Datalog and Its Modern Extensions: Semantics, Convergence, and Optimizations

Suggested Citation: (2024), "Datalog and Its Modern Extensions: Semantics, Convergence, and Optimizations", : Vol. XX, No. XX, pp 1–25. DOI: XXX.

now
the essence of knowledge
Boston — Delft

Contents

1

The Sum-Product Abstraction

By the late 1980s and early 1990s, it was recognized that there is a very expressive and powerful abstraction that can be used to capture a massive number of problems in constraint satisfaction, relational databases, logic, probabilistic inference, graph theory, information theory, signal processing, digital communications, and discrete optimization (Shenoy and Shafer, 1988; Shafer and Shenoy, 1991). The abstraction was powerful due to two main reasons: the algebraic formulation of the problem, and the common algorithmic technique used to solve the problem.

There were many different names given to this problem, including *local computation* (Lauritzen and Spiegelhalter, 1988), *marginalization the product function* (Aji and McEliece, 2000), and *sum-product computation* (Dechter, 1997), and another slightly more general *valuation algebra* (Kohlas and Shenoy, 2000). We shall use SumProd to refer to this problem. As shall be evident shortly, a SumProd query is a very natural generalization of a conjunctive query. Moreover, as we shall see in Chapter **??**, this problem shares an algebraic commonality with Datalog.

Since SumProd is defined over an algebraic structure called *semiring*, we will start this chapter with a brief introduction to semirings in

Section 1.1. We will then define the SumProd problem in Section 1.2 and present a host of example problems from many different domains that can be modeled as a SumProd problem.

In Section 1.3 and 1.4, we present two complementary "meta" algorithms for SumProd, one based on backtracking search, the other on dynamic programming. The backtracking search algorithm can be shown to be "worst-case optimal" in some precise sense, and it is tightly connected with the notion of "worst-case optimal joins" in relational database query processing. The dynamic programming technique is a powerful algorithmic technique called *variable elimination*.

Finally, Section 1.5 explains how *tree-decomposition* and *message-passing* algorithms are related to variable elimination, and how the back-tracking search algorithm can be used as a building block for answering general SumProd queries.

1.1 Semirings

If we look with a magnifying glass at how a standard relational query operates we notice that it combines tuples using two basic primitives: it *joins* tuples during the computation of joins or cartesian products, and it *unions* tuples during duplicate elimination or in order to express a union operator. An algebraic structure that supports these two basic operations is called a semiring.

Formally, a *semiring* is a tuple $\boldsymbol{S} = (S, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, where:

- S is a set.
- $(S, \oplus, \mathbf{0})$ is a commutative monoid, meaning that $\oplus$ is associative, commutative, and has identity $\mathbf{0}$,
- $(S, \otimes, \mathbf{1})$ is a (not necessarily commutative) monoid,
- $\otimes$ distributes over $\oplus$: $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and similarly $(y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)$, and,
- $\mathbf{0}$ is absorptive, meaning $\mathbf{0} \otimes x = x \otimes \mathbf{0} = \mathbf{0}$

Readers familiar with rings will recognize that a semiring is like a ring but misses the additive inversion. When the absorption rule is dropped,

then the semiring is called a *pre-semiring*. When the multiplicative operation $\otimes$ is commutative, then $\boldsymbol{S}$ is called a *commutative (pre-)semiring*. If $\boldsymbol{S}, \boldsymbol{S}'$ are two pre-semirings, then a *homomorphism* is a function $h : S \to S'$ that commutes with the semiring operations, $h(x \oplus y) = h(x) \oplus' h(y)$ and $h(x \otimes y) = h(x) \otimes' h(y)$, and also preserves the identities, meaning $h(\mathbf{0}) = \mathbf{0}'$ and $h(\mathbf{1}) = \mathbf{1}'$.

The following are classic examples of semirings:

- The Boolean semiring: $\boldsymbol{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$.
- The natural numbers with the standard operations, $(\mathbb{N}, +, \times, 0, 1)$ and, similarly, the real numbers $\mathbb{R}$.
- The semiring of non-negative real numbers $(\mathbb{R}_+, +, \times, 0, 1)$.
- The tropical semiring: $\mathsf{Trop} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$. Here "addition" is $x \oplus y = \min(x, y)$ and "multiplication" is $x \otimes y = x + y$. We note that some texts define the tropical semiring over $\mathbb{N} \cup \{\infty\}$ (Gunawardena, 1998) or over $\mathbb{R} \cup \{\infty\}$ (Gondran and Minoux, 2008). Some texts use the nation Trop^+ to emphasize that the semiring is over non-negative numbers. We will use the simpler notation Trop in this chapter.
- If $\boldsymbol{S}$ is a semiring, then for any $n \in \mathbb{N}$, the set of $n \times n$ $\boldsymbol{S}$-matrices forms a non-commutative semiring, denoted $\boldsymbol{S}^{n \times n}$, where addition and multiplication are the usual matrix operations, and the identities are $\mathbf{0}_n$ and I_n respectively (the zero-matrix, and the identity matrix).

  Notice that, if $\boldsymbol{S}$ is a pre-semiring instead of a semiring, then $\boldsymbol{S}^{n \times n}$ is neither a pre-semiring nor a semiring, because it does not have an identity for multiplication: $AI_n \neq A$, since $\mathbf{0} \cdot a_{ij} \neq \mathbf{0}$, and therefore $\boldsymbol{S}^{n \times n}$. Instead, the following weaker identity holds: $A(I_n + B) = A + AB$, see (Lehmann, 1977).

The semiring of *polynomials* $\mathbb{N}[\boldsymbol{x}]$ plays a special role in our discussion, and we introduce it here in some detail. Fix a set of N variables $\boldsymbol{x} = \{x_1, \ldots, x_N\}$. Then $(\mathbb{N}[\boldsymbol{x}], +, \cdot, 0, 1)$ is the semiring of *formal polynomials* with coefficients in $\mathbb{N}$, where each element $f \in \mathbb{N}[\boldsymbol{x}]$ is a formal

polynomial (e.g. $f = 3x_1^3x_3x_4 + x_1x_2^4$), and where addition and multiplication are defined in the standard way. When we discuss polynomials we denote variables by lower case, to distinguish them from the logical variables in datalog, which are denoted by upper case.

We say that $\mathbb{N}[\boldsymbol{x}]$ is the *freely generated commutative semiring* by the set $\boldsymbol{x}$ because it enjoys the following property. For any commutative semiring $\boldsymbol{S}$ and any function $h : \boldsymbol{x} \to \boldsymbol{S}$, there exists a homomorphism of semirings $\bar{h} : \mathbb{N}[\boldsymbol{x}] \to \boldsymbol{S}$ that extends h: $h(x) = \bar{h}(x)$ for all $x \in \boldsymbol{x}$. The function $\bar{h}$ is simply the evaluation function, which evaluates any polynomial on the values $h(x_1), \ldots, h(x_N)$. For a simple example, consider the polynomial:

$$f(x_1, x_2, x_3) = 2x_1^3x_2 + x_2^2x_3^5$$

Fix a commutative semiring $\boldsymbol{S}$, and let $h : \boldsymbol{x} \to S$ map the variables x_1, x_2, x_3 to the values $a_1, a_2, a_3 \in S$. Then, the value of the polynomial is:

$$f(a_1, a_2, a_3) = (a_1^3 \otimes a_2) \oplus (a_1^3 \otimes a_2) \oplus (a_2^2 \otimes a_3^5)$$

where we used the common convention $a_1^3 \stackrel{\text{def}}{=} a_1 \otimes a_1 \otimes a_1$.

Of course, $\mathbb{N}[\boldsymbol{x}]$ is a commutative semiring. The freely generated *non-commutative* semiring also exists: it consists of formal polynomials whose monomials are strings in $\boldsymbol{x}^*$. For example, $f = 3x_1x_3x_1x_1 + 5x_1x_1x_2$. We do not need the freely non-commutative semiring in this chapter, and will not consider it any further.

1.2 The Sum-Product Problem

We are now ready to define the SumProd query. Let $\mathcal{H} = (V, \mathcal{E})$ denote a hypergraph, where V is a set of variables. The domain of a variable $v \in V$ is denoted by $\mathsf{Dom}(v)$; abusing notations, for $S \subseteq V$, we define $\mathsf{Dom}(S) := \prod_{v \in S} \mathsf{Dom}(v)$. To each hyperedge $S \in \mathcal{E}$, there corresonds a "factor" function $\psi_S : \prod_{v \in S} \mathsf{Dom}(v) \to D$, where D denotes a common domain of these functions (such as $D = \mathbb{R}$, or $D = \{\texttt{true}, \texttt{false}\}$). For each subset S of variables, we use $\boldsymbol{x}_S \in \prod_{v \in S} \mathsf{Dom}(v)$ to denote a tuple of values over the domains of variables in S.

Definition 1.2.1 (The SumProd query)**.** The SumProd problem is to evaluate the following query (i.e. to compute the output factor $\varphi(\boldsymbol{x}_F)$), given the input factors ψ_S, $S \in \mathcal{E}$:

$$\varphi(\boldsymbol{x}_F) := \bigoplus_{\boldsymbol{x}_{V-F}} \bigotimes_{S \in \mathcal{E}} \psi_S(\boldsymbol{x}_S) \tag{1.1}$$

where the sum $\oplus$ and $\otimes$ operators are such that $(D, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a semiring.

Example 1.2.2 (Conjunctive query)**.** When the semiring is a Boolean semiring, the SumProd query is precisely a *conjunctive* query (i.e. a Datalog rule), where the summation corresponds to existential quantification, the product is a conjunction, and the factors are input relations.

From this observation, the connection to relational databases, logic programming and constraint satisfaction is apparent. Essentially, each factor ψ_S represents a *constraint* over the variables S, the product $\otimes$ becomes a conjunction of those constraints, and the summation $\oplus$ is an existential quantification which asks whether the constraints can be satisfied together.

We next present a few examples illustrating the power of this abstraction in expressing problems in a variety of domains. *Many* more examples can be found in (Pearl, 1989; Aji and McEliece, 2000; Kohlas and Wilson, 2008; Wainwright and Jordan, 2008; Khamis *et al.*, 2016) and references thereoff.

Example 1.2.3 (Matrix Chain Multiplication)**.** Given a series of matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n$ over some field $\mathbb{F}$, where the dimension of $\boldsymbol{A}_i$ is $p_i \times p_{i+1}$, $i \in [n]$, we wish to compute the product $\boldsymbol{A} = \boldsymbol{A}_1 \cdots \boldsymbol{A}_n$. This problem is a SumProd problem, set up as as follows. There are $n+1$ variables $X_1, \ldots, X_{n+1}$ with domains $\mathsf{Dom}(X_i) = [p_i]$, for $i \in [n+1]$. For each $i \in [n]$, the matrix $\boldsymbol{A}_i$ is a function of two variables

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

where $\psi_{i,i+1}(x, y) = (\boldsymbol{A}_i)_{xy}$. The 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 1.2.4 (SAT and #SAT, CSP and #CSP)**.** Let φ be a CNF formula over n Boolean variables $X_1, \ldots, X_n$. Let $\mathcal{H} = ([n], \mathcal{E})$ be the hypergraph of φ. Then, each clause of φ is a factor ψ_S, and the question of whether φ is satisfiable is the same as evaluating the constant function $\varphi = \bigvee_{\boldsymbol{x}} \bigwedge_{S \in \mathcal{E}} \psi_S(\boldsymbol{x}_S)$. To solve #SAT, we change the input factors ψ_S to map to $\{0, 1\}$, and the semiring to be sum-product: $\varphi = \sum_{\boldsymbol{x}} \prod_{S \in \mathcal{E}} \psi_S(\boldsymbol{x}_S)$. The more general problems CSP and #CSP are encoded similarly.

Example 1.2.5 (Probabilistic graphical models)**.** We consider *probabilistic graphical models* (PGMs) on discrete finite domains. Without loss of generality, consider only *undirected graphical models* (i.e. *Markov random fields*, *factor graphs*). Directed PGMs can be turned into this undirected form by the process of *moralizing* (Wainwright and Jordan, 2008).

The model can be represented by a hypergraph $\mathcal{H} = ([n], \mathcal{E})$, where there are n discrete random variables $X_1, \ldots, X_n$ on finite domains $\mathsf{Dom}(X_1), \ldots, \mathsf{Dom}(X_n)$ respectively, and m *factors* (also called *potential functions*):

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

Typically, we want to *learn* the model and perform *inference* from the model. For example, we might want to (1) Compute the marginal distribution of some set of variables, (2) Compute the conditional distribution $p(\boldsymbol{x}_A \mid \boldsymbol{x}_B)$ of some set of variables A given specific values to another set of variables $\boldsymbol{x}_B$, or (3) Compute $\arg\max_{\boldsymbol{x}_A} p(\boldsymbol{x}_A \mid \boldsymbol{x}_B)$ (for MAP queries, for example).

When we condition on some variables, we can restrict the factors to only those entries that match the conditioned variables. It is obvious that the first two questions above are special cases of the SumProd problem on the sum-product semiring. The third question is on the max-product semiring (Wainwright and Jordan, 2008).

Example 1.2.6 (Permanent)**.** #SAT is #**P**-complete. Another canonical #**P**-complete problem is Permanent, which is the problem of evaluating the *permanent* $\mathsf{perm}(\boldsymbol{A})$ of a given *binary* square matrix $\boldsymbol{A}$. Let S_n denote the symmetric group on $[n]$, then the permanent of $\boldsymbol{A} = (a_{ij})_{i,j=1}^n$ is defined as follows. $\mathsf{perm}(\boldsymbol{A}) := \sum_{\pi \in S_n} \prod_{i=1}^n a_{i\pi(i)}$.

The Permanent problem can be written in the sum-product form as follows. Say the input matrix is $\boldsymbol{A} = (a_{ij})$, there is a singleton factor ψ_i for each vertex i, where $\psi_i(j) = a_{ij}$. Then, there are $\binom{n}{2}$ factors ψ_{jk} for $j \neq k \in [n]$, where $\psi_{jk}(x, y) = 1$ if $x \neq y$ and 0 if $x = y$. The problem is to evaluate the constant function

$$\varphi = \sum_{\boldsymbol{x}} \prod_{i \in [n]} \psi_i(x_i) \prod_{(j \neq k)} \psi_{jk}(x_j, x_k).$$

Example 1.2.7 (Discrete Fourier Transform)**.** Recall that the discrete Fourier transform (DFT) is the matrix vector multiplication where $A_{xy} = e^{i2\pi \frac{x \cdot y}{n}}$. For this example, we will consider the case when $n = p^m$ for some prime p and integer $m \geq 1$. Recall that the DFT is defined as follows:

$$\varphi(x) = \sum_{y=0}^{n-1} b_y \cdot e^{i2\pi \frac{x \cdot y}{n}}.$$

Write $x = \sum_{i=0}^{m-1} x_i \cdot p^i$ and $y = \sum_{j=0}^{m-1} y_j \cdot p^j$ in their base-p form. Then we can re-write the transform above as follows:

$$\varphi(x_0, x_1, \ldots, x_{m-1}) = \sum_{(y_0, \ldots, y_{m-1}) \in \mathbb{F}_p^m} b_y \cdot e^{i2\pi \frac{\sum_{0 \leq j+k \leq 2m-2} x_j \cdot y_k \cdot p^{j+k}}{n}}.$$

Recalling that $n = p^m$, note that for any $j+k \geq m$, we have $e^{i2\pi \frac{x_j \cdot y_k \cdot p^{j+k}}{n}} = 1$. Thus, the above is equivalent to

$$\varphi(x_0, x_1, \ldots, x_{m-1}) = \sum_{(y_0, \ldots, y_{m-1}) \in \mathbb{F}_p^m} b_y \cdot \prod_{0 \leq j+k < m} e^{i2\pi \frac{x_j \cdot y_k}{p^{m-j-k}}}.$$

The above immediately suggests the following reduction to SumProd. Let $\mathcal{H} = ([n], \mathcal{E})$ where $[n] = \{X_0, X_1, \ldots, X_{m-1}, Y_0, Y_1, \ldots, Y_{m-1}\}$ and $\mathcal{E}$ has an edge (X_j, Y_k) for every $j, k \in \{0, 1, \ldots, m-1\}$ such that $j + k < m$. Further, there is another edge $(Y_0, Y_1, \ldots, Y_{m-1})$. The variable domains are $\mathsf{Dom}(X_i) = \mathsf{Dom}(Y_i) = \{0, 1, \ldots, p-1\}$. For every $j, k \in \mathbb{F}_p$ such that $j + k < m$, the corresponding factor

$$\psi_{X_j, Y_k} : \mathbb{F}_p^2 \to \mathbf{D}$$

is defined as

$$\psi_{X_j, Y_k}(x, y) = e^{i2\pi \frac{x \cdot y}{p^{m-j-k}}}.$$

Finally, the factor $\psi_{\boldsymbol{Y}} : \mathbb{F}_p^m \to \mathbf{D}$ is defined as

$$\psi_{\boldsymbol{Y}}(y_0, y_1, \ldots, y_{m-1}) = b_{(y_0, y_1, \ldots, y_{m-1})}.$$

Then the output is

$$\varphi(x_0, x_1, \ldots, x_{m-1}) = \sum_{(y_0, \ldots, y_{m-1}) \in \mathbb{F}_p^m} \psi_Y(y_0, y_1, \ldots, y_{m-1}) \cdot \prod_{0 \le j+k < m} \psi_{X_j, Y_k}(x_j, y_k).$$

The SumProd problem was shown to be NP-hard in (Cooper, 1990), which should not be surprising because we knew that answering conjunctive queries was NP-hard in query complexity (Chandra and Merlin, 1977). Beyond conjunctive queries, (Khamis *et al.*, 2016) generalized SumProd into Functional Aggregate Query (or FAQ), to capture even more problems where multiple semirings are required to model them. The same algorithmic framework presented in the next section can also be applied to FAQs.

1.3 Backtracking Search and Worst-Case Optimal Joins

In order to discuss the computational complexity of solving the SumProd problem (1.1), we need to be specific on how the input factors ψ_S are represented.

Listing Representation We shall assume the *listing* representation, where every input factor ψ_S is represented as an arity-$(|S|+1)$ relation: $|S|$ attributes to store the key $\boldsymbol{x}_S$, and the extra column storying the *value* $\psi_S(\boldsymbol{x}_S)$. For example, every input factor in Example 1.2.3 are represented as a table of triples (i, j, a_{ij}).

When the semiring is a Boolean semiring, the column storing the value is not needed; this case corresponds precisely to the *conjunctive* queries in relational databases. Furthermore, when the value $\psi_S(\boldsymbol{x}_S)$ is $\mathbf{0}$ (the zero element of the semiring), we will omit the corresponding tuple from the relation. This encoding is similar to that of sparse matrix represetation for arity-2 relations.

We use $\|\psi\|$ to denote the number of non-zero-valued tuples of the given function ψ. In data-complexity, $\|\psi\|$ is the *size* of the input factor ψ.

Join and backtracking search A simple way to solve the SumProd problem (1.1) is to enumerate all tuples $\boldsymbol{x}_V$ for which $\varphi(\boldsymbol{x}_S) \neq \mathbf{0}$, for all $S \in \mathcal{E}$. Then, the output factor $\varphi(\boldsymbol{x}_F)$ can be computed by aggregating over all such tuples with the $\oplus$ operator, grouping by the free variables $\boldsymbol{x}_F$.

If we think of $\psi_S(\boldsymbol{x}_S)$ as encoding constraints where $\boldsymbol{x}_S$ satisfies the constraint if $\psi_S(\boldsymbol{x}_S) \neq \mathbf{0}$, then one can identify all satisfying tuples $\boldsymbol{x}_V$ with a *backtracking search* algorithm. We will assume that the search is done via a fixed ordering of the variables in V. If we think of $\psi_S(\boldsymbol{x}_S)$ as a database relation, then enumerating the satisfying tuples is exactly a (natural) join query evaluation problem.

Backtracking search is one way to solve the join query evaluation problem. While join is a more general primitive, we have chosen to stick to the backtracking search angle in this section because it contrasts very well with the dynamic programming approach presented in the next section; furthermore, efficient join computation is a *much* deeper database topic, both in theory and in practice than what can be reasonably discussed in this section.

Backtracking search is a generic algorithmic technique to solve the SumProd problem (1.1), but it may not be the most efficient. However, there are some non-trivial properties of this algorithm that were only discovered recently (Ngo *et al.*, 2012; Veldhuizen, 2014). In particular, *how* backtracking search is implemented can have a significant impact on the runtime. We briefly present these properties here as they are needed to understand the notions of "width" in the next section.

Worst-case optimal join and fractional edge cover The backtracking search algorithm to solve (1.1) can be analyzed in terms of a combinatorial property of the input hypergraph $\mathcal{H} = (V, \mathcal{E})$. In database terminology, this is the *query hypergraph*. The *fractional edge cover* of $\mathcal{H}$ is an assignment of a non-negative "weight" λ_S for each $S \in \mathcal{E}$

such that for every $v \in V$, the total weight of edges containing v is at least 1; and the *fractional edge cover number* $\rho^*(\mathcal{H})$ is the minimum total weight over all fractional edge covers. In particular, $\rho^*(\mathcal{H})$ is the objective value of the following linear program:

$$\min \sum_{S \in \mathcal{E}} \lambda_S \tag{1.2}$$

$$\text{s.t.} \sum_{S \in \mathcal{E}: v \in S} \lambda_S \geq 1, \quad \forall v \in V, \tag{1.3}$$

$$\lambda_S \geq 0, \quad \forall S \in \mathcal{E}. \tag{1.4}$$

Example 1.3.1 (Loomis-Whitney hypergraphs)**.** The *Loomis-Whitney hypergraph* of order n, named after (Loomis and Whitney, 1949) is the hypergraph $\mathcal{H}$ on $V = [n]$ and $\mathcal{E} = \binom{[n]}{n-1}$, i.e. the hypergraph where every edge contains all but one vertex. The fractional edge cover number of the Loomis-Whitney hypergraph is $\rho^*(\mathcal{H}) = n/(n-1)$, where $\lambda_S = 1/(n-1)$ for all $S \in \mathcal{E}$. The triangle (non-hyper) graph is the Loomis-Whitney hypergraph of order 3, with $\rho^* = 3/2$.

The main property of the backtracking search algorithm that is of our interest is the following

Theorem 1.3.2. There is a way to implement backtracking search to solve (1.1) so that it runs in time $O(|\mathcal{E}| \cdot |V|^2 \cdot \log N \cdot N^{\rho^*(\mathcal{H})})$.

In some database applications, it is often convenient to subsume factors that are dependent on the size of the query hypergraph and $\log N$; in that case, we say that the backtracking search algorithm runs in time $\tilde{O}(N^{\rho^*(\mathcal{H})})$.

In the context of database query evaluation, such an algorithm is called a *worst-case optimal*, because in the worst case the output size is $N^{\rho^*(\mathcal{H})}$. The subject of worst-case optimal join algorithms is much more involved than what was presented here. In particular, it has an interesting history with deep connection to information theory. See (Ngo, 2018) for more historical context.

Example 1.3.3 (Loomis-Whitney queries)**.** Loomis-Whitney queries are SumProd queries whose hypergraphs are Loomis-Whitney. For example,

order-3 Loomis-Whiney query is the triangle query, such as

$$Q(A, B, C) \text{ :- } R(A, B) \wedge S(B, C) \wedge T(A, C).$$

for some relations R, S, T. If R, S, T are the same edge relation of a graph, then the query asks for listing all triangles in a graph. Worst-case optimal join returns all the triangles in time $\tilde{O}(N^{3/2})$, where N is the number of edges in the graph. This example also illustrates a key strength of worst-case optimal joins: there are input instances for which *any* join-project query plan will necessarily runs in $\Omega(N^2)$-time (Ngo *et al.*, 2012).

An order-4 Loomis-Whitney query is of the form

$$Q(A, B, C, D) \text{ :- } R(A, B, C) \wedge S(A, B, D) \wedge T(A, C, D) \wedge U(B, C, D).$$

This query can be answered in $O(N^{4/3})$-time.

It is worth noting that, if R, S, T, U are dense 3D-tensors, over domains of size M for each of the variables A, B, C, D, then $N = M^3$ (the number of non-zero entries in each tensor), and the bound is $N^{4/3} = M^4$, which is the same as the naïve bound. In particular, the fractional edge covering bound is good when the input factors (or tensors) are sparse.

1.4 Dynamic Programming with Variable elimination

Backtracking search has a well-known weakness: it may repeat the same computation many times. The other side of the coin is dynamic programming, which "caches" the results of subproblems and reuses them when needed. Given how expressive the SumProd abstraction is, it was equally amazing that there is a unified *meta* dynamic programming algorithm to solve (1.1): it is called the *variable elimination* algorithm, or equivalently the *message passing* or *belief propagation* algorithm (Pearl, 1982; Shafer and Shenoy, 1990; Zhang and Poole, 1994; Dechter, 1999; Aji and McEliece, 2000).

As surveyed and nicely presented in (Aji and McEliece, 2000; Kohlas and Wilson, 2008), this meta-algorithm has as special cases a massive number of well-known (even best-known) algorithms for problems that can be cast as SumProd, such as the fast Hadamard transform, the

turbo decoding algorithm, the FFT on any finite Abelian group, many best known algorithms for answering inference queries in graphical models (Wainwright and Jordan, 2008) (such as Viterbi's algorithm, Pearl's "belief propagation" algorithm, the Shafer-Shenoy probability propagation algorithm, the Baum-Welch "forward-backward" algorithm, and discrete-state Kalman filtering).

From the database perspective, variable elimination is a very general algorithmic abstraction for designing logical query optimizers, which has been adopted in a couple of commercial relational DBMSs and shown to be very robust in practice (Nguyen *et al.*, 2015). In fact, when formulated correctly, Yannakakis's classic algorithm for computing join queries (Yannakakis, 1981) is an earlier incarnation of this algorithm[1].

The variable elimination (meta) algorithm is very simple; it can be explained with a couple of examples. Suppose our problem is to count the number of tuples (x, y, z, u) satisfying the conjunction $R(x, y) \wedge S(y, z) \wedge T(z, u)$, where R, S, and T are input relations. This problem can be represented algebraically as follows. Overloading notations, we use R to denote the indicator function that $(x, y) \in R$, i.e. $R(x, y) = \mathbf{1}_{(x,y)\in R}$. Similarly, S and T are indicator functions. In particular, we "lifted" the relations from the Boolean domain to be real functions for the purpose of counting. Then, the counting query is of the form (1.1):

$$Q() = \sum_a \sum_b \sum_c \sum_d R(a, b) \cdot S(b, c) \cdot T(c, d). \tag{1.5}$$

Note the freedom we have with the range of a, b, c, d: since R, S, and T are indicator functions, we are now free to sum over the entire ranges of a, b, c, d as values that do not belong to the input relations are zeroed out. The problem can be solved by applying the distributive law, eliminating

[1] See (Khamis *et al.*, 2016) for an explanation

one variable at a time:

$$\begin{aligned} Q() &= \sum_a \sum_b \sum_c \sum_d R(a,b) \cdot S(b,c) \cdot T(c,d) \qquad (1.6) \\ &= \sum_a \sum_b \sum_c R(a,b) \cdot S(b,c) \cdot \sum_d T(c,d) \\ &= \sum_a \sum_b \sum_c R(a,b) \cdot S(b,c) \cdot W(c) \\ &= \sum_a \sum_b R(a,b) \cdot \sum_c S(b,c) \cdot W(c) \\ &= \sum_a \sum_b R(a,b) \cdot V(b) \end{aligned}$$

The algebraic reasoning leads to a query plan, where we first compute an intermediate result $W(c)$, then another intermediate result $V(b)$, and finally Q is computed. It is easy to see how this plan can be translated to relational query plans and the meta algorithm takes $\tilde{O}(N)$-time.

If the query was not a count query, but rather a proper conjunctive query

$$Q() = \bigvee_a \bigvee_b \bigvee_c \bigvee_d R(a,b) \wedge S(b,c) \wedge T(c,d).$$

Then, precisely the same query plan works. All we have to do is to replace $\sum$ with $\vee$ and $\cdot$ with $\wedge$. That is the power of algebraic abstraction. The above ideas are often expressed in the database textbooks as *pushing aggregates through joins* and *pushing existential quantifications through joins* (Ramakrishnan and Gehrke, 2003). They are all instance of the variable elimination framework.

While one can use variable orderings as representations of query plans based on variable elimination, in practice it is often more robust to use an equivalent representation called tree-decompositions, as we shall see in the next section.

Is that enough or do we need more examples / details? –Hung.

1.5 Tree-Decomposition and Message-Passing

Another way to express the query plan from variable elimination is the notion of *tree decomposition* (or *hypertree* decomposition). The reader is

referred to (Wainwright and Jordan, 2008; Gottlob *et al.*, 2016) for more detailed descriptions of tree-decompositions and their usage in graphical model inference and constraint satisfaction. This chapter explains the main intuition through an example.

The variable elimination strategy shown in (1.6) can be captured with a combinatorial object where we create a "bag" (a set of variables) for each sub-problem in the query plan. The first bag is $\{c, d\}$ because the sub-problem producing $W(c)$ involves variables $\{c, d\}$. This bag produces an intermediate result $W(c)$ which is called a *message* passed to the second bag $\{b, c\}$. Finally, the bag $\{b, c\}$ passes the message $V(b)$ to the bag $\{a, b\}$ in the same manner. The combinatorial structure connecting these bags, where there is an edge between two bags if one passes a message to the other, is called a *tree decomposition* of the input query (1.5). We shall define this notion formally below.

The variable elimination algorithm when expressed via tree decompositions is called the *message passing* algorithm. See Figure 1.1 for an illustration.

Figure 1.1: A Tree Decomposition for problem (1.5)

The main optimization problem one has to solve in order to apply the variable elimination framework is to decide the variable ordering under which to gradually push the summations in (to eliminate variables). To decide if a variable order is "better" than another, we then need a cost function to tell them apart, in the same way query optimizers have cost functions for selecting query plans.

One sensible cost function is the worst-case cardinalities over all intermediate result. (Minimizing the intermediate result sizes is a main goal of DBMS query optimizers.) Each intermediate result is computed over a bag of the corresponding tree decomposition. Thus, minimizing the maximum cardinality of the intermediate results is the same as minimizing the maximum number of satisfying tuples over each bag of the tree decomposition.

In the solution we just described for Problem (1.5), the worst-case

cardinalities of all intermeidate results is $O(N)$, i.e. linear in the input size. For a more interesting example, consider the query (in SumProd form):

$$Q = \sum_{a,b,c,d,e,f} R(a,b)S(a,c)T(b,c,d,e)W(e,f)V(d,f) \tag{1.7}$$

One way to answer this query is the following variable elimination strategy:

$$\begin{aligned}
Q &= \sum_{a,b,c,d,e,f} R(a,b)S(a,c)T(b,c,d,e)W(e,f)V(d,f) \\
&= \sum_{a,b,c,d,e} R(a,b)S(a,c)T(b,c,d,e) \sum_{f} W(e,f)V(d,f) \\
&= \sum_{a,b,c,d,e} R(a,b)S(a,c)T(b,c,d,e)M_1(e,d) \\
&= \sum_{a,b,c} R(a,b)S(a,c) \sum_{e,d} T(b,c,d,e)M_1(e,d) \\
&= \sum_{a,b,c} R(a,b)S(a,c)M_2(b,c) \\
&= \sum_{a,b} R(a,b) \sum_{c} S(a,c)M_2(b,c) \\
&= \sum_{a,b} R(a,b)M_3(a,b)
\end{aligned}$$

The corresonding tree decomposition (TD) is shown in Figure 1.2. This TD has a worst-case cost of N^2, because the bag $\{d,e,f\}$ is the join of two input relations $W(e,f)$ and $V(d,f)$, which is of size $O(N^2)$ in the worst-case. Similarly, one can reason straightforwardly that bag $\{b,c,d,e\}$ has $O(N)$-size at worst, bag $\{a,b,c\}$ has $O(N^2)$-size at worst, and bag $\{a,b\}$ has $O(N)$-size at worst. Overall, this query plan has cost $O(N^2)$. The exponent 2 is called the (generalized) *hypertree width* of the tree decomposition (Gottlob *et al.*, 2016).

There is, however, a better query plan that yields a cost less than quadratic. This query plan is based on the notion of *worst-case optimal join algorithms* (WCOJ, as presented in Section 1.3) and the so-called *fractional edge covering number* of a (sub-)query (Grohe and Marx, 2014). We illustrate these concepts with the following variable elimination

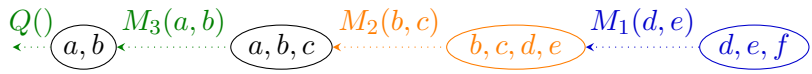

Figure 1.2: A Tree Decomposition for query (1.7)

strategy, where an *indicator projection* of the factor $T(b,c,d,e)$ is used to *reduce* the size of the intermediate result.

$$
\begin{aligned}
Q &= \sum_{a,b,c,d,e,f} R(a,b)S(a,c)T(b,c,d,e)W(e,f)V(d,f) \\
&= \sum_{a,b,c,d,e} R(a,b)S(a,c)T(b,c,d,e) \underbrace{\sum_{f} \pi_{de} T(d,e)W(e,f)V(d,f)}_{\tilde{O}(N^{3/2}),\ \text{WCOJ}} \\
&= \sum_{a,b,c,d,e} R(a,b)S(a,c)T(b,c,d,e)M_1(d,e) \\
&= \sum_{a,b,c} R(a,b)S(a,c) \sum_{d,e} T(b,c,d,e)M_1(d,e) \\
&= \sum_{a,b,c} R(a,b)S(a,c)M_2(b,c) \\
&= \underbrace{\sum_{a,b,c} R(a,b)S(a,c)M_2(b,c)}_{\tilde{O}(N^{3/2}),\ \text{WCOJ}}
\end{aligned}
$$

The indicator projection $\pi_{d,e}T$ is the projection of the factor T onto the variables d, e. It is a function of d and e, and $\pi_{d,e}T(d,e) = 1$ iff there are (b,c) for which $T(d,e,b,c) \neq \mathbf{0}$. It should be obvious that introducing this indicator projection does not change the semantic of the query. This idea was introduced in (Khamis *et al.*, 2016). The bag $\{e,f,d\}$ that is used to create the message M_1 is of size $O(N^{1.5})$. This number 1.5 is the *fractional edge covering* number of the triangle graph consisting of edges de, ef and df. Similarly, the bag $\{a,b,c\}$ is also bounded by $N^{1.5}$. Thus, overall, this query plan with TD shown in Figure 1.3 has cost $O(N^{1.5})$, and it is said to have a *fractional hypertree width* of 1.5 (Grohe and Marx, 2014).

The variable ordering could have been different, but with the same

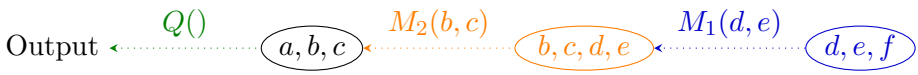

Figure 1.3: Tree Decomposition

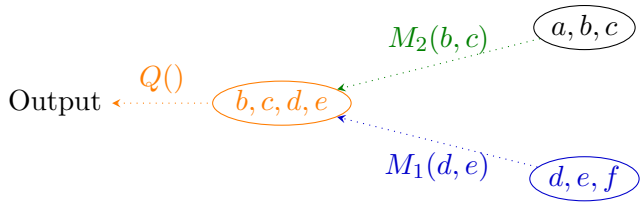

Figure 1.4: Tree Decomposition

cost, as shown in the following strategy:

$$
\begin{aligned}
Q &= \sum_{a,b,c,d,e,f} R(a,b)S(a,c)T(b,c,d,e)W(e,f)V(d,f) \\
&= \sum_{a,b,c,d,e} R(a,b)S(a,c)T(b,c,d,e) \underbrace{\sum_{f} \pi_{de} T(d,e)W(e,f)V(d,f)}_{M_1(d,e) \text{ in } \tilde{O}(N^{3/2}),\ \text{WCOJ}} \\
&= \sum_{b,c,d,e} T(b,c,d,e)M_1(d,e) \underbrace{\sum_{a} \pi_{bc} T(b,c)R(a,b)S(a,c)}_{M_2(b,c) \text{ in } O(N^{3/2}),\ \text{WCOJ}} \\
&= \sum_{b,c,d,e} M_1(d,e)T(b,c,d,e)M_2(b,c)
\end{aligned}
$$

This results in the TD shown in Figure 1.4

We now formalize the notion of tree-decompositions and how they can be used to represent a query plan for SumProd queries.

Definition 1.5.1 (Tree-decomposition)**.** Consider a SumProd query in the form (1.1), defined via a hypergraph $\mathcal{H} = (V, \mathcal{E})$. A *tree decomposition* of the query is a pair (T, χ), where $T = (V(T), E(T))$ is a tree and $\chi : V(T) \to 2^V$ is a map from the nodes of T to subsets of V that satisfies the following properties:

- for all hyperedge $S \in \mathcal{E}$, there is a node $t \in V(T)$ such that $S \subseteq \chi(t)$;

- and, for any vertex $v \in V$, the set $\{t \mid v \in \chi(t)\}$ forms a connected sub-tree of T.

Each set $\chi(t)$ is called a *bag* of the tree-decomposition, and we will assume w.l.o.g. that the bags are distinct, i.e. $\chi(t) \neq \chi(t')$ when $t \neq t'$ are nodes in T.

The *fractional hypertree width* of a tree decomposition (T, χ), denoted by $\mathsf{fhtw}(T, \chi)$, is the maximum fractional edge covering number over all bags of the tree decomposition:

$$\mathsf{fhtw}(T, \chi) = \max_{t \in V(T)} \rho^*_{\mathcal{H}}(\chi(t)). \tag{1.8}$$

Given a SumProd query $\varphi(\boldsymbol{x}_F)$ as defined in (1.1) a *free-connex* tree decomposition for the query is a tree decomposition for φ with the additional property that there is a connected sub-tree T' of T for which $F = \bigcup_{t \in V(T')} \chi(t)$.

From the examples described above, it is not hard to prove the following result:

Theorem 1.5.2 (Khamis *et al.*, 2016)**.** Given a free-connex tree decomposition (T, χ) of a SumProd query $\varphi(\boldsymbol{x}_F)$; let N denote the maximum input factor size; then, we can compute the output factor $\varphi(\boldsymbol{x}_F)$ in time

$$\tilde{O}\left(N^{\mathsf{fhtw}(T,\chi)} + \|\varphi\|\right) \tag{1.9}$$

TODOs: define faq-width, state a theorem relating these 2 notions. explain how fractional hypertree width and other widths are related –Hung.

1.6 Further Readings

The SumProd problem (with a variety of different names) was studied extensively starting in the late 1980s (Shenoy and Shafer, 1988; Shafer and Shenoy, 1991; Lauritzen and Spiegelhalter, 1988). Variable elimination and equivalently message passing as a general technique for solving SumProd queries were also studied extensively in the probabilistic graphical model literature (Dechter, 1997; Kohlas and Shenoy, 2000; Aji and McEliece, 2000)

More detailed background knowledge on semirings can be found in (Gondran and Minoux, 2008). Graphical models and their applications are in Wainwright and Jordan, 2008; Koller and Friedman, 2009. (Gottlob *et al.*, 2016) provides a comprehensive survey of tree-decompositions and their use in database query processing.

(Khamis *et al.*, 2016) extended the SumProd problem to capture computations over mutiple semirings. The fractional hypertree width runtime for SumProd queries can also be achieved using top-down (memoized) dynamic programming (Olteanu and Závodný, 2015). This approach, along with a novel compressed data representation was proposed in (Olteanu and Závodný, 2015), which has many practical applications (Olteanu and Schleich, 2016; Olteanu, 2020).

The notion of worst-case optimal join algorithms was developed through a series of works (Grohe and Marx, 2014; Atserias *et al.*, 2008; Ngo *et al.*, 2012), with NPRR (Ngo *et al.*, 2012) and LeapFrog TrieJoin (Veldhuizen, 2014) as two representative algorithms. High-level summary presentations can be found in (Ngo, 2018; Ngo *et al.*, 2013). These concepts and algorithms were extended to more general knowledge about the input databases (Abo Khamis *et al.*, 2017).

The notion of free-connex tree-decomposition was developed in (Bagan *et al.*, 2007) to answer conjunctive queries. (Khamis *et al.*, 2016) generalized the SumProd problem to capture computations over mutiple semirings, and explained how variable elimination can be used to solve these queries. Naturally, free-connex tree-decompositions can be applied to SumProd queries as well. As explained in (Khamis *et al.*, 2016), the classic Yannakakis algorithm (Yannakakis, 1981) is a variable elimination algorithm.

References

Abo Khamis, M., H. Q. Ngo, and D. Suciu. (2017). "What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?" In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by E. Sallinger, J. V. den Bussche, and F. Geerts. ACM. 429–444. DOI: 10.1145/3034786.3056105. URL: https://doi.org/10.1145/3034786.3056105.

Aji, S. M. and R. J. McEliece. (2000). "The generalized distributive law". *IEEE Transactions on Information Theory*. 46(2): 325–343. DOI: 10.1109/18.825794. URL: http://dx.doi.org/10.1109/18.825794.

Atserias, A., M. Grohe, and D. Marx. (2008). "Size Bounds and Query Plans for Relational Joins". In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society. 739–748. DOI: 10.1109/FOCS.2008.43. URL: https://doi.org/10.1109/FOCS.2008.43.

Bagan, G., A. Durand, and E. Grandjean. (2007). "On Acyclic Conjunctive Queries and Constant Delay Enumeration". In: *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*. Ed. by J. Duparc and T. A. Henzinger. Vol. 4646. *Lecture Notes in Computer Science*. Springer. 208–222. DOI: 10.1007/978-3-540-74915-8_18. URL: https://doi.org/10.1007/978-3-540-74915-8%5C_18.

Chandra, A. K. and P. M. Merlin. (1977). "Optimal Implementation of Conjunctive Queries in Relational Data Bases". In: *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*. Ed. by J. E. Hopcroft, E. P. Friedman, and M. A. Harrison. ACM. 77–90. DOI: 10.1145/800105.803397. URL: https://doi.org/10.1145/800105.803397.

Cooper, G. F. (1990). "The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks". *Artif. Intell.* 42(2-3): 393–405. DOI: 10.1016/0004-3702(90)90060-D. URL: https://doi.org/10.1016/0004-3702(90)90060-D.

Dechter, R. (1997). "Bucket Elimination: a Unifying Framework for Processing Hard and Soft Constraints". *Constraints An Int. J.* 2(1): 51–55. DOI: 10.1023/A:1009796922698. URL: https://doi.org/10.1023/A:1009796922698.

Dechter, R. (1999). "Bucket Elimination: A Unifying Framework for Reasoning". *Artif. Intell.* 113(1-2): 41–85. DOI: 10.1016/S0004-3702(99)00059-4. URL: https://doi.org/10.1016/S0004-3702(99)00059-4.

Gondran, M. and M. Minoux. (2008). *Graphs, dioids and semirings*. Vol. 41. *Operations Research/Computer Science Interfaces Series*. Springer, New York. xx+383. ISBN: 978-0-387-75449-9.

Gottlob, G., G. Greco, N. Leone, and F. Scarcello. (2016). "Hypertree Decompositions: Questions and Answers". In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by T. Milo and W. Tan. ACM. 57–74. DOI: 10.1145/2902251.2902309. URL: https://doi.org/10.1145/2902251.2902309.

Grohe, M. and D. Marx. (2014). "Constraint Solving via Fractional Edge Covers". *ACM Trans. Algorithms*. 11(1): 4:1–4:20. DOI: 10.1145/2636918. URL: https://doi.org/10.1145/2636918.

Gunawardena, J. (1998). "An introduction to idempotency". In: *Idempotency (Bristol, 1994)*. Vol. 11. *Publ. Newton Inst.* Cambridge Univ. Press, Cambridge. 1–49. DOI: 10.1017/CBO9780511662508.003. URL: https://doi-org.gate.lib.buffalo.edu/10.1017/CBO9780511662508.003.

Khamis, M. A., H. Q. Ngo, and A. Rudra. (2016). "FAQ: Questions Asked Frequently". In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by T. Milo and W.-C. Tan. ACM. 13–28. DOI: 10.1145/2902251.2902280. URL: https://doi.org/10.1145/2902251.2902280.

Kohlas, J. and P. P. Shenoy. (2000). "Computation in valuation algebras". In: *Algorithms for uncertainty and defeasible reasoning*. Vol. 5. *Handb. Defeasible Reason. Uncertain. Manag. Syst.* Kluwer Acad. Publ., Dordrecht. 5–39. ISBN: 0-7923-6672-7.

Kohlas, J. and N. Wilson. (2008). "Semiring induced valuation algebras: Exact and approximate local computation algorithms". *Artif. Intell.* 172(11): 1360–1399. DOI: 10.1016/j.artint.2008.03.003. URL: https://doi.org/10.1016/j.artint.2008.03.003.

Koller, D. and N. Friedman. (2009). *Probabilistic graphical models. Adaptive Computation and Machine Learning*. MIT Press, Cambridge, MA. xxxvi+1231. ISBN: 978-0-262-01319-2.

Lauritzen, S. L. and D. J. Spiegelhalter. (1988). "Local computations with probabilities on graphical structures and their application to expert systems". *Journal of the Royal Statistical Society: Series B (Methodological)*. 50(2): 157–194.

Lehmann, D. J. (1977). "Algebraic Structures for Transitive Closure". *Theor. Comput. Sci.* 4(1): 59–76. DOI: 10.1016/0304-3975(77)90056-1. URL: https://doi.org/10.1016/0304-3975(77)90056-1.

Loomis, L. H. and H. Whitney. (1949). "An inequality related to the isoperimetric inequality". *Bull. Amer. Math. Soc.* 55: 961–962. ISSN: 0002-9904.

Ngo, H. Q. (2018). "Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems". In: *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*. Ed. by J. V. den Bussche and M. Arenas. ACM. 111–124. DOI: 10.1145/3196959.3196990. URL: https://doi.org/10.1145/3196959.3196990.

Ngo, H. Q., E. Porat, C. Ré, and A. Rudra. (2012). "Worst-case optimal join algorithms: [extended abstract]". In: *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*. Ed. by M. Benedikt, M. Krötzsch, and M. Lenzerini. ACM. 37–48. DOI: 10.1145/2213556.2213565. URL: https://doi.org/10.1145/2213556.2213565.

Ngo, H. Q., C. Ré, and A. Rudra. (2013). "Skew strikes back: new developments in the theory of join algorithms". *SIGMOD Rec.* 42(4): 5–16. DOI: 10.1145/2590989.2590991. URL: https://doi.org/10.1145/2590989.2590991.

Nguyen, D. T., M. Aref, M. Bravenboer, G. Kollias, H. Q. Ngo, C. Ré, and A. Rudra. (2015). "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*. Ed. by J. L. Larriba-Pey and T. L. Willke. ACM. 2:1–2:8. DOI: 10.1145/2764947.2764948. URL: https://doi.org/10.1145/2764947.2764948.

Olteanu, D. (2020). "The Relational Data Borg is Learning". *Proc. VLDB Endow.* 13(12): 3502–3515. DOI: 10.14778/3415478.3415572. URL: http://www.vldb.org/pvldb/vol13/p3502-olteanu.pdf.

Olteanu, D. and M. Schleich. (2016). "Factorized Databases". *SIGMOD Rec.* 45(2): 5–16. DOI: 10.1145/3003665.3003667. URL: https://doi.org/10.1145/3003665.3003667.

Olteanu, D. and J. Závodný. (2015). "Size Bounds for Factorised Representations of Query Results". *ACM Trans. Database Syst.* 40(1): 2:1–2:44. DOI: 10.1145/2656335. URL: https://doi.org/10.1145/2656335.

Pearl, J. (1982). “Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach”. In: *Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, USA, August 18-20, 1982*. Ed. by D. L. Waltz. AAAI Press. 133–136. URL: http://www.aaai.org/Library/AAAI/1982/aaai82-032.php.

Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference. Morgan Kaufmann series in representation and reasoning*. Morgan Kaufmann.

Ramakrishnan, R. and J. Gehrke. (2003). *Database management systems (3. ed.)* McGraw-Hill. ISBN: 978-0-07-115110-8.

Shafer, G. R. and P. P. Shenoy. (1991). “Local computation in hypertrees”. URL: https://kuscholarworks.ku.edu/handle/1808/143.

Shafer, G. R. and P. P. Shenoy. (1990). “Probability propagation”. In: vol. 2. No. 1-4. 327–351. DOI: 10.1007/BF01531015. URL: https://doi.org/10.1007/BF01531015.

Shenoy, P. P. and G. Shafer. (1988). “Axioms for probability and belief-function proagation”. In: *UAI ’88: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence, Minneapolis, MN, USA, July 10-12, 1988*. Ed. by R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer. North-Holland. 169–198.

Veldhuizen, T. L. (2014). “Triejoin: A Simple, Worst-Case Optimal Join Algorithm”. In: *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*. Ed. by N. Schweikardt, V. Christophides, and V. Leroy. OpenProceedings.org. 96–106. DOI: 10.5441/002/icdt.2014.13. URL: https://doi.org/10.5441/002/icdt.2014.13.

Wainwright, M. J. and M. I. Jordan. (2008). “Graphical Models, Exponential Families, and Variational Inference”. *Found. Trends Mach. Learn.* 1(1-2): 1–305. DOI: 10.1561/2200000001. URL: https://doi.org/10.1561/2200000001.

Yannakakis, M. (1981). “Algorithms for Acyclic Database Schemes”. In: *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*. IEEE Computer Society. 82–94.

Zhang, N. L. and D. Poole. (1994). "A simple approach to Bayesian network computations". In: *Proc. of the Tenth Canadian Conference on Artificial Intelligence*.