PODS 2016 Best Paper

FAQ: Questions Asked Frequently

Mahmoud Abo Khamis
LogicBlox Inc., and
SUNY at Buffalo

Hung Q. Ngo
LogicBlox Inc., and
SUNY at Buffalo

Atri Rudra
Comp. Sci. and Eng.
SUNY at Buffalo

ABSTRACT

We define and study the **F**unctional **A**ggregate **Q**uery (FAQ) problem, which encompasses many frequently asked questions in constraint satisfaction, databases, matrix operations, probabilistic graphical models and logic. This is our main conceptual contribution.

We then present a simple algorithm called InsideOut to solve this general problem. InsideOut is a variation of the traditional dynamic programming approach for constraint programming based on variable elimination. Our variation adds a couple of simple twists to basic variable elimination in order to deal with the generality of FAQ, to take full advantage of Grohe and Marx's fractional edge cover framework, and of the analysis of recent worst-case optimal relational join algorithms.

As is the case with constraint programming and graphical model inference, to make InsideOut run efficiently we need to solve an optimization problem to compute an appropriate *variable ordering*. The main technical contribution of this work is a precise characterization of when a variable ordering is 'semantically equivalent' to the variable ordering given by the input FAQ expression. Then, we design an approximation algorithm to find an equivalent variable ordering that has the best 'fractional FAQ-width'. Our results imply a host of known and a few new results in graphical model inference, matrix operations, relational joins, and logic.

We also briefly explain how recent algorithms on beyond worst-case analysis for joins and those for solving SAT and #SAT can be viewed as variable elimination to solve FAQ over compactly represented input functions.

Keywords

Join algorithm; Quantified conjunctive query; Logic; Semiring; Probabilistic graphical model; Constraint satisfaction

PODS'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4191-2/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2902251.2902280

1. INTRODUCTION

1.1 Motivating examples

The following fundamental problems from three diverse domains share a common algebraic structure.

Example 1. (Matrix Chain Multiplication (MCM)) Given a series of matrices $\mathbf{A}_1, \dots, \mathbf{A}_n$ over some field $\mathbb{F}$, where the dimension of $\mathbf{A}_i$ is $p_i \times p_{i+1}$, $i \in [n]$, we wish to compute the product $\mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_n$. The problem can be reformulated as follows. There are $n+1$ variables $X_1, \dots, X_{n+1}$ with domains $\mathsf{Dom}(X_i) = [p_i]$, for $i \in [n+1]$. For $i \in [n]$, matrix $\mathbf{A}_i$ can be viewed as a function of two variables

$$\psi_{i,i+1} : \mathsf{Dom}(X_i) \times \mathsf{Dom}(X_{i+1}) \to \mathbb{F},$$

where $\psi_{i,i+1}(x, y) = (\mathbf{A}_i)_{xy}$. The MCM problem is to compute the output function

$$\varphi(x_1, x_{n+1}) = \sum_{x_2 \in \mathsf{Dom}(X_2)} \cdots \sum_{x_n \in \mathsf{Dom}(X_n)} \prod_{i=1}^{n} \psi_{i,i+1}(x_i, x_{i+1}).$$

Example 2. (Maximum A Posteriori (MAP) queries in probabilistic graphical models (PGM)) Consider a discrete graphical model represented by a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. There are n discrete random variables $\mathcal{V} = \{X_1, \dots, X_n\}$ on finite domains $\mathsf{Dom}(X_i)$, $i \in [n]$, and $m = |\mathcal{E}|$ *factors*

$$\psi_S : \prod_{i \in S} \mathsf{Dom}(X_i) \to \mathbb{R}_+, \ S \in \mathcal{E}.$$

A typical inference task is to compute the marginal MAP estimates, written in the form

$$\varphi(x_1, \dots, x_f) = \max_{x_{f+1} \in \mathsf{Dom}(X_{f+1})} \cdots \max_{x_n \in \mathsf{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S).$$

Example 3. (# Quantified Conjunctive Query (#QCQ)) Let Φ be a first order formula of the form

$$\Phi(X_1, \dots, X_f) = Q_{f+1} X_{f+1} \cdots Q_n X_n \left(\bigwedge_{R \in \mathsf{atoms}(\Phi)} R \right),$$

where $Q_i \in \{\exists, \forall\}$, for $i > f$. The #QCQ problem is to *count* the number of tuples in relation Φ on the free variables $X_1, \dots, X_f$. To reformulate #QCQ, construct a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows: $\mathcal{V}$ is the set of all variables $X_1, \dots, X_n$, and for each $R \in \mathsf{atoms}(\Phi)$ there is a hyperedge $S = \mathsf{vars}(R)$ consisting of all variables in R. The atom R can be viewed as a function indicating whether an assignment $\mathbf{x}_S$ to its variables is satisfied by the atom; namely $\psi_S(\mathbf{x}_S) = 1$ if $\mathbf{x}_S \in R$ and 0 otherwise.

Now, for each $i \in \{f+1,\dots,n\}$ we define an aggregate operator

$$\bigoplus^{(i)} = \begin{cases} \max & \text{if } Q_i = \exists, \\ \times & \text{if } Q_i = \forall. \end{cases}$$

Then, the #QCQ problem above is to compute the *constant* function

$$\varphi = \sum_{x_1 \in \mathsf{Dom}(X_1)} \cdots \sum_{x_f \in \mathsf{Dom}(X_f)} \bigoplus^{(f+1)}_{x_{f+1}\in\{0,1\}} \cdots \bigoplus^{(n)}_{x_n\in\{0,1\}} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S).$$

It turns out that these and dozens of other fundamental problems from constraint satisfaction (CSP), databases, matrix operations, PGM inference, logic, coding theory, and complexity theory can be viewed as special instances of a generic problem we call the **F**unctional **A**ggregate **Q**uery, or the FAQ problem, which we define next. The first two columns in Table 1 present eight of these problems. The full version of the paper [3] presents many more.

1.2 The FAQ problem

Throughout the paper, we use the following convention. Uppercase X_i denotes a variable, and lowercase x_i denotes a value in the domain $\mathsf{Dom}(X_i)$ of the variable. Furthermore, for any subset $S \subseteq [n]$, define

$$\mathbf{X}_S = (X_i)_{i\in S}, \qquad \mathbf{x}_S = (x_i)_{i\in S} \in \prod_{i\in S} \mathsf{Dom}(X_i).$$

In particular, $\mathbf{X}_S$ is a tuple of variables and $\mathbf{x}_S$ is a tuple of specific values with support S. The input to FAQ is a set of functions and the output is a function computed using a series of aggregates over the variables and input functions. More specifically, for each $i \in [n]$, let X_i be a variable on some discrete domain $\mathsf{Dom}(X_i)$, where $|\mathsf{Dom}(X_i)| \geq 2$. The FAQ problem is to compute the following function

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus^{(f+1)}_{x_{f+1}\in\mathsf{Dom}(X_{f+1})} \cdots \bigoplus^{(n)}_{x_n\in\mathsf{Dom}(X_n)} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S), \qquad (1)$$

where

- $\mathcal{H} = (\mathcal{V},\mathcal{E})$ is a multi-hypergraph. $\mathcal{V} = [n]$ is the index set of the variables X_i, $i \in [n]$. Overloading notation, $\mathcal{V}$ is also referred to as the set of variables.
- The set $F = [f]$ is the set of *free variables* for some integer $0 \le f \le n$. Variables in $\mathcal{V} - F$ are called *bound variables.*
- $\mathbf{D}$ is a fixed domain, such as $\{0,1\}$ or $\mathbb{R}^+$.
- For every hyperedge $S \in \mathcal{E}$, $\psi_S : \prod_{i\in S} \mathsf{Dom}(X_i) \to \mathbf{D}$ is an *input* function (also called a *factor*).
- For every bound variable $i > f$, $\oplus^{(i)}$ is a binary (aggregate) operator on the domain $\mathbf{D}$.
- And, for each bound variable $i > f$ either $\oplus^{(i)} = \otimes$ or $(\mathbf{D}, \oplus^{(i)}, \otimes)$ forms a commutative semiring [1] (with the same $\mathbf{0}$ and $\mathbf{1}$).

[1]A triple $(\mathbf{D},\oplus,\otimes)$ is a *commutative semiring* if $\oplus$ and $\otimes$ are commutative binary operators over $\mathbf{D}$ satisfying the following: (1) $(\mathbf{D},\oplus)$ is a commutative monoid with an additive identity, denoted by $\mathbf{0}$. (2) $(\mathbf{D},\otimes)$ is a commutative monoid with a multiplicative identity, denoted by $\mathbf{1}$. (In the usual semiring definition, we do not need the multiplicative monoid to be commutative.) (3) $\otimes$ distributes over $\oplus$. (4) For any element $e \in \mathbf{D}$, we have $e \otimes \mathbf{0} = \mathbf{0} \otimes e = \mathbf{0}$.

If $\oplus^{(i)} = \otimes$, then $\oplus^{(i)}$ is called a *product aggregate*; otherwise, it is a *semiring aggregate.* (We assume that there is at least one semiring aggregate.)

Because for $i > f$ every variable X_i has its own aggregate $\bigoplus^{(i)}$ over all values $x_i \in \mathsf{Dom}(X_i)$, in the rest of the paper we will write $\bigoplus^{(i)}_{x_i}$ to mean $\bigoplus^{(i)}_{x_i\in\mathsf{Dom}(X_i)}$.

We will often refer to φ as an FAQ-*query.* We use FAQ-SS (for FAQ on *single semiring*) to denote the special case of FAQ when there is only *one* variable aggregate, i.e. $\oplus^{(i)} = \oplus, \forall i > f$, and $(\mathbf{D},\oplus,\otimes)$ is a semiring.

Due to space limitation, in this extended abstract we will assume that the input factors are given using the *listing representation*, i.e. each factor is a table of all tuples of the form $\langle \mathbf{x}_S, \psi_S(\mathbf{x}_S)\rangle$, such that $\psi_S(\mathbf{x}_S) \neq \mathbf{0}$. (Entries not in the table are $\mathbf{0}$-entries.) This representation is commonly used in the CSP, databases, and sparse matrix computation domains. It should be noted that our results can handle even more succinct representations such as GDNFs and decision diagrams [15] in CSP literature and algebraic decision diagrams in the PGM literature [8]. (See [3] for details.)

1.3 Paper organization

Section 2 summarizes the contributions of the paper and sketches the line of attack. Related works are discussed in Section 3. Section 4 discusses the main ideas behind InsideOut and states its runtime *given* a variable ordering. Section 5 presents most of our technical details on computing a good variable ordering for a special case of FAQ where for every $i > f$, $\oplus^{(i)}$ is a semiring aggregate. This special case illustrates many of the key ideas and concepts. Unfortunately, due to space limitation we have to leave out the analytical details of the most general case of FAQ. All the missing details can be found in [3].

2. SUMMARY OF CONTRIBUTIONS

2.1 Conceptual contribution

The formulation of FAQ has its roots in the SumProd problem formulated by Dechter [20] who solved it using variable elimination. The SumProd problem is *exactly* FAQ-SS. A later work by Aji and McEliece [5] also addressed SumProd, which they renamed marginalization over semirings.

FAQ substantially generalizes FAQ-SS, as FAQ can now capture problems in logic such as QCQ (quantified conjunctive query) or #QCQ (sharp quantified conjunctive query). We argue that FAQ is a very powerful way of thinking about these problems and related issues. FAQ can be thought of as a declarative query language over functions. For example, we show in [3] how different input representations can vastly affect the landscape of tractability of the problem, and how the output representation is related to the notion of factorized databases [43].

2.2 Algorithmic contribution

We present a single algorithm, called InsideOut, to solve FAQ. InsideOut is a variation of the variable elimination algorithm [20, 54, 53]. In PGM, variable elimination was first proposed by Zhang and Poole [53]. Then Dechter [20] observed that this strategy can be applied to problems on other semirings such as constraint satisfaction and SAT solving. In the database literature, Yannakakis' algorithm [52] can also

Problem	FAQ formulation	Previous Algo.	**Our Algo.**
#QCQ	$\sum_{(x_1,\dots,x_f)} \bigoplus^{(f+1)}_{x_{f+1}} \cdots \bigoplus^{(n)}_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	No non-trivial algo	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
QCQ	$\bigoplus^{(f+1)}_{x_{f+1}} \cdots \bigoplus^{(n)}_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	$\tilde{O}(N^{\mathsf{PW}(\mathcal{H})} + Z)$ [14]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
#CQ	$\sum_{(x_1,\dots,x_f)} \max_{x_{f+1}} \cdots \max_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{DM}(\mathcal{H})} + Z)$ [23]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
Joins	$\bigcup_{\mathbf{x}} \bigcap_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}\left(N^{\mathsf{fhtw}(\mathcal{H})} + Z\right)$ [28]	$\tilde{O}\left(N^{\mathsf{fhtw}(\mathcal{H})} + Z\right)$
Marginal Distribution	$\sum_{(x_{f+1},\dots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{htw}(\varphi)} + Z)$ [32]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
MAP query	$\max_{(x_{f+1},\dots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{htw}(\varphi)} + Z)$ [32]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
Matrix Chain Mult.	$\sum_{x_2,\dots,x_n} \prod_{i=1}^{n} \psi_{i,i+1}(x_i, x_{i+1})$	DP bound [18]	DP bound
DFT	$\sum_{(y_0,\dots,y_{m-1})\in\mathbb{Z}_p^m} b_y \cdot \prod_{0\le j+k<m} e^{i2\pi \frac{x_j \cdot y_k}{p^{m-j-k}}}$	$O(N \log_p N)$ [17]	$O(N \log_p N)$

Table 1: Runtimes of algorithms assuming optimal variable ordering is given. Problems shaded red are in CSPs and logic ($\mathbf{D} = \{0, 1\}$ for CSP and $\mathbf{D} = \mathbb{N}$ for #CSP), problems shaded green fall under PGMs ($\mathbf{D} = \mathbb{R}_+$), and problems shaded blue fall under matrix operations ($\mathbf{D} = \mathbb{C}$). N denotes the size of the largest factor (assuming they are represented with the listing format). $\mathsf{htw}(\varphi)$ is the notion of integral cover width defined in [32] for PGM. $\mathsf{PW}(\mathcal{H})$ is the *optimal width of a prefix graph* of $\mathcal{H}$ from [14] and $\mathsf{DM}(\mathcal{H}) = \text{poly}(F\text{-}\mathsf{ss}(\mathcal{H}), \mathsf{fhtw}(\mathcal{H}))$, where F-$\mathsf{ss}(\mathcal{H})$ is the $[f]$-quantified star size [23]. Z is the output size in listing representation. Our width $\mathsf{faqw}(\varphi)$ is never worse than any of the three and there are classes of queries where ours is unboundedly better than all three. In DFT, $N = p^m$ is the length of the input vector. $\tilde{O}$ hides a logarithmic factor in data complexity and polynomial factor in query complexity.

be cast as variable elimination under the set semiring or Boolean semiring.[2]

InsideOut adds three minor twists to the basic variable elimination strategy. **First**, we use a backtracking-search strategy called OutsideIn to compute the intermediate results. This strategy allows us to use the *analysis* of recent worst-case optimal join algorithms [51, 40, 41] to bound the time it takes to compute the intermediate results using the fractional edge cover bound [7, 28]. **Second**, we introduce the idea of a **01**-*projection* of a function onto a support to obtain the fractional hypertree width style of runtime guarantee [29]. **Third**, in addition to making use of the distributive law to 'fold' common factors [5] when we face a semiring aggregate, we apply a swap between an aggregate and the inside product when that aggregate is also a product.

We show that InsideOut runs in time $\tilde{O}(N^{\mathsf{faqw}(\sigma)} + Z)$, where σ is a variable ordering that we choose to run the algorithm on, N is the input size, and Z is the output size (under the 'listing representation' of input and output factors), and $\mathsf{faqw}(\sigma)$ is a parameter called the (fractional) FAQ-*width* of σ. FAQ-width is the FAQ-analog of the induced fractional edge cover width of a variable ordering. (See Definition 2.)

In fact, we show (in [3]) that the variable elimination framework is still powerful in cases when the fractional hypertree width bounds are no longer applicable. These are special cases of FAQ where the input functions are compactly represented. In particular, we explain how – with a suitable modification – InsideOut can be used to recover recently known beyond worst-case results in join algorithms (Minesweeper [39], and Tetris [2]), and results on the tractability of SAT and #SAT for β-acyclic formulas [44, 11].

On the practical side, preliminary implementations of join algorithms based on fractional hypertree width for counting graph patterns have been very encouraging, as they are faster than existing commercial systems by at least an order of magnitude [1, 42] for selected queries. Far beyond graph patterns, we have implemented InsideOut within the LogicBlox database system [6] with great performance results.

2.3 Main technical contributions

The key technical problem is choosing a σ that minimizes $\mathsf{faqw}(\sigma)$. This is where the generality of FAQ requires new ideas. Traditional variable elimination for CSPs or PGM inference also requires computing a good variable ordering to minimize the *treewidth* [35] or *fractional hypertree width* [29] of the variable ordering. However, in those cases all variable orderings are valid. Hence, all we have to do in this traditional setting is to compute a tree decomposition whose maximum bag size (or maximum fractional edge cover number over the bags) is minimized, then the GYO-elimination procedure will produce a good variable ordering (see [3]). In the general setting of FAQ, just like in logic where there are alternating quantifiers, the set of semantically equivalent variable orderings depends on both the input query expression and the structure of the query hypergraph.

To see what equivalent ordering could mean, consider the following. A natural class of valid permutations to consider are those that only permute aggregates in a maximal block of identical aggregates in the query expression. However, taking the query hypergraph into account, one can do much

[2]It is well-known [5, 35] that variable elimination and message passing are equivalent in the special case of FAQ-SS.

better. Consider, for example, the FAQ-query

$$\varphi = \max_{x_1} \sum_{x_2} \max_{x_3} \cdots \sum_{x_{2k}} \psi_{\{1,3,\dots,2k-1\}} \psi_{\{2,4,\dots,2k\}}$$

where both factors have range $\mathbb{R}_+$. In this case, even though max and $\sum$ do not commute with one another, we can rewrite φ using any of the $(2k)!$ variable orderings and still obtain the same result. (The aggregates have to be permuted along with the variables to which they are attached.)

Even for the special case of FAQ-SS, where there is only one type of semiring aggregates hence all permutations are valid, computing the optimal variable ordering is already **NP**-hard in query complexity, because computing the (fractional hyper) treewidth of the query hypergraph is **NP**-hard. Hence, the extra complication of only considering 'valid' orderings for FAQ seems to make our task much harder. Somewhat surprisingly, we are able to show that the complexity of computing the optimal ordering for general FAQ is essentially the same as the complexity of computing the optimal ordering for FAQ-SS instances. As a brief summary, we have the following technical contributions:

- Given an FAQ-query φ we define the set $\mathsf{EVO}(\varphi)$ of all variable orderings *semantically equivalent* to φ. Roughly, for any $\sigma \in \mathsf{EVO}(\varphi)$, if we rewrite the expression for φ using the ordering σ (with all aggregates permuted along), then we obtain a function identical to φ, **no matter** what the input factors are. The FAQ-width of φ is $\mathsf{faqw}(\varphi) = \min_{\sigma\in\mathsf{EVO}(\varphi)} \mathsf{faqw}(\sigma)$.

- We describe how to (in poly-time, query complexity) construct an *expression tree* for the input FAQ-query. The expression tree induces a partially ordered set on the variables called the *precedence poset*. By defining a notion called *component-wise equivalence*, we completely characterize $\mathsf{EVO}(\varphi)$: $\sigma \in \mathsf{EVO}(\varphi)$ if and only if it is component-wise equivalent to some linear extension of the precedence poset.

- To prepare for the approximation algorithm for $\mathsf{faqw}(\varphi)$, we show that going through all orderings in $\mathsf{EVO}(\varphi)$ is *not* necessary. The set of linear extensions of the precedence poset is sufficient for this purpose: every linear extension of the precedence poset is semantically equivalent to φ, and every $\sigma \in \mathsf{EVO}(\varphi)$ has the same FAQ-width as some linear extension of this poset.

- Finally, using an approximation algorithm for the fractional hypertree width (fhtw) with approximation ratio $g(\mathsf{fhtw})$ as a blackbox,[3] and using the expression tree as a guide, we give an approximation algorithm computing an ordering σ such that $\mathsf{faqw}(\sigma) \leq \mathsf{faqw}(\varphi) + g(\mathsf{faqw}(\varphi))$.

2.4 Highlights of our corollaries

In light of the fact that many problems can be reduced to FAQ, Table 1 presents a selected subset of corollaries that our results imply.[4] We list the results assuming the optimal variable ordering is already given. (This holds true for both known results and our results.) When the optimal variable ordering is not given, the exponents of N in all cases have to be changed to the best known approximating factors for the corresponding width. In the FAQ case, that would be $O(\mathsf{faqw}^3(\varphi))$.

For each problem, the table lists the corresponding FAQ instance, the runtime of the previously best known algorithm, and the runtime of InsideOut. These are the problems that we would like to highlight, as they yield either new results or an alternative interpretation of known results in the FAQ framework.

The results in Table 1 roughly span three areas: (1) CSPs and Logic; (2) PGMs and (3) Matrix operations. Except for joins, problems in area (1) need the full generality of our FAQ formulation, where InsideOut either improves upon existing results or yields new results. Problems in area (2) can already be reduced to FAQ-SS. Here, InsideOut improves upon known results since it takes advantage of Grohe and Marx's more recent fractional hypertree width bounds. Finally, problems in area (3) of Table 1 are classic. InsideOut does not yield anything new here, but it is intriguing to be able to explain the textbook dynamic programming algorithm for Matrix-Chain Multiplication [18] as an algorithm to find a good variable ordering for the corresponding FAQ-instance. The DFT result is a re-writing of Aji and McEliece's observation [5].[5]

It should be noted that the prior results on #CQ [23] and QCQ [14] focused on dichotomy theorems for bounded-arity classes of input hypergraphs, not just on the best possible runtime one can get. Our faqw notion is a generalization of fractional hypertree width, which steps into the unbounded arity world. See Marx [38] for a more detailed discussion of known results on CSP in the unbounded-arity case.

3. RELATED WORK

Since FAQ encompasses so many areas, our related work discussion is necessarily incomplete. The full version [3] has pointers to many more related works.

3.1 Problems on one semiring

As was mentioned in the introduction, the FAQ-SS problem was explicitly defined by Aji and McEliece [5] who called it the MPF problem (for Marginalize the product function). They presented a message passing algorithm for FAQ-SS and essentially showed that their algorithm meets the treewidth bound. Their paper also lists a number of problems that are FAQ-SS instances, including Matrix Chain Multiplication (less specific than our result, they just argue that essentially different variable orderings give rise to different ways of parenthesizing the matrix chain multiplication) and Matrix Vector Multiplication. They showed that their general algorithm contains FFT as a special case. We re-phrase their interpretation of the FFT using InsideOut. They also showed that many basic decoding problems in coding theory can be cast as FAQ-SS instances.

Kohlas and Wilson [34] presented even more applications of the FAQ-SS problem. The paper categorized various existing message passing algorithms depending on what extra properties they need beyond $(\mathbf{D}, \oplus, \otimes)$ being a commutative semiring. Their paper also explored algorithms for approximate computations (while in this work we solely deal with

[3]The best known such algorithm due to Marx [36] has $g(\mathsf{fhtw}) = O(\mathsf{fhtw}^3)$

[4]The results listed in the table implicitly assumed the listing representation of input factors.

[5]Note that we have further results on matrix vector multiplication for structured matrices (see [3]).

exact computation). Approximate computations in PGMs have been explored under the semiring framework [48].

Most of the results in the PGM literature present algorithms that are shown to obtain the treewidth bound. To the best of our knowledge, the finest notion of hypergraph width used to bound the performance of algorithms is the integral hypertree width bound of [26], which appeared in [32, 21]. See Section 3.3 and [3] for a more detailed discussion on the various notions of widths.

In the database literature, recently Koch [33] described an algebraic query language called AGCA over 'rings of databases' which is somewhat similar in spirit to FAQ. This framework makes use of additive inverses to allow for efficient view maintenance.

3.2 Factorized databases

Bakibayev et al. [9] and Olteanu and Závodný [43] introduced the notion of *factorized databases*, and showed how one can efficiently compute join and aggregates over factorized databases. In hindsight there is much in common between their approach and InsideOut applied to the single semiring case of FAQ-SS. Both approaches have the same runtime complexity, because both are dynamic programming algorithms, InsideOut is bottom-up, and factorized database computation is top-down (memoized).

The FAQ framework is more general in that it can handle multiple aggregate types. Our contribution also involves the characterization of EVO and an approximation algorithm for faqw. On the other hand, aspects of factorized database that FAQ does not handle include the evaluation of SQL queries and output size bounds on the factorized representations.

3.3 Notions of widths

Various notions of hypergraph 'widths' have been developed over the years in PGM, CSP, and database theory. In particular, two often-used properties of the input query are *acyclicity* and *bounded width.*

When the query is acyclic, the classic algorithm of Yannakakis [52] for relational joins (and CSPs) runs in time linear in the input plus output size, modulo a log factor. Similarly, Pearl's belief propagation algorithm [45] works well for acyclic graphical models. As we briefly touch upon in [3], Yannakakis' algorithm is essentially belief propagation on the Boolean semiring or set semiring. The algorithm can also be reinterpreted using InsideOut.

Subsequent works on databases and CSPs further expand the classes of queries that can be evaluated in polynomial time. These works define progressively more general notions of width for a query, which intuitively measure how far a query is from being acyclic. Roughly, these results state that if the corresponding notion of 'width' is bounded by a constant, then the query is 'tractable,' i.e. there is a polynomial-time algorithm to evaluate it. For example, Gyssens et al. [30, 31] showed that queries with bounded *degree of acyclicity* are tractable. Then came *query width* (qw) from Chekuri and Rajaraman [12], *hypertree width* and *generalized hypertree width* (ghw) from Gottlob et al. [50, 26], and *fractional hypertree width* from Grohe and Marx [29]. More recently, Marx developed the most general width notions known to date, called *adaptive width* and *submodular width* [37, 38].

In the PGM literature, the most common notion is *treewidth* as the textbook variable elimination and message passing algorithms are often stated to run in time $O(N^{w+1})$ where w is the tree width of the model [35]. Freuder [24] and Dechter and Pearl [22] showed in late 1980s that CSP instances with bounded treewidth are tractable.

In the logic/finite model theory literature, several notions of widths were also developed [4, 23, 14]. We describe them in the relevant sections of the full version [3].

3.4 Finite model theory

In [47], Pichler and Skritek studied the #CQ problem in the special case where the query is acyclic. We refer to this special case as #ACQ. In particular, they showed that #ACQ is tractable in data complexity (i.e. when the number of variables that we are counting over is a constant) and in query complexity (i.e. when all relations have constant sizes) but not in combined complexity where the problem turns out to be #**P**-complete.

In [23], Durand and Mengel introduced a new parameter for #CQ called the *quantified star size*. It is basically a measure of how free variables are connected in the query's hypergraph. Along with bounded generalized hypertree width (or fractional hypertree width), bounded quantified star size characterizes the classes of #CQ instances that are tractable in the bounded arity (or bounded generalized hypertree width) case.

The quantified star size idea has been expanded later by applying it to the *core* of the query instead of the original query [27]. The core is a minimal subquery that is homomorphic to the original query. Because homomorphism does not preserve counts (i.e. it is not a one-to-one mapping), free variables have to be explicitly preserved by taking the core of the *color query* of the original query. Further development along with new lower bounds appeared in [16].

QCQ (second row of Table 1) has its own long line of research. An early and interesting result was the tractability of QCQ when the domain size, treewidth, and number of quantifier alternations are all constants [13]. More recently, Chen and Dalmau introduced a notion of width for QCQ based on elimination orderings [14]. In particular, they take the minimum width over some variable orderings that are equivalent to the original query. They showed the tractability of QCQ when their width is bounded.

The runtime of InsideOut for CQ, #CQ, and QCQ are unboundedly better than the above results as shown in Table 1. Our result on #QCQ is new: to the best of our knowledge no non-trivial efficient algorithms for #QCQ were known prior to our work. Essentially, InsideOut is able to unify the above results under the same umbrella.

4. THE ALGORITHM

4.1 Notation

Recall that $\mathbf{0}$ is the additive identity of the semiring(s) which also annihilates any element of $\mathbf{D}$ under multiplication. Unless specified otherwise, we also assume that there are n variables and m factors. Given any Boolean predicate P, define the *Kronecker delta* δ_P to be $\delta_P = 1$ if P holds and 0 otherwise. (Note that these are numbers 1 and 0, not the multiplicative or additive identity.)

Let $W \subseteq [n]$ be some subset of variables, and $\mathbf{y}_W \in \prod_{i\in W} \mathsf{Dom}(X_i)$ be some given value tuple. The *conditional factor* $\psi_S(\cdot \mid \mathbf{y}_W)$ is a function from $\prod_{i\in S} \mathsf{Dom}(X_i)$ to $\mathbf{D}$

defined by

$$\psi_S(\mathbf{x}_S \mid \mathbf{y}_W) = \begin{cases} \mathbf{0} & \text{if } S \cap W \neq \emptyset \text{ and } \mathbf{x}_{S\cap W} \neq \mathbf{y}_{S\cap W} \\ \psi_S(\mathbf{x}_S) & \text{otherwise.} \end{cases}$$

For each factor ψ_S, define its *size* to be the number of non-zero points under its domain:

$$|\psi_S| := \left|\{\mathbf{x}_S \mid \psi_S(\mathbf{x}_S) \neq \mathbf{0}\}\right|.$$

We often use N to denote the sum of all input factor sizes.

4.2 The algorithmic building block

Consider the FAQ-SS form of the FAQ expression (1) when there is no free variable and all aggregates are the same semiring aggregate. In this case, we write the expression as

$$\varphi = \bigoplus_{\mathbf{x}} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S) = \bigoplus_{x_1 \in \mathsf{Dom}(X_1)} \left(\bigoplus_{\mathbf{x}_{[n]-\{1\}}} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S \mid x_1) \right).$$

We can evaluate this expression by going through each value of x_1 and computing the inner expression 'conditioned' on this x_1. The naïve implementation of this strategy wastes time if there is any x_1 for which *some* conditional factor $\psi_S(\cdot \mid x_1)$ is identically $\mathbf{0}$. Thus, the obvious idea is to first compute the set I_1 of values x_1 for which $\psi_S(\cdot \mid x_1) \not\equiv \mathbf{0}$ *for all* factors ψ_S. Then, recursively compute the expression

$$\varphi = \bigoplus_{x_1 \in I_1} \left(\bigoplus_{\mathbf{x}_{[n]-\{1\}}} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S \mid x_1) \right).$$

This is called the OutsideIn algorithm, as it evaluates the expression from the outer-most aggregate to the inner-most. It serves as the algorithmic building block of InsideOut.

Given that the input factors are represented using the listing format (i.e. only non-$\mathbf{0}$ entries are listed), computing the above expression recursively is a join algorithm in disguise, and any of the algorithms from [40, 51, 41] works. In fact, the OutsideIn algorithm works even if there were free variables. Using the Hölder inequality analysis approach from [41], we can easily show the following. (See [3] for more details including the proofs.)

THEOREM 4.1. *Let $(\lambda_S)_{S\in\mathcal{E}}$ be any fractional edge cover of the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with m edges and n vertices. Algorithm* OutsideIn *applied to* FAQ-SS *runs in time*

$$\tilde{O}(mn \prod_{S\in\mathcal{E}} |\psi_S|^{\lambda_S}).$$

The expression $\prod_{S\in\mathcal{E}} |\psi_S|^{\lambda_S}$ with an optimal choice of the fractional edge cover $(\lambda_S)_{S\in\mathcal{E}}$ is called the AGM-*bound* for the query [7].

OutsideIn is backtracking search [19, 25] which was known 50 years ago in the AI and constraint programming world. In the PGM literature, the method of conditioning search is similar, but the main theoretical objective is so that the conditioning graph is acyclic [46]. The main advantage of backtracking search is that it requires very little extra space. The main disadvantage is that it might have to resolve the same subproblem multiple times. The duality between backtracking search and dynamic programming is well-known in the constraint programming literature [49]. Next section explores the other side of this duality: the dynamic programming side.

4.3 The InsideOut algorithm

4.3.1 *The* FAQ-SS *case*

To describe the intuition, we first explain InsideOut for the special instance FAQ-SS (or SumProd). The idea behind variable elimination [20, 54, 53] is to 'fold' common factors, exploiting the distributive law:

$$\bigoplus_{x_{f+1}} \cdots \bigoplus_{x_n} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$$

$$= \bigoplus_{x_{f+1}} \cdots \bigoplus_{x_{n-1}} \bigotimes_{S\in\mathcal{E}-\partial(n)} \psi_S(\mathbf{x}_S) \otimes \underbrace{\left(\bigoplus_{x_n} \bigotimes_{S\in\partial(n)} \psi_S(\mathbf{x}_S) \right)}_{\text{new factor } \psi_{U_n-\{n\}}},$$

where the equality follows from the fact that $\otimes$ distributes over $\oplus$, $\partial(n)$ denotes all edges incident to n in $\mathcal{H}$ and $U_n = \cup_{S\in\partial(n)} S$. Assume for the moment that we can somehow efficiently compute the intermediate factor $\psi_{U_n-\{n\}}$. Then, the resulting problem is another instance of FAQ-SS on a modified multi-hypergraph $\mathcal{H}_{n-1}$, constructed from $\mathcal{H}$ by removing vertex n along with all edges in $\partial(n)$, and *adding back* a new hyperedge $U_n - \{n\}$. Recursively, we continue this process until all variables $X_n, \dots, X_{f+1}$ are eliminated. Textbook treewidth-based results for PGM inference are obtained this way [35]. In the database context (i.e. given an FAQ-query over the Boolean semiring), the intermediate result $\psi_{U_n-\{n\}}$ is essentially an intermediate materialized relation of a query plan.

While correct, basic variable elimination as described above is potentially not very efficient for sparse input factors, i.e. factors whose sizes are smaller than the product of the domain sizes. The main reason is that the product that was factored out might annihilate many entries of the intermediate result $\psi_{U_n-\{n\}}$, while we have spent so much time computing $\psi_{U_n-\{n\}}$. For example, for an $S \notin \partial(n)$ and tuple $\mathbf{y}_S$ such that $S \subseteq U_n$ and $\psi_S(\mathbf{y}_S) = \mathbf{0}$, we do not need to compute the entries $\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}})$ for which $\mathbf{y}_S = \mathbf{x}_S$: those entries will be killed later anyhow. The idea is then to only compute those $\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}})$ values that will 'survive' the other factors. One simple way to achieve this would be to compute the following factor instead of $\psi_{U_n-\{n\}}$:

$$\psi'_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}}) = \bigoplus_{x_n} \left[\left(\bigotimes_{S\in\partial(n)} \psi_S(\mathbf{x}_S) \right) \otimes \left(\bigotimes_{\substack{S\notin\partial(n),\\ S\cap U_n\neq\emptyset}} \bigoplus_{\mathbf{x}_{S-U_n}} \psi_S(\mathbf{x}_S) \right) \right],$$

where $\bigoplus_{\mathbf{x}_{S-U_n}} \psi_S(\mathbf{x}_S)$ results from $\psi_S(\mathbf{x}_S)$ by marginalizing away variables in $S - U_n$. If $\mathbf{D} = \{0, 1\}$ or $\{\mathsf{true}, \mathsf{false}\}$, then the above computation of $\psi'_{U_n-\{n\}}$ is enough; indeed, this idea is precisely what is behind relational join and CSP algorithms achieving the *fractional hypertree width bound* [29]. However, for a general range $\mathbf{D}$ as in PGM inference or an aggregate query in a database, the change above can produce an incorrect result because the same $\psi_S(\mathbf{x}_S)$ can participate in the product multiple times during different elimination steps.

Our proposed fix is very simple: instead of multiplying with the actual value of $\psi_S(\mathbf{x}_S)$ for $S \notin \partial(n)$, we only multiply with an indicator factor that checks if $\psi_S(\mathbf{x}_S)$ is $\mathbf{0}$ or not. More precisely, let $\psi_{S/U_n}(\mathbf{x}_S)$ be the *projection* of ψ_S to variables in $S \cap U_n$ where all the non-zero entries become $\mathbf{1}$. (See Definition 9 in the appendix.) Armed with this new indicator factor, InsideOut computes the following factor when marginalizing X_n away:

$$\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}}) = \bigoplus_{x_n} \left[\left(\bigotimes_{S \in \partial(n)} \psi_S(\mathbf{x}_S) \right) \otimes \left(\bigotimes_{\substack{S \notin \partial(n), \\ S \cap U_n \neq \emptyset}} \psi_{S/U_n}(\mathbf{x}_{S \cap U_n}) \right) \right] \quad (2)$$

and similarly for eliminating the remaining variables.

4.3.2 The general case

The above strategy does not care if the variable aggregates where the same or different: As long as $(\mathbf{D}, \oplus^{(n)}, \otimes)$ is a semiring, we can fold the common factors. Thus, InsideOut works almost as is for a general FAQ instance (as opposed to FAQ-SS).

We are left to describe how InsideOut deals with the case when $\oplus^{(n)} = \otimes$. In this case, the simple idea is to swap the two (identical) operators:

$$\begin{aligned} & \varphi(\mathbf{x}_{[f]}) \\ = & \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigoplus_{x_n}^{(n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S) \\ = & \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{x_n \in \mathsf{Dom}(X_n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S) \\ = & \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{S \in \mathcal{E}} \bigotimes_{x_n \in \mathsf{Dom}(X_n)} \psi_S(\mathbf{x}_S) \\ = & \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{S \notin \partial(n)} (\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|} \bigotimes_{S \in \partial(n)} \bigotimes_{x_n} \psi_S(\mathbf{x}_S). \end{aligned}$$

This results in a *product marginalization* of individual factors ψ_S for $S \in \partial(n)$. In particular, for each $S \in \partial(n)$, we compute the factor with support $S - \{n\}$ defined by

$$\psi_{S-\{n\}}(\mathbf{x}_{S-\{n\}}) = \bigotimes_{x_n \in \mathsf{Dom}(X_n)} \psi_S(\mathbf{x}_S).$$

This can be computed in linear time in $|\psi_S|$. The product marginalization step is algorithmically much easier because it does not create the intermediate factor $\psi_{U_n-\{n\}}$.

As for $S \notin \partial(n)$, we replace ψ_S by the power factor

$$\psi'_S(\mathbf{x}_S) = (\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|},$$

which can be done in linear time with a log $|\mathsf{Dom}(X_n)|$ blowup using the repeated squaring algorithm. Note the key fact that this power is with respect to the product aggregate $\otimes$. In most (if not all) applications of FAQ, there is one additional property: most of the time, $\otimes$ is an idempotent operator over the effective domain. For example, in the #QCQ problem $\otimes$ is the usual product operator and the domain that it aggregates over is $\{0, 1\}$ (before there is a sum outside). In this case, $\psi'_S(\mathbf{x}_S) = (\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|} = \psi_S(\mathbf{x}_S)$, and we do not need to spend the linear nor log-blowup time.

The definition of what it means for an operator to act idempotently on a domain and its relationship to the position of the operator in the FAQ-formula are subtle. We leave the details to [3]. Finally, note that in this case we recurse on the hypergraph $\mathcal{H}_{n-1} = \mathcal{H} - \{n\}$.

Thus far we left open the question of how to compute $\psi_{U_n-\{n\}}$ efficiently, assuming $\oplus^{(n)}$ is a semiring aggregate. We simply apply the OutsideIn algorithm from Section 4.2. Note that computing $\psi_{U_n-\{n\}}$ is an instance of FAQ with $|U_n| - 1$ free variables.

4.3.3 Output representation

Let $\mathcal{E}_f$ be the set of supports of factors remaining after eliminating the non-free variables. The output to FAQ is now the expression

$$\varphi(\mathbf{x}_{[f]}) = \bigotimes_{S \in \mathcal{E}_f} \psi_S(\mathbf{x}_S), \quad (3)$$

which is an FAQ instance with all free variables. We can compute the output directly by running OutsideIn on (3). However, inspired by Olteanu and Závodný [43] we can first compute the output in the factorized representation, and then report it.

One way to think about this idea is to first compute the set of tuples $\mathbf{x}_{[f]}$ for which $\varphi(\mathbf{x}_{[f]}) \neq \mathbf{0}$, and then for each such tuple compute its value $\varphi(\mathbf{x}_{[f]})$ using (3) by multiplying the corresponding ψ_S. Computing the non-$\mathbf{0}$ output tuples *is* a join problem, which in the FAQ setting is the $(\cup, \cap)/(\vee, \wedge)$ semiring, with the runtime $\tilde{O}(N^{\mathsf{fhtw}} + Z)$ as shown in Table 1.

Implementation-wise, we can use the **01**-projection idea from Section 4.3.1 as follows. For $k = f, \ldots, 1$, we will eliminate the free variable X_k and use OutsideIn to compute the factors $\psi_{U_k}, \psi_{U_k-\{k\}}$ defined as

$$\begin{aligned} \psi_{U_k}(\mathbf{x}_{U_k}) &:= \bigotimes_{S : S \cap U_k \neq \emptyset} \psi_{S/U_k}(\mathbf{x}_{S \cap U_k}), \\ \psi_{U_k-\{k\}}(\mathbf{x}_{U_k-\{k\}}) &:= \bigoplus_{x_k}^{(k)} \psi_{U_k}(\mathbf{x}_{U_k}), \end{aligned}$$

where $\bigoplus_{x_k}^{(k)}$ is the **01**-OR (See Defintion 10). Finally, we run OutsideIn on the following expression instead of (3).

$$\varphi(\mathbf{x}_{[f]}) = \left(\bigotimes_{S \in \mathcal{E}_f} \psi_S(\mathbf{x}_S) \right) \otimes \left(\bigotimes_{k \in [f]} \psi_{U_k}(\mathbf{x}_{U_k}) \right). \quad (4)$$

Eliminating free variables and then recovering them back by (4) are equivalent to the two phases of Yannakakis algorithm [52]. Our implementation of InsideOut in LogicBlox follows the elimination/recovery method.

4.3.4 Runtime analysis

The runtime of InsideOut is the sum of the runtimes of n variable elimination steps plus the time needed to report the output at the end. For $k = n, ..., 1$, the cost of the kth step is analyzed as follows. If $k > f$ and the aggregate $\bigoplus^{(k)}$ is a semiring aggregate, or if $k \leq f$, then the runtime is dominated by the OutsideIn algorithm's runtime to compute the intermediate factor $\psi_{U_k-\{k\}}$. From Theorem 4.1, this is bounded by the AGM-bound on the set U_k, denoted by $\mathsf{AGM}(U_k)$. On the other hand, if $k > f$ and $\bigoplus^{(k)} = \bigotimes$, then the runtime is linear in the input size modulo a log factor. The final invocation of OutsideIn on (4) reports the output (in listing representation) in linear time, just like the output phase of Yannakakis algorithm.

From there, with a bit of care we can write down a precise expression for the runtime of InsideOut. Minimizing the resulting (somewhat complicated) expression leads to the dynamic programming algorithm for the MCM problem and the FFT algorithm for the DFT. See [3] for details.

One way to simplify the runtime expression is to upper-bound each bound $\mathsf{AGM}(U_k)$ by the fractional edge cover number of the set U_k, i.e. $\mathsf{AGM}(U_k) \leq N^{\rho^*_{\mathcal{H}}(U_k)}$ [29].[6] Let

[6]For any subset $B \subseteq \mathcal{V}$ of vertices, $\rho^*_{\mathcal{H}}(B)$ is the optimal ob-

K denote the indices $f < i \leq n$ such that $\oplus^{(i)} \neq \otimes$. Then, it follows that the algorithm runs in time

$$\tilde{O}\left(N^{\max_{k\in K\cup[f]} \rho^*_{\mathcal{H}}(U_k)} + Z\right), \qquad (5)$$

where Z is the output size in listing representation. In the above discussion, we eliminated the variables in order $X_n, X_{n-1}, \dots, X_1$. However, there is no reason to force InsideOut to follow this particular order. In particular, there might be a different variable ordering for which expression (5) is *a lot* smaller and the algorithm still works correctly on that ordering. (See [3] for examples.)

This is where the main technical contributions of this paper begin. We need to answer the following questions:

1. How do we know which variable orderings are equivalent to the original FAQ-query expression?

2. How do we find the best variable ordering among all equivalent variable orderings?

The next section formalizes the above two questions.

4.4 Fractional width of an FAQ-query

We first define precisely the notion of 'semantic equivalence' between two variable orderings.

Definition 1. (φ-equivalent variable ordering) Let φ be an FAQ-query written in the form (1). A *φ-equivalent variable ordering* is a vertex ordering $\sigma = (\sigma(1), \dots, \sigma(n))$ of the hypergraph $\mathcal{H}$ satisfying the following conditions:

(a) The set $\{\sigma(1), \dots, \sigma(f)\}$ is exactly $F = [f]$. In other words, in a φ-equivalent variable ordering, the free variables come first (in any order).

(b) The function φ^σ defined by

$$\varphi^\sigma(\mathbf{x}_F) := \bigoplus_{x_{\sigma(f+1)}}^{(\sigma(f+1))} \cdots \bigoplus_{x_{\sigma(n)}}^{(\sigma(n))} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$$

is identical to the function φ irrespective of the input factors.[7]

Let $\mathsf{EVO}(\varphi)$ denote the set of all φ-equivalent variable orderings.

For any φ-equivalent variable ordering σ, we could have run InsideOut on φ^σ instead of φ. In that case, the sets U_k are denoted by U_k^σ, and the recursive sequence of hypergraphs is denoted by $\mathcal{H}_k^\sigma$, $k = n, n-1, \dots,$ as expected.

Definition 2. (Fractional FAQ-width of a variable ordering) Let σ be a φ-equivalent variable ordering. Define the sequence of hypergraphs $\mathcal{H}_k^\sigma$ along with the sets U_k^σ. The *fractional* FAQ *width* of a variable ordering σ is the quantity

$$\mathsf{faqw}(\sigma) := \max_{k\in K\cup[f]} \rho^*_{\mathcal{H}}(U_k^\sigma). \qquad (6)$$

As analyzed in (5), InsideOut runs in time $\tilde{O}(N^{\mathsf{faqw}(\sigma)} + Z)$, where σ is the original variable ordering given in expression (1). Thus, to have the best runtime we would like to select an equivalent ordering σ with the smallest $\mathsf{faqw}(\sigma)$.

jective value of the following linear program: $\min \sum_{F\in\mathcal{E}} x_e$ subject to $\sum_{F\ni v} x_F \geq 1$ for every $v \in B$ and $x_F \geq 0$ for every $F \in \mathcal{E}$.

[7]Here, we assume that the variable domains, the range $\mathbf{D}$, and the aggregates are fixed and known in advance, but the input factors are not.

Definition 3. (FAQ-width of an FAQ-query) The FAQ-*width* of an FAQ-query φ is defined by

$$\mathsf{faqw}(\varphi) := \min\{\mathsf{faqw}(\sigma) \mid \sigma \in \mathsf{EVO}(\varphi)\}$$

It is easy to see that if φ is an instance of FAQ-SS without free variables then $\mathsf{faqw}(\varphi) = \mathsf{fhtw}(\mathcal{H})$.

The question of determining whether a given variable ordering σ belongs to $\mathsf{EVO}(\varphi)$ is a tricky question to answer formally. In particular, the answer depends on what exactly we meant by a variable aggregate, a product aggregate, the variable domain sizes, and the range $\mathbf{D}$. This difficulty is analogous to the situation in logic when one wants to decide whether two (first-order, e.g.) formulas of specific forms are logically equivalent [10][8].

We derive a complete characterization of EVO, based on a combinatorial concept of *component-wise equivalence*. However, to facilitate the approximation algorithm for the FAQ-width we show that one does not need the complete characterization.

Our approach is as follows. In the most general case, we will define a class of variable orderings for a given input FAQ-query φ. This class will be precisely the set of linear extensions $\mathsf{LinEx}(P)$ of a partially ordered set (poset) on variables called the *precedence poset* P. The precedence poset is defined on a tree called the *expression tree* of the input query φ. Every variable ordering in this class will be shown to be φ-equivalent. This is the 'soundness' of the variable ordering class.

Then, we prove that this class of variable orderings is all we need to consider to compute/approximate $\mathsf{faqw}(\varphi)$, because every φ-equivalent variable ordering σ either belongs to $\mathsf{LinEx}(P)$ or is component-wise equivalent to π (which implies that $\mathsf{faqw}(\sigma) = \mathsf{faqw}(\pi)$) for some $\pi \in \mathsf{LinEx}(P)$. This shows the 'completeness' of $\mathsf{LinEx}(P)$. The completeness result rests on the assumption that different variable aggregates do not commute. (Even this simple statement needs a technical clarification, which is done in Proposition 5.1.)

Because the final result is a bit technically involved, we shall present in the next section a special case that illustrates many of our main ideas. It should be noted, however, that in the end there is only one theorem and one approximation algorithm for all FAQ instances.

5. VARIABLE ORDERING FOR FAQ WITH ONLY SEMIRING AGGREGATES

This section presents the variable ordering results for FAQ where every variable aggregate forms a semiring with the product aggregate. (Note that there can be an arbitrary number of different types of semirings.) In particular, we consider the FAQ-query of the form

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus_{x_{f+1}}^{(f+1)} \cdots \bigoplus_{x_n}^{(n)} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S) \qquad (7)$$

where $(\mathbf{D}, \oplus^{(i)}, \otimes)$ is a semiring for every $i > f$ (i.e. the set K as defined in (5) is $\{f+1, \dots, n\}$).

The main aim in this section is to illustrate the key technical idea of the *expression tree*. The expression tree defines the *precedence poset*. One key component of our completeness results is the notion of *component-wise equivalence* between two variable orderings. Component-wise equivalence

[8]We thank Balder ten Cate for pointing out to us the essence of the difficulty and the reference.

preserves FAQ-width and φ-equivalence. We will show two important facts about the precedence poset:

- (Soundness) Every linear extension of the precedence poset is a φ-equivalent variable ordering.
- (Completeness) Every φ-equivalent variable ordering is component-wise equivalent to (hence has the same FAQ-width as) some linear extension of the precedence poset. Therefore, $\mathsf{EVO}(\varphi)$ is *completely* characterized using component-wise equivalence and the precedence poset. Moreover, to compute/approximate $\mathsf{faqw}(\varphi)$, we *only* need to consider linear extensions of the precedence poset.

We use the structure of the expression tree to compute a variable ordering to approximate $\mathsf{faqw}(\varphi)$, using an approximation algorithm for fhtw as a blackbox. Thus, the expression tree is crucial in guiding the construction of a good variable ordering.

5.1 Expression tree and and precedence poset

The expression tree is defined on a sequence of *tagged variables* along with a hypergraph. In such a sequence, every vertex i (or equivalently variable X_i) is tagged with its corresponding operator $\oplus^{(i)}$; or, if the variable is a free variable then its tag is *free*. Given a sequence σ of tagged variables, a *tag block* is a maximal subsequence of consecutive variables in σ with the same tag. The first tag block of a sequence σ of tagged variables is the longest prefix of σ consisting of variables of the same tag.

Definition 4. (Expression tree) The *expression tree* for φ is a rooted tree P. Every node of the tree is a set of variables. We construct the expression tree using two steps: the *compartmentalization step* and the *compression step*. In the compartmentalization step, we construct the expression tree based on the connected component structures of the FAQ-query relative to the hypergraph structure $\mathcal{H}$. In the compression step we collapse the tree to make it shorter whenever possible.

Compartmentalization. In this step, initially we start off with the sequence of variables with their corresponding tags exactly as written in (7). In particular, the sequence starts with f free variables (whose tags are 'free'), and then the i'th variable with tag $\oplus^{(i)}$ for $i = f+1,\dots,n$. For technical reasons, we add a dummy variable X_0 to the beginning of the sequence with a free tag too. So the sequence we start off with is the following

$$\sigma = \Big\langle (X_0, \text{'free'}), \dots, (X_f, \text{'free'}), (X_{f+1}, \oplus^{(f+1)}), \dots, (X_n, \oplus^{(n)}) \Big\rangle.$$

The vertex X_0 is an isolated vertex of the hypergraph $\mathcal{H}$. Now given a tagged variable sequence σ and a hypergraph $\mathcal{H}$, we build the tree by constructing a node L containing all variables in the first tag block L of σ. This node L will be the root of the expression tree. (The effect of the dummy variable X_0 is that, even if the original query has no free variable, the first block L still has X_0 in it.) If L contains all variables already, then naturally the tree has only one node. Otherwise, for each connected component $C = (\mathcal{V}(C), \mathcal{E}(C))$ of $\mathcal{H} - L$ we construct a sequence σ_C of tagged variables by listing all variables in $\mathcal{V}(C)$ in exactly the same relative order as they appeared in σ. From the sequence σ_C and the hypergraph C, we recursively construct the expression tree P_C. Finally, we connect all the roots of (sub)expression trees P_C to the node L. This completes the compartmentalization step. After everything is done, we also remove the dummy variable X_0 from the expression tree P. If originally there was no free variable, the tree has an empty root node and the subtrees correspond to the connected components of $\mathcal{H}$. (See Example 4 and Figure 1 in Appendix A.)

Compression. Now, in the expression tree P that resulted from the compartmentalization step, as long as there is still a node L whose tag is the same as a child node L' of the tree P, we merge the child into L; namely, we set $L := L \cup L'$, remove L', and connect all subtrees under L' to become subtrees of L. Repeat this step until no further merging is possible. (See Figure 2.)

Note that the compression step can make some nodes L larger and the final tree T shorter than the tree that resulted from the compartmentalization step alone. This step is crucial for getting the correct expression tree. If φ is an instance of FAQ-SS, then P is a tree of depth ≤ 1 where the root node contains all free variables (if any) and its children (if any) contain the rest of the variables.

Definition 5. (Precedence poset) The expression tree defines a partial order on the variables. Abusing notation we will also use P to denote the partial order $([n], \preceq)$. In this poset, $u \prec v$ whenever $u \in L$, $v \in L'$, and L' is a (strict) descendant of L in the expression tree P. In particular, variables in the same node of the expression tree are not comparable in this partial order. We call this partial order the *precedence poset*. Let $\mathsf{LinEx}(P)$ denote the set of all linear extensions of the poset.

5.2 What makes two aggregates different?

Before proving soundness and completeness, we need a small technical detour. Recall that aggregates are simply binary operators under $\mathbf{D}$.

Definition 6. (Different aggregates) Two aggregates $\oplus$ and $\bar{\oplus}$ are *different* if there is a pair $a, b \in \mathbf{D}$ such that

$$a \oplus b \neq a \bar{\oplus} b.$$

Otherwise they are *identical*.

Definition 7. (Commutative aggregates) Two aggregates $\oplus$ and $\bar{\oplus}$ are said to be *commutative* if for every $a, b, c, d \in \mathbf{D}$, we have $(a \oplus b) \bar{\oplus} (c \oplus d) = (a \bar{\oplus} c) \oplus (b \bar{\oplus} d)$.

Recall that in FAQ, all semirings share the same $\mathbf{0}$ (since one '$\mathbf{0}$' must annihilate the rest). Thus, if we select $a = d = \mathbf{0}$ in the above equality, then we obtain $b \bar{\oplus} c = b \oplus c$, for every $b, c \in \mathbf{D}$. This means

Proposition 5.1. *Commutative aggregates are identical aggregates. Conversely, non-commutative aggregates are different aggregates.*

(Note that it is possible for semantically different aggregates to be identical under $\mathbf{D}$ by accident. For example, in the $\{0, 1\}$ domain min and $\times$ are identical.) In this paper, we assume that two different aggregates in the input FAQ-expression are not functionally identical. Recall that we also assumed $|\mathsf{Dom}(X_i)| \geq 2$ for every $i \in [n]$ (otherwise the aggregate on that X_i is trivial and can be ignored).

PROPOSITION 5.2. *Suppose $\oplus$ and $\bar{\oplus}$ are different binary operators (under the domain $\mathbf{D}$), then for every $i, j \in [n]$, there is a function $\phi_{ij} : \mathsf{Dom}(X_i) \times \mathsf{Dom}(X_j) \to \mathbf{D}$ for which*

$$\bigoplus_{x_i \in \mathsf{Dom}(X_i)} \bar{\bigoplus}_{x_j \in \mathsf{Dom}(X_j)} \phi_{ij}(x_i, x_j) \neq \bar{\bigoplus}_{x_j \in \mathsf{Dom}(X_j)} \bigoplus_{x_i \in \mathsf{Dom}(X_i)} \phi_{ij}(x_i, x_j). \quad (8)$$

PROOF. From the analysis above, the two operators do not commute. Hence, there are four members $a, b, c, d \in \mathbf{D}$ so that $(a \oplus b)\bar{\oplus}(c \oplus d) \neq (a\bar{\oplus}c) \oplus (b\bar{\oplus}d)$. Fix arbitrary elements $x_i^1 \neq x_i^2 \in \mathsf{Dom}(X_i)$ and $x_j^1 \neq x_j^2 \in \mathsf{Dom}(X_j)$. Define

$$\phi_{ij}(x_i, x_j) = \begin{cases} a & \text{if } (x_i, x_j) = (x_i^1, x_j^1) \\ b & \text{if } (x_i, x_j) = (x_i^2, x_j^1) \\ c & \text{if } (x_i, x_j) = (x_i^1, x_j^2) \\ d & \text{if } (x_i, x_j) = (x_i^2, x_j^2) \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (9)$$

Then, $\bigoplus_{x_i} \bar{\bigoplus}_{x_j} \phi_{ij}(x_i, x_j) = (a\bar{\oplus}c) \oplus (b\bar{\oplus}d) \neq (a \oplus b)\bar{\oplus}(c \oplus d) = \bar{\bigoplus}_{x_j} \bigoplus_{x_i} \phi_{ij}(x_i, x_j)$. $\square$

Given $i, j \in [n]$, $x_i^1 \neq x_i^2 \in \mathsf{Dom}(X_i)$, $x_j^1 \neq x_j^2 \in \mathsf{Dom}(X_j)$, we define an 'identity' function $\phi_{ij}^{(\mathbf{I})} : \mathsf{Dom}(X_i) \times \mathsf{Dom}(X_j) \to \mathbf{D}$ as follows.

$$\phi_{ij}^{(\mathbf{I})}(x_i, x_j) = \begin{cases} \mathbf{1} & \text{if } (x_i, x_j) = (x_i^1, x_j^1) \text{ or } (x_i, x_j) = (x_i^2, x_j^2) \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (10)$$

We will use both ϕ_{ij} and $\phi_{ij}^{\mathbf{I}}$ in the proofs below.

5.3 Soundness and completeness

We are now fully equipped to show that $\mathsf{LinEx}(P)$ is sound.

THEOREM 5.3 ($\mathsf{LinEx}(P) \subseteq \mathsf{EVO}(\varphi)$). *Every linear extension of the precedence poset P is φ-equivalent.*

PROOF. Let P be the expression tree constructed using only the compartmentalization step. This expression tree already defines a poset on variables. We will first show that every linear extension of this *compartmentalization poset* is φ-equivalent. We prove this claim by induction on the number of tag blocks of the input sequence. Let σ denote the input sequence of tagged variables with input hypergraph $\mathcal{H}$.

In the base case σ has only one tag block. All variables in the sequence belong to the same node of the compartmentalization expression tree. This means every permutation of variables is a linear extension of the poset, which is what we expect because every permutation is φ-equivalent.

In the inductive step, suppose σ has at least two tag blocks with the first block being the set L of variables. Then, each sub-sequence σ_C for each connected component of $\mathcal{H} - L$ defines an FAQ-expression φ_C on the set of conditional factors $\{\psi_S(\cdot \mid \mathbf{x}_L) \mid S \in \mathcal{E} \wedge S \cap \mathcal{V}(C) \neq \emptyset\}$. When we condition on the first L variables, the expression $\varphi(\cdot \mid \mathbf{x}_L)$ completely factorizes into a product of the FAQ-expressions φ_C. (Another way to put this is that φ can be written as a series of aggregates on variables in L, with a product of φ_C inside.) By induction, every linear extension of the compartmentalization poset for σ_C is φ_C-equivalent. Those linear extensions can be put together in arbitrary interleaving way to form $\varphi(\cdot \mid \mathbf{x}_L)$. This observation completes the proof of the claim, because every linear extension of the expression poset for φ consists of variables in L, followed by arbitrary interleavings of linear extensions of the expression posets for the φ_C.

Next, we consider the expression tree after the compression step. We show that every linear extension of the final precedence poset is φ-equivalent by induction on the number of merges of a child node L' to a parent node L (which both must have the same tag). To see this, we can take a linear extension σ of the expression tree *before* the merge where all variables in L and in L' are consecutive in σ. Then, because all variables in $L \cup L'$ have the same tag, we can permute them in any way and still obtain a φ-equivalent variable ordering. $\square$

It would have been nice if every φ-equivalent variable ordering is a linear extension of P. Unfortunately this is not true. Consider the following FAQ-query

$$\varphi = \sum_{x_1} \sum_{x_2} \max_{x_3} \max_{x_4} \sum_{x_5} \psi_{15}\psi_{25}\psi_{13}\psi_{24},$$

where all factors have range $\mathbb{R}_+$ so that all variables are semiring variables. In this case the expression tree has three nodes: one empty root, a node containing $\{1, 2, 5\}$ and two children $\{3\}$ and $\{4\}$. The linear extensions will enforce that X_1, X_2, X_5 come before X_3 and X_4. However, it is easy to see that we can rewrite φ as follows.

$$\begin{aligned} \varphi &= \sum_{x_1} \sum_{x_2} \max_{x_3} \max_{x_4} \sum_{x_5} \psi_{15}\psi_{25}\psi_{13}\psi_{24} \\ &= \sum_{x_5} \sum_{x_1} \sum_{x_2} \max_{x_3} \max_{x_4} \psi_{15}\psi_{25}\psi_{13}\psi_{24} \\ &= \sum_{x_5} \left(\sum_{x_1} \max_{x_3} \psi_{15}\psi_{13} \right) \cdot \left(\sum_{x_2} \max_{x_4} \psi_{25}\psi_{24} \right). \end{aligned}$$

This means when conditioned on X_5, the expression factorizes and we can multiply them back allowing for 4 to come before 1, or for 3 to come before 2; namely,

$$\begin{aligned} \varphi &= \sum_{x_5} \sum_{x_2} \max_{x_4} \sum_{x_1} \max_{x_3} \psi_{15}\psi_{25}\psi_{13}\psi_{24} \\ &= \sum_{x_5} \sum_{x_1} \max_{x_3} \sum_{x_2} \max_{x_4} \psi_{15}\psi_{25}\psi_{13}\psi_{24}. \end{aligned}$$

However, it can be verified that

$$\mathsf{faqw}(5, 1, 3, 2, 4) = \mathsf{faqw}(5, 2, 4, 1, 3) = \mathsf{faqw}(\sigma)$$

for any $\sigma \in \mathsf{LinEx}(P)$ where 5 comes first in σ, and where P is the expression tree of the query. Note that we can take the linear extensions of the factorized components $\{1, 3\}$ and $\{2, 4\}$ and interleave them in any way, as long as we still respect their relative order within each component. However, these interleavings do not add anything of value as far as the faqw is concerned.

Another way to think about the above example is that we could have arbitrarily selected one variable in the first tag block of φ, construct a compartmentalization expression tree with that variable as the root. (One variable at a time instead of one tag block at a time.) Then, by the same reasoning we used in the proof of Theorem 5.3, every linear extension of this 'variable-wise' poset is φ-equivalent. However, this idea alone also does not work because it will forbid the selection of X_5 as the first variable in the above example. Thus, it is crucial that we construct the compressed expression tree first, to determine which variable *can* come first in a φ-equivalent variable ordering. Ultimately, the set $\mathsf{LinEx}(P)$ gives us a canonical way of listing the variable orderings that really matter in evaluating φ.

In what follows, we implement the above informal discussion and intuition by showing that every φ-equivalent variable ordering has the same width as some ordering in $\mathsf{LinEx}(\varphi)$. The following lemma says that the expression tree indeed gives us a complete list of variables that can occur first (after the free variables) in any φ-equivalent ordering.

LEMMA 5.4. *For every variable ordering $\pi = (u_1, \dots, u_n) \in \mathsf{EVO}(\varphi)$, the variable u_{f+1} must belong to a child node of the root of the expression tree.*[9]

PROOF. Let φ^π denote the function defined by the FAQ-query with π as the variable ordering (over the same input factors as φ). Our aim is to show that if the conclusion of the lemma does not hold then there exist input factors ψ_S for which $\varphi^\pi \not\equiv \varphi$.

Suppose for the sake of contradiction that u_{f+1} belongs to a node L whose parent is L_p, and L_p is not the root of the expression tree P. Let $\overline{L}_a$ denote union of all the (strict) ancestors of L_p in the expression tree. From the construction of the expression tree, the vertices in the set $\{u_{f+1}\} \cup L_p$ belong to the same connected component of the graph $\mathcal{H} - \overline{L}_a$. Let $i_0 := u_{f+1}, i_1, \dots, i_k \in L_p$ be the shortest path in the Gaifman graph of $\mathcal{H} - \overline{L}_a$ from u_{f+1} to L_p. Then, the vertices $i_1, \dots, i_{k-1}$ do not belong to $L_p \cup \overline{L}_a$; and, there are distinct hyperedges $S_1, \dots, S_k$ of $\mathcal{H}$ such that $\{i_{j-1}, i_j\} \subseteq S_j$ for all $j \in [k]$.

For each variable $\ell \in \{i_0, \dots, i_k\}$, we fix two arbitrary values $x_\ell^1 \neq x_\ell^2 \in \mathsf{Dom}(X_\ell)$ and for each $\ell' \in [n] \setminus \{i_0, \dots, i_k\}$, we fix one arbitrary value $e_{\ell'} \in \mathsf{Dom}(X_{\ell'})$. For the sake of brevity, denote $\bigoplus = \bigoplus^{(i_0)}$ and $\bar{\bigoplus} = \bigoplus^{(i_k)}$.

Now, we construct an input set of factors ψ_S, $S \in \mathcal{E}$, for which $\varphi^\pi \not\equiv \varphi$.

- Define the factor ψ_{S_k} by
$$\psi_{S_k}(\mathbf{x}_{S_k}) := \begin{cases} \phi_{i_{k-1}i_k}(x_{i_{k-1}}, x_{i_k}) & \text{if } x_{\ell'} = e_{\ell'} \text{ for all } \ell' \\ & \quad \in S_k \setminus \{i_0, \dots, i_k\} \\ \mathbf{0} & \text{otherwise,} \end{cases}$$
where $\phi_{i_{k-1}i_k}$ is the function defined in (9).
- For every $j \in [k-1]$, define
$$\psi_{S_j}(\mathbf{x}_{S_j}) := \begin{cases} \phi^{(\mathbf{I})}_{i_{j-1}i_j}(x_{i_{j-1}}, x_{i_j}) & \text{if } x_{\ell'} = e_{\ell'} \text{ for all } \ell' \\ & \quad \in S_j \setminus \{i_0, \dots, i_k\} \\ \mathbf{0} & \text{otherwise,} \end{cases}$$
where $\phi^{(\mathbf{I})}_{i_{j-1}i_j}$ is the 'identity' function defined in (10). (Think of these factors as little 2×2 identity matrices.)
- Finally, for every $S' \in \mathcal{E} \setminus \{S_1, \dots, S_k\}$, define
$$\psi_{S'}(\mathbf{x}_{S'}) := \begin{cases} \mathbf{1} & \text{if } x_{\ell'} = e_{\ell'} \text{ for all } \ell' \in S' \setminus \{i_0, \dots, i_k\} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

[9]Recall that $\{u_1, \dots, u_f\}$ are the free variables, which are located in the root of the expression tree. And, if $f = 0$ then the root of the expression tree is empty.

Because i_0 is the first in π after the free variables, $\varphi^\pi(e_1, \dots, e_f)$ will evaluate to the left hand side of (8). (Imagine running the $\mathsf{InsideOut}$ algorithm to evaluate φ^π.) Next, to get a contradiction, we pick an ordering $\sigma \in \mathsf{LinEx}(P)$ such that i_0 precedes all variables that are at the same or lower level in the expression tree. By Theorem 5.3, we know that $\sigma \in \mathsf{EVO}(\varphi)$. In σ, i_k precedes i_0 which in turn precedes all of $\{i_1, \dots, i_{k-1}\}$. Hence, when we compute $\varphi(e_1, \dots, e_f)$ using the ordering σ (and the $\mathsf{InsideOut}$ algorithm) we get the right hand side of (8). Thus, $\varphi^\pi \not\equiv \varphi$ as desired. $\square$

The next definition realizes the intuition that if we construct the precedence tree using the one-variable-at-a-time strategy (as opposed to the one-tag-block-at-a-time strategy), then we can interleave the linear extensions of each connected components arbitrarily and still get a variable ordering which is φ-equivalent with the same FAQ-width. Since the interleaving can happen at any level, the definition is inductive.

Definition 8. (Component-wise equivalence) Let φ be an FAQ-query where all variable aggregates are semiring aggregates. Let $\sigma = (v_1, \dots, v_n) \in \mathsf{EVO}(\varphi)$ be a variable ordering. Let $\pi = (u_1, \dots, u_n)$ be another variable ordering with $\{u_1, \dots, u_f\} = F$. Then, π is said to be *component-wise equivalent* (or CW-equivalent) to σ if and only if:

- either $n = 1$,
- or $\mathcal{H}$ has ≥ 2 connected components, and for each connected component $C = (\mathcal{V}(C), \mathcal{E}(C))$ of $\mathcal{H}$, π_C is CW-equivalent to σ_C, where σ_C (respectively, π_C) is the variable ordering ordering of $\mathcal{V}(C)$ that is consistent with σ (respectively, π),
- or $u_1 = v_1$, and for each connected component $C = (\mathcal{V}(C), \mathcal{E}(C))$ of $\mathcal{H} - \{v_1\}$, π_C is CW-equivalent to σ_C, where σ_C (respectively, π_C) is the ordering of $\mathcal{V}(C)$ that is consistent with σ (respectively, π).

Given a set of variable orderings $\Lambda \subseteq \mathsf{EVO}(\varphi)$, we use $\mathsf{CWE}(\Lambda)$ to denote the set of all variable orderings that are CW-equivalent to some variable ordering in Λ.

PROPOSITION 5.5. *Let π be a variable ordering that is CW-equivalent to $\sigma \in \mathsf{EVO}(\varphi)$. Then, we have $\pi \in \mathsf{EVO}(\varphi)$ and $\mathsf{faqw}(\sigma) = \mathsf{faqw}(\pi)$.*

The following theorem shows the completeness part. (The proof and an example are in Appendices B.2 and B.3.)

THEOREM 5.6 $(\mathsf{EVO}(\varphi) = \mathsf{CWE}(\mathsf{LinEx}(P)))$. *A variable ordering σ is φ-equivalent if and only if it is CW-equivalent to some linear extension of the precedence poset P.*

Proposition 5.5 and Theorem 5.6 imply the following result, which is precisely what we need to approximate $\mathsf{faqw}(\varphi)$.

COROLLARY 5.7. $\mathsf{faqw}(\varphi) = \min \{\mathsf{faqw}(\sigma) \mid \sigma \in \mathsf{LinEx}(P)\}$.

5.4 Approximating $\mathsf{faqw}(\varphi)$

Let P be the expression tree constructed from the query φ in the form (7). We define notation that will be used through out this section. Let C be a node of the expression tree P. (This means C is a set of variables of φ, and all variables in C have the same tag.) Let L be the parent node of C (if any) in the expression tree P. We define the following sets:

$$\bar{\mathcal{E}}(C) := \{S \in \mathcal{E} \mid S \cap C' \neq \emptyset \text{ for some } C' \text{ node in the subtree of } P \text{ rooted at } C.\} \tag{11}$$

$$S_{L,C} := L \cap \left(\bigcup_{S \in \bar{\mathcal{E}}(C)} S \right) \tag{12}$$

$$U(C) := \bigcup_{L' \text{ an ancestor of } C} \left(L' \cap \bigcup_{S \in \bar{\mathcal{E}}(C)} S \right). \tag{13}$$

If C has no parent then $U(C) = \emptyset$ by default. We think of the set $S_{L,C}$ as the contribution of all the nodes in the C-branch to L, and the set $U(C)$ as the contribution of all the nodes in the C-branch to all the (strict) ancestors of C. Next, for every node L in the expression tree P, define the hypergraph $\mathcal{H}_L$ as follows.

- If L is a leaf node of P, then $\mathcal{H}_L = \mathcal{H}[L]$, the subgraph of $\mathcal{H}$ induced by L.
- If L is not a leaf node of P, then $\mathcal{H}_L = (L, \mathcal{E}_L)$, where

$$\begin{aligned}\mathcal{E}_L := & \big\{S \cap L \mid (S \in \mathcal{E}) \wedge (S \cap L \neq \emptyset) \wedge \\ & (S \cap C = \emptyset, \forall \text{ descendant } C \text{ of } L)\big\} \\ & \cup \big\{S_{L,C} \mid C \text{ a child of } L\big\}.\end{aligned}$$

  In other words, $\mathcal{E}_L$ is the set of all projections of S down to L for all hyperedges S for which S intersects L but not any descendant of L in the expression tree; and for each child C of L, $\mathcal{E}_L$ also contains the projection onto L of the union of all hyperedges S that intersect (some descendant of) C.

LEMMA 5.8. *For any node L in the expression tree,*

$$\mathsf{faqw}(\varphi) \geq \mathsf{fhtw}(\mathcal{H}_L) \quad \text{and} \quad \mathsf{faqw}(\varphi) \geq \rho^*_{\mathcal{H}}(U(L)).$$

PROOF. To show the first inequality, by Corollary 5.7, it is sufficient to prove that $\mathsf{faqw}(\sigma) \geq \mathsf{fhtw}(\mathcal{H}_L)$ for any variable ordering $\sigma = (v_1, \dots, v_n) \in \mathsf{LinEx}(P)$ and for any node L in the expression tree.

If L is a leaf node of the expression tree, then $\mathsf{faqw}(\sigma) \geq \mathsf{fhtw}(\mathcal{H}) \geq \mathsf{fhtw}(\mathcal{H}_L)$ because $\mathcal{H}_L$ is an induced subgraph of $\mathcal{H}$. Now, suppose L is not a leaf node. For any child C of L, let k be the smallest integer such that v_k belongs to some node in the subtree rooted at C. Then, due to the fact that $\sigma \in \mathsf{LinEx}(P)$, if $v_j \in L$ then $j < k$. The set U^σ_k is *precisely* equal to $U(C) \cup \{v_k\}$. This is because each time we eliminate a vertex belonging to any node in the subtree rooted at C, we insert back a hyperedge interconnecting all its neighbors (to the next hypergraph in the hypergraph sequence). And so by the time we reach U^σ_k all of the nodes in $U(C)$ belong to the same hyperedge of $\mathcal{H}^\sigma_k$. It follows that $(U^\sigma_k - \{v_k\}) \cap L = U(C) \cap L = S_{L,C}$ (since $L \cap L' = \emptyset$ for any ancestor L' of L). From this observation we obtain:

$$\begin{aligned}
\mathsf{faqw}(\varphi) &= \min_\sigma \left(\max \{\rho^*_{\mathcal{H}}(U^\sigma_k) \mid k \in [n]\}\right) \\
&\geq \min_\sigma \left(\max \{\rho^*_{\mathcal{H}}(U^\sigma_j) \mid j \in L\}\right) \\
(\rho^* \text{ is monotone}) &\geq \min_\sigma \left(\max \{\rho^*_{\mathcal{H}}(U^\sigma_j \cap L) \mid j \in L\}\right) \\
&\geq \min_\tau \left(\max \{\rho^*_{\mathcal{H}}(U^\tau_j) \mid j \in L\}\right) \\
&= \mathsf{fhtw}(\mathcal{H}_L).
\end{aligned}$$

In the above, τ is taken only over all variable orderings of L instead of the entire set $[n]$.

We next prove the second inequality that, for any $\sigma = (v_1, \dots, v_n) \in \mathsf{LinEx}(\varphi)$, and any node C in the expression tree, we have $\mathsf{faqw}(\sigma) \geq \rho^*_{\mathcal{H}}(U(C))$. As we have observed above, let k be the smallest integer such that $v_k \in C$, then $U^\sigma_k = U(C) \cup \{v_k\}$. Hence, because ρ^* is monotone, $\mathsf{faqw}(\sigma) = \max_{j \in [n]} \rho^*_{\mathcal{H}}(U^\sigma_j) \geq \rho^*_{\mathcal{H}}(U^\sigma_k) \geq \rho^*_{\mathcal{H}}(U(C))$. □

THEOREM 5.9. *Let φ be any FAQ query whose hypergraph is $\mathcal{H}$ and all variable aggregates are semiring aggregates. Suppose there is an approximation algorithm that, given any hypergraph $\mathcal{H}'$, outputs a tree decomposition of $\mathcal{H}'$ with fractional hypertree width at most $g(\mathsf{fhtw}(\mathcal{H}'))$ in time $t(|\mathcal{H}'|, \mathsf{fhtw}(\mathcal{H}'))$ for some non-decreasing functions g, t. Then, we can in time $|\mathcal{H}| \cdot t(|\mathcal{H}|, \mathsf{faqw}(\varphi))$ compute a φ-equivalent vertex ordering σ such that*

$$\mathsf{faqw}(\sigma) \leq \mathsf{faqw}(\varphi) + g(\mathsf{faqw}(\varphi)).$$

PROOF. We use the blackbox approximation algorithm for fhtw to construct a tree decomposition (T_L, χ_L) for every hypergraph $\mathcal{H}_L$ where L is a node in the expression tree. Then, from each of those tree decompositions, we construct a variable ordering σ_L for variables in the set L in the standard way. Finally, we construct the variable ordering σ for $[n]$ by concatenating all the σ_L together in any way that respects the precedence partial order.

Suppose $\sigma = (v_1, \dots, v_n)$ is the resulting variable ordering. Consider an arbitrary vertex v_k. Let L be the node of the precedence tree that contains v_k. Let B be the bag in (T_L, χ_L) that v_k belonged to when it was eliminated to construct σ_L. Then, using the same argument as in the proof of Lemma 5.8, we can show that

$$U^\sigma_k \subseteq B \cup U(L). \tag{14}$$

To see this, first consider the simpler case when L is a leaf node of the expression tree. Then, when we eliminate v_k the set U_k is the union of the sets $S \in \partial(v_k)$. The part $U_k \cap L$ is covered by B because within L the elimination algorithm works on $\mathcal{H}_L$. The part $U_k \setminus L$ is covered by the maximum *residue* left over from eliminating all vertices in L. The residue is precisely the set $U(L)$, because every time we eliminate a vertex we collect all its neighbors together into a hyperedge. By the time the last vertex from L is eliminated, the entire set $U(L)$ becomes a hyperedge. Now, if L is not a leaf node, the situation is exactly the same except for the fact that we work on the graph $\mathcal{H}_L$ which is not necessarily the same as $\mathcal{H}[L]$. The graph $\mathcal{H}_L$ contains the restrictions on L of all the residues of the subtrees under L.

Next, from Lemma 5.8 and from the fact that ρ^* is subadditive, relation (14) implies

$$\begin{aligned}\rho^*_{\mathcal{H}}(U^\sigma_k) &\leq \rho^*(B) + \rho^*(U(L)) \leq g(\mathsf{fhtw}(\mathcal{H}_L)) + \mathsf{faqw}(\varphi) \\ &\leq g(\mathsf{faqw}(\varphi)) + \mathsf{faqw}(\varphi).\end{aligned}$$

Finally, $\mathsf{faqw}(\sigma) = \max_{k \in [n]} \rho^*_{\mathcal{H}}(U^\sigma_k) \leq g(\mathsf{faqw}(\varphi)) + \mathsf{faqw}(\varphi)$. □

Acknowledgments

This work was partly supported by NSF grants CNF-1409551, CCF-1319402, and by DARPA under agreement #FA8750-15-2-0009. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright thereon.

6. REFERENCES

[1] C. R. Aberger, A. Nötzli, K. Olukotun, and C. Ré. EmptyHeaded: Boolean Algebra Based Graph Processing. *ArXiv e-prints*, Mar. 2015.

[2] M. Abo Khamis, H. Q. Ngo, C. Ré, and A. Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of the 34rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'15, Melbourne, VIC, AustraliaS, May 31-June 04, 2015*, 2015.

[3] M. Abo Khamis, H. Q. Ngo, and A. Rudra. FAQ: questions asked frequently. *CoRR*, abs/1504.04044, 2015.

[4] I. Adler and M. Weyer. Tree-width for first order formulae. *Logical Methods in Computer Science*, 8(1), 2012.

[5] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[6] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the LogicBlox system. In T. Sellis, S. B. Davidson, and Z. G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1371–1382. ACM, 2015.

[7] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748. IEEE Computer Society, 2008.

[8] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[9] N. Bakibayev, T. Kociský, D. Olteanu, and J. Zavodny. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.

[10] E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem.* Universitext. Springer-Verlag, Berlin, 2001. Reprint of the 1997 original.

[11] J. Brault-Baron, F. Capelli, and S. Mengel. Understanding model counting for β-acyclic cnf-formulas. *CoRR*, abs/1405.6043, 2014.

[12] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[13] H. Chen. Quantified constraint satisfaction and bounded treewidth. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 161–165, 2004.

[14] H. Chen and V. Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 205–214. IEEE Computer Society, 2012.

[15] H. Chen and M. Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010.

[16] H. Chen and S. Mengel. A trichotomy in the complexity of counting answers to conjunctive queries. *CoRR*, abs/1408.0890, 2014.

[17] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, second edition, 2001.

[19] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[20] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

[21] R. Dechter, L. Otten, and R. Marinescu. On the practical significance of hypertree vs. treewidth. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 913–914. IOS Press, 2008.

[22] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[23] A. Durand and S. Mengel. Structural tractability of counting of solutions to conjunctive queries. In W. Tan, G. Guerrini, B. Catania, and A. Gounaris, editors, *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 81–92. ACM, 2013.

[24] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In H. E. Shrobe, T. G. Dietterich, and W. R. Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, July 29 - August 3, 1990, 2 Volumes.*, pages 4–9. AAAI Press / The MIT Press, 1990.

[25] S. W. Golomb and L. D. Baumert. Backtrack programming. pages 516–524, 1965.

[26] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.

[27] G. Greco and F. Scarcello. Counting solutions to conjunctive queries: structural and hybrid tractability. In R. Hull and M. Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 132–143. ACM, 2014.

[28] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298. ACM Press, 2006.

[29] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4, 2014.

[30] M. Gyssens, P. Jeavons, and D. A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, 1994.

[31] M. Gyssens and J. Paredaens. A decomposition methodology for cyclic databases. In *Advances in Data Base Theory*, pages 85–122, 1982.

[32] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2):165–193, 2005.

[33] C. Koch. Incremental query evaluation in a ring of databases. In J. Paredaens and D. V. Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 87–98. ACM, 2010.

[34] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[35] D. Koller and N. Friedman. *Probabilistic graphical models*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2009. Principles and techniques.

[36] D. Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, Apr. 2010.

[37] D. Marx. Tractable structures for constraint satisfaction with truth tables. *Theory Comput. Syst.*, 48(3):444–464, 2011.

[38] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42, 2013.

[39] H. Q. Ngo, D. T. Nguyen, C. Re, and A. Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 234–245, 2014.

[40] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In M. Lenzerini and M. Benedikt, editors, *PODS*, pages 37–48. ACM, 2012.

[41] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. In *SIGMOD RECORD*, pages 5–16, 2013.

[42] D. T. Nguyen, M. Aref, M. Bravenboer, G. Kollias, H. Q. Ngo, C. Ré, and A. Rudra. Join processing for graph patterns: An old dog with new tricks. In *Proceedings of the Third International Workshop on Graph Data Management Experiences and Systems, GRADES 2015, Melbourne, VIC, Australia, May 31 - June 4, 2015*, pages 2:1–2:8, 2015.

[43] D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *ACM Trans. Datab. Syst.*, 40(1), 2015.

[44] S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of Acyclic and Almost Acyclic CNF Formulas. In *FSTTCS 2010*, volume 8, 2010.

[45] J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In D. L. Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982.*, pages 133–136. AAAI Press, 1982.

[46] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, Sept. 1986.

[47] R. Pichler and S. Skritek. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001, Sept. 2013.

[48] E. Rollon, J. Larrosa, and R. Dechter. Semiring-based mini-bucket partitioning schemes. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.

[49] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[50] F. Scarcello. Query answering exploiting structural properties. *SIGMOD Record*, 34(3):91–99, 2005.

[51] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

[52] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

[53] N. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.

[54] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *J. Artificial Intelligence Res.*, 5:301–328, 1996.

APPENDIX

A. INTUITION BEHIND THE EXPRESSION TREE

Example 4. (Intuition behind expression tree) Consider the following FAQ query that has two different semiring aggregates ($\sum$ and max) and no free variables:

$$\varphi = \sum_{x_1}\sum_{x_2}\max_{x_3}\sum_{x_4}\sum_{x_5}\max_{x_6}\max_{x_7}\psi_{12}\psi_{135}\psi_{14}\psi_{246}\psi_{27}\psi_{37}.$$

(ψ_{12} above denotes a factor whose support is $\{X_1, X_2\}$, and so on.) The hypergraph of φ is depicted in Figure 1a. The compartmentalization step of the construction of the expression tree is depicted in Figures 1b through 1d. Figures 2a and 2b depict the compression step. The final expression tree appears on the right of Figure 2b.

B. OMITTED TECHNICAL DETAILS

B.1 The InsideOut Algorithm

Definition 9. (**01**-projection) For each $S \in \mathcal{E}$ and any set $U \subseteq \mathcal{V}$ such that $S \cap U \neq \emptyset$, define the **01**-*projection* of ψ_S onto U as the function

$$\psi_{S/U} : \prod_{i \in S\cap U} \mathsf{Dom}(X_i) \to \{\mathbf{0}, \mathbf{1}\},$$

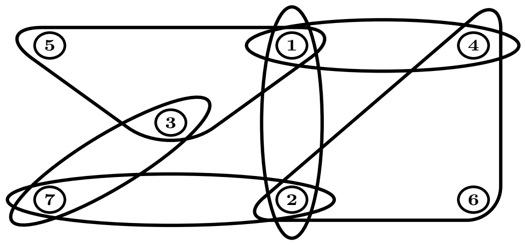
where

$$\psi_{S/U}(\mathbf{x}_{S\cap U}) = \begin{cases} \mathbf{1} & \text{if } \exists \mathbf{y}_S \text{ s.t. } \psi_S(\mathbf{y}_S) \neq \mathbf{0} \text{ and } \mathbf{y}_{S\cap U} = \mathbf{x}_{S\cap U} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

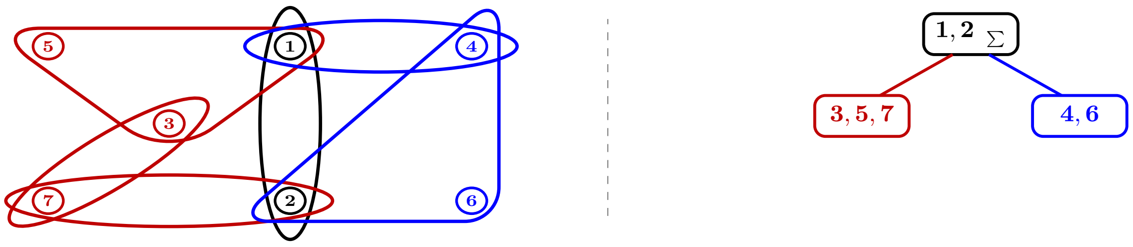
Definition 10. (**01**-OR) Given $a, b \in \{\mathbf{0}, \mathbf{1}\}$, the **01**-OR of a and b is $\mathbf{0}$ if $a = b = \mathbf{0}$, and $\mathbf{1}$ otherwise.

B.2 Proof of Completeness

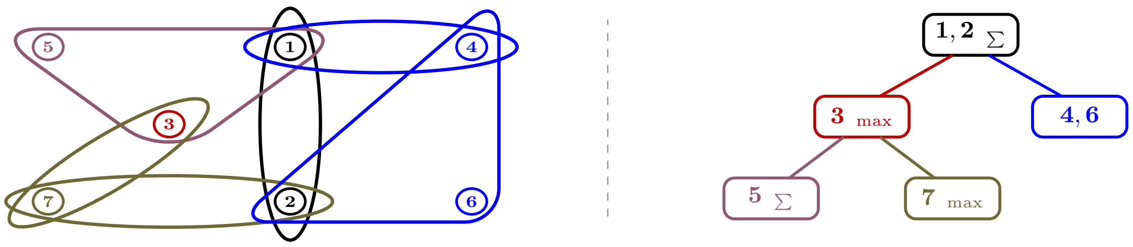
In this section, we prove Theorem 5.6, which basically says that our characterization of $\mathsf{EVO}(\varphi)$ is complete. We start with proving Proposition 5.5, which almost immediately follows from Definition 8.



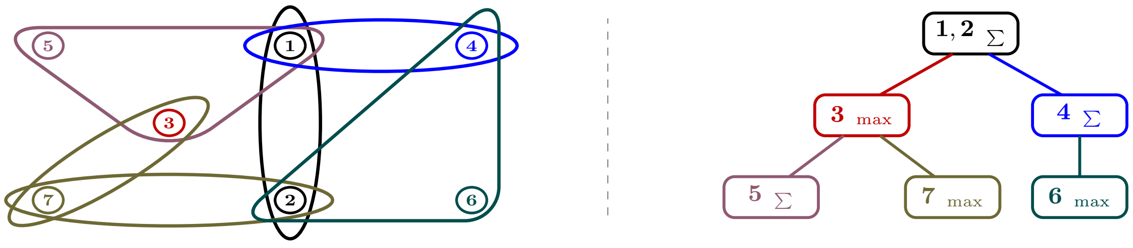
(a) $\varphi = \sum_{x_1}\sum_{x_2}\max_{x_3}\sum_{x_4}\sum_{x_5}\max_{x_6}\max_{x_7} \psi_{12}\;\psi_{135}\;\psi_{14}\;\psi_{246}\;\psi_{27}\;\psi_{37}$



(b) $\varphi = \sum_{x_1}\sum_{x_2}\psi_{12}\left(\max_{x_3}\sum_{x_5}\max_{x_7}\psi_{135}\;\psi_{27}\;\psi_{37}\right)\left(\sum_{x_4}\max_{x_6}\psi_{14}\;\psi_{246}\right)$

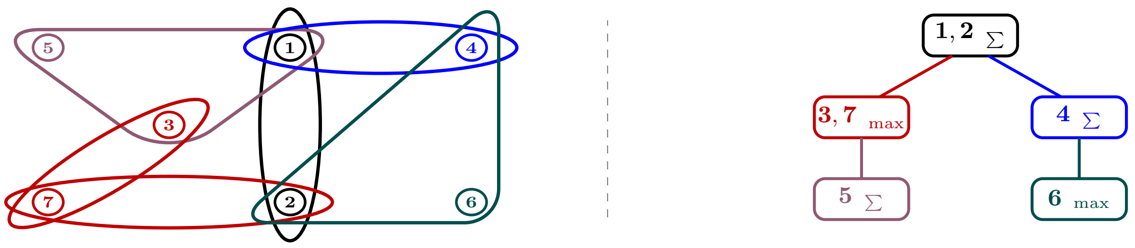


(c) $\varphi = \sum_{x_1}\sum_{x_2}\psi_{12}\left(\max_{x_3}\sum_{x_5}\psi_{135}\max_{x_7}\psi_{27}\;\psi_{37}\right)\left(\sum_{x_4}\max_{x_6}\psi_{14}\;\psi_{246}\right)$

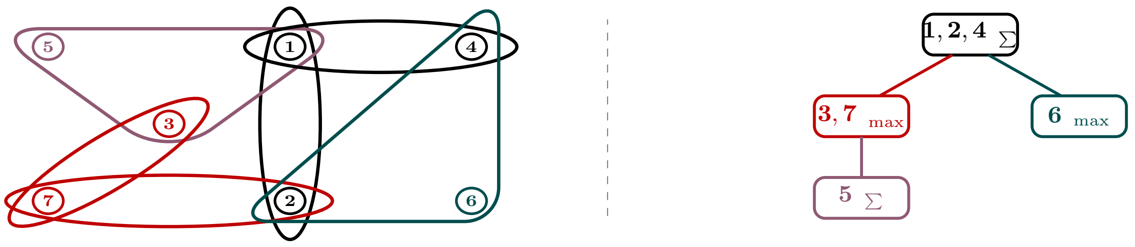


(d) $\varphi = \sum_{x_1}\sum_{x_2}\psi_{12}\left(\max_{x_3}\sum_{x_5}\psi_{135}\max_{x_7}\psi_{27}\;\psi_{37}\right)\left(\sum_{x_4}\psi_{14}\max_{x_6}\psi_{246}\right)$

Figure 1: The compartmentalization step of the expression tree from Example 4, depicted using colors. (1a) depicts the hypergraph of the FAQ query: $\varphi = \sum_{x_1}\sum_{x_2}\max_{x_3}\sum_{x_4}\sum_{x_5}\max_{x_6}\max_{x_7}\psi_{12}\psi_{135}\psi_{14}\psi_{246}\psi_{27}\psi_{37}$. For simplicity, the dummy free variable X_0 is ignored in this example. (1b) shows the first part of the compartmentalization step, where the first tag block is $L = \{1, 2\}$. After removing L, the query breaks into two connected components: the red and the blue. The expression tree at this point appears on the right. Each color is used to denote correspondence between parts of the query expression, hypergraph, and expression tree having that color. (1c) shows how to apply compartmentalization recursively on the red component, while (1d) shows the blue component compartmentalization.



(a) $\varphi = \sum_{x_1}\sum_{x_2}\psi_{12}\left(\max_{x_3}\max_{x_7}\psi_{27}\,\psi_{37}\sum_{x_5}\psi_{135}\right)\left(\sum_{x_4}\psi_{14}\max_{x_6}\psi_{246}\right)$



(b) $\varphi = \sum_{x_1}\sum_{x_2}\sum_{x_4}\psi_{12}\,\psi_{14}\left(\max_{x_3}\max_{x_7}\psi_{27}\,\psi_{37}\sum_{x_5}\psi_{135}\right)\left(\max_{x_6}\psi_{246}\right)$

Figure 2: The compression step of the expression tree from Example 4. Recall that Figure (1d) (right) depicted the expression tree at the end of the compartmentalization step. (2a) shows a compression where node $\{7\}$ is merged into its parent $\{3\}$ (since they both have the same tag "max"). (2b) shows another compression where $\{4\}$ is merged into $\{1, 2\}$. Since no further compression is possible, (2b) (right) depicts the final expression tree.

Proof of Proposition 5.5. Due to the fact that the different (conditional) connected components do not interact, when we run InsideOut on σ and π for all $k \in [n]$, we have $U_k^\sigma = U_k^\pi$. This observation proves both claims. □

Proof of Theorem 5.6. We only need to show that $\mathsf{EVO}(\varphi) \subseteq \mathsf{CWE}(\mathsf{LinEx}(P))$ because the reverse containment follows from Proposition 5.5 and Theorem 5.3. Also, without loss of generality we can assume that the root of the expression tree F is empty, and it has one child node L. (If there were different connected components, they can interleave arbitrarily and we prove each of them separately.)

Fix an arbitrary $\sigma = (v_1, \dots, v_n) \in \mathsf{EVO}(\varphi)$. Then $v_1 \in L$ by Lemma 5.4. For each connected component C of $\mathcal{H} - \{v_1\}$, define a sub-query φ_C with variable ordering σ_C on the conditional factors $\{\psi_S(\cdot \mid x_{v_1}) \mid S \in \mathcal{E} \wedge S \cap \mathcal{V}(C) \neq \emptyset\}$, where σ_C is the subsequence of σ obtained by picking out vertices in $\mathcal{V}(C)$. Let P_C be the precedence poset of the expression tree for φ_C. By induction on the number of variables, we know $\sigma_C \in \mathsf{EVO}(\varphi_C) \subseteq \mathsf{CWE}(\mathsf{LinEx}(P_C))$. Hence, there exists $\pi_C \in \mathsf{LinEx}(P_C)$ that is CW-equivalent to σ_C.

The expression tree P_C for φ_C consists of an empty root (with 'free' tag). This root has only one child node L_C. The subtree rooted at L_C is called an L_C-subtree. Now, if we attach all the roots of all the L_C-subtrees together, we will create a tree whose root R is the union of the L_C. Then, we add v_1 to the root R and add an empty parent to R then we will obtain exactly the expression tree P.

From this observation, we can pick a variable ordering π that is consistent with all π_C such that π starts with v_1, followed by variables in $L - \{v_1\}$. It follows that $\pi \in \mathsf{LinEx}(P)$ and π is CW-equivalent to σ. □

B.3 Example of Completeness

Here, we give an example of component-wise equivalence (Definition 8) and the role it plays in completeness (Theorem 5.6).

Example 5. (Component-wise equivalence and completeness) Consider the following FAQ query with two different semiring aggregates ($\sum$ and max) and three variables, all are bound.

$$\varphi = \sum_{x_1}\max_{x_2}\sum_{x_3}\psi_{12}\psi_{13}.$$

(ψ_{ij} above denotes an input factor whose support is $\{X_i, X_j\}$.) For this query, $\mathsf{EVO}(\varphi) = \{(1,2,3), (1,3,2), (3,1,2)\}$. Ignoring the dummy free variable X_0, the expression tree of φ consists of a root with the tag '$\sum$' containing the variables $\{X_1, X_3\}$ and a single child node with tag 'max' containing $\{X_2\}$. Therefore, $\mathsf{LinEx}(P) = \{(1,3,2), (3,1,2)\} \subseteq \mathsf{EVO}(\varphi)$, as suggested by Theorem 5.3. Note that the original ordering $(1,2,3) \notin \mathsf{LinEx}(P)$. However, $(1,2,3)$ is component-wise equivalent to $(1,3,2)$ (See Definition 8). Therefore,

$$\mathsf{CWE}(\mathsf{LinEx}(P)) = \{(1,3,2), (1,2,3), (3,1,2)\} = \mathsf{EVO}(\varphi),$$

just as predicted by Theorem 5.6. Moreover, by Proposition 5.5, $\mathsf{faqw}((1,2,3)) = \mathsf{faqw}((1,3,2)) = 1$. Therefore, when searching for the variable ordering with the best FAQ-width, the ordering $(1,2,3)$ is redundant, hence it is sufficient to consider $\mathsf{LinEx}(P)$, just as suggested by Corollary 5.7.