In addition, the support of each factor Φ_t has size bounded by N^w.

Next we compute (26) in time $\tilde{O}(N^w)$. We will make use of the fact that (T, χ) is a relaxed TD. Fix an arbitrary root of the tree decomposition (T, χ); following InsideOut, we compute (26) by eliminating variables from the leaves of (T, χ) up to the root. Without loss of generality, we assume that the tree decomposition is non-redundant, i.e., no bag is a subset of another in the tree decomposition (otherwise the contained bag factor can be "absorbed" into the containee bag factor). Let t_1 be any leaf of (T, χ), t_2 be its parent, where $L = \chi(t_1)$ and $U = \chi(t_2)$. Now write (26) as follows:

$$\bigoplus_{\boldsymbol{x}_\mathcal{V}} \left(\bigotimes_{t \in V(T)} \Phi_t(\boldsymbol{x}_{\chi(t)}) \right) \otimes \left(\bigotimes_{S \in \mathcal{E}_\ell} \mathbf{1}_{\sum_{v \in S} \theta^S_v(x_v) \leq 0} \right)$$

$$= \bigoplus_{\boldsymbol{x}_{\mathcal{V} \setminus (L \setminus U)}} \bigoplus_{\boldsymbol{x}_{L \setminus U}} \left(\bigotimes_{t \in V(T)} \Phi_t(\boldsymbol{x}_{\chi(t)}) \right) \otimes \left(\bigotimes_{S \in \mathcal{E}_\ell} \mathbf{1}_{\sum_{v \in S} \theta^S_v(x_v) \leq 0} \right) \quad (31)$$

$$= \bigoplus_{\boldsymbol{x}_{\mathcal{V} \setminus (L \setminus U)}} \left(\bigotimes_{t \in V(T) \setminus \{t_1, t_2\}} \Phi_t(\boldsymbol{x}_{\chi(t)}) \right) \otimes \left(\bigotimes_{\substack{S \in \mathcal{E}_\ell \\ S \cap (L \setminus U) = \emptyset}} \mathbf{1}_{\sum_{v \in S} \theta^S_v(x_v) \leq 0} \right)$$

$$\otimes \underbrace{\left[\bigoplus_{\boldsymbol{x}_{L \setminus U}} \Phi_{t_1}(\boldsymbol{x}_L) \otimes \Phi_{t_2}(\boldsymbol{x}_U) \otimes \left(\bigotimes_{\substack{S \in \mathcal{E}_\ell \\ S \cap (L \setminus U) \neq \emptyset}} \mathbf{1}_{\sum_{v \in S} \theta^S_v(x_v) \leq 0} \right) \right]}_{\text{sub-query of the form (15)}}. \quad (32)$$

The third equality uses the semiring's distributive law. (Note that $S \cap (L \setminus U) \neq \emptyset$ implies that $S \subseteq (L \cup U)$ thanks to Definition 3.3 and the fact that t_2 is the only neighbor of t_1.) Lemma 3.1 implies that we can compute the sub-query in the allotted time. The above step eliminates all variables in $L \setminus U$. Repeatedly applying the above step yields the desired output $Q()$.

When the query has free variables, the algorithm proceeds similarly to the case of an FAQ query with free variables [4, 5]. □

Example 3.6. Given 3 binary relations R, S and T, consider a query Q about the number of tuples (a, b, c, d) that satisfy:

$$R(a,b) \wedge S(b,c) \wedge T(c,d) \wedge (a \leq c) \wedge (c \leq b) \wedge (d \leq b). \quad (33)$$

The query Q has $\mathcal{E}_s = \mathcal{E}_{\not\infty} = \{\{a,b\},\{b,c\},\{c,d\}\}$ and $\mathcal{E}_\ell = \mathcal{E}_\infty = \{\{a,c\},\{b,c\},\{b,d\}\}$. Let $N = \max\{|R|,|S|,|T|\}$. Note that

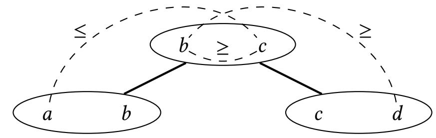


Figure 1: An optimal relaxed tree decomposition for the query in Example 3.6. *Ligament* edges are dashed. Each *skeleton* edge is held in one bag.

$\mathsf{faqw}(Q) = 2$. In fact, any of the previously known algorithms, e.g. [4, 5], would take time $O(N^2)$ to answer Q. However, this query has $\mathsf{faqw}_\ell(Q) = 1$, and by Theorem 3.5, it can be answered in time $O(N \cdot \log N)$. (Note that here $2 = k < |\mathcal{E}_\ell| = 3$.) An optimal relaxed tree decomposition is shown in Figure 1. □

We next give a couple of simple lower and upper bounds for faqw_ℓ. The upper bound shows that, effectively faqw_ℓ is the best we can hope for, if the FAQ-AI query is arbitrary. The lower bound shows that, while the relaxed tree decomposition idea can improve the runtime by a polynomial factor, it cannot improve the runtime over straightforwardly applying InsideOut (over non-relaxed tree decompositions) by more than a polynomial factor.

Proposition 3.7. *For any positive integer m, there exists an* FAQ-AI *query of the form* (3) *for which $F = \emptyset$, $\mathsf{faqw}_\ell(Q) \geq m$ and it cannot be answered in time $o(N^{\mathsf{faqw}_\ell(Q)})$, modulo k-*sum *hardness.*

Proposition 3.8. *For any* FAQ-AI *query Q of the form* (3)*, we have $\mathsf{faqw}_\ell(Q) \geq \frac{1}{2}\mathsf{faqw}(Q)$; in particular, when Q has no free variables $\mathsf{fhtw}_\ell(Q) \geq \frac{1}{2}\mathsf{fhtw}(Q)$.*

3.2.2 FAQ-AI *on the Boolean semiring.* Before formally explaining how we can adapt PANDA to solve an FAQ-AI query on the Boolean semiring, we give the intuition with an example.

Example 3.9. Consider the following FAQ-AI (written in Datalog):

$$Q() \leftarrow R(a,b) \wedge S(b,c) \wedge T(c,d) \wedge a \leq d. \quad (34)$$

Here $\mathsf{faqw}_\ell(Q) = \mathsf{faqw}(Q) = 2$. Using fractional hypertree width measure and InsideOut (even with relaxed TDs and Theorem 3.5), the best runtime is $O(N^2)$, because no matter which (relaxed) TD we choose, the worst-case bag relation size is $\Theta(N^2)$. A key idea of the PANDA framework [6] is the use of a *disjunctive Datalog rule.* Consider the following disjunctive Datalog rule:

$$U(a,b,c) \vee W(b,c,d) \leftarrow R(a,b) \wedge S(b,c) \wedge T(c,d). \quad (35)$$

There are two relations in the head U and W, and they form a solution to the rule iff the following holds: if (a,b,c,d) satisfies the body, then either $(a,b,c) \in U$ or $(b,c,d) \in W$. Via information-theoretic inequalities [6], we are able to show that PANDA can compute a solution (U, W) to the above disjunctive Datalog rule in time $\tilde{O}(N^{1.5})$. In particular, both $|U|$ and $|W|$ are bounded by $N^{1.5}$.

Given the solution (U, W) to (35), it is straightforward to verify that the following also holds, using the distributivity of $\vee$ over $\wedge$:

$$(R(a,b) \wedge W(b,c,d)) \vee (U(a,b,c) \wedge T(c,d)) \leftarrow R(a,b) \wedge S(b,c) \wedge T(c,d). \quad (36)$$

By semijoin-reducing W against S, T, and semjoin-reducing U against R, S, we conclude that

$$(R(a,b) \wedge W(b,c,d)) \vee (U(a,b,c) \wedge T(c,d)) \equiv R(a,b) \wedge S(b,c) \wedge T(c,d).$$

Finally, we have a rewrite of the original body:

$$[R(a,b) \wedge W(b,c,d) \wedge a \le d] \vee [U(a,b,c) \wedge T(c,d) \wedge a \le d] \equiv R(a,b) \wedge S(b,c) \wedge T(c,d) \wedge a \le d. \quad (37)$$

By defining intermediate rules, we can compute Q from them:

$$Q_1() \leftarrow R(a,b) \wedge W(b,c,d) \wedge a \le d, \quad (38)$$
$$Q_2() \leftarrow U(a,b,c) \wedge T(c,d) \wedge a \le d, \quad (39)$$
$$Q() \leftarrow Q_1() \vee Q_2(). \quad (40)$$

The key point is that Q_1 and Q_2 are of the form (15), and thus they each can be answered in $\tilde{O}(N^{1.5})$-time (since $|U|, |W| \le N^{1.5}$). This implies that Q can be answered in $\tilde{O}(N^{1.5})$-time overall. □

The strategy outlined in the above example uses PANDA to evaluate an FAQ-AI query over the Boolean semiring. The resulting algorithm achieves a natural generalization of the submodular FAQ-width defined in (14):

Definition 3.10. Given an FAQ-AI query Q (3) over the Boolean semiring. The *relaxed submodular* FAQ-*width* of Q is defined by

$$\mathsf{smfw}_\ell(Q) := \max_{h \in \mathsf{ED}_{\not\emptyset} \cap \Gamma_n} \min_{(T,\chi) \in \mathsf{TD}_F^\ell} \max_{t \in V(T)} h(\chi(t)). \quad (41)$$

(Recall that the set of relaxed tree decompositions TD_F^ℓ was defined in Definition 3.3.)

Theorem 3.11. *Any* FAQ-AI *query* Q *of the form* (3) *on the Boolean semiring can be answered in time* $\tilde{O}(N^{\mathsf{smfw}_\ell(Q)} + |Q|)$.

Proof. As in the proof of Theorem 3.5, we first assume there are no free variables; the case when $F \neq \emptyset$ is a trivial extension. When $F = \emptyset$, the query (3) is written in Datalog as:

$$Q() \leftarrow \bigwedge_{K \in \mathcal{E}_s} R_K \wedge \bigwedge_{S \in \mathcal{E}_\ell} \left[\sum_{v \in S} \theta_v^S \le 0 \right]. \quad (42)$$

Here, we write only R_K instead of $R_K(\boldsymbol{x}_K)$ and θ_v^S instead of $\theta_v^S(x_v)$ to avoid clutter. It will be implicit throughout this proof that the subscript of a factor/function indicates its arguments. To answer query (42), we first rewrite the skeleton of the body into a disjunction over all relaxed tree decompositions:

$$\bigwedge_{K \in \mathcal{E}_s} R_K \equiv \bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} \bigwedge_{t \in V(T)} S_{\chi(t)}^{(T,\chi)}. \quad (43)$$

Note that the right-hand side of (43) is a *Boolean tensor decomposition* of the left-hand side. The idea of using Boolean tensor decomposition to speed up query evaluation was used in [3] in the context of queries with *dis*equalities. Assuming that we can compute the intermediate relations $S_{\chi(t)}^{(T,\chi)}$ efficiently satisfying (43), then (42) can be answered by answering for each $(T,\chi) \in \mathsf{TD}_\emptyset^\ell$ an intermediate query:

$$Q^{(T,\chi)}() \leftarrow \bigwedge_{t \in V(T)} S_{\chi(t)}^{(T,\chi)} \wedge \bigwedge_{S \in \mathcal{E}_\ell} \left[\sum_{v \in S} \theta_v^S \le 0 \right]. \quad (44)$$

The final answer Q is obtained by the trivial Datalog rule:

$$Q() \leftarrow \bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} Q^{(T,\chi)}(). \quad (45)$$

The key point to notice here is that each intermediate query (44) is an FAQ-AI query (3) with $\mathsf{faqw}_\ell = 1$, and thus from Theorem 3.5 each one of them can be answered in time $\tilde{O}(M)$ where

$$M = \max_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} \max_{t \in V(T)} |S_{\chi(t)}^{(T,\chi)}|. \quad (46)$$

It remains to compute a solution to (43); to do so, we apply distributivity of $\vee$ over $\wedge$ to rewrite the left-hand side as follows. Let $\mathcal{M}$ be the collection of *all* maps $\beta : \mathsf{TD}_\emptyset^\ell \to 2^{\mathcal{V}}$ such that $\beta(T,\chi) = \chi(t)$ for some $t \in V(T)$; in other words, β selects one bag $\chi(t)$ out of each tree decomposition (T,χ). Then, from the distributive law we have

$$\bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} \bigwedge_{t \in V(T)} S_{\chi(t)}^{(T,\chi)} \equiv \bigwedge_{\beta \in \mathcal{M}} \bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} S_{\beta(T,\chi)}^{(T,\chi)}, \quad (47)$$

which means to solve the relational equation (43) we can instead solve the equation

$$\bigwedge_{\beta \in \mathcal{M}} \bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} S_{\beta(T,\chi)}^{(T,\chi)} \equiv \bigwedge_{K \in \mathcal{E}_s} R_K. \quad (48)$$

We seek the equivalence by solving for each of the clauses on the left-hand side separately, because the left-hand side is a conjunction. In particular, we need to compute solutions to the following disjunctive Datalog rules, one for each $\beta \in \mathcal{M}$:

$$\bigvee_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} W_{\beta(T,\chi)}^{(T,\chi)} \leftarrow \bigwedge_{K \in \mathcal{E}_s} R_K. \quad (49)$$

Once we obtain the relations $W_{\beta(T,\chi)}^{(T,\chi)}$, we can semijoin-reduce them against the input relations, and define $S_{\beta(T,\chi)}^{(T,\chi)}$ to be the union of all the corresponding $W_{\beta(T,\chi)}^{(T,\chi)}$ over all β.

Finally, we evaluate each disjunctive Datalog rule (49) by running the PANDA algorithm, which computes the rule in time bounded by $\tilde{O}(N^{e(\beta)})$, where

$$e(\beta) = \max_{h \in \mathsf{ED}_{\not\emptyset} \cap \Gamma_n} \min_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} h(\beta(T,\chi)). \quad (50)$$

Maximizing over $\beta \in \mathcal{M}$, the runtime is bounded by $\tilde{O}(N^w)$, where

$$\begin{aligned} w &= \max_{\beta \in \mathcal{M}} e(\beta) && (51)\\ &= \max_{\beta \in \mathcal{M}} \max_{h \in \mathsf{ED}_{\not\emptyset} \cap \Gamma_n} \min_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} h(\beta(T,\chi)) && (52)\\ &= \max_{h \in \mathsf{ED}_{\not\emptyset} \cap \Gamma_n} \max_{\beta \in \mathcal{M}} \min_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} h(\beta(T,\chi)) && (53)\\ &= \max_{h \in \mathsf{ED}_{\not\emptyset} \cap \Gamma_n} \min_{(T,\chi) \in \mathsf{TD}_\emptyset^\ell} \max_{t \in V(T)} h(\chi(t)) = \mathsf{smfw}_\ell(Q). && (54) \end{aligned}$$

Equality (53) follows from the minimax lemma in [6]. Our reasoning above also shows that M from (46) is bounded by $N^{\mathsf{smfw}_\ell(Q)}$. □

3.3 Relaxed polymatroids

A key step in the proof of Theorem 3.11 is to find the Boolean tensor decomposition (43) of the product over R_K. In a non-Boolean semiring, this becomes a tensor decomposition on this semiring:

$$\bigotimes_{K\in\mathcal{E}_s} R_K = \bigoplus_{(T,\chi)\in\mathsf{TD}_F^\ell} \bigotimes_{t\in V(T)} S^{(T,\chi)}_{\chi(t)}. \tag{55}$$

In order to compute this tensor decomposition, we can still follow the script of the proof of Theorem 3.11, working on the parameter space of the input factors R_K; however, for the equality in (55) to hold (it is an identity over the value-space of the factors), we have to ensure the following property:

> For any $\boldsymbol{x}_{\mathcal{V}}$ s.t. $\bigotimes_{K\in\mathcal{E}_s} R_K(\boldsymbol{x}_K) \neq \mathbf{0}$, there is *exactly one* tree decomposition $(T,\chi)\in\mathsf{TD}_F^\ell$ for which
>
> $$\bigotimes_{t\in V(T)} S^{(T,\chi)}_{\chi(t)}(\boldsymbol{x}_{\chi(t)}) = \bigotimes_{K\in\mathcal{E}_s} R_K(\boldsymbol{x}_K), \tag{56}$$
>
> while for the other TDs, the left-hand side above is $\mathbf{0}$.

Essentially, the property ensures that we do not have to perform inclusion-exclusion (IE) over the tree decompositions in TD_F^ℓ.[6] We do not know how to ensure this property in general. However, under a relaxed notion of polymatroids, the property above holds. Since this idea applies to FAQ queries in general, we start with our result on FAQ queries first, before specializing it to FAQ-AI.

3.3.1 FAQ on non-Boolean semirings. To explain how we can guarantee the property (56) for an FAQ query over a non-Boolean semiring, consider the following example. Suppose that we would like to evaluate the (aggregate) query

$$Q() = \sum_{\boldsymbol{x}_{[4]}} R_{12}(x_1,x_2)R_{23}(x_2,x_3)R_{34}(x_3,x_4)R_{41}(x_4,x_1). \tag{57}$$

We write R_{ij} instead of $R_{ij}(x_i,x_j)$ for short. The factors R_{ij} are functions of two variables $R_{ij} : \mathsf{Dom}(X_i)\times\mathsf{Dom}(X_j)\to\mathbb{R}$, and they are represented by *ternary* relations in a database. Abusing notation we will also use R_{ij} to refer to its support, i.e., the binary relation over (X_i,X_j) such that $(x_i,x_j)\in R_{ij}$ iff $R_{ij}(x_i,x_j)\neq 0$.

There are only two non-trivial tree decompositions for the "4-cycle" query (57): one with bags $\{1,2,3\}$ and $\{3,4,1\}$, and the other with bags $\{1,2,4\}$ and $\{2,3,4\}$. To evaluate the query, we first solve the relation equation (55), but only on the supports; i.e., we would like to find relations $S_{123}, S_{341}, S_{234}$, and S_{412} such that

$$\begin{aligned} &R_{12}\wedge R_{23}\wedge R_{34}\wedge R_{41} \equiv (S_{123}\wedge S_{341})\vee(S_{234}\wedge S_{412}) \equiv \\ &(S_{123}\vee S_{234})\wedge(S_{123}\vee S_{412})\ \wedge(S_{341}\vee S_{234})\wedge(S_{341}\vee S_{412}). \end{aligned} \tag{58}$$

The second $\equiv$ is due to the distributivity of $\vee$ over $\wedge$. Since the last formula is in CNF, we can solve each term separately by solving 4 different disjunctive Datalog rules:

$$(S_{123}\vee S_{234}) \leftarrow R_{12}\wedge R_{23}\wedge R_{34}\wedge R_{41}, \tag{59}$$
$$(S_{123}\vee S_{412}) \leftarrow R_{12}\wedge R_{23}\wedge R_{34}\wedge R_{41}, \tag{60}$$
$$(S_{341}\vee S_{234}) \leftarrow R_{12}\wedge R_{23}\wedge R_{34}\wedge R_{41}, \tag{61}$$
$$(S_{341}\vee S_{412}) \leftarrow R_{12}\wedge R_{23}\wedge R_{34}\wedge R_{41}. \tag{62}$$

[6] IE is difficult for two reasons: (1) IE computation explodes the runtime, and (2) in a general semiring there may not be additive inverses and thus IE may not even apply.

Applying the proof-to-algorithm conversion idea from [6], the above disjunctive Datalog rules can be solved with the PANDA algorithm. It is beyond the scope of the main body of this paper to describe the PANDA algorithm in full details. However, we can describe a solution. Let $N = \max\{|R_{12}|,|R_{23}|,|R_{34}|,|R_{41}|\}$. For each input relation/factor, define their "light" parts as follows.

$$R^\ell_{ij} := \{(x_i,x_j)\in R_{ij} \ :\ |\sigma_{X_i=x_i}R_{ij}| \le \sqrt{N}\}. \tag{63}$$

Also, for every R_{ij}, define $R^h_{ij} := R_{ij}\setminus R^\ell_{ij}$. Then, one can verify that the following is a solution to the relational equations (59)…(62) (and by semijoin-reducing each one of them with relations R_{ij}, they become a solution to (58) as well):

$$S_{ijk} = R_{ij}\bowtie R^l_{jk}\cup\pi_i R^h_{ij}\bowtie R_{jk}. \tag{64}$$

Furthermore, it is straightforward to verify that each S_{ijk} can be computed in $\tilde{O}(N^{1.5})$-time. Once we have obtained this solution to the relational equation, i.e., we have the relations S_{ijk}, we can extend them naturally into factors (so that they are represented by 4-ary relations) satisfying (56). In particular, as functions with range $\mathbb{R}$, they are defined by

$$S_{ijk}(x_i,x_j,x_k) := R_{ij}(x_i,x_j)\cdot R_{jk}(x_j,x_k). \tag{65}$$

The above sketch does not work for a generic FAQ query because we do not know how to guarantee that (56) is satisfied given the relational solution returned by PANDA. (If we were able to do so, then the notion of submodular width would apply also to #CSP and not just CSP.) However, we *are* able to prove that this strategy works (i.e., (56) can be ensured) under a relaxed notion of polymatroids and a corresponding "sharp submodular (FAQ) width".

Definition 3.12. Given a collection $\mathcal{E}$ of subsets of $\mathcal{V}$, a set function $h : 2^{\mathcal{V}}\to\mathbb{R}_+$ is said to be a *$\mathcal{E}$-polymatroid* if it satisfies the following: (i) $h(\emptyset)=0$, (ii) $h(X)\le h(Y)$ whenever $X\subseteq Y$, and (iii) $h(X\cup Y)+h(X\cap Y)\le h(X)+h(Y)$ for every pair $X,Y\subseteq\mathcal{V}$ such that $X\cap Y\subseteq S$ for some $S\in\mathcal{E}$. In particular, a $2^{\mathcal{V}}$-polymatroid is a polymatroid as defined in Section 2.1. For $\mathcal{V}=[n]$, let $\Gamma_{n|\mathcal{E}}$ denote the set of all $\mathcal{E}$-polymatroids on $\mathcal{V}$.

The following definition is a straightforward generalization of smfw from (14), where we replace Γ_n by the relaxed polymatroids.

Definition 3.13. Given an FAQ query (2) whose hypergraph is $\mathcal{H}=(\mathcal{V},\mathcal{E}=\mathcal{E}_{\not\infty}\cup\mathcal{E}_\infty)$, its *#-submodular* FAQ-*width*, denoted by $\#\mathsf{smfw}(Q)$, is defined by

$$\#\mathsf{smfw}(Q) := \max_{h\in\mathsf{ED}_{\not\infty}\cap\Gamma_{n|\mathcal{E}_{\not\infty}}}\ \min_{(T,\chi)\in\mathsf{TD}_F}\ \max_{t\in V(T)} h(\chi(t)). \tag{66}$$

When there are no free variables, i.e., $F=\emptyset$, we define $\#\mathsf{subw}(Q) := \#\mathsf{smfw}(Q)$, to mirror the case when $\mathsf{faqw}(Q)=\mathsf{fhtw}(Q)$.

Under the above more restricted width parameter[7], our vision above with condition (56) can now be realized:

Theorem 3.14. *Any* FAQ *query Q of the form* (2) *on any semiring can be answered in time $\tilde{O}(N^{\#\mathsf{smfw}(Q)}+|Q|)$.*

[7] When we relax the polymatroids, the width goes *up*, and thus it is more restricted.

Appendix A gives the proof of Theorem 3.14, which involves an appropriate adaptation of PANDA called #PANDA. It also shows that while #smfw(Q) can be larger than smfw(Q), it is not larger than faqw(Q) and can be unboundedly smaller for classes of queries.

Proposition 3.15 (Connecting #smfw to both smfw and faqw)**.**

(a) *For any* FAQ *query Q, the following holds:*

$$\mathsf{smfw}(Q) \leq \#\mathsf{smfw}(Q) \leq \mathsf{faqw}(Q). \tag{67}$$

In particular, when Q has no free variables, we have

$$\mathsf{subw}(Q) \leq \#\mathsf{subw}(Q) \leq \mathsf{fhtw}(Q). \tag{68}$$

(b) *Furthermore, there are classes of queries Q for which the gap between* #smfw(Q) *and* faqw(Q) *is unbounded, and so is the gap between* #subw(Q) *and* fhtw(Q).

Example 3.16. Consider again the count query Q in (57), which we showed earlier how to compute in time $\tilde{O}(N^{1.5})$. Since Q has no free variables, faqw(Q) = fhtw(Q) = 2 and #smfw(Q) = #subw(Q). In the proof of Proposition 3.15, we show that #subw(Q) $\leq$ 1.5. Therefore, the #PANDA algorithm from the proof of Theorem 3.14 can compute (57) in time $\tilde{O}(N^{1.5})$. In fact, the $\tilde{O}(N^{1.5})$ algorithm we described earlier for (57) is just a specialization of #PANDA. The proof of Proposition 3.15 offers a family of similar examples. □

3.3.2 FAQ-AI *on non-Boolean semirings.* Finally, we put everything together to solve the FAQ-AI problem. The only (very natural) change is to replace the tree decompositions by their relaxed version, and the technical details flow through.

Definition 3.17. Given an FAQ-AI query (3) whose hypergraph is $\mathcal{H} = (\mathcal{V}, \mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_\ell = \mathcal{E}_{\not\infty} \cup \mathcal{E}_\infty)$, its *relaxed #-submodular* FAQ-*width*, denoted by #smfw$_\ell(Q)$, is defined by

$$\#\mathsf{smfw}_\ell(Q) := \max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_{n|\mathcal{E}_{\not\infty}}} \min_{(T,\chi)\in \mathsf{TD}^\ell_F} \max_{t \in V(T)} h(\chi(t)). \tag{69}$$

When $F = \emptyset$, we define #subw$_\ell(Q)$:= #smfw$_\ell(Q)$.

Theorem 3.18. *Any* FAQ-AI *query Q of the form* (3) *on any semiring can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)} + |Q|)$.*

Example 3.19. Consider the following count query:

$$Q() = \sum_{a,b,c,d} R(a, b) \cdot S(b, c) \cdot T(c, d) \cdot \mathbf{1}_{a+b+c+d \leq 0}. \tag{70}$$

Let $N := \max\{|R|, |S|, |T|\}$. For the above query faqw(Q) = faqw$_\ell(Q)$ = #smfw(Q) = 2. Any of the previously known algorithms, including the one from Theorem 3.5 *and* the one from Theorem 3.14, would need time $O(N^2)$ to compute Q. In Appendix A, we show that #smfw$_\ell(Q)$ $\leq$ 1.5. As an example of Theorem 3.18, we also show how to compute the above query in $\tilde{O}(N^{1.5})$. (Using the same method, we can also solve the counting version of Q_3 from Example 1.2 in the same time.) □

4 APPLICATIONS TO RELATIONAL MACHINE LEARNING

Our FAQ-AI formalism and solution are directly applicable to learning a class of machine learning models, which includes supervised models (e.g., robust regression, SVM classification), and unsupervised models (e.g., clustering via k-means). In this section, we show that the core computation of these optimization problems can be formulated in FAQ-AI over the sum-product semiring.

4.1 Training ML models over Databases

A typical machine learning model is learned over a training dataset $\boldsymbol{G}$. We consider the common scenario where the input data is a relational database $\boldsymbol{I}$, and the training dataset $\boldsymbol{G}$ is the result of a feature extraction join query Q over $\boldsymbol{I}$ [1, 2, 18, 23]. Each tuple $(\boldsymbol{x}, y) \in \boldsymbol{G}$ consists of a vector of features $\boldsymbol{x}$ and a label y. We consider that the feature extraction query Q has the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}_s)$, where $\mathcal{E}_s$ is the set of its skeleton hyperedges.

A supervised machine learning model is a function $f_{\boldsymbol{\beta}}(\boldsymbol{x})$ with parameters $\boldsymbol{\beta}$ that is used to predict the label y for unlabeled data. The parameters are obtained by minimizing the objective function:

$$J(\boldsymbol{\beta}) = \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} \mathcal{L}\left(y, f_{\boldsymbol{\beta}}(\boldsymbol{x})\right) + \lambda\Omega(\boldsymbol{\beta}), \tag{71}$$

where $\mathcal{L}(a, b)$ is a loss function, Ω is a regularizer, e.g., ℓ_1 or ℓ_2 norm, and the constant $\lambda \in (0, 1)$ controls the influence of regularization.

Previous work has shown that for polynomial loss functions, such as square loss $\mathcal{L}(a, b) = (a - b)^2$, the core computation for optimizing the objective $J(\boldsymbol{\beta})$ amounts to FAQ evaluation [2]. In many instances, however, the loss function is non-polynomial, either due to the structure of the loss, or the presence of non-polynomial components embedded within the model structure (e.g., ReLU activation function in neural nets) [27].

Examples of commonly used non-polynomial loss functions are: (1) hinge loss, used to learn classification models like linear support vector machines (SVM) [27], or generalized low rank models (glrm) with boolean principal component analysis (PCA) [35]; (2) Huber loss, used to learn regression models that are robust to outliers [27]; (3) scalene loss, used to learn quantile regression models [35]; (4) epsilon insensitive loss, used to learn SVM regression models [27]; and (5) ordinal hinge loss, used to learn ordinal regression models or ordinal PCA (another glrm) [35].

Any optimization problem with the above non-polynomial loss functions can benefit from our evaluation algorithm for FAQ-AI by reformulating computations in the optimization algorithm as FAQ-AI expressions over the feature extraction join query Q. We next exemplify this reformulation for two such problems: (1) learning a robust linear regression model using Huber loss, which can be solved with gradient-descent optimization, and (2) learning a linear support vector machine (SVM) for binary classification using hinge loss, which can be solved with subgradient-based optimization algorithms or with a cutting-plane algorithm for the primal formulation of linear SVM classification. Appendix B details the cases of the scalene, epsilon insensitive, and ordinal hinge loss functions.

We also consider the k-means unsupervised clustering algorithm and give an FAQ-AI reformulation of the computation done in an iteration of the algorithm over the dataset $\boldsymbol{G}$.

The advantage of FAQ-AI reformulation is that the FAQ-AI expressions for the aforementioned optimization problems can be evaluated over relaxed tree decompositions of the feature extraction query Q and do not require the explicit materialization of its result $\boldsymbol{G}$. The size of and time to compute G is $O(|I|^{\rho^*(Q)})$ [29]. The solution to these optimization problems can be computed in time sub-linear in the size of $\boldsymbol{G}$, using InsideOut or #PANDA.

4.2 Robust linear regression with Huber Loss

A linear regression model is a linear function $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x} = \sum_{i\in[n]} \beta_i x_i$ with features $\boldsymbol{x} = (x_1 = 1, x_2, \ldots, x_n)$ and parameters $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$. For a given feature vector $\boldsymbol{x}$, the model is used to estimate the (continuous) label $y \in \mathbb{R}$. We learn the model parameters by minimizing the objective $J(\boldsymbol{\beta})$ with the Huber loss function, which is defined as:

$$\mathcal{L}(a,b) = \begin{cases} \frac{1}{2}(a-b)^2 & \text{if } |a-b| \leq 1, \\ \frac{1}{2}|a-b| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{72}$$

Huber loss is equivalent to the square loss when $|a - b| \leq 1$ and to the absolute loss otherwise[8]. The advantage of Huber loss is that it is differentiable at all points (as opposed to the absolute loss), and more robust to outliers than the square loss.

To learn the parameters, we use batch gradient descent optimization, which repeatedly updates the parameters in the direction of the gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ until convergence. We provide details on gradient-based optimization in Appendix B.1. In this section, we focus on the core computation of the algorithm, which is the repeated computation of the objective $J(\boldsymbol{\beta})$ and its gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$.

The gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ is the vector of partial derivatives with respect to parameters $(\beta_j)_{j\in[n]}$. The objective function $J(\boldsymbol{\beta})$ (with ℓ_2 regularization) and its partial derivative with respect to β_j are:

$$\begin{aligned} J(\boldsymbol{\beta}) = \frac{1}{2} \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} & (y - f_{\boldsymbol{\beta}}(\boldsymbol{x}))^2 \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} \\ & + (|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| - 1) \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| > 1} + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2, \end{aligned} \tag{73}$$

$$\begin{aligned} \frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j} &= \lambda \beta_j + \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x})) \cdot x_j \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} \\ &\quad + \frac{1}{2}(x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0} - x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 0}) \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| > 1} \\ &= \lambda \beta_j + \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x})) \cdot x_j \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} \\ &\quad + 1/2 \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 1} - 1/2 \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) < -1}. \end{aligned} \tag{74}$$

Our observation is that we can compute $J(\boldsymbol{\beta})$ and $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ without materializing $\boldsymbol{G}$, by reformulating their data dependent computation as a few FAQ-AI expressions. We exemplify the rewriting for $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$; the rewriting for $J(\boldsymbol{\beta})$ is presented in Appendix B.3. The first of the three summations in $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ is rewritten as follows:

$$\begin{aligned} & \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} (y - \sum_{i\in[n]} \beta_i x_i) \cdot x_j \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} \\ &= \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} y \cdot x_j \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} - \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} \beta_i \cdot x_i \cdot x_j \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1} \\ &= \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} y \cdot x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) \leq 1} \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0} \\ &\quad + \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} y \cdot x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) \geq -1} \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 0} \\ &\quad - \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} \beta_i \cdot x_i \cdot x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) \leq 1} \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0} \\ &\quad - \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in \boldsymbol{G}} \beta_i \cdot x_i \cdot x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) \geq -1} \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 0}. \end{aligned} \tag{75}$$

The four terms can be expressed as $O(n)$ FAQ-AI expressions of the form (3). For instance, the first part of the expression is equivalent to the following FAQ-AI query:

$$Q() = \sum_{y, \boldsymbol{x}_{\mathcal{V}}} y \cdot x_j \cdot \underbrace{\mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) \leq 1} \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0}}_{\text{ligaments } \mathcal{E}_\ell} \cdot \left(\prod_{F\in \mathcal{E}_s} R_F(\boldsymbol{x}_F) \right).$$

The other two summations in $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ both aggregate over x_j and have one inequality that defines a ligament in $\mathcal{E}_\ell$. They can be expressed as FAQ-AI expressions. Overall, the gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ can be expressed as $O(n^2)$ FAQ-AI expressions. Appendix B.3 shows that the same holds for $J(\boldsymbol{\beta})$.

Theorem 4.1. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. For any robust linear regression model $\boldsymbol{\beta}^\top \boldsymbol{x}$, the objective $J(\boldsymbol{\beta})$ and gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ with Huber loss can be computed in time $\tilde{O}(N^{\#\text{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\text{faqw}_\ell(Q)} \log N)$ with* InsideOut.

4.3 Learning linear SVM classification models

A linear SVM classification model is used for binary classification problems where the label $y \in \{\pm 1\}$. For the features $\boldsymbol{x} = (x_1 = 1, x_2, \ldots, x_n)$, the model learns the parameters $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ of a linear discriminant function $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$ such that $f_{\boldsymbol{\beta}}(\boldsymbol{x})$ separates the data points in $\boldsymbol{G}$ into positive and negative classes with a maximum margin. The parameters can be learned by minimizing the objective function (71) with the hinge loss function:

$$\mathcal{L}(a,b) = \max\{0, 1 - a \cdot b\}. \tag{76}$$

Hinge loss is non-differentiable, and thus standard gradient descent optimization is not applicable. We next discuss two alternative approaches for solving this optimization problem.

The first approach is based on the observation that the loss function is convex, and the objective admits subgradient vectors, which generalize the standard notion of gradient. The optimization problem can be solved with subgradient based updates. Pegasos is a well-know algorithm for this approach [33].

The alternative approach is to solve the primal formulation of the problem, which avoids the non-differentiable objective by turning it into a constraint optimization problem with slack variables.

[8]Without loss of generality, we use a simplified Huber loss. The threshold between absolute and square loss is given by a constant δ and the absolute loss is $\frac{\delta}{2}|a-b| - \frac{\delta^2}{2}$.

Joachims proposed a cutting-plane algorithm which solves this optimization problem efficiently [21].

For both approaches, the number of iterations of the optimization algorithm is independent of the size $|\boldsymbol{G}|$ of training dataset $\boldsymbol{G}$ [21, 33]. Thus, the time complexity for finding the solution is $O(|\boldsymbol{G}|)$.

Despite the fact that the two approaches solve the same problem, they have been hugely influential in their own right. We therefore consider both approaches, and show that by reformulating their computation as FAQ-AI we can solve them asymptotically faster than materializing the training dataset $\boldsymbol{G}$, i.e., sublinear in $|\boldsymbol{G}|$.

4.3.1 Subgradient-based optimization for linear SVM classification. We first use subgradient-based optimization to compute the parameters of the SVM model; Appendix B.1 gives the details. The core of the optimization is the repeated computation of the objective and the partial derivatives in terms of $(\beta_j)_{j\in[n]}$. The objective $J(\boldsymbol{\beta})$ (with ℓ_2 regularization) and the partial derivative $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ are:

$$J(\boldsymbol{\beta}) = \sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \max\{0, 1 - y(\boldsymbol{\beta}^\top \boldsymbol{x})\} + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2, \tag{77}$$

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} y \cdot x_j \cdot \mathbf{1}_{y(\boldsymbol{\beta}^\top \boldsymbol{x})\le 1} + \lambda\beta_j. \tag{78}$$

Our observation is that $J(\boldsymbol{\beta})$ and $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ can be reformulated as FAQ-AI expressions and computed without materializing $\boldsymbol{G}$. We first rewrite the objective (derivation steps shown in Appendix B.5):

$$\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \max\{0, 1 - y(\boldsymbol{\beta}^\top \boldsymbol{x})\} + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 \tag{79}$$

$$= \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 + \underbrace{\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \mathbf{1}_{y=1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\le 1}}_{\text{FAQ-AI of the form (3)}} - \underbrace{\sum_{i=1}^{n}\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \beta_i x_i \mathbf{1}_{y=1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\le 1}}_{\text{FAQ-AI of the form (3)}}$$

$$+ \underbrace{\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \mathbf{1}_{y=-1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\ge -1}}_{\text{FAQ-AI of the form (3)}} + \underbrace{\sum_{i=1}^{n}\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \beta_i x_i \mathbf{1}_{y=-1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\ge -1}}_{\text{FAQ-AI of the form (3)}}.$$

Similarly, $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ can be rewritten into two FAQ-AI expressions:

$$\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} y \cdot x_j \cdot \mathbf{1}_{y(\boldsymbol{\beta}^\top \boldsymbol{x})\le 1} + \lambda\beta_j \tag{80}$$

$$= \lambda\beta_j + \underbrace{\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} x_j \cdot \mathbf{1}_{y=1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\le 1}}_{\text{FAQ-AI of the form (3)}} - \underbrace{\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} x_j \cdot \mathbf{1}_{y=-1}\mathbf{1}_{\boldsymbol{\beta}^\top\boldsymbol{x}\ge -1}}_{\text{FAQ-AI of the form (3)}}.$$

Theorem 4.2. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. For any linear SVM classification model $\boldsymbol{\beta}^\top\boldsymbol{x}$, the objective $J(\boldsymbol{\beta})$ and gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ with hinge loss can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)}\log N)$ with* InsideOut.

Algorithm 1: Training classification SVM via (82)

1 $\mathcal{W} \leftarrow \emptyset$; // Working set
2 $t \leftarrow 0$;
3 **repeat**
4 $\quad t \leftarrow t + 1$;
5 $\quad (\boldsymbol{\beta}^{(t)}, \xi^{(t)}) \leftarrow \arg\min_{\boldsymbol{\beta},\xi\ge 0}\left\{\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + C\xi\right.$;
6 $\quad\quad$ s.t. $\left.\frac{1}{|\boldsymbol{G}|}\left\langle \boldsymbol{\beta}, \sum_{(\boldsymbol{x},y)\in\boldsymbol{T}} y\boldsymbol{x}\right\rangle \ge \frac{|\boldsymbol{T}|}{|\boldsymbol{G}|} - \xi, \forall \boldsymbol{T}\in\mathcal{W}\right\}$;
7 $\quad \boldsymbol{T}^{(t)} := \{(\boldsymbol{x}, y) \in \boldsymbol{G} \mid y\left\langle \boldsymbol{\beta}^{(t)}, \boldsymbol{x}\right\rangle < 1\}$;
8 $\quad \mathcal{W} \leftarrow \mathcal{W} \cup \{\boldsymbol{T}^{(t)}\}$
9 **until** $\frac{|\boldsymbol{T}^{(t)}|}{|\boldsymbol{G}|} - \frac{1}{|\boldsymbol{G}|}\left\langle \boldsymbol{\beta}^{(t)}, \sum_{(\boldsymbol{x},y)\in\boldsymbol{T}^{(t)}} y\boldsymbol{x}\right\rangle \le \xi^{(t)} + \epsilon$;

4.3.2 Cutting-plane algorithm for linear SVM classification in primal space. An alternative to learning linear SVM via subgradient-based optimization is to pose the problem as a constraint optimization problem. The equivalent formulation for minimizing the objective (77) is the primal formulation of linear SVM [27]:

$$\min_{\boldsymbol{\beta},\xi_{\boldsymbol{x},y}\ge 0} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{|D|}\sum_{(\boldsymbol{x},y)\in\boldsymbol{G}} \xi_{\boldsymbol{x},y} \tag{81}$$
$$\text{s.t.} \quad y f_{\boldsymbol{\beta}}(\boldsymbol{x}) \ge 1 - \xi_{\boldsymbol{x},y}, \qquad \forall(\boldsymbol{x}, y) \in \boldsymbol{G}.$$

where $\xi_{\boldsymbol{x},y}$ are slack variables and C is the regularization parameter.

The optimization problem solves for the hyperplane $f_{\boldsymbol{\beta}}(\boldsymbol{x})$ that classifies the data points $(\boldsymbol{x}, y) \in \boldsymbol{G}$ into two classes, so that the margin between the hyperplane and the nearest data point for each class is maximized. For each $(\boldsymbol{x}, y) \in \boldsymbol{G}$, the slack variable $\xi_{\boldsymbol{x},y}$ encodes how much the point violates the margin of the hyperplane.

Joachims' cutting-plane algorithm solves (81) in linear time over the training dataset [21]. The algorithm solves the following *structural classification* SVM formulation, which is equivalent to (81):

$$\min_{\boldsymbol{\beta},\xi\ge 0} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\xi \tag{82}$$
$$\text{s.t.} \quad \frac{1}{|\boldsymbol{G}|}\left\langle \boldsymbol{\beta}, \sum_{(\boldsymbol{x},y)\in\boldsymbol{T}} y\boldsymbol{x}\right\rangle \ge \frac{1}{|\boldsymbol{G}|}|\boldsymbol{T}| - \xi, \qquad \forall \boldsymbol{T} \subseteq \boldsymbol{G}.$$

This formulation has $2^{|\boldsymbol{G}|}$ constraints, one for each possible subset $\boldsymbol{T} \subseteq \boldsymbol{G}$, and a single slack variable ξ that is shared by all constraints.

Algorithm 1 presents Joachims' cutting-plane algorithm for solving (82). It iteratively constructs a set of constraints $\mathcal{W}$, which is a subset of all constraints in (82). In each round t, it first computes the optimal value for $\boldsymbol{\beta}^{(t)}$ and $\xi^{(t)}$ over the current working set $\mathcal{W}$. Then, it identifies the constraint $\boldsymbol{T}^{(t)}$ that is most violated for the current $\boldsymbol{\beta}^{(t)}$, and adds this constraint to $\mathcal{W}$. It continues until $\boldsymbol{T}^{(t)}$ is violated by at most ϵ. Joachims showed that Algorithm 1 finds the ϵ-approximate solution to (82) in $O(1)$-many iterations [21]. Hence $|\mathcal{W}|$ and the number of constraints of the optimization problem are bounded by a number *independent* of $|\boldsymbol{G}|$.

Next, we consider the inner optimization problem at line 5. Although $|\mathcal{W}|$ is small, the number n of variables can still be large. This prohibits solving with quadratic programming as it can take up to $O(n^3)$ [27]. Its Wolfe dual, on the other hand, is a quadratic program with only a constant number of variables that is independent of n and one constraint. Let $\boldsymbol{x}_T = \sum_{(\boldsymbol{x},y)\in T} y\boldsymbol{x}$. We next present the derived Wolfe dual (its derivation from (82) is in Appendix B.8):

$$\max_{\boldsymbol{\alpha}\geq 0} \quad -\frac{1}{2}\left\langle \sum_{T\in\mathcal{W}} \alpha_T \boldsymbol{x}_T, \sum_{T\in\mathcal{W}} \alpha_T \boldsymbol{x}_T \right\rangle + \sum_{T\in\mathcal{W}} |T|\alpha_T \tag{83}$$

$$\text{s.t.} \quad \sum_{T\in\mathcal{W}} \alpha_T \leq \frac{C}{|\boldsymbol{G}|}$$

where $\boldsymbol{\alpha} = (\alpha_T)_{T\in\mathcal{W}}$ is the vector of constraints.

Theorem 4.3. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. A linear SVM classification model can be learned over the training dataset $Q(\boldsymbol{I})$ with Joachims' cutting-plane algorithm in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)}\log N)$ with* InsideOut.

4.4 k-means clustering

Next we consider the popular k-means clustering algorithm, which is an example of an unsupervised machine learning algorithm.

An unsupervised machine learning model is computed over a dataset $\boldsymbol{G} \subseteq \mathbb{R}^n$, for which each tuple $\boldsymbol{x} \in \boldsymbol{G}$ is a vector of features without a label. A clustering task aims to divide $\boldsymbol{G}$ into k clusters of "similar" data points with respect to the ℓ_2 norm: $\boldsymbol{G} = \cup_{i=1}^k \boldsymbol{G}_i$, where k is a given fixed positive integer. Each cluster $\boldsymbol{G}_i$ is represented by a cluster mean $\boldsymbol{\mu}_i \in \mathbb{R}^n$. One of the most ubiquitous clustering methods, Lloyd's k-means clustering algorithm (also known as the k-means method), involves the optimization problem (1) with respect to the partition $(\boldsymbol{G}_i)_{i\in[k]}$ and the k means $(\boldsymbol{\mu}_i)_{i\in[k]}$. Other norms or distance measures can be used, e.g., if we replace ℓ_2 with ℓ_1-norm, then we get the k-median problem. The subsequent development considers the ℓ_2-norm.

Lloyd's algorithm can be viewed as a special instantiation of the *Expectation-Maximization* (EM) algorithm. It iteratively computes two updating steps until convergence. First, it updates the cluster assignments for each $(\boldsymbol{G}_i)_{i\in[k]}$:

$$\boldsymbol{G}_i = \left\{ \boldsymbol{x} \in \boldsymbol{G} \mid \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \leq \left\|\boldsymbol{x} - \boldsymbol{\mu}_j\right\|^2, \forall j \in [k] \setminus \{i\} \right\} \tag{84}$$

and then it updates the corresponding k-means $(\boldsymbol{\mu}_i)_{i\in[k]}$:

$$\boldsymbol{\mu}_i = \frac{1}{|\boldsymbol{G}_i|} \sum_{\boldsymbol{x}\in\boldsymbol{G}_i} \boldsymbol{x}. \tag{85}$$

Our observation is that we can reformulate the updating of the k-means as FAQ-AI expressions, without explicitly computing the partitioning $(\boldsymbol{G}_i)_{i\in[k]}$. For a given set of k-means $(\boldsymbol{\mu}_j)_{j\in[k]}$, let $c_{ij}(\boldsymbol{x})$ be the following function:

$$\begin{aligned} c_{ij}(\boldsymbol{x}) &= \sum_{\ell\in[n]} [(x_\ell - \mu_{i,\ell})^2 - (x_\ell - \mu_{j,\ell})^2] \\ &= \sum_{\ell\in[n]} [\mu_{i,\ell}^2 - 2x_\ell(\mu_{i,\ell} + \mu_{j,\ell}) - \mu_{j,\ell}^2]. \end{aligned} \tag{86}$$

where $\mu_{j,\ell}$ is the ℓ'th component of mean vector $\boldsymbol{\mu}_j$. A data point $\boldsymbol{x} \in \boldsymbol{G}$ is closest to center $\boldsymbol{\mu}_i$ if and only if $c_{ij}(\boldsymbol{x}) \leq 0$ holds $\forall j \in [k]$.

We use this inequality to reformulate the mean vector $\boldsymbol{\mu}_i$ as $O(n)$ FAQ-AI expressions. First, we express $|\boldsymbol{G}_i|$ as:

$$Q_i() = \sum_{\boldsymbol{x}} \left(\prod_{j\in[k]} \mathbf{1}_{c_{ij}(\boldsymbol{x})\leq 0} \right) \left(\prod_{F\in\mathcal{E}_s} R_F(\boldsymbol{x}_F) \right). \tag{87}$$

Then, for each $\ell \in [n]$, the sum $\sum_{\boldsymbol{x}\in\boldsymbol{G}_i} x_\ell$ can be reformulated in FAQ-AI as follows (similarly to (4)):

$$Q_{i\ell}() = \sum_{\boldsymbol{x}} x_\ell \left(\prod_{j\in[k]} \mathbf{1}_{c_{ij}(\boldsymbol{x})\leq 0} \right) \left(\prod_{F\in\mathcal{E}_s} R_F(\boldsymbol{x}_F) \right). \tag{88}$$

Each component $(\mu_{i,\ell})_{\ell\in[n]}$ is equal to the division of $Q_{i\ell}$ by Q_i.

Overall, the mean vector $\boldsymbol{\mu}_i$ can be computed with $O(n)$ FAQ-AI expressions of the form (3).

Theorem 4.4. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query where n is the number of its variables. Each iteration of Lloyd's k-means algorithm can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)}\log^{k-1} N)$ with* InsideOut .

REFERENCES

[1] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. AC/DC: In-Database Learning Thunderstruck. In *2nd Workshop on Data Mgt for End-To-End ML (DEEM'18)*. 8:1–8:10.
[2] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. In-Database Learning with Sparse Tensors. In *PODS*. 325–340.
[3] Mahmoud Abo Khamis, Hung Q. Ngo, Dan Olteanu, and Dan Suciu. 2019. Boolean Tensor Decomposition for Conjunctive Queries with Negation. In *ICDT*.
[4] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *PODS*. 13–28.
[5] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2017. Juggling Functions Inside a Database. *SIGMOD Rec.* 46, 1 (2017), 6–13.
[6] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?. In *PODS*. 429–444.
[7] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*. 208–222.
[8] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. 2011. Orthogonal Range Searching on the RAM, Revisited. In *SoCG*. 1–10.
[9] Bernard Chazelle. 1988. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17, 3 (1988), 427–462.
[10] Bernard Chazelle. 1990. Lower bounds for orthogonal range searching. II. The arithmetic model. *J. ACM* 37, 3 (1990), 439–463.
[11] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational geometry* (third ed.). Springer-Verlag, Berlin. xii+386 pages. Algorithms and applications.
[12] Lukasz Golab and M. Tamer Özsu. 2010. *Data Stream Management.* Morgan & Claypool Publishers.
[13] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *PODS*. 57–74.
[14] Georg Gottlob, Christoph Koch, and Reinhard Pichler. 2002. Efficient Algorithms for Processing XPath Queries. In *VLDB*. 95–106.
[15] Martin Grohe and Dániel Marx. 2014. Constraint Solving via Fractional Edge Covers. *ACM Trans. Alg.* 11, 1 (2014), 4.
[16] Torsten Grust. 2002. Accelerating XPath Location Steps. In *SIGMOD*. 109–120.
[17] Terry Halpin and Spencer Rugaber. 2015. *LogiQL: A Query Language for Smart Databases.* CRC Press.
[18] Joseph M. Hellerstein et al. 2012. The MADlib Analytics Library or MAD Skills, the SQL. *PVLDB* 5, 12 (2012), 1700–1711.
[19] N. Ishakbeyoglu and Z. Ozsoyoglu. 1997. Testing satisfiability of a conjunction of inequalities. In *Int. Symposium on Computer and Inf. Syst. (ISCIS XII)*. 148–154.
[20] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31, 8 (2010), 651–666.
[21] Thorsten Joachims. 2006. Training Linear SVMs in Linear Time. In *SIGKDD*. 217–226.
[22] Anthony C. Klug. 1988. On conjunctive queries containing inequalities. *J. ACM* 35, 1 (1988), 146–160.
[23] Arun Kumar, Jeffrey F. Naughton, and Jignesh M. Patel. 2015. Learning Generalized Linear Models Over Normalized Data. In *SIGMOD*. 1969–1984.
[24] Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. 2016. Deterministic Time-Space Trade-Offs for k-SUM. In *ICALP*. 58:1–58:14.
[25] Nikos Mamoulis. 2011. *Spatial Data Management.* Morgan & Claypool Publishers.
[26] Dániel Marx. 2013. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM* 60, 6 (2013), Art. 42, 51.
[27] Kevin P. Murphy. 2013. *Machine learning : a probabilistic perspective.* MIT Press, Cambridge, Mass.
[28] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *PODS*. 111–124.
[29] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case optimal join algorithms: [extended abstract]. In *PODS*. 37–48.
[30] Dan Olteanu and Jiewen Huang. 2009. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*. 389–402.
[31] Mihai Patrascu and Ryan Williams. 2010. On the Possibility of Faster SAT Algorithms. In *SODA*. 1065–1075.
[32] Luc Segoufin. 2013. Enumerating with Constant Delay the Answers to a Query. In *ICDT*. 10–20.
[33] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming* 127, 1 (01 Mar 2011), 3–30.
[34] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases.* Morgan & Claypool Publishers.
[35] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. 2016. Generalized Low Rank Models. *Foundations and Trends in Machine Learning* 9, 1 (2016), 1–118.
[36] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *ICDT*. 96–106.

A MISSING DETAILS FROM SECTION 3

A.1 Proof of Proposition 3.7

Proposition 3.7. *For any positive integer m, there exists an* FAQ-AI *query of the form* (3) *for which* $F = \emptyset$, $\mathsf{faqw}_\ell(Q) \geq m$ *and it cannot be answered in time* $o(N^{\mathsf{faqw}_\ell(Q)})$*, modulo* k-sum *hardness.*

Proof. It is widely assumed [24, 31] that $O(N^{\lceil k/2 \rceil})$ is the best runtime for k-sum, which is the following problem: given k number sets $R_1, \ldots, R_k$ of maximum size N, determine whether there is a tuple $\boldsymbol{t} \in R_1 \times \cdots \times R_k$ such that $\sum_{i \in [k]} t_i = 0$. We can reduce k-sum to our problem: Consider the query Q over the Boolean semiring:

$$Q() \leftarrow \left(\bigwedge_{i \in [k]} R_i(x_i) \right) \wedge \left(\sum_{i \in [k]} x_i \leq 0 \right) \wedge \left(\sum_{i \in [k]} x_i \geq 0 \right). \tag{89}$$

The answer to Q is true iff there is a tuple $(x_1, \ldots, x_k) \in R_1 \times \cdots \times R_k$ such that $\sum_{i \in [k]} x_i = 0$. The reduction shows that our query (89) is k-sum-hard. For this query, $\mathsf{faqw}_\ell(Q) = \lceil k/2 \rceil$. □

A.2 Proof of Proposition 3.8

Proposition 3.8 *For any* FAQ-AI *query* Q *of the form* (3)*, we have* $\mathsf{faqw}_\ell(Q) \geq \frac{1}{2}\mathsf{faqw}(Q)$*; in particular, when* Q *has no free variables* $\mathsf{fhtw}_\ell(Q) \geq \frac{1}{2}\mathsf{fhtw}(Q)$.

Proof. Let (T, χ) denote a relaxed tree decomposition of $\mathcal{H}$ with fractional hypertree width $\mathsf{faqw}_\ell(\mathcal{H})$. Construct a new (non-relaxed) tree decomposition (T', χ') for $\mathcal{H}$ as follows. Each vertex t in $V(T)$ is also a vertex in $V(T')$ with $\chi'(t) = \chi(t)$. Moreover, to each edge $\{s, t\} \in E(T)$ there corresponds an additional vertex st in $V(T')$ whose bag is $\chi'(st) = \chi(s) \cup \chi(t)$. As for the edge set of T', for each edge $\{s, t\} \in E(T)$, there are two corresponding edges in $E(T')$, namely $\{s, st\}$ and $\{t, st\}$. It is easy to see that (T', χ') is a (non-relaxed) tree decomposition of $\mathcal{H}$ with width at most $2\mathsf{faqw}(\mathcal{H})$. Moreover, if (T, χ) is F-connex, then so is (T', χ'). □

A.3 Proof of Proposition 3.15

Proposition 3.15.

(a) *For any* FAQ *query* Q*, the following holds:*

$$\mathsf{smfw}(Q) \leq \#\mathsf{smfw}(Q) \leq \mathsf{faqw}(Q). \tag{90}$$

In particular, when Q *has no free variables, we have*

$$\mathsf{subw}(Q) \leq \#\mathsf{subw}(Q) \leq \mathsf{fhtw}(Q). \tag{91}$$

(b) *Furthermore, there are classes of queries* Q *for which the gap between* $\#\mathsf{smfw}(Q)$ *and* $\mathsf{faqw}(Q)$ *is unbounded, and so is the gap between* $\#\mathsf{subw}(Q)$ *and* $\mathsf{fhtw}(Q)$.

Proof. First we prove part (a). The first inequality in (90) follows directly from the definitions of #smfw and smfw along with the fact that $\Gamma_n \subseteq \Gamma_{n|\mathcal{E}_{\not\infty}}$. To prove the second inequality in (90), we use the following variant of the *Modularization Lemma* from [6]:

Claim 1 (Variant of the Modularization Lemma [6]). *Given a hypergraph $\mathcal{H} = (\mathcal{V} = [n], \mathcal{E})$ and a set $B \subseteq \mathcal{V}$, we have*

$$\max_{h \in \mathsf{ED} \cap \Gamma_{n|\mathcal{E}}} h(B) = \max_{h \in \mathsf{ED} \cap \mathsf{M}_n} h(B), \tag{92}$$

where ED *is given by* (6) *and* M_n *denotes the set of all modular functions* $h : 2^{\mathcal{V}} \to \mathbb{R}_+$. *(A function* $h : 2^{\mathcal{V}} \to \mathbb{R}_+$ *is modular if* $h(X) = \sum_{i \in X} h(i), \forall X \subseteq \mathcal{V}$.*)*

Proof of Claim 1. Obviously, the LHS of (92) is lowerbounded by the RHS. Next, we prove LHS ≤ RHS. WLOG we assume $B = [k]$ for some $k \in [n]$. Let $h^* = \arg\max_{h \in \mathsf{ED} \cap \Gamma_{n|\mathcal{E}}} h(B)$. Define a function $\bar{h} : 2^{\mathcal{V}} \to \mathbb{R}_+$ as follows:

$$\bar{h}(F) = \sum_{i \in F} (h^*([i]) - h^*([i-1])).$$

Obviously $\bar{h} \in \mathsf{M}_n$ and $\bar{h}(B) = h^*(B)$. Next, we prove $\bar{h} \in \mathsf{ED}$ by proving that for every $F \subseteq [n]$ where $F \subseteq E$ for some $E \in \mathcal{E}$, the following holds:

$$\bar{h}(F) \leq h^*(F).$$

The proof is by induction on $|F|$. The base case when $|F| = 0$ is trivial. For the inductive step, consider some F where $F \subseteq E$ for some $E \in \mathcal{E}$. Let j be the maximum integer in F, then by noting that $|F \cap [j-1]| < |F|$, we have

$$\begin{aligned}
\bar{h}(F) &= h^*([j]) - h^*([j-1]) + \sum_{i \in F - \{j\}} (h^*([i]) - h^*([i-1])) \\
&= h^*([j]) - h^*([j-1]) + \bar{h}(F \cap [j-1]) \\
&= h^*(F \cup [j-1]) - h^*([j-1]) + \bar{h}(F \cap [j-1]) \\
&\leq h^*(F \cup [j-1]) - h^*([j-1]) + h^*(F \cap [j-1]) \\
&\leq h^*(F).
\end{aligned}$$

The first inequality above is by induction hypothesis, and the second inequality follows from the fact that h^* is a $\mathcal{E}$-polymatroid (recall Definition 3.12). Both steps rely on the fact that $F \cap [j-1] \subseteq E$ for some $E \in \mathcal{E}$. Consequently, $\bar{h} \in \mathsf{ED} \cap \mathsf{M}_n$. Since $\bar{h}(B) = h^*(B)$, this proves Claim 1. □

Now we prove the second inequality in (90):

$$\begin{aligned}
\#\mathsf{smfw}(Q) &= \max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_{n|\mathcal{E}_{\not\infty}}} \min_{(T,\chi) \in \mathsf{TD}_F} \max_{t \in V(T)} h(\chi(t)) \\
\text{(Max-min inequality)} &\leq \min_{(T,\chi) \in \mathsf{TD}_F} \max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_{n|\mathcal{E}_{\not\infty}}} \max_{t \in V(T)} h(\chi(t)) \\
&= \min_{(T,\chi) \in \mathsf{TD}_F} \max_{t \in V(T)} \max_{h \in \mathsf{ED}_{\not\infty} \cap \Gamma_{n|\mathcal{E}_{\not\infty}}} h(\chi(t)) \\
\text{(Claim 1)} &= \min_{(T,\chi) \in \mathsf{TD}_F} \max_{t \in V(T)} \max_{h \in \mathsf{ED}_{\not\infty} \cap \mathsf{M}_n} h(\chi(t)) \\
\text{(Strong duality in LP)} &= \min_{(T,\chi) \in \mathsf{TD}_F} \max_{t \in V(T)} \rho^*_{\mathcal{E}_{\not\infty}}(\chi(t)) \\
&= \mathsf{faqw}(Q).
\end{aligned}$$

The fact that $\max_{h \in \mathsf{ED}_{\not\infty} \cap \mathsf{M}_n} h(\chi(t)) = \rho^*_{\mathcal{E}_{\not\infty}}(\chi(t))$ follows from the two sides being dual linear programs. (Recall the definition of ρ^* from Section 2.1.)

Now, we prove part (b) of Proposition A.3. In [6], we constructed a class of graphs/queries where the gap between fhtw and subw is unbounded. We will re-use the same construction here and prove that the upperbound on subw that we proved in [6] is also an upperbound on #subw. The upperbound proof is going to be different from [6] though since here we can only use $\mathcal{E}$-polymatroid properties to prove the bound (recall Definition 3.12).

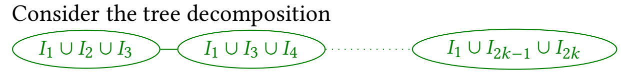
Given integers m and k, consider a graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ which is an "m-fold $2k$-cycle": The vertex set $\mathcal{V} := I_1 \cup I_2 \cup \ldots \cup I_{2k}$ is a disjoint union of $2k$-sets of vertices. Each set I_j has m vertices in it, namely $I_j := \{I_j^1, I_j^2, \ldots, I_j^m\}$. There is no edge between any two vertices within the set I_j for every $j \in [2k]$, namely I_j is an independent set. The edge set $\mathcal{E}$ of the hypergraph is the union of $2k$ complete bipartite graphs $K_{m,m}$:

$$\mathcal{E} := (I_1 \times I_2) \cup (I_2 \times I_3) \cup \cdots \cup (I_{2k-1} \times I_{2k}) \cup (I_{2k} \times I_1).$$

Finally consider an FAQ query Q that has a finite-sized input factor R_K for every $K \in \mathcal{E}$, i.e. $\mathcal{E}_{\not\infty} = \mathcal{E}$ and $\mathcal{E}_\infty = \emptyset$. (Recall notation from Section 2.2.) Moreover, assume Q has no free variables, hence $\mathsf{faqw}(Q) = \mathsf{fhtw}(Q)$ and $\#\mathsf{smfw}(Q) = \#\mathsf{subw}(Q)$.

We proved in [6] that $\mathsf{fhtw}(Q) \geq 2m$. Next we prove that $\#\mathsf{subw}(Q) \leq m(2 - 1/k)$. Let h be any function in $\mathsf{ED}_{\not\infty} \cap \Gamma_{n|\mathcal{E}_{\not\infty}}$. We recognize two cases:

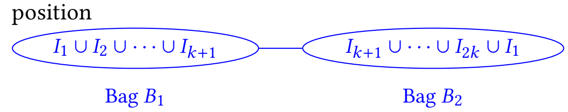
- Case 1: $h(I_i) \leq \theta$ for some $i \in [2k]$. WLOG assume $h(I_1) \leq \theta$. Consider the tree decomposition



For bag $B = I_1 \cup I_i \cup I_{i+1}$, using $\mathcal{E}_{\not\infty}$-polymatroid properties (Definition 3.12), we have

$$\begin{aligned}
h(B) &\leq h(I_1) + h(I_i \cup I_{i+1}) \\
&\leq h(I_1) + \sum_{j=1}^{m} h\left(\left\{I_i^j, I_{i+1}^j\right\}\right) \\
&\leq \theta + m.
\end{aligned}$$

- Case 2: $h(I_i) > \theta$ for all $i \in [2k]$. Consider the tree decomposition



For convenience, given any vertex I_i^j, define the vertex set $\mathcal{V}_i^j$ as follows:

$$\mathcal{V}_i^j := I_1 \cup I_2 \cup \ldots \cup I_{i-1} \cup \left\{I_i^1, I_i^2, \ldots, I_i^{j-1}\right\}.$$

From $\mathcal{E}_{\not\infty}$-polymatroid properties, we have

$$\begin{aligned}
h(B_1) &= h(I_1 \cup I_2) + \sum_{i=3}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_i^j\right\} \cup \mathcal{V}_i^j \mid \mathcal{V}_i^j\right) \\
&\leq h(I_1 \cup I_2) + \sum_{i=3}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_i^j, I_{i-1}^j\right\} \mid \left\{I_{i-1}^j\right\}\right) \\
&= h(I_1 \cup I_2) + \sum_{i=3}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_i^j, I_{i-1}^j\right\}\right) - \sum_{i=3}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_{i-1}^j\right\}\right) \\
&\leq h(I_1 \cup I_2) + \sum_{i=3}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_i^j, I_{i-1}^j\right\}\right) - \sum_{i=3}^{k+1} h(I_{i-1}) \\
&\leq \sum_{i=2}^{k+1} \sum_{j=1}^{m} h\left(\left\{I_i^j, I_{i-1}^j\right\}\right) - \sum_{i=3}^{k+1} h(I_{i-1}) \\
&\leq km - (k-1)\theta.
\end{aligned}$$

In a symmetric way, we can also show that $h(B_2) \leq km - (k-1)\theta$. By setting $\theta = (1 - 1/k)m$, we prove that $\#\text{subw}(Q) \leq m(2 - 1/k)$. Since $\text{fhtw}(Q) \geq 2m$, this proves part (b) of the proposition. □

A.4 Proof of Theorem 3.14

Theorem 3.14. *Any* FAQ *query Q of the form* (2) *on any semiring can be answered in time* $\tilde{O}(N^{\#\text{smfw}(Q)} + |Q|)$.

Proof. The PANDA algorithm [6] takes as input a disjunctive Datalog query of the form

$$\bigvee_{B \in \mathcal{B}} G_B(\boldsymbol{x}_B) \leftarrow \bigwedge_{K \in \mathcal{E}} R_K(\boldsymbol{x}_K). \tag{93}$$

The above query has an input relation R_K corresponding to each hyperedge $K \in \mathcal{E}$ in the query's hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. The output to the above query is a collection of tables G_B, one for each "goal" (or "target") B in the collection of goals $\mathcal{B}$. The output tables $(G_B)_{B \in \mathcal{B}}$ must satisfy the logical implication in (93): In particular, for each tuple $\boldsymbol{x}_{\mathcal{V}}$ that satisfies the conjunction $\bigwedge_{K \in \mathcal{E}} R_K(\boldsymbol{x}_K)$, the disjunction $\bigvee_{B \in \mathcal{B}} G_B(\boldsymbol{x}_B)$ must hold. Query (35) is an example of (93). A disjunctive Datalog query (93) can have many valid outputs. The PANDA algorithm computes one such output in time $\tilde{O}(N^e)$, where

$$e = \max_{h \in \text{ED}_{\not\sim} \cap \Gamma_n} \min_{B \in \mathcal{B}} h(B). \tag{94}$$

(Recall notation from Section 2.2.)

In what follows, we describe a variant of PANDA, called #PANDA, that takes a disjunctive Datalog query (93), and computes the following:

- A collection of tables $(G_B)_{B \in \mathcal{B}}$ that form a valid output to query (93), i.e. that satisfy the logical implication in (93).
- Moreover, associated with each output table G_B, #PANDA additionally computes a collection of "filter" tables $\left(F_K^{(B)}\right)_{K \in \mathcal{E}}$, one table $F_K^{(B)}$ for each hyperedge $K \in \mathcal{E}$ in the input hypergraph $\mathcal{H}$. The output tables G_B along with the associated filters $\left(F_K^{(B)}\right)_{K \in \mathcal{E}}$ satisfy the following condition: For each tuple $\boldsymbol{x}_{\mathcal{V}}$ that satisfies the conjunction $\bigwedge_{K \in \mathcal{E}} R_K(\boldsymbol{x}_K)$, there is *exactly one* target $B \in \mathcal{B}$ where the conjunction $\bigwedge_{K \in \mathcal{E}} F_K^{(B)}(\boldsymbol{x}_K)$ holds, and for that target B, $G_B(\boldsymbol{x}_B)$ holds as well. In particular, the following equivalences hold:

$$\veebar_{B \in \mathcal{B}} \left[\bigwedge_{K \in \mathcal{E}} F_K^{(B)}(\boldsymbol{x}_K)\right] \equiv \bigwedge_{K \in \mathcal{E}} R_K(\boldsymbol{x}_K), \tag{95}$$

$$\left[\bigwedge_{K \in \mathcal{E}} F_K^{(B)}(\boldsymbol{x}_K)\right] \equiv \left[G_B(\boldsymbol{x}_B) \wedge \bigwedge_{K \in \mathcal{E}} F_K^{(B)}(\boldsymbol{x}_K)\right], \quad \forall B \in \mathcal{B}, \tag{96}$$

where $\veebar$ above denotes the exclusive OR.

#PANDA computes the above output tables $(G_B)_{B \in \mathcal{B}}$ and $\left(\left(F_K^{(B)}\right)_{K \in \mathcal{E}}\right)_{B \in \mathcal{B}}$ in time $\tilde{O}(N^{e'})$ where

$$e' = \max_{h \in \text{ED}_{\not\sim} \cap \Gamma_{n|\mathcal{E}_{\not\sim}}} \min_{B \in \mathcal{B}} h(B). \tag{97}$$

Now we briefly explain how to tweak the PANDA algorithm into #PANDA satisfying the above characteristics. We refer the reader to [6] for more details about PANDA. At a high level, the PANDA algorithm starts with proving an exact upperbound on e from (94) using a sequence of proof steps, called the *proof sequence*. Then PANDA interprets each step in the proof sequence as a relational operator, and then uses this sequence of relational operators as a query plan to actually compute the query in time $\tilde{O}(N^e)$. One of the proof steps used in PANDA is *the decomposition step* $h(Y) \to h(X) + h(Y|X)$ for some $X \subseteq Y \subseteq \mathcal{V}$. The relational operator corresponding to this decomposition step is the "partitioning" operator, in which we take an input (or intermediate) table R_Y and partition it into a small number $k = O(\log |R|)$ of tables $R_Y^{(1)}, \ldots, R_Y^{(k)}$, based on the degrees of variables in Y with respect to variables in $X \subseteq Y$. In particular, define the degree of Y w.r.t. a tuple $\boldsymbol{t}_X \in \pi_X R_Y$ as

$$\deg_{R_Y}(Y|\boldsymbol{t}_X) := \left|\left\{\boldsymbol{t}'_Y \in R_Y \mid \boldsymbol{t}'_X = \boldsymbol{t}_X\right\}\right| \tag{98}$$

In the partitioning step, we partition tuples $\boldsymbol{t}_X \in \pi_X R_Y$ into k buckets based on $\deg_{R_Y}(Y|\boldsymbol{t}_X)$ and partition R_Y accordingly. After partitioning, PANDA creates k independent branches of the problem, where in the j-th branch, R_Y is replaced by $R_Y^{(j)}$, for each $j \in [k]$. PANDA continues on each branch independently and end up computing a target G_B for some $B \in \mathcal{B}$ that is potentially different for each branch.

From the proof sequence construction described in [6], we note the following: If the constructed proof sequence that is used to prove the bound on e in (94) contains a decomposition step $h(Y) \to h(X) + h(Y|X)$, then the proof of the bound on e must have relied on some submodularity constraint on h of the form $h(X) + h(Z \cup Y) \leq h(Y) + h(Z \cup X)$ for some $Z \subseteq \mathcal{V}$ where $Z \cap Y = \emptyset$. However, the new bound (97) used in #PANDA only relies on submodularities $h(X) + h(Z \cup Y) \leq h(Y) + h(Z \cup X)$ where $X \subseteq K$ for some $K \in \mathcal{E}$. (Recall $\Gamma_{n|\mathcal{E}_{\not\sim}}$ from Definition 3.12.) Therefore, in #PANDA, whenever we apply a partitioning step of R_Y into $R_Y^{(1)}, \ldots, R_Y^{(k)}$ based on the degrees $\deg_{R_Y}(Y|\boldsymbol{t}_X)$ of $\boldsymbol{t}_X \in \pi_X R_Y$, we can add $\pi_X R_Y^{(j)}$ into the filter $F_K^{(B)}$ for some $K \in \mathcal{E}$, i.e. we can set $F_K^{(B)} \leftarrow F_K^{(B)} \ltimes \pi_X R_Y^{(j)}$ on the j-th branch. Semijoin-reducing $\pi_X R_Y^{(j)}$ into some $F_K^{(B)}$ is possible thanks to the fact that $X \subseteq K$ for some $K \in \mathcal{E}$. Moreover, this semijoin-reduction of filters $F_K^{(B)}$ maintains (95). (Initially, we start with filters $F_K^{(B)}$ that are identical to the input relations R_K, which trivially satisfies (95).)

Now that we have described the #PANDA algorithm satisfying the above properties, we explain how to use it as a blackbox to solve an FAQ query Q of the form (2) in time $\tilde{O}(N^{\#\text{smfw}(Q)} + |Q|)$. Following the same notation as in the proof of Theorem 3.11, let $\mathcal{M}$ be the collection of *all* maps $\beta : \text{TD}_F^{\ell} \to 2^{\mathcal{V}}$ such that $\beta(T, \chi) = \chi(t)$ for some $t \in V(T)$; in other words, β selects one bag $\chi(t)$ out of each tree decomposition (T, χ). For each $\beta \in \mathcal{M}$, we use #PANDA to solve the following rule:

$$\veebar_{(T,\chi) \in \text{TD}_F} \left[G_{\beta(T,\chi)} \wedge \bigwedge_{K \in \mathcal{E}} F_K^{(\beta(T,\chi))}\right] \equiv \bigwedge_{K \in \mathcal{E}} R_K. \tag{99}$$

The solutions collectively satisfy the following:

$$\bigwedge_{\beta \in \mathcal{M}} \veebar_{(T,\chi) \in \text{TD}_F} \left[G_{\beta(T,\chi)} \wedge \bigwedge_{K \in \mathcal{E}} F_K^{(\beta(T,\chi))}\right] \equiv \bigwedge_{K \in \mathcal{E}} R_K.$$

Now we distribute the outer conjunction $\bigwedge_{\beta \in \mathcal{M}}$ over the exclusive OR $\veebar$, which results in an exclusive OR outside and a big conjunction inside. Using the same diagonalization argument from [6], we know that for this inner conjunction there must exist some tree decomposition $(\bar{T}, \bar{\chi}) \in \mathsf{TD}_F$ where the conjunction contains $G_{\bar{\chi}(t)}$ for all $t \in V(\bar{T})$. Thanks to (96), we can keep those terms $G_{\bar{\chi}(t)}$ in the conjunction and drop out all other terms $G_{\beta(T,\chi)}$ to get an equivalent conjunction. We interpret the resulting conjunction as an FAQ query: The input factors to this FAQ query are all filter $F_K^{(\beta(T,\chi))}$ in the conjunction along with $G_{\bar{\chi}(t)}$ for all $t \in V(\bar{T})$; all other $G_{\beta(T,\chi)}$ have been dropped. Now we solve this FAQ query by running InsideOut over the tree decomposition $(\bar{T}, \bar{\chi})$. We repeat the above for every conjunction. Afterwards, because different conjunctions are joined together with an *exclusive* OR, we can simply add up individual query results.

From (97), the total runtime is $\tilde{O}(N^w + |Q|)$, where

$$w = \max_{\beta \in \mathcal{M}} \max_{h \in \mathsf{ED}_{\not\perp} \cap \Gamma_{n|\mathcal{E}_{\not\perp}}} \min_{(T,\chi) \in \mathsf{TD}_F} h(\beta(T,\chi)) \tag{100}$$

$$= \max_{h \in \mathsf{ED}_{\not\perp} \cap \Gamma_{n|\mathcal{E}_{\not\perp}}} \max_{\beta \in \mathcal{M}} \min_{(T,\chi) \in \mathsf{TD}_F} h(\beta(T,\chi)) \tag{101}$$

$$= \max_{h \in \mathsf{ED}_{\not\perp} \cap \Gamma_{n|\mathcal{E}_{\not\perp}}} \min_{(T,\chi) \in \mathsf{TD}_F} \max_{t \in V(T)} h(\chi(t)) \tag{102}$$

$$= \#\mathsf{smfw}(Q). \tag{103}$$

□

A.5 Proof of Theorem 3.18

Theorem 3.18. *Any* FAQ-AI *query Q of the form* (3) *on any semiring can be computed in time* $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)} + |Q|)$.

The proof is very similar to that of Theorem 3.14. The key difference is that instead of running InsideOut on individual FAQ queries obtained after applying #PANDA, we now run the InsideOut variant from Theorem 3.5. The proof is thus omitted.

A.6 More details on Example 3.19

Consider the count query from Example 3.19:

$$Q() = \sum_{a,b,c,d} R(a,b) \cdot S(b,c) \cdot T(c,d) \cdot \mathbf{1}_{a+b+c+d \le 0}. \tag{104}$$

First we prove that $\#\mathsf{smfw}_\ell(Q) \le 1.5$. Here $F = \emptyset$. We will use two *relaxed* tree decompositions in TD_F^ℓ: The first (T_1, χ_1) has two bags $\{a,b,c\}$ and $\{c,d\}$. The second (T_2, χ_2) has two bags $\{a,b\}$ and $\{b,c,d\}$. (Both are relaxed TDs because the ligament edge $\mathbf{1}_{a+b+c+d \le 0}$ is not contained in any bag; recall Definition 3.3.) Following (69), for each $h \in \mathsf{ED}_{\not\perp} \cap \Gamma_{n|\mathcal{E}_{\not\perp}}$, we will pick one TD or the other. In particular, given some $h \in \mathsf{ED}_{\not\perp} \cap \Gamma_{n|\mathcal{E}_{\not\perp}}$:

- If $h(b) \ge 1/2$, then $h(bc|b) \le 1/2$. We pick (T_1, χ_1). From $\mathcal{E}_{\not\perp}$-polymatroid properties (Definition 3.12), we have

$$\begin{aligned} h(abc) &= h(ab) + h(abc|ab) \le h(ab) + h(bc|b) \le 1.5, \\ h(cd) &\le 1. \end{aligned}$$

- If $h(b) < 1/2$, we pick (T_2, χ_2).

$$\begin{aligned} h(ab) &\le 1, \\ h(bcd) &= h(b) + h(bcd|b) \le h(b) + h(cd) \le 1.5. \end{aligned}$$

This proves that $\#\mathsf{smfw}_\ell(Q) \le 1.5$.

Finally, as a special case of #PANDA, we explain how to solve the above query in time $\tilde{O}(N^{1.5})$ (where recall $N := \max\{|R|, |S|, |T|\}$). Let

$$\begin{aligned} S^\ell &:= \left\{(b,c) \in S \mid |\{c' \mid (b,c') \in S\}| \le \sqrt{N}\right\}, \\ S^h &:= S \setminus S^\ell. \end{aligned}$$

Now we can write

$$\begin{aligned} Q() &= \sum_{a,b,c,d} R(a,b) \cdot \left(S^\ell(b,c) + S^h(b,c)\right) \cdot T(c,d) \cdot \mathbf{1}_{a+b+c+d \le 0} \\ &= Q^\ell() + Q^h(), \text{ where} \\ Q^\ell() &:= \sum_{a,b,c,d} \underbrace{R(a,b) \cdot S^\ell(b,c)}_{U(a,b,c)} \cdot T(c,d) \cdot \mathbf{1}_{a+b+c+d \le 0}, \\ Q^h() &:= \sum_{a,b,c,d} R(a,b) \cdot \underbrace{S^h(b,c) \cdot T(c,d)}_{W(b,c,d)} \cdot \mathbf{1}_{a+b+c+d \le 0}. \end{aligned}$$

Note that both U and W above have sizes $\le N^{1.5}$. Using the algorithm from the proof of Theorem 3.5, Q^ℓ can be answered in time $O(N^{1.5} \log N)$ using the relaxed TD (T_1, χ_1), while Q^h can be answered in the same time using (T_2, χ_2).

B RELATIONAL MACHINE LEARNING

B.1 Gradient-based Optimization

In this section, we overview gradient-based optimization algorithms for convex and differentiable objective functions of the form (71). A gradient-based optimization algorithm employs the first-order gradient information to optimize $J(\boldsymbol{\beta})$. It repeatedly updates the parameters $\boldsymbol{\beta}$ by some step size α in the direction of the gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ until convergence. To guarantee convergence, it is common to use backtracking line search to ensure that the step size α is sufficiently small to decrease the loss for each step. Each update step requires two computations: (1) *Point evaluation*: Given $\boldsymbol{\theta}$, compute the scalar $J(\boldsymbol{\theta})$; and (2) *Gradient computation*: Given $\boldsymbol{\theta}$, compute the vector $\boldsymbol{\nabla} J(\boldsymbol{\theta})$.

There exist several variants of gradient descent algorithms, e.g., batch gradient descent or stochastic gradient descent, as well as many different algorithms to choose a valid step size [27]. For this work, we consider the batch gradient descent (BGD) algorithm with the Armijo backtracking line search condition, as depicted in Algorihtm 2. A common choice for setting the step size is a function that is inversely related to number of iterations of the algorithm, for instance $\alpha = \frac{1}{\lambda t}$ at iteration t, where λ is the regularization parameter from (71) [33].

B.1.1 Subgradient Descent. If the objective function $J(\boldsymbol{\beta})$ is convex but not differentiable, the gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ is not defined. Such objective functions do, however, admit a subgradient, which can be used in subgradient-based optimization algorithms. Algorihtm 2 naturally captures the batch subgradient-descent algorithm, if the parameters are updated in the direction of the subgradient as opposed to the gradient.

Algorithm 2: BGD with Armijo line search.

$\boldsymbol{\beta} \leftarrow$ a random point;
while *not converged yet* **do**
    $\alpha \leftarrow$ next step size;
    $\boldsymbol{d} \leftarrow \nabla J(\boldsymbol{\beta})$;
    `// Line search with Armijo condition;`
    while $\left(J(\boldsymbol{\beta} - \alpha \boldsymbol{d}) \geq J(\boldsymbol{\beta}) - \frac{\alpha}{2} \|\boldsymbol{d}\|_2^2\right)$ **do**
        $\alpha \leftarrow \alpha/2$;
    end
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \alpha \boldsymbol{d}$;
end

A popular application for subgradient-descent optimization algorithms is the learning of linear SVM models. One such algorithm is the Pegasos algorithm [33], which showed that subgradient methods can learn the parameters of the model significantly faster than other approaches, including Joachims' cutting plane algorithm [21].

B.2 Other non-polynomial loss functions

In this section, we overview the following non-polynomial loss functions, which were introduced in Section 4: (1) epsilon insensitive loss; (2) ordinal hinge loss; and (3) scalene loss. For each function, we define the loss function $\mathcal{L}$, the corresponding objective function $J(\boldsymbol{\beta})$, and the partial (sub)derivative $\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j}$ which is used in (sub)gradient-based optimization algorithms. In the derivations for the objective $J(\boldsymbol{\beta})$, we will focus on the loss function and ignore the regularizer for better readability.

As in Section 4, the objective and (sub)dervative can be reformulate into a few FAQ-AI expressions of the form (3). Instead of writing out the expressions explicitly, we annotate those terms that can be reformulated. The actual reformulation should be clear from the examples in Section 4 and Appendix B.3.

Epsilon insensitive loss. The epsilon insensitive loss function [27] is defined as:

$$\mathcal{L}(a, b) = \begin{cases} 0 & \text{if } |a - b| \leq \epsilon \\ |a - b| - \epsilon & \text{otherwise} \end{cases}$$

This loss function is used to learn SVM regression models. We consider learning a linear regression model $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$. The objective function and the corresponding partial subderivative with respect to β_j are given by:

$$\begin{aligned} J(\boldsymbol{\beta}) &= \sum_{(\boldsymbol{x},y) \in G} (|y - \boldsymbol{\beta}^\top \boldsymbol{x}| - \epsilon) \cdot \mathbf{1}_{|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| > \epsilon} \\ &= \underbrace{\sum_{(\boldsymbol{x},y) \in G} (y - \boldsymbol{\beta}^\top \boldsymbol{x} - \epsilon) \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > \epsilon}}_{O(n) \text{ FAQ-AI queries of the form (3)}} + \\ &\quad + \underbrace{\sum_{(\boldsymbol{x},y) \in G} (\boldsymbol{\beta}^\top \boldsymbol{x} - y - \epsilon) \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) - y > \epsilon}}_{O(n) \text{ FAQ-AI queries of the form (3)}} \end{aligned}$$

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j} = \underbrace{\sum_{(\boldsymbol{x},y) \in G} x_j \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) - y < \epsilon}}_{\text{FAQ-AI query of form (3)}} - \underbrace{\sum_{(\boldsymbol{x},y) \in G} x_j \cdot \mathbf{1}_{y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) > \epsilon}}_{\text{FAQ-AI query of form (3)}}$$

The objective and partial subderivative can thus be reformulated as $O(n)$ FAQ-AI expressions.

Ordinal hinge loss. The ordinal hinge loss [35] is defined as:

$$\begin{aligned} \mathcal{L}(a, b) &= \sum_{t=1}^{a-1} \max(0, 1 - b + t) + \sum_{t=a+1}^{d} \max(0, 1 + b - t) \\ &= \sum_{t=1}^{d} \max(0, 1 - b + t) \cdot \mathbf{1}_{t<a} + \max(0, 1 + b - t) \cdot \mathbf{1}_{t>a} \\ &= \sum_{t=1}^{d} (1 - b + t) \cdot \mathbf{1}_{t<a} \cdot \mathbf{1}_{b<t+1} + (1 + b - t) \cdot \mathbf{1}_{t>a} \cdot \mathbf{1}_{b>1-t} \end{aligned}$$

The loss function is used to learn ordinal regression models or ordinal PCA [35]. A linear ordinal regression model is linear function $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$ which predicts an ordinal label $y \in [d]$. The objective function and the partial subderivative with respect to β_j are given by:

$$\begin{aligned} J(\boldsymbol{\beta}) &= \sum_{t=1}^{d} \underbrace{\sum_{(\boldsymbol{x},y) \in G} (1 - f_{\boldsymbol{\beta}}(\boldsymbol{x}) + t) \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 1+t} \cdot \mathbf{1}_{y<t}}_{O(n) \text{ FAQ-AI queries of form (3)}} \\ &\quad + \sum_{t=1}^{d} \underbrace{\sum_{(\boldsymbol{x},y) \in G} (1 + f_{\boldsymbol{\beta}}(\boldsymbol{x}) - t) \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) > t-1} \cdot \mathbf{1}_{y>t}}_{O(n) \text{ FAQ-AI queries of form (3)}} \end{aligned}$$

$$\begin{aligned} \frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j} &= \sum_{t=1}^{d} \underbrace{\sum_{(\boldsymbol{x},y) \in G} x_j \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) > t-1} \cdot \mathbf{1}_{y>t}}_{\text{FAQ-AI query of form (3)}} \\ &\quad - \sum_{t=1}^{d} \underbrace{\sum_{(\boldsymbol{x},y) \in G} x_j \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 1+t} \cdot \mathbf{1}_{y<t}}_{\text{FAQ-AI query of form (3)}} \end{aligned}$$

The objective and partial subderivative can thus be reformulated as $O(d \cdot n)$ FAQ-AI expressions.

Scalene loss. The scalene loss function [35] is defined as:

$$\begin{aligned} \mathcal{L}(a, b) &= \alpha \cdot \max(0, a - b) + (1 - \alpha) \cdot \max(0, b - a) \\ &= \alpha \cdot (a - b) \cdot \mathbf{1}_{a>b} + (1 - \alpha) \cdot (b - a) \cdot \mathbf{1}_{b>a} \end{aligned}$$

where $\alpha \in (0, 1)$ is a constant.

The loss function is used to learn quantile regression models. We again consider a linear regression model $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$. The objective function and the partial subderivative with respect to β_j

are given by:

$$J(\boldsymbol{\beta}) = \alpha \underbrace{\sum_{(\boldsymbol{x},y)\in G} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x})) \cdot \mathbf{1}_{y>f_{\boldsymbol{\beta}}(\boldsymbol{x})}}_{O(n) \text{ FAQ-AI queries of form (3)}} + (1-\alpha) \underbrace{\sum_{(\boldsymbol{x},y)\in G} (f_{\boldsymbol{\beta}}(\boldsymbol{x}) - y) \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}(\boldsymbol{x})>xy}}_{O(n) \text{ FAQ-AI queries of form (3)}}$$

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \beta_j} = (1-\alpha) \underbrace{\sum_{(\boldsymbol{x},y)\in G} x_j \cdot \mathbf{1}_{f_{\boldsymbol{\beta}}>y}}_{\text{FAQ-AI query of form (3)}} - \alpha \underbrace{\sum_{(\boldsymbol{x},y)\in G} x_j \cdot \mathbf{1}_{y>f_{\boldsymbol{\beta}}(\boldsymbol{x})}}_{\text{FAQ-AI query of form (3)}}$$

The objective and partial subderivative can thus be reformulated as $O(n)$ FAQ-AI expressions.

B.3 Reformulating the objective with Huber loss into FAQ-AI expressions

We consider the objective $J(\boldsymbol{\beta})$ with Huber loss for linear regression models as defined in Section 4.2, and show how it can be reformulated into $O(n^2)$ FAQ-AI expressions of the form (3). The objective $J(\boldsymbol{\beta})$ is defined as follows:

$$J(\boldsymbol{\beta}) = \frac{1}{2} \sum_{(\boldsymbol{x},y)\in G} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x}))^2 \cdot \mathbf{1}_{|y-f_{\boldsymbol{\beta}}(\boldsymbol{x})|\leq 1} + (|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| - 1) \cdot \mathbf{1}_{|y-f_{\boldsymbol{\beta}}(\boldsymbol{x})|>1} + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2$$

First, we consider the case where $|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| \leq 1$, i.e. the square loss term of $J(\boldsymbol{\beta})$. For ease of notation, let $c_1(y, \boldsymbol{x}) = y - f_{\boldsymbol{\beta}}(\boldsymbol{x})$.

$$\begin{aligned}
&\sum_{(\boldsymbol{x},y)\in G} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x}))^2 \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&= \sum_{(\boldsymbol{x},y)\in G} y^2 - 2y f_{\boldsymbol{\beta}}(\boldsymbol{x}) + (f_{\boldsymbol{\beta}}(\boldsymbol{x}))^2 \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&= \sum_{(\boldsymbol{x},y)\in G} y^2 \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} - 2 \sum_{(\boldsymbol{x},y)\in G} y \cdot f_{\boldsymbol{\beta}}(\boldsymbol{x}) \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&\quad + \sum_{(\boldsymbol{x},y)\in G} (f_{\boldsymbol{\beta}}(\boldsymbol{x}))^2 \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&= \sum_{(\boldsymbol{x},y)\in G} y^2 \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} - 2 \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot y \cdot x_i \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&\quad + \sum_{i\in[n]} \sum_{j\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot \beta_j \cdot x_i \cdot x_j \cdot \mathbf{1}_{c_1(y,\boldsymbol{x})} \\
&= \sum_{(\boldsymbol{x},y)\in G} y^2 \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\leq 1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq 0} \\
&\quad + \sum_{(\boldsymbol{x},y)\in G} y^2 \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq -1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<0} \\
&\quad - 2 \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot y \cdot x_i \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\leq 1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq 0} \\
&\quad - 2 \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot y \cdot x_i \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq -1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<0} \\
&\quad + \sum_{i\in[n]} \sum_{j\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot \beta_j \cdot x_i \cdot x_j \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\leq 1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq 0} \\
&\quad + \sum_{i\in[n]} \sum_{j\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot \beta_j \cdot x_i \cdot x_j \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq -1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<0}
\end{aligned}$$

Each of summation over the training dataset G in the final reformulation above can be expressed as one FAQ-AI query with two ligament hyperedges. For instance, the first summation over G is equivalent to the following FAQ-AI expression:

$$Q() = \sum_{y,\boldsymbol{x}_{\mathcal{V}}} y^2 \cdot \underbrace{\mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\leq 1} \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})\geq 0}}_{\text{ligaments in } \mathcal{E}_\ell} \cdot \left(\prod_{F\in\mathcal{E}_s} R_F(x_F) \right)$$

The absolute loss function for the case $|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| > 1$ can be reformulated similarly:

$$\begin{aligned}
&\sum_{(\boldsymbol{x},y)\in G} (|y - f_{\boldsymbol{\beta}}(\boldsymbol{x})| - 1) \cdot \mathbf{1}_{|y-f_{\boldsymbol{\beta}}(\boldsymbol{x})|>1} \\
&= \sum_{(\boldsymbol{x},y)\in G} (y - f_{\boldsymbol{\beta}}(\boldsymbol{x}) - 1) \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})>1} \\
&\quad + \sum_{(\boldsymbol{x},y)\in G} (f_{\boldsymbol{\beta}}(\boldsymbol{x}) - y - 1) \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<-1} \\
&= \sum_{(\boldsymbol{x},y)\in G} y \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})>1} - \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot x_i \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})>1} \\
&\quad - \sum_{(\boldsymbol{x},y)\in G} y \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<-1} + \sum_{i\in[n]} \sum_{(\boldsymbol{x},y)\in G} \beta_i \cdot x_i \cdot \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<-1} \\
&\quad - \sum_{(\boldsymbol{x},y)\in G} \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})>1} - \sum_{(\boldsymbol{x},y)\in G} \mathbf{1}_{y-f_{\boldsymbol{\beta}}(\boldsymbol{x})<-1}
\end{aligned}$$

All of these terms can be reformulated as $O(n)$ FAQ-AI expressions of the form 3.

Overall, the objective $J(\boldsymbol{\beta})$ with Huber loss for learning robust linear regression models can be computed with $O(n^2)$ FAQ-AI expressions, and without materializing the training dataset G.

B.4 Proof of Theorem 4.1

Theorem 4.1. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. For any robust linear regression model $\boldsymbol{\beta}^\top \boldsymbol{x}$, the objective $J(\boldsymbol{\beta})$ and gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ with Huber loss can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)} \log N)$ with* InsideOut.

Proof. Let n be the number of variables in Q. We show in Section 4.2 and Appendix B.3 that we can rewrite of the objective $J(\boldsymbol{\beta})$ and gradient $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ into $O(n^2)$ FAQ-AI expressions with at most $|\mathcal{E}_\ell| = 2$ ligament hyperedges. The overall runtime bound for computing $J(\boldsymbol{\beta})$ and $\boldsymbol{\nabla} J(\boldsymbol{\beta})$ with #PANDA follows from Theorem 3.18, which states that #PANDA can compute each FAQ-AI expression in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$.

The overall runtime bound for computing $J(\boldsymbol{\beta})$ and $\nabla J(\boldsymbol{\beta})$ with InsideOut follows from Theorem 3.5, which states that InsideOut can compute each FAQ-AI expression in time $O(N^{\mathsf{faqw}_\ell(Q)} \log N)$. □

B.5 Derivation steps for reformulating (79)

We show the derivation steps of the reformulation of (79).

$$\sum_{(\boldsymbol{x},y)\in G} \max\{0, 1 - y(\boldsymbol{\beta}^\top \boldsymbol{x})\} + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2 \tag{105}$$

$$= \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2 + \sum_{(\boldsymbol{x},y)\in G} (1 - y(\boldsymbol{\beta}^\top \boldsymbol{x})) \cdot \mathbf{1}_{y(\boldsymbol{\beta}^\top \boldsymbol{x})\leq 1} \tag{106}$$

$$\begin{aligned}
&= \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2 + \sum_{(\boldsymbol{x},y)\in G} (1 - (\boldsymbol{\beta}^\top \boldsymbol{x})) \cdot \mathbf{1}_{y=1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \leq 1} \\
&\quad + \sum_{(\boldsymbol{x},y)\in G} (1 + (\boldsymbol{\beta}^\top \boldsymbol{x})) \cdot \mathbf{1}_{y=-1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \geq -1} \\
&= \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2 + \underbrace{\sum_{(\boldsymbol{x},y)\in G} \mathbf{1}_{y=1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \leq 1}}_{\text{query of the form (3)}} - \sum_{i=1}^{n} \underbrace{\sum_{(\boldsymbol{x},y)\in G} \beta_i x_i \mathbf{1}_{y=1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \leq 1}}_{\text{query of the form (3)}} \\
&\quad + \underbrace{\sum_{(\boldsymbol{x},y)\in G} \mathbf{1}_{y=-1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \geq -1}}_{\text{query of the form (3)}} + \sum_{i=1}^{n} \underbrace{\sum_{(\boldsymbol{x},y)\in G} \beta_i x_i \mathbf{1}_{y=-1} \mathbf{1}_{\boldsymbol{\beta}^\top \boldsymbol{x} \geq -1}}_{\text{query of the form (3)}}.
\end{aligned}$$

B.6 Proof of Theorem 4.2

Theorem 4.2. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. For any linear SVM classification model $\boldsymbol{\beta}^\top \boldsymbol{x}$, the objective $J(\boldsymbol{\beta})$ and gradient $\nabla J(\boldsymbol{\beta})$ with hinge loss can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)} \log N)$ with* InsideOut.

Proof. Let n be the number of variables in Q. We show in Section 4.3.1 that $J(\boldsymbol{\beta})$ and $\nabla J(\boldsymbol{\beta})$ can be rewritten into $O(n)$ FAQ-AI expressions with a single ligament hyperedge (i.e. $|\mathcal{E}_\ell| = 1$). The overall runtime bound for computing $J(\boldsymbol{\beta})$ and $\nabla J(\boldsymbol{\beta})$ with #PANDA follows from Theorem 3.18, which states that #PANDA can compute each FAQ-AI query in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$. The runtime for computing $J(\boldsymbol{\beta})$ and $\nabla J(\boldsymbol{\beta})$ with InsideOut follows from Theorem 3.5: This is $O(N^{\mathsf{faqw}_\ell(Q)} \cdot \log N)$ for a FAQ-AI query Q. □

B.7 Proof of Theorem 4.3

Theorem 4.3. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query. A linear SVM classification model can be learned over the training dataset $Q(I)$ with Joachims' cutting-plane algorithm in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)} \log N)$ with* InsideOut.

Proof. Recall that for each iteration t of Algorithm 1, we add one set $\boldsymbol{T}^{(t)}$ to $\mathcal{W}$, and $\boldsymbol{T}^{(t)}$ is associated with a coefficient vector $\boldsymbol{\beta}^{(t)}$. Our main observation is that we do not have to materialize the set $\boldsymbol{T}^{(t)}$, since it is completely determined by the data and the coefficient vector $\boldsymbol{\beta}^{(t)}$. Thus, instead of storing $\boldsymbol{T}^{(t)}$ we can simply store $\boldsymbol{\beta}^{(t)}$ and reformulate the data dependent term $\boldsymbol{x}_{T^{(t)}}$ in (83) as a computation over $\boldsymbol{G}$:

$$\forall \boldsymbol{T}^{(t)} \in \mathcal{W} \ : \quad \boldsymbol{x}_{T^{(t)}} = \sum_{(\boldsymbol{x},y)\in \boldsymbol{T}^{(t)}} y\boldsymbol{x} = \sum_{(\boldsymbol{x},y)\in G} y\boldsymbol{x} \cdot \mathbf{1}_{y\langle \boldsymbol{\beta}^{(t)}, \boldsymbol{x}\rangle < 1}.$$

The vector $\boldsymbol{x}_{T^{(t)}}$ has size n. For each $j \in [n]$, we can compute the j'th component of $\boldsymbol{x}_{T^{(t)}}$ as the summation of the following two FAQ-AI expressions, which are of form (3):

$$Q_1() = \sum_{\boldsymbol{x}_{\mathcal{V}}, y} y \cdot x_j \cdot \mathbf{1}_{y=1} \cdot \mathbf{1}_{\sum_{j\in[n]} \beta_j^{(t)} \cdot x_j < 1} \cdot \left(\prod_{F\in\mathcal{E}_s} R_F(\boldsymbol{x}_F) \right),$$

$$Q_2() = \sum_{\boldsymbol{x}_{\mathcal{V}}, y} y \cdot x_j \cdot \mathbf{1}_{y=-1} \cdot \mathbf{1}_{\sum_{j\in[n]} \beta_j^{(t)} \cdot x_j > -1} \cdot \left(\prod_{F\in\mathcal{E}_s} R_F(\boldsymbol{x}_F) \right).$$

Q_1 and Q_2 have a single ligament hyperedge (i.e. $|\mathcal{E}_\ell| = 1$). Theorem 3.18 states that #PANDA computes Q_i for $i \in [2]$ in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q_i)})$. Consequently, the optimization problem at line 5 of Algorithm 1 can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q_i)})$. This determines the runtime of Algorithm 1.

Using InsideOut, the runtime of Algorithm 1 follows from Theorem 3.5: This is $O(N^{\mathsf{faqw}_\ell(Q_i)} \log N)$ for Q_i. □

B.8 Wolfe dual for optimization problem at line 5 of Algorithm 1

We consider the inner optimization problem at line 5 of Algorithm 1, show how to derive the Wolfe dual (83) from the structural SVM classification formulation (82). Recall that $\boldsymbol{x}_{\boldsymbol{T}} = \sum_{(\boldsymbol{x},y)\in \boldsymbol{T}} y\boldsymbol{x}$, the inner optimization problem at line 5 of Algorithm 1 is of the form:

$$\begin{aligned}
\min_{\boldsymbol{\beta},\xi} \quad & \frac{1}{2} \|\boldsymbol{\beta}\|^2 + C\xi \\
\text{s. t.} \quad & \langle \boldsymbol{\beta}, \boldsymbol{x}_{\boldsymbol{T}} \rangle \geq |\boldsymbol{T}| - |\boldsymbol{G}|\xi \qquad \boldsymbol{T} \in \mathcal{W} \\
& \xi \geq 0.
\end{aligned} \tag{107}$$

The Lagrangian function if this optimization problem is:

$$\begin{aligned}
& L(\boldsymbol{\beta}, \xi, \boldsymbol{\alpha}, \gamma) \\
&= \frac{1}{2} \|\boldsymbol{\beta}\|^2 + C\xi + \sum_{\boldsymbol{T}\in\mathcal{W}} \alpha_{\boldsymbol{T}} (|\boldsymbol{T}| - |G|\xi - \langle \boldsymbol{\beta}, \boldsymbol{x}_{\boldsymbol{T}} \rangle) - \gamma\xi \\
&= \frac{1}{2} \|\boldsymbol{\beta}\|^2 - \left\langle \boldsymbol{\beta}, \sum_{\boldsymbol{T}\in\mathcal{W}} \alpha_{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{T}} \right\rangle + \sum_{\boldsymbol{T}\in\mathcal{W}} |\boldsymbol{T}|\alpha_{\boldsymbol{T}} \\
&\quad + \left(C - |\boldsymbol{G}| \sum_{\boldsymbol{T}\in\mathcal{W}} \alpha_{\boldsymbol{T}} - \gamma \right) \xi.
\end{aligned}$$

where $\boldsymbol{\alpha} = (\alpha_{\boldsymbol{T}})_{\boldsymbol{T}\in\mathcal{W}}$ and γ are Lagrange multipliers.

Since the Lagrangian is convex and continuously differentiable, we can define the Wolfe dual as the following optimization problem:

$$\begin{aligned}
\max_{\boldsymbol{\beta},\xi} \quad & L(\boldsymbol{\beta}, \xi, \boldsymbol{\alpha}, \gamma) && (108)\\
\text{s. t.} \quad & \nabla_{\boldsymbol{\beta}} L = \boldsymbol{\beta} - \sum_{T \in \mathcal{W}} \alpha_T \boldsymbol{x}_T = 0 \\
& \nabla_{\xi} L = C - |G| \sum_{T \in \mathcal{W}} \alpha_T - \gamma = 0 \\
& \boldsymbol{\alpha} \geq \mathbf{0}, \gamma \geq 0.
\end{aligned}$$

The optimal condition for $\boldsymbol{\beta}$ is $\boldsymbol{\beta} = \sum_{T \in \mathcal{W}} \alpha_T \boldsymbol{x}_T$. We use this equality to rewrite the above dual formulation and attain the optimization problem (83) from Section 4.3.2:

$$\begin{aligned}
\max_{\boldsymbol{\alpha} \geq \mathbf{0}} \quad & -\frac{1}{2}\left\langle \sum_{T \in \mathcal{W}} \alpha_T \boldsymbol{x}_T, \sum_{T \in \mathcal{W}} \alpha_T \boldsymbol{x}_T \right\rangle + \sum_{T \in \mathcal{W}} |T|\alpha_T && (109)\\
\text{s.t.} \quad & \sum_{T \in \mathcal{W}} \alpha_T \leq \frac{C}{|G|}
\end{aligned}$$

B.9 Proof of Theorem 4.4

Theorem 4.4. *Let $\boldsymbol{I}$ be an input database where N is the largest relation in $\boldsymbol{I}$, and Q be a feature extraction query where n is the number of its variables. Each iteration of Lloyd's k-means algorithm can be computed in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$ with* #PANDA *and in time $O(N^{\mathsf{faqw}_\ell(Q)} \log^{k-1} N)$ with* InsideOut.

Proof. We shown in Section 4.4 that each mean vector $(\mu_j)_{j\in[k]}$ can be computed with $O(n)$ FAQ-AI expression of the form (3), where each query has $|\mathcal{E}_\ell| = k$ ligament hyperedges. For #PANDA, the overall runtime to update all k-means follows from Theorem 3.18 (respectively Theorem 3.5), which states that the algorithm can compute each FAQ-AI expression of form (3) in time $\tilde{O}(N^{\#\mathsf{smfw}_\ell(Q)})$. Using InsideOut, the runtime follows from Theorem 3.5: Any FAQ-AI query Q of form (3) can be computed in time $O(N^{\mathsf{faqw}_\ell(Q)} \log^{k-1} N)$. □

C RECOVERING TWO EXISTING RESULTS

In this section we review two prior results concerned with the evaluation of queries with inequalities: the evaluation of Core XPath queries over XML documents via relational encoding in the pre/post plane and the exact inference for IQ queries with inequality joins over probabilistic databases. Our main observation is that their linearithmic complexity is due to the same structural property behind relaxed tree decompositions: Such queries admit trivially a relaxed tree decomposition, where each bag corresponds to one relation in the query and the ligament edges, i.e., the inequality joins, are covered by neighboring bags.

C.1 Core XPath Queries

We consider the problem of evaluating Core XPath queries over XML documents. An XML document is represented as a rooted tree whose nodes follow the *document order*. Core XPath queries define traversals of such trees using two constructs: (1) a *context node* that is the starting point of the traversal; and (2) a tree of *location steps* with one distinguished branch that selects nodes and all other branches conditioning this selection. Given a context node v, a location step selects a set of the nodes in the tree that are accessible from v via the step's *axis*. This set of nodes provides the context nodes for the next step, which is evaluated for each such node in turn. The result of the location step is the set of nodes accessible from any of its input context nodes, sorted in document order.

The *preorder rank* $pre(v)$ of a node v is the index of v in the list of all nodes in the tree that are visited in the (depth-first, left-to-right) preorder traversal of the tree; this order is the document order. Similarly, the *postorder rank* $post(v)$ of v is its index in the list of all nodes in the tree that are visited in the (depth-first, left-to-right) postorder tree traversal. We can use the pre/post-order ranks of nodes to define the main axes `descendant`, `ancestor`, `following`, and `preceding` [16]. Given two nodes v and v' in the tree, the four axes are defined using the pre/post two-dimensional plane:

- v' is a descendant of v or equivalently v is an ancestor of v'

$$\text{iff } pre(v) < pre(v') \wedge post(v') < post(v)$$

- v' follows v or equivalently v precedes v'

$$\text{iff } pre(v) < pre(v') \wedge post(v) < post(v')$$

The remaining axes `parent`, `child`, `following-sibling`, and `preceding-sibling` are restrictions of the four main axes, where we also use the parent information *par* for each node:

- v' is a child of v or equivalently v is a parent of v'

$$\text{iff } v = par(v')$$

- v' is a following sibling of v or equivalently v is a preceding sibling of v'

$$\text{iff } pre(v) < pre(v') \wedge post(v) < post(v') \wedge par(v) = par(v')$$

We follow the standard approach to reformulate XPath evaluation in the relational domain [16]. We represent the document by a factor $\boldsymbol{G}$ in the Boolean semiring with schema $(pre, post, par, tag)$. For each node in the tree there is one tuple in $\boldsymbol{G}$ with *pre* and *post* ranks, label *tag*, and preorder rank *par* of the parent node. A query with n location steps is mapped to an FAQ-AI expression Q that is a join of $n+1$ copies of $\boldsymbol{G}$ where the join conditions are the inequalities encoding the axes of the n steps. The first copy $\boldsymbol{G}_0$ is for the initial context node(s). The axis of the i-th step is translated into the conjunction of inequalities between pre/post rank variables of the copies $\boldsymbol{G}_{i-1}$ and $\boldsymbol{G}_i$. The query Q has one free variable: This is the preorder rank variable from the copy of $\boldsymbol{G}$ corresponding to the location step that selects the result nodes.

Example C.1. The Core XPath query

$$v/\text{descendant} :: a[\text{descendant} :: c]/\text{following} :: b$$

selects all b-labeled nodes following a-labeled nodes that are descendants of the given context node v and that have at least one c-labeled descendant node. The steps in the above textual representation of the query are separated by /. The brackets [ ] delimit a condition on the selection of the a-labeled nodes. We can reformulate this query in FAQ-AI over the Boolean semiring as follows:

$$\begin{aligned} Q(pre_b) \leftarrow & G_v(pre_v, post_v, p_v, tag_v) \wedge G_a(pre_a, post_a, p_a, \text{'a'}) \wedge \\ & G_c(pre_c, post_c, p_c, \text{'c'}) \wedge G_b(pre_b, post_b, p_b, \text{'b'}) \wedge \\ & pre_v < pre_a \wedge post_a < post_v \wedge \quad \text{// a is descendant of v} \\ & pre_a < pre_c \wedge post_c < post_a \wedge \quad \text{// c is descendant of a} \\ & pre_a < pre_b \wedge post_a < post_b \quad \text{// b is following a} \end{aligned}$$

□

The hypergraph of a relational encoding of a Core XPath query has one skeleton hyperedge for each copy of the document factor and one ligament edge for each pair of inequalities over two of these copies. Any two skeleton hyperedges may only have one node, i.e., query variable, in common to express the parent/child or sibling relationship between their corresponding steps. This hypergraph admits a trivial relaxed tree decomposition, which mirrors the tree structure of the query. In particular, there is one bag of the decomposition consisting of the variables of each copy of the document factor. Each ligament edge represents a pair of inequalities over variables of two neighboring bags. The running intersection property holds since the equalities are by construction only over variables from neighboring bags.

It is known that the time complexity of answering a Core XPath query Q with n location steps over an XML document $\boldsymbol{G}$ is $O(n \cdot |\boldsymbol{G}|)$ (Theorem 8.5 [14]; it assumes the document factor sorted). We can show a linearithmic time complexity result using our FAQ-AI reformulation of Core XPath queries and the trivial relaxed tree decomposition.

Proposition C.2. *For any Core XPath query Q with n location steps and XML document $\boldsymbol{G}$, the query answer can be computed in time $O(n \cdot |\boldsymbol{G}| \cdot \log |\boldsymbol{G}|)$.*

Proof. Let φ be the FAQ-AI reformulation of Q and F the factor representing the XML document $\boldsymbol{G}$. There is a one-to-one correspondence between the trivial relaxed tree decomposition and the XPath query, with one bag per location step. Let n be the number of location steps in Q, or equivalently the number of bags in the tree decomposition. We consider this trivial tree decomposition and choose its root as the bag corresponding to the location step that selects the answer node set. Our evaluation algorithm proceeds in a bottom-up left-to-right traversal of the tree decomposition and eliminates one bag at a time. This bag elimination is a variant

We index the bags and their corresponding factors in this traversal order. The first factor to eliminate is then denoted by F_1 while the last factor, which corresponds to the location step selecting the answer node set, is denoted by F_n.

We initially create factors S_j that are copies of factors F_j corresponding to leaf bags in the tree. Consider now two factors S_j and F_i corresponding to a leaf bag and respectively to its parent bag. Let $\phi_{i,j}$ be the conjunction of inequalities defining the axis relationship between the location steps corresponding to these bags. We then compute a new factor S_i that consists of those tuples in F_i that join with some tuples in S_j. This is expressed in FAQ-AI over the Boolean semiring:

$$\begin{aligned} S_i(pre_i, post_i, p_i, t_i) \leftarrow & F_i(pre_i, post_i, p_i, t_i) \wedge S_j(pre_j, post_j, p_j, t_j) \\ & \wedge \phi_{i,j} \end{aligned}$$

The conjunction $\phi_{i,j}$ only has two inequalities on variables between the two bags. Computing S_i takes time $O(|F| \log |F|)$ following the algorithm from the proof of Theorem 3.5. We can sort both F_i and S_j in ascending order on the preorder column and in descending order on the postorder column. For each tuple t in F_i, the tuples in S_j that join with t form a contiguous range in S_j. To assert whether t is in S_i, it suffices to check that this range is not empty. There are n such steps and $|F| = |F_i| = |\boldsymbol{G}|$, with an overall time complexity of $O(n \cdot |\boldsymbol{G}| \log |\boldsymbol{G}|)$. □

C.2 Probabilistic Queries with Inequalities

The problem of query evaluation in probabilistic databases is #P-hard for general queries and probabilistic database formalisms [34]. Extensive prior work focused on charting the tractability frontier of this problem, with positive results for several classes of queries on so-called tuple-independent probabilistic databases. We discuss here one such class of queries with inequality joins called IQ [30].

A tuple-independent probabilistic database is a database where each tuple t is associated with a Boolean random variable $v(t)$ that is independent of the other tuples in the database. This is the database formalism of choice for studies on query tractability since inference is hard already for trivial queries on more expressive probabilistic database formalisms [34].

FAQ factors naturally capture tuple-independent probabilistic databases: A tuple-independent probabilistic relation R is a factor that maps each tuple t in R to the probability that the associated random variable $v(t)$ is true.

We next define the class IQ of inequality queries and later show how to recover the linearithmic time complexity for their inference.

Definition C.3 (adapted from Definitions 3.1, 3.2 [30]). Let a hypergraph $\mathcal{H} = (\mathcal{V} = [n], \mathcal{E}_s \cup \mathcal{E}_\ell)$, where $\mathcal{E}_s$ and $\mathcal{E}_\ell$ are disjoint, $\mathcal{E}_s$ consists of pairwise disjoint sets, $\mathcal{E}_\ell$ consists of sets $\{i, j\}$ for which there is a vector $c_{i,j} \in \{[1, -1]^{\mathrm{T}}, [-1, 1]^{\mathrm{T}}\}$, and $\forall F \in \mathcal{E}_s$: $|(\bigcup_{I \in \mathcal{E}_\ell} I) \cap F| \leq 1$. An IQ query has the form

$$Q() \leftarrow \bigwedge_{F \in \mathcal{E}_s} R_F(\boldsymbol{X}_F) \wedge \bigwedge_{\{i,j\} \in \mathcal{E}_\ell} [X_i, X_j]^{\mathrm{T}} \cdot c_{i,j} \leq 0 \qquad (110)$$

where $(R_F)_{F \in \mathcal{E}_s}$ are distinct factors. □

The edges (i.e., binary hyperedges) in $\mathcal{E}_\ell$ correspond to inequalities of the query variables. These inequalities are restricted so that there is at most one node (query variable) from any hyperedge in $\mathcal{E}_s$. Inequalities on variables of the same factor are not in $\mathcal{E}_\ell$; they can be computed trivially in a pre-processing step.

The inequalities may only have the form $X_i \leq X_j$ or $X_j \leq X_i$. They induce an *inequality graph* where X_i is a parent of X_j if $X_i \leq X_j$. This graph can be minimized by removing edges corresponding to redundant inequalities implied by other inequalities [19]. Each graph node thus corresponds to precisely one factor. We categorize the IQ queries based on the structural complexity of their inequality graphs into (forests of) paths, trees, and graphs.

Example C.4. Consider the following IQ queries:

$$Q_1() \rightarrow R(A) \wedge S(B) \wedge T(C) \wedge A \leq B \wedge B \leq C$$
$$Q_2() \rightarrow R(A) \wedge S(B) \wedge T(C) \wedge A \leq B \wedge A \leq C$$

The inequalities form a path in Q_1 and a tree in Q_2.

The probability a query over a probabilistic database $\boldsymbol{I}$ is the probability of its *lineage* [34]. The lineage is a propositional formula over the random variables associated with the input tuples. It is equivalent to the disjunction of all possible derivations of the query answer from the input tuples.

Example C.5. Consider the factors R, S, T, where r_i, s_j, t_k denote the variables associated with the tuples in these factors and for a random variable a, p_a denotes the probability that a = true:

R	A	
r_1	1	p_{r_1}
r_2	2	p_{r_2}
r_3	3	p_{r_3}

S	B	
s_1	2	p_{s_1}
s_2	3	p_{s_2}
s_3	4	p_{s_3}

T	C	
t_1	3	p_{t_1}
t_2	4	p_{t_2}
t_3	5	p_{t_3}

The lineage of Q_1 and Q_2 over these factors is:

$$\underbrace{\begin{array}{l} r_1[s_1(t_1+t_2+t_3)+s_2(t_2+t_3)+s_3t_3]+ \\ r_2[\qquad\qquad\quad s_2(t_2+t_3)+s_3t_3]+ \\ r_3[\qquad\qquad\qquad\qquad\qquad\; s_3t_3] \end{array}}_{\text{lineage of } Q_1} \qquad \underbrace{\begin{array}{l} r_1(s_1+s_2+s_3)(t_1+t_2+t_3)+ \\ r_2(\quad\; s_2+s_3)(t_1+t_2+t_3)+ \\ r_3(\qquad\quad s_3)(\quad\; t_2+t_3) \end{array}}_{\text{lineage of } Q_2}$$

Prior work (Theorem 4.7 [30]) showed that the probability of an IQ query Q with an inequality tree with k nodes over a tuple-independent probabilistic database of size N can be computed in time $O(2^k \cdot N \log N)$ using a construction of the query lineage in an Ordered Binary Decision Diagram (OBDD). We show next that a variant of the algorithm in the proof of Lemma 3.1, adapted from counting to *weighted counting*, i.e., probability computation, can compute the probability in time $O(N \log N)$, thus shaving off an exponential factor in the number of inequalities.

We first explain this result using two examples, which draw on a crucial observation made in prior work [30]: The lineage of IQ queries has a chain structure: For each factor, there is an order on its random variables that defines a chain of logical implications between their cofactors in the lineage: the cofactor of the first variable implies the cofactor of the second variable, which implies the cofactor of the third variable, and so on.

Example C.6. We continue Example C.5. The lineage of Q_1 and Q_2 is arranged so that the chain structure becomes apparent. This structure allows for an equivalent rewriting of the lineage [30], as shown next for the lineage ϕ_{r_1} of Q_1 (for a random variable a, $\overline{a}$ denotes its negation):

$$\phi_{r_i} = r_i\phi_{s_i} + \overline{r_i}\phi_{r_{i+1}}, \forall i \in [3]; \qquad \phi_{r_4} = \text{false}$$
$$\phi_{s_j} = s_j\phi_{t_j} + \overline{s_j}\phi_{s_{j+1}}, \forall j \in [3]; \qquad \phi_{s_4} = \text{false}$$
$$\phi_{t_k} = t_k \qquad + \overline{t_k}\phi_{t_{k+1}}, \forall k \in [3]; \qquad \phi_{t_4} = \text{false}$$

In disjunctive normal form, the lineage of Q_1 may have size cubic in the size of the database. The factorization of the lineage in Example C.5 lowers the size to quadratic. The above rewriting further reduces the size to linear. The rewritten form can be read directly from the input factors following the structure of the inequality tree.

Since the above expressions are sums of two mutually exclusive formulas, their probabilities are the sums of the probabilities of their respective two formulas. Their probabilities can be computed in one bottom-up right-to-left pass: First for ϕ_{t_k} in decreasing order of k, then for ϕ_{s_j} in decreasing order of j, and finally for ϕ_{r_i} in decreasing order of i. We extend the probability function p from input random variables to formulas over these variables. The probability of Q_1's lineage, which is also the probability of Q_1, is $(\forall i, j, k \in [3])$:

$$p(\phi_{r_i}) = p(r_i) \cdot p(\phi_{s_i}) + [1 - p(r_i)] \cdot p(\phi_{r_{i+1}})$$
$$p(\phi_{s_j}) = p(s_j) \cdot p(\phi_{t_j}) + [1 - p(s_j)] \cdot p(\phi_{s_{j+1}})$$
$$p(\phi_{t_k}) = p(t_k) \qquad\quad + [1 - p(t_k)] \cdot p(\phi_{t_{k+1}})$$

Since there are no variables r_4, s_4, and t_4, we use $p(\phi_{r_4}) = p(\phi_{s_4}) = p(\phi_{t_4}) = 0$. This computation corresponds to a decomposition of ϕ_{r_1} that can be captured by a linear-size OBDD [30].

The probability of the lineage ψ_{r_1} of Q_2 is computed similarly $(\forall i, j, k \in [3])$:

$$p(\psi_{r_i}) = p(r_i) \cdot p(\psi_{s_i}) \cdot p(\psi_{t_i}) + [1 - p(r_i)] \cdot p(\psi_{r_{i+1}})$$
$$p(\psi_{s_j}) = p(s_j) + [1 - p(s_j)] \cdot p(\psi_{s_{j+1}})$$
$$p(\psi_{t_k}) = p(t_k) + [1 - p(t_k)] \cdot p(\psi_{t_{k+1}})$$

This computation would correspond to a decomposition of ψ_{r_1} that can be captured by an OBDD with several nodes for a random variable from S and T; in general, such an OBDD would have a size linear in N but with an additional exponential factor in the size of the inequality tree due to the inability to represent succinctly the products of lineage over T and of lineage over S [30]. (OBDDs with AND nodes can capture such products without this exponential factor, though in this paper we do not use them.) □

Proposition C.7. *Given a tuple-independent probabilistic database $\boldsymbol{I}$ of size N and an IQ query Q with a forest of inequality trees, we can compute the probability of Q over $\boldsymbol{I}$ in time $O(N \log N)$.*

Proof. We next present the inference algorithm for a given IQ query Q with an inequality tree. It uses a minor variant of the algorithm from the proof of Lemma 3.1 to compute a functional aggregate query with additive inequalities over two factors.

We first reduce the input database $\boldsymbol{I}$ to a simplified database of unary and nullary factors that is constructed by aggregating away all query variables that do not contribute to inequalities.

Let us partition $\mathcal{E}_s$ into the hyperedges $\mathcal{E}_1$ that contain query variables involved in inequalities and all other hyperedges $\mathcal{E}_2$.

We reduce each factor $(R_F)_{F \in \mathcal{E}_1}$ with a query variable X_i occurring in inequalities to a unary factor $S_{\{i\}}$ by aggregating away all other query variables. For an X_i-value x_i, $S_{\{i\}}(x_i)$ gives the probability of the disjunction of the independent random variables associated with the tuples in R_F that have the X_i-value x_i:

$$S_{\{i\}}(x_i) = 1 - \prod_{\boldsymbol{x} \in \text{Dom}(X_F - \{i\})} \big(1 - R_F(\boldsymbol{x}_F)\big)$$

We also reduce all factors $(R_F)_{F \in \mathcal{E}_2}$ with no query variable occurring in inequalities to one nullary factor $S_\emptyset$ by aggregating away all query variables. $S_\emptyset()$ gives the probability of the conjunction of all factors without query variables in inequalities:

$$S_\emptyset() = \prod_{F \in \mathcal{E}_2 l} \left[1 - \prod_{\boldsymbol{x} \in \text{Dom}(\boldsymbol{X}_F)} \left(1 - R_F(\boldsymbol{x}_F)\right)\right]$$

This simplification reduces the set $\mathcal{E}_s$ of hyperedges to a new set $\mathcal{E}_u$ of unary edges, one per query variable in the inequalities, and one nullary edge: $\mathcal{E}_u = \{\emptyset\} \cup \bigcup_{\{i,j\} \in \mathcal{E}_\ell} \{\{i\}, \{j\}\}$. The simplification does not affect the inference problem: The probability of Q is the same as the probability of the query Q' over $\mathcal{E}_u \cup \mathcal{E}_\ell$:

$$Q'() \leftarrow \bigwedge_{F \in \mathcal{E}_u} S_F(\boldsymbol{X}_F) \wedge \bigwedge_{\{i,j\} \in \mathcal{E}_\ell} [X_i, X_j]^{\mathrm{T}} \cdot c_{i,j} \leq 0 \qquad (111)$$

The hypergraph of Q' trivially admits the relaxed tree decomposition whose structure is that of the inequality tree of Q' (and of Q): The skeleton edges are $\mathcal{E}_u$ and the ligament edges are $\mathcal{E}_\ell$.

The inference algorithm traverses the inequality tree bottom-up and eliminates one level of query variables at a time. For a variable X_p with children $X_{c_1}, \ldots, X_{c_k}$, it computes recursively the factor

$$Q_p(x_p) = S_p(x_p) \cdot \prod_{i \in [k]} S_{c_i}(\text{lub}_i(x_p)) + (1 - S_p(x_p)) \cdot Q_p(\text{lsub}_p(x_p))$$

We use $\text{lub}_i(x_p)$ to find the value in S_{c_i} that is the least upper bound of x_p and $\text{lsub}_p(x_p)$ to find the value in Q_p that is the least strict upper bound of x_p, i.e., the next value in ascending order. The definition of Q_p is recursive: It first computes the probability for x_p and then for its previous values. In case X_p has no children, i.e., $k = 0$ the product over S_{c_i} is one.

The probability of Q is then the product of $S_\emptyset$ and the probability of the first tuple in the factor of the root variable. If Q has a forest of inequality trees, then the subqueries for the trees would be disconnected and thus correspond to independent random variables. The probability of Q is then the product of the probabilities of the independent subqueries. □

The case of inequality graphs can be reduced to that of inequality trees by variable elimination. The elimination of a variable X_i repeatedly replaces it in the query by a value from its domain. The inequality graph of this residual query has no node for X_i and none of its edges. By removing k variables to obtain an inequality tree, the complexity of computing the query probability increases by at most the product of the sizes of the factors having these k variables.