Optimizing MPF Queries: Decision Support and Probabilistic Inference

Hector Corrada Bravo
Raghu Ramakrishnan

Technical Report #1567

August 2006

Optimizing MPF Queries: Decision Support and Probabilistic Inference

Héctor Corrada Bravo[†] Raghu Ramakrishnan[†‡]
[†] University of Wisconsin-Madison [‡] Yahoo! Research
{hcorrada, raghu}@cs.wisc.edu

Abstract

We identify a broad class of aggregate queries, called MPF queries, inspired by the literature on marginalizing product functions. MPF queries operate on "functional relations," where a measure attribute is functionally determined by the other relation attributes. An MPF query is an aggregate query over a stylized join of several functional relations. In the motivating literature on probabilistic inference, this join corresponds to taking the product of several probability distributions, and the grouping step corresponds to marginalization. Thus, MPF queries represent probabilistic inference in a relational setting.

While they play a central role in probabilistic inference, and our work complements recent work that provides a framework for probabilistic inference in a database setting, we present MPF queries in a general form where arbitrary functions other than probability distributions are handled. We demonstrate the value of MPF queries for decision support applications through a number of illustrative examples. We exploit the relationship to probabilistic inference in query evaluation by combining database optimization techniques for aggregate queries with traditional algorithms from the probabilistic inference literature, such as Variable Elimination and Belief Propagation. We consider how to optimize individual queries, combining features from Variable Elimination and Chaudhuri and Shim's algorithm for optimizing Group By queries. We also present an algorithm to find a cache of materialized views in order to efficiently evaluate a workload of MPF queries, combining Belief Propagation, Junction Trees, and database-style Group By optimizations. These results are especially interesting and timely because of the growing interest in managing data with uncertainty using probabilistic frameworks.

1. Introduction

We consider a class of queries that generalize probabilistic inference on some graphical models [2]. Functions over discrete domains are naturally represented as relations where an attribute (the value of the function) is determined by the remaining attributes (the inputs to the function) via a Functional Dependency (FD). We define such relations, called Functional Relations, and present an extended Relational Algebra that operates on them. A functional view V is defined by the join of a set S of smaller, 'local' functional relations, and specifies a joint function over the union of domains of functions in S. MPF (Marginalize a Product Function) queries are a type of aggregate query that compute V's value in arbitrary subregions of its domain:

```
select Vars, Agg(V[f]) from V group by Vars.
```

There are two options for evaluating an MPF query: 1) the relation defined by V is materialized, and maintained as base relations are updated; or, 2) each query is rewritten using V's definition and evaluated, so that constructing the relation defined by V is an intermediate step. The problem of view maintenance is avoided, but this approach is prohibitive if computing V's relation is too expensive. The latter option is likely to be appropriate for answering individual queries, and variations of the former might be appropriate if we have knowledge of the anticipated query workload. In this paper, we study the second approach.

A simple extension of the algorithm of Chaudhuri and Shim [4, 5] for optimizing aggregate queries yields significant gains over how MPF queries must be evaluated in existing systems (see Section 7). We also get similar gains from using the Variable Elimination technique [26] from the literature on optimizing probabilistic inference. Additionally, we present extensions to VE based on ideas in the Chaudhuri and Shim algorithm that yield even better plans than traditional VE. Finally, we present an algorithm for defining a collection of materialized views to efficiently support a workload of MPF queries. This algorithm builds on Belief Propagation [17] and Junction Trees [14, 1, 6], which are techniques for optimizing a specific type of probabilistic inference workload.

The contributions of this paper are as follows:

1. We introduce MPF queries which significantly generalize the relational framework introduced by

Wong [21] for probabilistic models. The generalized class of queries is motivated by decision support applications, and also allows us to represent probabilistic inference in a relational setting.

2. We extend the optimization algorithm of Chaudhuri and Shim for aggregate queries to the MPF setting, taking advantage of the semantics of functional relations and the extended algebra over these relations. This extension produces better quality plans for MPF queries than those given by the procedure in [4, 5],

3. We build on the connection to probabilistic inference and extend existing inference techniques to develop novel optimization techniques for MPF queries. Even for the restricted class of MPF queries that correspond to probabilistic inference, to the best of our knowledge this is the first approach that addresses scalability and cost-based plan selection.

4. We implement our optimization techniques in a modified Postgres system, and present a thorough evaluation that demonstrates the significant gains they yield.

The paper is organized as follows: in the next section we formally define the MPF query setting; in Sections 3 and 4 we introduce two application settings for MPF queries, decision support and probabilistic inference; Sections 5 and 6 describe and analyze optimization schemes for single MPF queries and workloads of MPF queries respectively; experimental results for single query optimization are shown in Section 7.

2. MPF Query Definition

We now formalize the MPF query setting. First, we define functional relations:

Definition 1. *A relation s with schema $\{A_1, A_2, \ldots, A_m, f\}$ is a **functional relation** (FR) if $f \in \mathbb{R}$, and the FD $A_1 A_2 \cdots A_m \to f$ holds. The attribute f is referred to as the **measure** attribute of s.*

We make several observations about FRs. First, any dependency of the form $A_i \to f$ can be extended to the maximal FD in Definition 1 and is thus sufficient to define an FR. Second, we do not assume relations contain the entire cross product of domains of $A1, \ldots, A_m$, although this is required in principle for probability functions. We refer to such relations as *complete*. Third, $f \in \mathbb{R}$ is not necessary. We will see that for the purposes of the optimizations we present, a number of domains are allowable. Finally, any relation can be considered an FR where f is implicit and assumed to take the value 1 (the multiplicative identity element of f's domain).

Functional relations may be joined to create functions with larger domains:

Definition 2. *Let s_1 and s_2 be functional relations, the **product join** of s_1 and s_2 is defined as:*

$$s_1 \overset{*}{\bowtie} s_2 = \pi_{\text{Var}(s_1) \cup \text{Var}(s_2), s_1[f] * s_2[f]}(s_1 \bowtie s_2),$$

where $\text{Var}(s)$ *is the set of non-measure attributes of* s.

This definition is clearer when expressed in SQL:

```
select A1,...,Am,(s1[f] * s2[f]) as f
from s1,s2
where s1[A1]= s2[A1],..., s1[Ak]= s2[Ak]
```

where $\{A1, \ldots, Am\} = \text{Var}(s_1) \cup \text{Var}(s_2)$, and $\{A1, \ldots, Ak\} = \text{Var}(s_1) \cap \text{Var}(s_2)$.

Some observations: Implicit in the Relational Algebra expression for product join is the fact that each table defines a unique measure; measure fields are never included in the set of join conditions; and, the product join of two FRs is itself an FR. Although we have defined this operation in terms of products over $\mathbb{R}$, it may be defined over the multiplicative operation of allowable domains (see below).

We are now in position to define the MPF problem.

Definition 3. ***The MPF Problem.*** *Given view definition r over base functional relations s_i, $i = 1, 2, \ldots, n$ such that $r = s_1 \overset{*}{\bowtie} s_2 \overset{*}{\bowtie} \cdots \overset{*}{\bowtie} s_n$, compute*

$$\pi_{X, \text{AGG}(r[f])} \text{GroupBy}_X(r)$$

*where $X \subseteq \bigcup_{i=1}^{n} \text{Var}(s_i)$, and AGG is a suitable additive aggregate function. We refer to X as the **query variables**.*

Note that the result of an MPF query is an FR; thus MPF queries may be used as subqueries defining further MPF problems.

We propose the following SQL extension to define views in the MPF setting:

```
create mpfview r as
  (select vars, measure = (* s1.f,s2.f,...,sn.f)
  from s1, s2, ..., sn
  where joinquals)
```

where the last argument in the select clause lists the measure attributes of base relations and the multiplicative operation used in the product join.

We used two operations on measure attributes of FRs to define the MPF setting: the multiplicative operation in the product join, and the additive aggregate $\text{AGG}(\cdot)$ in the MPF problem. This setting can be equivalently defined on measures in arbitrary commutative semi-rings [1, 15]. A *semiring* is a set closed on additive and multiplicative operations; both operations are associative and commutative; the additive operation distributes with respect to the multiplicative operation; and, the set contains the identity elements of

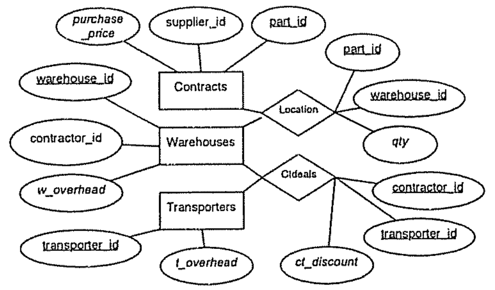


Figure 1. A supply chain decision support schema

both operations. Another pertinent allowable domain is the set $\{0, 1\}$ with $\wedge$ and $\vee$ as the multiplicative and additive operations. The core idea behind the optimization procedures we present in Sections 5 and 6 takes advantage of the distributivity of operations on semi-rings. In the next two sections, we motivate the study of MPF queries.

3. MPF Queries and Decision Support

Consider the following enterprise schema: 1) *Contracts:* stores terms for a part's purchase from a supplier; 2) *Warehouses:* each warehouse is operated by a contractor, and has an associated multiplicative factor determining the storage overhead for parts; 3) *Transporters:* transporters entail an overhead for transporting a part; 4) *Location:* the quantity of each part sent to a warehouse; 5) *Ctdeals:* contractors may have special contracts with transporters which reduce the cost of shipping to their warehouses using that transporter. Since contracts with suppliers, storage and shipping overheads and deals between contractors and transporters are not exclusively controlled by the company, it draws these pieces of information from diverse sources and combines them to make decisions about supply chains. Figure 1 shows an ER diagram of this schema, with measure attributes set in *italic* font.

Total investment on each supply chain may be computed by the view

```
create mpfview invest(pid,sid,wid,cid,tid,inv) as
  select pid, sid, wid, cid, tid,
      measure=(* p_price, w_overhead, t_overhead,
        qty, ct_discount)
  from contracts c, warehouses w, transporters t,
      location l,ctdeals ct
  where c.pid = l.pid and l.wid = w.wid ...
```

3.1. MPF Query Forms

Using this schema we present templates and examples for a number of MPF query variants that arise in a decision support context. In the following, we assume that r is as in Definition 3:

Basic: This is the query form used in the definition of the MPF problem above:

```
select X,AGG(r.f) from r group by X
```

Example: What is the minimum investment on each part?

```
select pid, min(inv) from invest group by pid
```

Restricted answer set: Here we are only interested in a subset of a query's answer as given by specific values of the query variables. We add a `where X=c` clause to the Basic query above. *Example*: How much would it cost for warehouse w1 to go off-line?

```
select wid, sum(inv) from invest where wid=w1
  group by wid
```

Constrained domain: Here we compute the function's value for the query variables conditioned on given values for other variables. We add a `where Y=c` clause to the Basic query for $Y \notin X$. *Example*: How much money would each contractor lose if transporter t1 went off-line?

```
select cid, sum(inv) from invest where tid=t1
  group by cid
```

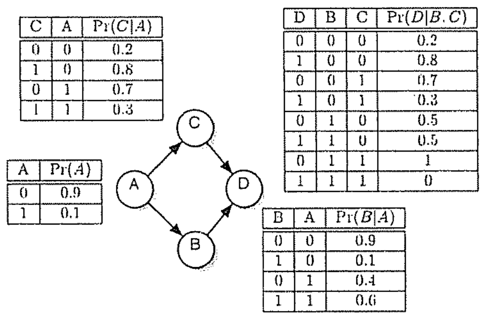
The optimization schemes we present in Sections 5 and 6 are for the three query types above. There are other useful types of MPF queries; optimizing them is a challenge for future work:

Constrained range: Here function values in the result are restricted. This is useful, for example, when only values that satisfy a given threshold are required. This is accomplished by adding a `having f<c` clause to the basic query.

The next two query types are of a hypothetical nature where alternate measure or domain values are considered. **Alternate measure:** here the measure value of a given base relation is hypothetically updated. For example, how much money would contractor c1 lose if warehouse w1 went off-line if, hypothetically, part p1 was a different price? **Alternate domain:** alternatively, variable values in base relations may be hypothetically updated. For example, how much money would contractor c1 lose if warehouse w1 went off-line under a hypothetical transfer of c1's contractor-transporter deal with t1 to t2?

4. MPF Queries and Probabilistic Inference

In this section, we show how MPF queries can be used to query Bayesian Network (BN) models of uncertain data. BNs [17, 14, 6] are widely-used probabilistic models of distributions that satisfy some conditional independence properties, allowing the distribution to be factored into local distributions over subsets of random variables.



C	A	$\Pr(C\mid A)$
0	0	0.2
1	0	0.8
0	1	0.7
1	1	0.3

A	$\Pr(A)$
0	0.9
1	0.1

D	B	C	$\Pr(D\mid B.C)$
0	0	0	0.2
1	0	0	0.8
0	0	1	0.7
1	0	1	0.3
0	1	0	0.5
1	1	0	0.5
0	1	1	1
1	1	1	0

B	A	$\Pr(B\mid A)$
0	0	0.9
1	0	0.1
0	1	0.4
1	1	0.6

Figure 2. A simple Bayesian Network

To understand the intuition behind BNs, consider a probabilistic model over the cross product of large discrete domains. A functional relation can represent this distribution but its size makes its use infeasible. However, if the function were factored, we could use the MPF setting to express the distribution using smaller local functional relations. For probability distributions, factorization is possible if some conditional independence properties hold; a BN represents such properties graphically. We now discuss how to efficiently compute properties of the BN distribution when the functional relations that define the local distributions are so large that they are disk-resident.

For binary random variables A, B, C, D a functional relation of size 2^4 can represent a joint probability distribution. If, however, a set of conditional independencies exists such that $\Pr(A, B, C, D) = \Pr(A)\Pr(B|A)\Pr(C|A)\Pr(D|B, C)$ then the BN in Figure 2 may be used instead. For this admittedly small example, the gains of factorization are not significant, but for a large number of large domains, factorization can yield a significant size reduction. The joint distribution is specified by the MPF view:

```
create mpfview joint as (
  select A,B,C,D, measure = (* tA.p, tB.p,
      tC.p, tD.p) as p
  from tA, tB, tC, tD
  where tA.A=tB.A and tA.A=tC.A ...  )
```

The set of conditional independence properties that induce a factorization may be given by domain knowledge, or estimated from data [12]. Given the factorization, the local function values themselves are estimated from data [12]. In either case, counts from data are required to derive these estimates. For data in multiple tables where a join dependency holds, the MPF setting can be used to compute the required counts.

After the estimation procedure computes the local functional relations we can use MPF queries to infer exact values of marginal distributions. An example inference task is given by the MPF query

```
select C,SUM(p) from joint where A=0 group by C
```

which computes the marginal probability distribution of variable C when $A = 0$ is observed, $\Pr(C|A = 0)$.

4.1. Discussion and Related Work

In a number of publications, Wong et al. [21, 22, 23] address the probabilistic inference task in relational terms and propose an extended relational model and algebra capable of expressing this problem. The MPF setting we present here is a generalization and reworking of their formulation. A major benefit of framing this task in a relational setting is that existing and new techniques for efficient query evaluation can then be used. This opportunity has not, to the best of our knowledge, been investigated; our study of query and workloadoptimization of MPF queries in Sections 5 and 6 is a first step in this direction.

Modeling and managing data with uncertainty has drawn considerable interest recently. A number of models have been proposed by the Statistics and Machine Learning [2, 10, 13, 20] and Database [7, 8, 3, 11] communities to define probability distributions over relational domains. For example, the DAPER formulation [13] extends Entity-Relationship models to define *classes* of conditional independence constraints and local distribution parameters. The general applicability of MPF queries to these approaches is an important issue for future investigation.

In particular, Dalvi and Suciu [7] and Ré et al. [18] define relational operators for probabilistic databases [11] and give a method for query answering. Part of their evaluation scheme produces queries which utilize a join operator that corresponds to our product join, and a projection operator that corresponds to our marginalization operator. They push as much of the evaluation as possible into the DB engine by posing these rewritten queries. Our optimization schemes are a first attempt at using cost-based query optimization in the DB engine to evaluate queries resulting from this rewriting.

Finally, we remark that in this paper, we consider the problem of scaling exact inference. This is required in settings where results are composed with other functions that are not monotonic with respect to likelihood. An example are systems where computation of expected risk or utility is performed. In these settings approximate values are not sufficient. However, for other systems where only relative likelihood suffices, e.g., ranking in information extraction, approximate inference procedures are sufficient and can be more efficient.

5. Single Query Optimization

The Generalized Distributive Law (GDL) has been proposed as a generic algorithm for efficiently solving MPF problems [1, 15]. The key property is the distributivity of

Table 1. Example cardinalities and domain sizes

Table	# tuples
contracts	100K
warehouses	5K
transporters	500
location	1M
ctdeals	500K

Variable	# ids
part_ids	100K
supplier_ids	10K
warehouse_ids	5K
contractor_ids	1K
transporter_ids	500

the addition and multiplication operations over measures. In relational terms, the GroupBy ('additive') operation distributes with the product join ('multiplicative') operation so that GroupBy's can be pushed into a query's join tree reducing the size of join operands.

We study two algorithms and their variants for implementing GDL: *(CS)* Chaudhuri and Shim's algorithm for optimizing aggregate queries [4, 5]; *(CS+)* a simple extension of CS that yields significant gains over the original; *(VE)* the Variable Elimination algorithm [26] proposed for probabilistic inference; and *(VE+)* an extension to VE based on the CS algorithm that finds better plans than VE.

These algorithms optimize basic, restricted answer and restricted domain MPF query types. We use Q1 as a running example:

```
Q1: select wid, SUM(inv) from invest group by wid;
```

and consider an instance with table cardinalities and variable domain sizes given in Table 1.

Chaudhuri and Shim [4, 5] define an optimization scheme for aggregate queries that pushes GroupBy nodes into join trees. They consider the usual non-functional join and describe correctness conditions for these plan transformations. This space of linear plans is explored using an extension of the dynamic programming optimization algorithm of Selinger et al. [19].

Algorithm 1 illustrates the CS procedure. The dynamic programming algorithm is modified so that in line 2, joinplan() finds the best linear plan that joins new relation r_j to the optimal plan for relation set S_j, while in line 3 it finds the best linear plan that joins r_j to the optimal plan for relation set S_j modified to include a GroupBy node as its topmost node. Grouping in this new node is done on query variables and variables appearing in a join condition for any relation not yet joined into S_j, ensuring the semantic correctness of the plan transformation. The cheapest of these two candidates is selected in line 4. The authors showed that this greedy-conservative heuristic produces a plan that is no worse in terms of IO cost than the original single GroupBy node plan.

As defined, the CS procedure cannot evaluate MPF queries efficiently. It does not consider the distributivity of GroupBy and functional join nodes since it assumes that aggregates are computed on a single column; not on the result of a function of many columns. The resulting evaluation plan would be the plan in Figure 3, which is the best plan without any GDL optimization.

Algorithm 1 The CS optimization algorithm

```
1: for all $r_j$, $S_j$ such that $Q' = S_j \cup \{r_j\}$ do
2:     $q_{1j} = \mathrm{joinplan}(\mathrm{optPlan}(S_j), r_j)$
3:     $q_{2j} = \mathrm{joinplan}(\mathrm{GroupBy}(\mathrm{optPlan}(S_j)), r_j)$
4:     $p_j = \mathrm{minCost}_i(q_{ij})$
5: end for
6: $\mathrm{optPlan}(Q') = \mathrm{minCost}_j(p_j)$
```

The CS+ algorithm is an extension of CS where joins are annotated as product joins, the distributive property of the aggregate and product join is verified, and the correctness condition of line 3 is retained. The CS+ algorithm implements the GDL restricted to left-linear plans. Figure 4 shows the CS+ plan for Q1. A GroupBy node is added after the join of *Location* and *Contracts* since the subplan joining *Warehouses* is cheaper. This simple extension can produce much better plans than unmodified CS.

The Variable Elimination algorithm [26] is based on a purely functional interpretation of MPF queries; our paper is the first to apply VE to relational query optimization. The domain of the function defined by the MPF view is reduced one variable at a time until only the query variables remain. While this is an entirely different approach to query optimization, not based on transformations between equivalent Relational Algebra expressions, we can cast it in relational terms: to eliminate a variable, all the tables that include it are product-joined, and the result is aggregated and grouped by the variables not eliminated yet. Algorithm 2 lists the VE algorithm. We abuse notation in this definition: in line 6, joinplan() returns the best plan joining the relations in the set given as argument, and in line 9, p denotes the relation resulting from the execution of plan p in line 6. The set rels(v_j, S) contains the relations in S where variable v_j appears. Figure 5 shows the VE plan for Q1 with elimination order *tid,pid,cid*.

Algorithm 2 The Variable Elimination Algorithm

```
1: Set $S = \{s_1, s_2, \ldots, s_n\}$
2: Set $V = \mathrm{Var}(r) \setminus X$
3: set $p$ = null
4: while $V \neq \emptyset$ do
5:     select $v_j \in V$ according to heuristic order
6:     set $p = \mathrm{GroupBy}(\mathrm{joinplan}(\mathrm{rels}(v_j, S)))$
7:     set $V = V \setminus \{v_j\}$
8:     remove relations containing $v_j$ from $S$
9:     set $S = S \cup \{p\}$
10: end while
```

The efficiency of VE for query evaluation is determined by the variable elimination order (see Section 5.5). In the GDL literature, the cost metric being minimized is the number of additions and multiplications used in evaluating the query. This is a valid cost metric in the original setting since operands are assumed to be memory-resident, and more significantly, single algorithms are assumed to implement each of the multiplication and marginalization operations. These are not valid assumptions in the relational case, where there

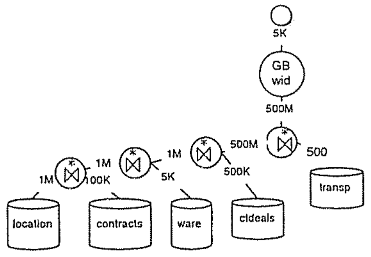


Figure 3. A CS plan for Q1

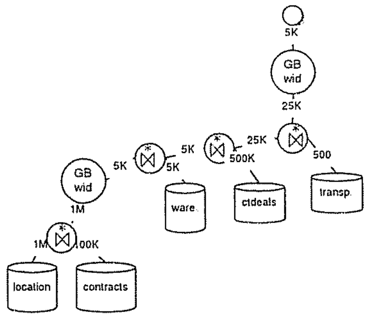


Figure 4. A CS+ plan for Q1

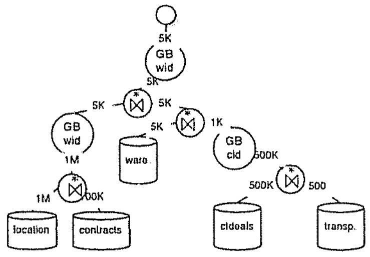


Figure 5. A VE plan for Q1

are multiple algorithms to implement join (multiplication) and aggregation (summation), and the choice of algorithm is based on the cost of accessing disk-resident operands.

5.1. Plan Linearity

Including nonlinear plans in the space searched by an optimization algorithm for MPF queries is essential since there are join operand reductions available to these plans that are not available to linear plans. When query variables are of small domain, but appear in large tables, this is a significant advantage. The example plan in Figure 4 illustrates this point. Also note that the elimination order in Figure 5 induces a non-linear join order. In fact, an advantage of VE is that it produces nonlinear plans with, in many cases, small optimization time overhead.

For an MPF query on variable X we can, conservatively, determine if a linear plan can efficiently evaluate it. We can check this using an expression that depends on the domain size of X, $\sigma_X = |X|$, and the size of the smallest base relation containing X, $\hat{\sigma}_X$. Both of these statistics are readily available in the catalog of RDBMs systems. To see the intuition behind this test, consider the following example: X occurs in only two base relations s_1 and s_2, where $|s_1| > |s_2|$, thus $\hat{\sigma}_X = |s_2|$. A linear plan must, at best, join s_2 to an intermediate relation s' of size σ_X resulting from a join or GroupBy node where s_1 is already included. On the other hand, a nonlinear plan is able to reduce s_2 to size σ_X before joining to s'. Under a simple cost model where joining R and S costs $|R||S|$ and computing an aggregate on R costs $|R|\log|R|$, a linear plan is admissible if the following inequality holds:

$$\sigma_X^2 + \hat{\sigma}_X \log \hat{\sigma}_X \geq \sigma_X \hat{\sigma}_X. \qquad (1)$$

We extend the CS+ procedure to consider nonlinear plans as follows: (a) extend the search strategy to a dynamic programming algorithm for nonlinear plans where for relation set S_j we consider joining every relation set of size $< j$; (b) instead of comparing two plans we now compare four: one without any GroupBy nodes (corresponding to line 2); another with a GroupBy on S_j (corresponding to line 3; another with a GroupBy on the operand (say, s') being joined to S_j; and finally, a plan with GroupBy nodes on both S_j and s'. The cheapest of these four plans is selected.

5.2. Plan Spaces

We now turn to a characterization of the plan spaces explored by nonlinear CS+ and VE.

Definition 4 (GDL Plan Space). *Denote as GDLPlan the space of all nonlinear evaluation plans where either GroupBy or join nodes are inner nodes, and are equivalent to a plan with only inner join nodes and a single GroupBy node at the root.*

CS+ performs a complete (but bounded) search of nonlinear join orders using dynamic programming with a local greedy heuristic that adds inner GroupBy nodes. This space, denoted *GDLPlan(CS+)* has the property that for any plan, if a *single* inner GroupBy node is removed, the cost of the subplan at its parent join node is greater. As before, CS+ yields a plan that is no worse than the plan with a single GroupBy at the root.

VE searches through a region of *GDLPlan* where a) all joins where a variable appears as a join condition are contiguous, and b) a GroupBy node immediately follows the last join on a variable. Denote this space as *GDLPlan(VE)*. No guarantee of optimality is given by VE due to its greedy heuristic search, and finding the variable ordering that yields the minimum cost plan is NP-complete in the number of view variables.

The following theorem characterizes these plan spaces:

Theorem 1. *[Inclusion Relationships] Using the notation above, we have:*

$$\textit{GDLPlan} \supset \textit{GDLPlan(CS+)} \supset \textit{GDLPlan(VE)}.$$

We need the following Lemma:

Lemma 1. *Consider relations $S_n = \{r_1, \ldots, r_n\}$ and variable v which only appears in r_k. Let $S'_n = \{r_1, \ldots, \mathrm{GroupBy}(r_k), \ldots, r_n\}$. The following holds:* $\mathrm{Cost}(\mathrm{optPlan}(S_n)) \leq \mathrm{Cost}(\mathrm{optPlan}(S'_n))$.

Proof. By induction on n. If $n = 2$ the Lemma follows since the plans are compared directly in line 4. Now assume true for $m \leq n - 1$. If $r_k = r_n$ then the Lemma follows since, again, the plans are compared directly in line 4. Otherwise, if $r_k \neq r_n$ then $r_k \in S_{n-1}$ we have by the inductive hypothesis $\mathrm{Cost}(\mathrm{optPlan}(S_{n-1})) \leq \mathrm{Cost}(\mathrm{optPlan}(S'_{n-1}))$ and the Lemma follows. □

Proof. (Theorem 1) We say that a plan is in a space $GDLPlan(\cdot)$ if the optimization algorithm either computes its cost, or can guarantee that it is more expensive than a plan for which it has computed cost.

- ($GDLPlan(CS+) \subseteq GDLPlan$) This follows by definition of CS+ and the semantic correctness of its plan transformation.
- ($GDLPlan(CS+) \neq GDLPlan$) By the greedy heuristic, any plan p' extending the plan not chosen in line 4 is not included in $GDLPlan(CS+)$. However, no guarantee is given that p' is more expensive than the plans extending the least expensive plan of line 4.
- ($GDLPlan(VE) \subseteq GDLPlan(CS+)$) Let p be the VE plan for elimination order $v_1, \ldots, v_n$. We prove this by induction on n. If $n = 1$, the theorem holds trivially. Now assume true for $m = n - 1$ and consider variables v_m and v_n and $S_m = \mathrm{rels}(v_m, S,)$. By the inductive hypothesis we have that the subplan in p that eliminates v_m has been considered by CS+. But since v_m only appears in the relation resulting from $\mathrm{optPlan}(S_m)$, by Lemma 1 we have that CS+ considers the subplan eliminating v_n as well. Thus $p \in GDLPlan(CS+)$.
- ($GDLPlan(VE) \neq GDLPlan(CS+)$) Consider a variable ordering where v_1 is preceded by v_2 but $\mathrm{rels}(v_1) \subseteq \mathrm{rels}(v_2)$. In this case VE does not consider adding Group By nodes in the subplan that eliminates v_2, but CS+ considers Group By nodes to 'eliminate' v_1 once $\mathrm{rels}(v_1)$ are joined.

□

It is easy to construct plans with cost lower than the minimum of the enclosed set at the non-overlapping regions of these spaces. There is no guarantee that the minimum cost plan for a query is contained in *GDLPlan(CS+)*; if it is, the dynamic programming procedure in CS+ will return it. Also, even if the minimum cost plan is contained in *GDLPlan(VE)*, there is no guarantee that VE will return it for an arbitrary ordering.

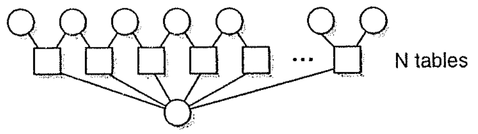


Figure 6. An example star MPF view.

5.3. Optimization Complexity

Another dimension of comparison between these two procedures is planning time. Since search for optimal subplans in VE only occurs in line 6, for views where variables exhibit low connectivity, that is, they appear only in a small subset of base relations, the cost of finding a VE plan is low.

As opposed to CS+, VE optimization time can be insensitive to variables that have high connectivity if average connectivity is low. Consider the star schema in Figure 6. This is the classic example where the optimization time of Selinger-type dynamic programming procedures degrades. In fact, the optimization time complexity for CS+ is $O(N2^N)$ for N relations. For VE with a proper ordering, only two relations have to be joined at a time for each variable, yielding optimization time complexity of $O(M)$ for M variables.

We summarize these findings by the following theorem:

Theorem 2 (Optimization Time Complexity). *Let S be the average variable connectivity, M be the number of variables, and N the number of tables. The worst-case optimization time complexity of VE with a proper heuristic computable in linear time is $O(MS2^S)$. The worst-case optimization time complexity of CS+ is $O(N2^N)$.*

Proof. The CS+ result is the standard complexity result for Salinger-type dynamic programming algorithms. For the VE result, a proper heuristic chooses a variable v_j in line 5 of Algorithm 2 where, on average, $|\mathrm{rels}(v_j)| = S$. Finding a plan for these tables in line 6 takes $O(S2^S)$. At worst, this is done M times, once for each variable. □

5.4. Extending the Variable Elimination Plan Space

We saw in Section 5.2 that the plan space considered by VE is a subset of the plan space considered by CS+. In this section we extend VE to close this gap. The first addition we make consists of delaying the elimination of variables if that results in better quality plans. There are two ways in which we implement this: by using Functional Dependencies on the base relations and by using cost-based local decisions similar to that used by CS+.

As defined, VE considers all variables as candidates for elimination; however, variables that satisfy the following property need not be considered since their elimination has no effect:

Proposition 1. *Let r be an MPF view over base relations $s_1, \ldots, s_n$, and $Y \in \mathrm{Var}(r)$. If for each $i, 1 \leq i \leq n$ an FD $X_i \rightarrow s_i[f]$ holds and $Y \notin X_i$ for $X_i \subseteq \mathrm{Var}(s_i)$, then* $\mathrm{GroupBy}_{\mathrm{Var}(r)\setminus Y}(r) = \pi_{\mathrm{Var}(r)\setminus Y}(r)$.

Proof. First, we note that for any functional relation s with $XY = \mathrm{Var}(()s)$ where the FD $X \rightarrow s[f]$ holds, then $\mathrm{GroupBy}_{X'}(s) = \pi_{X'}(s)$ for all $X' \supseteq X$ since the FD implies that there is only one row per value of X'. By the condition that FD's $X_i \rightarrow s_i[f]$ hold, we have that $\cup_i X_i \rightarrow r[f]$ holds. That means we may divide $\mathrm{Var}(r)$ into $\cup_i X_i$ and Z with $Y \in Z$. The Proposition follows by the first statement above. □

A sufficient condition for this Proposition is that for each base relation a primary key is given where Y is not part of any key. Furthermore, this Proposition holds for any set of relations, that is, at any point in the VE algorithm if a variable satisfies the Proposition for the current set of relations, that variable can be removed from the set of elimination candidates.

In the absence of FD information, we present an extension to Variable Elimination that uses cost-estimation to delay variable and push GroupBy nodes. Algorithm 2 requires two changes: 1) in line 6 we assume that the function joinplan() uses the local greedy conservative heuristic of CS+, and 2) we substitute line 6 with the line "set $p = \mathrm{joinplan}(\mathrm{rels}(v_j, S))$" to delay elimination. These additions have the effect of extending *GDLPlan(VE)* , so the following holds:

Theorem 3 (Extended Variable Elimination Space). *Denote by GDLPlan(VE+) the space of plans explored by Variable Elimination with the two cost-based extensions above, then*

GDLPlan(VE) ⊂ GDLPlan(VE+) ⊂ GDLPlan(CS+).

Proof.

- ($GDLPlan(VE) \subseteq GDLPlan(VE+)$) Given the same elimination order, the same proof for $CS+$ and VE shows this case.
- ($GDLPlan(VE) \neq GDLPlan(VE+)$) Consider an elimination order where v_i follows v_j but $\mathrm{rels}(v_i) \subset \mathrm{rels}(v_j)$, $VE+$ considers adding GroupBy nodes to eliminate v_i while creating the joinplan for $\mathrm{rels}(v_j)$, where VE does not. This is the same argument given above for VE and $CS+$.
- ($GDLPlan(VE+) \subseteq GDLPlan(CS+)$) The proof for this is the same as the proof of $GDLPlan(VE) \subseteq GDLPlan(CS+)$.
- ($GDLPlan(VE+) \neq GDLPlan(CS+)$) The issue here is that $VE+$ only considers plans where the joins for a given variable are contiguous, whereas $CS+$ does not follow that constraint. In the presence of indices and alternative access methods, contiguous joins are not necessarily optimal, therefore $CS+$ is able to produce plans that are not reachable to $VE+$.

□

Although there is still a gap between *GDLPlan(VE+)* and *GDLPlan(CS+)* corresponding to plans where joins for a variable are not necessarily contiguous, our experimental results in Section 7, show that $CS+$ rarely produces plans that are not reachable by $VE+$.

5.5. Elimination Heuristics

Finally, we turn to elimination heuristics. In the VE literature [9] there are two main heuristics: a) degree, which, in relational terms, estimates the size of post-elimination relation p in line 6, and b) width, which estimates the size of the pre-elimination relation $\mathrm{joinplan}(\mathrm{rels}(v_j, S))$ on the same line. In the literature, these estimates are given by the domain sizes of variables. For example, the degree heuristic computes the size of the cross-product of the domains of variables in p.

The degree heuristic greedily minimizes the size of join operands higher in the join tree. However, there are cases where executing the plan that yields these small operands is costly, whereas plans that use a different order are less expensive. In this case looking at estimates of the cost of eliminating a variable as an ordering heuristic is sensible. We call this ordering heuristic *Elimination Cost*.

A straightforward way of implementing the elimination cost heuristic is to call the query optimizer on the set of relations that need to be joined to estimate the cost of the plan required to eliminate a variable. However, for this heuristic to be computed efficiently, both average variable connectivity *and* maximum variable connectivity must be much lower than the number of tables, otherwise Variable Elimination would exhibit the same optimization time complexity as CS+.

While *width* and *elimination cost* estimate the cost of eliminating variables, the *degree* heuristic seeks to minimize the cost of future variable eliminations. There is a tradeoff between greedily minimizing the cost of the current elimination plan vs. minimizing the cost of subsequent elimination plans. To address this tradeoff we combine the *degree* and either *width* or *elimination cost* heuristics to derive elimination orders. We study the effect of these heuristics and their combinations in Section 7.

To summarize the contributions of this central section: 1) We presented a necessary condition for requiring nonlinear plans when answering a query; 2) We characterized the plan spaces explored by each of the algorithms given; 3) We analyzed the optimization time complexity of both algorithms, and gave conditions based on schema characteristics where one would be better than the other; 4) We extended VE so that its plan space is the space of CS+ plans without adding much optimization overhead; and 5) We proposed a cost-based ordering heuristic for Variable Elimination.

6. MPF Query Workload Optimization

MPF queries are stylized aggregate queries that follow a strict syntax. This means that workloads of MPF queries have a common structure that we want to exploit for efficient evaluation. In this section we describe an algorithm that decides on a set of materialized views for this purpose.

MPF Workload Problem Given an MPF view definition $r = s_1 \overset{*}{\bowtie} \cdots \overset{*}{\bowtie} s_n$ we define a workload as follows: (1) a set of queries $Q = \{q_1, \ldots, q_n\}$ where each q_i is a single variable basic or restricted answer MPF query, (2) each q_i is associated with a probability p_i that gives the likelihood of a user posing q_i. We want to build a set S of materialized views such that $C(S) + \mathbb{E}cost(Q(q,S))$ where $C(S)$ is the cost of materializing S, $cost(Q(q,S))$ is the cost of answering query q with respect to materialized views in S, and expectation is taken over the probability distribution specified by (2) above.

To ensure correctness of query answering with respect to S, we constrain S to satisfy the following invariant:

Definition 5. *A set of functional relations S satisfies the workload correctness invariant if for every functional relation $s \in S$ that includes X_i as a variable, computing an MPF query q on X_i using s yields the same result as evaluating q view r.*

To build S so that it satisfies the invariant of Definition 5, we extend the GDL all-vertex algorithm [1]. This algorithm first modifies the given view if it is not acyclic by using the Junction Tree algorithm. It then updates each of the resulting local functions using Belief Propagation (BP), a message passing algorithm that gathers in each local function information about the joint function. After the message passing algorithm is completed each local function now satisfies the invariant in Definition 5. See Appendix A for the BP and JT algorithms in the relational setting.

Our algorithm, VE-cache (Algorithm 3) creates a VE plan p from which S is induced. Here all variables are candidates for elimination, that is, no variable is treated as a query variable. While executing p it materializes and includes in S tables that precede a GroupBy node. At this point the relation resulting from the VE-cache plan has been reduced with respect to all base relations, but it is required that the tables in S be reduced as well. The correctness of VE-cache follows from the aciclicity of tables in S.

The following operations based on functional join are used in the algorithm.

Definition 6. *Let $U = \mathrm{Var}(t) \cap \mathrm{Var}(s)$, define the* ***product semijoin*** *of t and s as*

$$t \overset{*}{\ltimes} s = t \overset{*}{\bowtie} (\mathrm{GroupBy}_{U,\mathrm{SUM}(s[f])}(s)).$$

Also, define ***update semijoin*** *as*

$$t \ltimes s = t \overset{*}{\bowtie} (\mathrm{GroupBy}_{U,\mathrm{SUM}(t[f])}(t)) \overset{\div}{\bowtie} (\mathrm{GroupBy}_{U,\mathrm{SUM}(s[f])}(s)),$$

where $\overset{\div}{\bowtie}$ is defined exactly like product join, but uses the division operation instead of the product operation.

In short, these are extensions of the product join that use aggregation to reduce operators with respect to common variable subsets.

Algorithm 3 The VE-cache Optimization Scheme

```
Output:Set of cached relations that satisfy the correctness invariant
1: Create a no-query-variable Variable Elimination plan (Algorithm 2)
2: Cache all tables that precede a Group By node, say $t_1, \ldots, t_k$
3: for all $t_j, j = k, \ldots, 1$ do
4:     for all $t_i$, such that $j > i$ and GroupBy($t_i$) was used to create $t_j$ do
5:         compute $t_i \ltimes t_j$
6:     end for
7: end for
```

Given the VE plan of Figure 5, S contains three tables $t1(sid, pid, wid)$, $t2(wid, cid)$ and $t3(cid, tid)$, while the propagation required the operations $t2 \ltimes t3$ and $t1 \ltimes t2$. Since the materialized tables resulting from this algorithm satisfy the correctness invariant then evaluating Q1 on $t2$ gives the correct answer.

Theorem 4 (Correctness of VE-cache). *The set S of materialized tables in VE-cache satisfies the correctness invariant of Definition 5*

Proof of this theorem is given as Appendix A.

We can add restricted range queries to workloads and use the VE-cache scheme for optimization. The cached tables containing query variables have been reduced with respect to other tables where the full domain of the other tables are used. Thus, further reduction is required to absorb information about the function under the constrained domain.

The protocol to carry this out is the following: 1) apply the selection predicate to any VE-cache containing the constrained variable, 2) perform reductions along the schema between the cache table where the selection predicate was applied and every other cache table.

In our running example, if the following query were part of the workload:

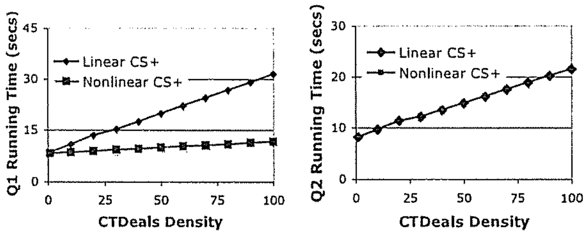


Figure 7. Plan Linearity Experiment

```
select wid, min(inv) from investment where tid=1
   group by wid
```

then, after applying the selection on *t3*, the reduction $t2 \ltimes t3$ is required.

Theorem 5. *After carrying out the given protocol, the new VECS-cache tables satisfy the correctness invariant of Definition 5.*

Proof. This protocol defines a BP semijoin program over an acyclic schema, so result follows from Theorem 10. □

7. Experimental Results

We now present experimental results illustrating the discussion in Section 5. We modified the PostgreSQL 8.1 optimizer and implemented each algorithm at the server (not middleware) level. The extensions in Section 2 were added to the PostgreSQL language with an extension that specifies the evaluation strategy. Experiments were performed on a 3 GHz Pentium IV Linux desktop with 2.4 GB of RAM and 38 GB of hard disk space. In most of these experiments, we do not compare the CS algorithm since its performance is substantially worse and distorts the scale of the plots, making it harder to see the relative performance of the other (much better) algorithms. However, the results in Section 7.4 make this comparison and illustrate the significant difference in performance.

7.1 Plan Linearity

Section 5.1 showed the benefit of nonlinear plans for MPF query evaluation. The experiment in Figure 7 illustrates how the plan linearity test heuristic is applied. On our example decision support schema we run two queries:

```
Q1:select cid, SUM(inv) from invest group by cid;
Q2:select tid, SUM(inv) from invest group by tid;
```

We plot evaluation time as the density of the *CTdeals* relation is increased. For Q1, we see that as density increases nonlinear plans execute faster, whereas for Q2, a linear plan is optimal for all densities. Since the nonlinear version of

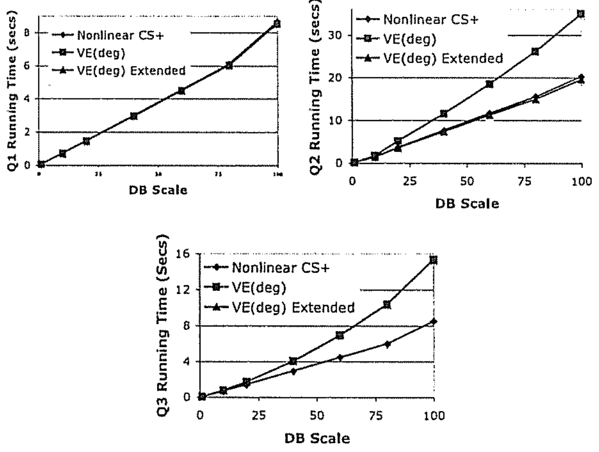


Figure 8. VE Extended Space Experiment

CS+ also considers linear plans, the Q2 running times for both plans coincide. For Q1, we have that $\sigma_{cid} = 1000$ and $\hat{\sigma}_{cid} = 5000$, so the inequality in Eq. 1 does not hold, whereas for Q2, we have $\sigma_{tid} = \hat{\sigma}_{tid} = 500$ which makes the inequality hold showing the applicability of the linearity condition.

7.2 Extended Variable Elimination Space

Section 5.4 showed how to extend the VE plan space to that of nonlinear CS+. Figure 8 compares the resulting plan quality for CS+ and VE with the degree heuristic with and without the space extension. We ran the following three queries as the total scale of the database is increased:

```
Q1:select cid, SUM(inv) from invest group by cid;
Q2:select sid, SUM(inv) from invest group by sid;
Q3:select wid, SUM(inv) from invest group by wid;
```

For Q1, the degree heuristic produced the optimal CS+ nonlinear plan without the VE extension. For Q2, the degree heuristic produced a suboptimal plan, but with the space extension we obtain the optimal plan. Q3 is a different case where we have that the degree heuristic is not able to find the optimal plan even with the extended space. While for the extended version of VE there exists a elimination order that includes the optimal plan from CS+, it is not the degree heuristic in this case. Also, the extension to VE guarantees that we find a plan no worse than the plan obtained by VE without extensions; this is reflected in the results shown here.

7.3 Elimination Heuristics

We now show experimental results on the effect of ordering heuristic on plan quality. Using our example schema we run two queries and plot their running time as a function of the scale of the DB:

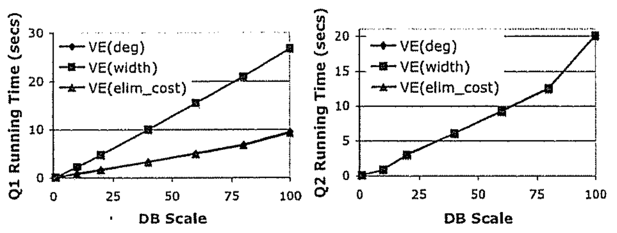


Figure 9. Ordering Heuristics Experiment

```
Q1:select cid, SUM(inv) from invest group by cid;
Q2:select pid, SUM(inv) from invest group by pid;
```

For Q1, the width heuristic which minimizes GroupBy operands yields a plan worse than both degree and elimination cost. Interestingly, width can be seen as an estimate of elimination cost, whereas degree seeks to minimize join operands, or, equivalently, minimize the cost of future variable eliminations. For Q2, all heuristics derived the same plan.

Table 2 summarizes another experiment on order heuristics. Three views were created similar to that in Figure 6: a) a star view exactly like Figure 6, b) a linear view where the variable connecting all tables is removed, and c) a 'multistar' schema where instead of a single common variable there are several common variables each connecting to three different tables. The number of tables $N = 5$, all variables had domain size 10 and all functional relations were complete. A query on the first variable in the linear section was run on each schema. For each of the *degree, width* and *elimination cost* heuristics described in Section 5.5 we ran both the original VE algorithm and its extended space version described in Section 5.4. We implement the elimination cost heuristic using an overestimate: we fix a linear join ordering and allow choice of access paths and operator algorithms. We also include results for combinations of the *degree and width* and *degree and elimination cost* heuristics[1]. We report the cost of the plan selected by the nonlinear CS+ algorithm, which is optimal in the plan space considered.

We see that for the star schema, the width heuristic performs best. This is not surprising since the degree heuristic selects the common variable: after joining all of its corresponding tables, all but the query variable can be eliminated and the resulting relation is small (10 tuples). This requires joining all base tables thus performing no GDL optimization. However, we see that by combining the degree and width heuristics we are able to produce a much better plan than degree but only slightly worse than width. The elimination cost heuristic performs better than the degree heuristic, but due to its overestimate, does not perform as well as the width heuristic. The difference in performance lessens as maximum variable connectivity drops.

[1]Combinations are implemented by normalizing each estimate (dividing by the largest among candidates) and multiplying the normalized values

Table 2. Ordering Heuristics Experiment Result

Ordering	star	multistar	linear
Nonlinear CS+	**429.62**	**363.02**	**21.23**
VE(deg)	240225.15	843.84	34.57
VE(deg) ext.	**429.62**	**363.02**	**21.23**
VE(width)	705.03	593.43	34.57
VE(width) ext.	**429.62**	**363.02**	**21.23**
VE(elim_cost)	1045.44	936.34	73.78
VE(elim_cost) ext.	**429.62**	**363.02**	**21.23**
VE(deg & width)	950.44	843.84	34.57
VE(deg & width) ext.	**429.62**	**363.02**	**21.23**
VE(deg & elim_cost)	240225.15	843.84	34.57
VE(deg & elim_cost) ext.	**429.62**	**363.02**	**21.23**

Table 3. Random Heuristic Experiment Result

Ordering	star	multistar	linear
VE(random)	30830.42 ± 1470.78	11730.35 ± 298.86	72.037 ± 0.29
VE(random) ext.	770.78 ± 5.60	4559.58 ± 149.03	51.783 ± 0.36

Interestingly, for all schemas, the extended VE algorithm with any heuristic produces optimal plans. This might indicate that the choice of elimination ordering becomes irrelevant when the extended version of VE is used. To study this phenomenon we implemented a heuristic that selects variables to eliminate at random. We ran the same query ten times using the random heuristic with and without the space extension. Table 3 reports the result. The cost displayed is the mean of the 10 runs and an estimated 95% confidence interval around the mean. We see that the minimum cost is not within the confidence interval in either case, which suggests that elimination ordering is still significant in the extended plan space version of VE.

7.4 Optimization Cost

The following experiment illustrates the trade-off between plan quality and optimization time of the algorithms. For each view in the previous experiment (with $N = 7$), we query all variables in the linear part. In Figure 10 we plot the average estimated cost of evaluating the query against the average time required to derive the execution plan. Points closer to the origin are best.

We first note significant gains provided by the algorithms proposed here compared to the CS algorithm. Next we note that nonlinear plans provide gains of around one order of magnitude compared to linear plans. The degree heuristic performs better when maximum variable connectivity is low, but still achieves quality plans when considering the extended space. The width and elimination cost heuristics are not affected by maximum variable connectivity indicating that their performance is controlled by average connectivity. Finally we note the lower optimization time, in gen-

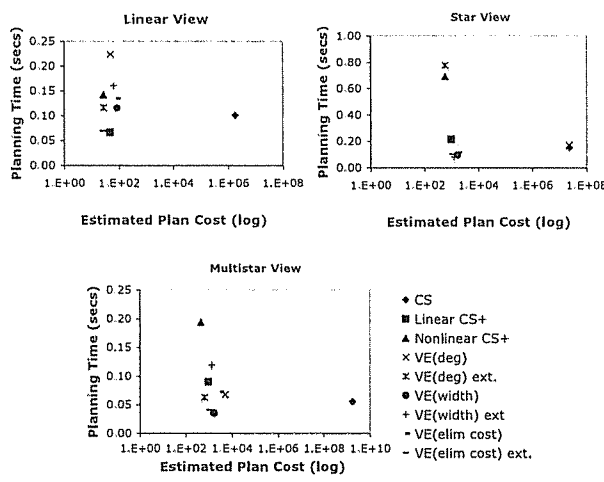


Figure 10. Optimization Time Tradeoff Experiment

eral, for VE compared to nonlinear CS+.

8. Conclusion and Future Work

We introduced the MPF class of queries, showed its value in a variety of settings, and presented optimization techniques to evaluate MPF queries and view materialization strategies for evaluating workloads of MPF queries.

Our work is an early step in synthesizing powerful ideas from database query evaluation and probabilistic inference. A number of models have recently been proposed for defining probability distributions over relational domains, e.g., Plate Models [2], PRMs [10], DAPER [13], and MLNs [20]. Applying MPF query optimization to support inference in such settings is a promising and valuable next step.

Theoretical properties of MPF queries, for example, the complexity of deciding containment, are intriguing. While general results for arbitrary aggregate queries exist, we think that the MPF setting specifies a constrained class of queries that might allow for interesting and useful results.

References

[1] S. Aji and R. McEliece. The generalized distributive law. *IEEE Trans. Info. Theory*, 46(2):325–343, March 2000.

[2] W. L. Buntine. Operations for learning with graphical models. *J. Artif. Intell. Res. (JAIR)*, 2:159–225, 1994.

[3] D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. In *VLDB*, pages 970–981, 2005.

[4] S. Chaudhuri and K. Shim. Including Group-By in Query Optimization. In *VLDB*, pages 354–366, 1994.

[5] S. Chaudhuri and K. Shim. Optimizing queries with aggregate views. In *Proc. 5th Int'nl. Conf. on Extending DB Technology*, pages 167–182. Springer-Verlag, 1996.

[6] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York, 1999.

[7] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[8] N. N. Dalvi and D. Suciu. Answering queries from statistics and probabilistic views. In *VLDB*, pages 805–816, 2005.

[9] Y. E. Fattah and R. Dechter. An evaluation of structural parameters for probabilistic reasoning: Results on benchmark circuits. In *UAI*, pages 244–251, 1996.

[10] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[11] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[12] D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1999.

[13] D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, prms and plate models. In *SRL2004*. ICML, August 2004.

[14] F. V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag, 2001.

[15] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Info. Theory*, 47(2):498–519, 2001.

[16] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[17] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[18] C. Ré, N. Dalvi, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

[19] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[20] P. Singla and P. Domingos. Discriminative training of markov logic networks. In *AAAI*, pages 868–873, 2005.

[21] S. K. M. Wong. The relational structure of belief networks. *J. Intell. Inf. Syst.*, 16(2):117–148, 2001.

[22] S. K. M. Wong, C. J. Butz, and Y. Xiang. A method for implementing a probabilistic model as a relational database. In *UAI*, pages 556–564, 1995.

[23] S. K. M. Wong, D. Wu, and C. J. Butz. Probabilistic reasoning in bayesian networks: A relational database approach. In *Canadian Conference on AI*, pages 583–590, 2003.

[24] D. Wu and S. K. M. Wong. Local propagation in bayesian networks versus semi-join program in databases. In *FLAIRS*, 2004.

[25] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, March 1981 1981.

[26] N. L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *JAIR*, 5:301–328, 1996.

A. Proof of Theorem 4

We now prove the correctness of the VE-cache algorithm by showing that it implements the GDL all-vertex algorithm. We first present the Belief Propagation algorithm to motivate the need for the acyclic schema the Junction Tree algorithm creates. Algorithm 4 is an adaptation of the Belief Propagation (BP) message passing algorithm to the relational setting.

BP selects an order of the relations in the schema according to some heuristic and reduces each functional relation in the order with respect to any table that precedes it with which it shares variables using the product semijoin operation ($\overset{*}{\ltimes}$) defined above. This step propagates values for variable subsets from one function to another if they have common variables, in a sense, propagating information about those variables to the latter function. Once this first pass is completed, the reverse reductions are done, so that function values are propagated in the reverse direction for all pairs of overlapping functions. This reverse reduction uses the update semijoin operation above so that values propagated in the first pass are not propagated again in the second pass.

Algorithm 4 The Belief Propagation Algorithm

```
1: Choose a table order $s_1, s_2, \ldots, s_n$
2: for all Table $s_i$ in order do
3:     for all Table $s_j$, such that $i < j$ and $s_i$ and $s_j$ share variables do
4:         compute $s_j \overset{*}{\ltimes} s_i$
5:     end for
6: end for
7: for all Table $s_j$ in reverse order do
8:     for all Table $s_i$, such that $j > i$ and $s_i$ and $s_j$ share variables do
9:         compute $s_i \ltimes s_j$
10:    end for
11: end for
```

Belief Propagation defines a semijoin program reduction on the set of base relations which, as opposed to the classical semijoin setting where projection is used, grouping and aggregation is used to 'project' tables. This connection between Belief Propagation and semijoin programs was made by Wu et al. [24].

Theorem 6. *[Pearl [17]] The updated base relations resulting from BP satisfy the invariant of Definition 5.*

Figure 11 shows the program resulting from BP with the order *Transporters (t), Ctdeals (ct), Warehouses (w), Location (l), Contracts (c)*. For illustration we expand the functional semijoins for the first and last steps of the program:

$$
\begin{aligned}
ct \overset{*}{\ltimes} t &= ct \overset{*}{\bowtie} (\mathrm{GroupBy}_{tid,\mathrm{SUM}(t.t\_overhead)}(t)) \\
t \ltimes ct &= t \overset{*}{\bowtie} (\mathrm{GroupBy}_{tid,\mathrm{SUM}(ct.ct\_discount)}(ct) \\
&\quad \overset{\div}{\bowtie} (\mathrm{GroupBy}_{tid,\mathrm{SUM}(t.t\_overhead)}(t)).
\end{aligned}
$$

1. $ct \overset{*}{\ltimes} t$	5. $l \ltimes c$
2. $w \overset{*}{\ltimes} ct$	6. $w \ltimes l$
3. $l \overset{*}{\ltimes} w$	7. $ct \ltimes w$
4. $c \overset{*}{\ltimes} l$	8. $t \ltimes ct$

Figure 11. A BP semijoin program

1. $st \overset{*}{\ltimes} t$	7. $l \ltimes c$
2. $ct \overset{*}{\ltimes} t$	8. $w \ltimes l$
3. $c \overset{*}{\ltimes} st$	9. $ct \ltimes w$
4. $w \overset{*}{\ltimes} ct$	10. $st \ltimes c$
5. $l \overset{*}{\ltimes} w$	11. $t \ltimes ct$
6. $c \overset{*}{\ltimes} l$	12. $t \ltimes st$

Figure 12. A BP semijoin program on a cyclic schema

The Belief Propagation algorithm is not correct for cyclic schemas. Consider an extension to our Decision Support schema that adds the table
Stdeals(supplier_id,transporter_id,st_discount)
which stores agreements between suppliers and transporters. Using the order *Transporters (t), Stdeals (st), Ctdeals (ct), Warehouses (w), Location (l), Contracts (c)* we get the program in Figure 12. In step 1, *st* is reduced with respect to *t*, and in step 3, *c* is reduced with respect to *st*, thus by step 3, *c* has been reduced with respect to *t*. However, in step 2, *ct* is reduced with respect to *t*, in steps 4,5 and 6 we have reductions from *ct* to *c* through *w* and *l*. Thus in step 6, *c* is reduced with respect to *t* again. Since each step involves the product of the measure attribute of the relations involved, the measure field of *c* has been incorrectly updated with the measure of *t* twice.

Acyclic schemas have the running intersection property:

Theorem 7. *(Maier [16]) Given schema $S = \{s_1, \ldots, s_n\}$ create undirected graph $G = (V, E)$ where $V = S$ and $(s_i, s_j) \in E$ if $\mathrm{Var}(s_i) \cap \mathrm{Var}(s_j) \neq \emptyset$, that is, the nodes of G are relations and an edge exists between two relations if they share variables. S is an acyclic schema if and only if there exists a tree T that spans G with the property that for vertices s_i, s_j, $\mathrm{Var}(s_i) \cap \mathrm{Var}(s_j)$ is contained in every relation in the path between s_i and s_j.*

The spanning tree with this property is also called a Junction Tree. Our original example schema has this property, while the schema with the addition of *Stdeals* does not.

Acyclic schemas have a further property:

Theorem 8. *(Jensen [14]) Given schema $S = \{s_1, \ldots, s_n\}$ create undirected graph $G = (V, E)$ where $V = \bigcup_i \mathrm{Var}(s_i)$ and $(v_i, v_j) \in E$ if there exists a relation s_k*

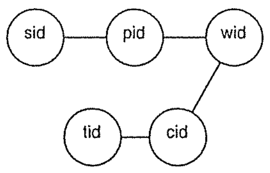


Figure 13. Variable graph for acyclic schema

such that $v_i, v_j \in \mathrm{Var}(s_k)$, that is, the nodes of G are the variables appearing in the schema and there is an edge between two variables if they co-occur in a relation. S is an acyclic schema if and only if G is chordal.

A chordal graph is one where every cycle of length greater than 3 has a chord, that is, an edge between two non-consecutive nodes in the cycle. Figure 13 has the variable graph for our original acyclic schema. The addition of *Stdeals* would add an edge between *sid* and *tid* which creates a cycle of length 5 that has no chord. We refer the reader to Cowell et al. [6] and Jensen [14] for a more extended discussion of chordal graphs and junction trees in the context of probabilistic inference, and to Wu [24] for further discussion on the links between Junction Trees, Belief Propagation and acyclic database schemas.

The Junction Tree algorithm creates an acyclic schema by transforming the variable graph of a cyclic schema into a chordal graph. The acyclic schema is then induced from this resulting chordal graph. Algorithm 5 lists the Junction Tree algorithm. Step 2 modifies the variable graph of the input schema to create a chordal graph using triangulization [2] which is listed as Algorithm 6. It adds edges to the graph by choosing a vertex, connecting any of its disconnected neighbors and then removing it from the graph. Figure 14 shows a chordal graph resulting from triangulization for our example cyclic schema using the vertex order *tid,sid* and added edges drawn dotted. Figure 15 shows the new schema and the Junction Tree resulting from that chordal graph. The final step of the algorithm populates the tables of the new schema by assigning relation s_i of the original schema to a relation s_j of the new schema such that $\mathrm{Var}(s_i) \subseteq \mathrm{Var}(s_j)$, and then computing the product join of tables assigned to each relation of the new schema.

The size of the resulting schema, and thus the complexity of Belief Propagation on the resulting schema, is determined by the size of the cliques in the new graph. This in turn is determined by the order in which vertices are chosen during triangulization. The size of the largest clique in the resulting graph is called the induced width of the new graph.

Theorem 9. *(Yannakakkis [25]) Finding the chordal graph with minimum induced width is NP-complete in the number*

[2]A chordal graph is also said to be triangulated

Algorithm 5 The Junction Tree Algorithm

```
1: Construct variable graph $G$ from schema $S$
2: Triangulate $G$ to create new graph $G'$
3: Create new schema $S'$ where each maximal clique in $G'$ is a relation
4: Assign relations from schema $S$ to relations in $S'$ that contain all of its variables
5: Create the new relation by product joining all $S$ tables assigned to each relation in $S'$
```

Algorithm 6 The Triangulization Procedure

```
Input: Graph $G = (V, E)$
Output: Chordal graph $G' = (V', E')$
1: Set $G' = (V', E')$ where $V' = V$ and $E' = E$
2: while $V \neq \emptyset$ do
3:     select vertex $v \in V$ from a non-chordal cycle
4:     for every pair $(v, u_1)$ and $(v, u_2) \in E$, add $(u_1, u_2)$ to $E$ and $E'$
5:     remove $v$ from $V$
6: end while
```

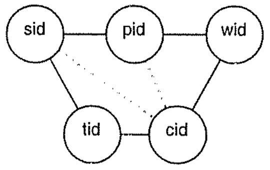


Figure 14. A chordal graph for the cyclic schema

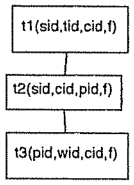


Figure 15. The resulting Junction Tree

of variables.

The equivalence between the Triangulization and the Variable Elimination algorithms is clear. Choosing a vertex and connecting any unconnected neighbors in triangulization is equivalent to selecting a variable v and joining the tables where it appears in Variable Elimination. The clique resulting from the added edges will be a relation in the new schema, caching the result of this join in Variable Elimination creates the relation in the new schema. Removing the vertex from the graph in triangulization yields a clique of its neighbors equivalent to the relation resulting from marginalizing, or, eliminating the chosen variable.

Theorem 10. *Denote the set of cached tables in VE-cache as $T = \{t_i : i = 1, \dots, k\}$. Then the following hold:*

1. *T is the schema result of triangulating using the variable order given by the VE plan of line 1,*
2. *T is an acyclic schema, and*
3. *VE-cache performs a BP semijoin program over T*

Proof. (1) follows from the equivalence of triangulation and variable elimination and the fact that the relations that precede Group Bys give the relations from triangulation. (2) follows from (1) since triangulation results in an acyclic schema. For (3) we first note that VE-cache implements directly the backward pass of lines 7 through 10, and that by the definition of $\overset{*}{\ltimes}$ we have that VE-cache also performs the forward pass when it executes the given VE plan. □

Proof. (Theorem 4). Follows directly from Theorems 10 and 6. □