Optimizing MPF Queries: Decision Support and Probabilistic Inference

Héctor Corrada Bravo[†]
[†] University of Wisconsin-Madison
Madison, WI USA
hcorrada@cs.wisc.edu

Raghu Ramakrishnan[†‡]
[‡] Yahoo! Research
Santa Clara, CA USA
raghu@cs.wisc.edu

ABSTRACT

Managing uncertain data using probabilistic frameworks has attracted much interest lately in the database literature, and a central computational challenge is probabilistic inference. This paper presents a broad class of aggregate queries, called MPF queries, inspired by the literature on probabilistic inference in statistics and machine learning. An MPF (Marginalize a Product Function) query is an aggregate query over a stylized join of several relations. In probabilistic inference, this join corresponds to taking the product of several probability distributions, while the aggregate operation corresponds to marginalization. Probabilistic inference can be expressed directly as MPF queries in a relational setting, and therefore, by optimizing evaluation of MPF queries, we provide scalable support for probabilistic inference in database systems. To optimize MPF queries, we build on ideas from database query optimization as well as traditional algorithms such as Variable Elimination and Belief Propagation from the probabilistic inference literature.

Although our main motivation for introducing MPF queries is to support easy expression and efficient evaluation of probabilistic inference in a DBMS, we observe that this class of queries is very useful for a range of decision support tasks. We present and optimize MPF queries in a general form where arbitrary functions (i.e., other than probability distributions) are handled, and demonstrate their value for decision support applications through a number of illustrative and natural examples.

Categories and Subject Descriptors:
H.2.8 [Database Management]: Database Applications
H.2.4 [Database Management]: Systems – *Query Processing*

General Terms: Algorithms, Management, Performance.

Keywords: Probabilistic Inference, Aggregate Queries.

1. INTRODUCTION

Recent proposals for managing uncertain information require the evaluation of probability measures defined over a large number of discrete random variables. This paper presents MPF queries, a broad class of aggregate queries capable of expressing this probabilistic inference task. By optimizing query evaluation in the MPF (Marginalize a Product Function) setting we provide direct support for scalable probabilistic inference in database systems. Further, looking beyond probabilistic inference, we define MPF queries in a general form that is useful for Decision Support, and demonstrate this aspect through several illustrative queries.

The MPF setting is based on the observation that functions over discrete domains are naturally represented as relations where an attribute (the value, or measure, of the function) is determined by the remaining attributes (the inputs, or dimensions, to the function) via a Functional Dependency (FD). We define these *Functional Relations*, and present an extended Relational Algebra to operate on them. A view V can then be created in terms of a stylized join of a set of 'local' functional relations such that V defines a joint function over the union of the domains of the 'local' functions. MPF queries are a type of aggregate query that computes view V's joint function value in arbitrary subsets of its domain:

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

We optimize the evaluation of MPF queries by extending existing database optimization techniques for aggregate queries to the MPF setting. In particular, we show how a modification to the algorithm of Chaudhuri and Shim [4, 5] for optimizing aggregate queries yields significant gains over evaluation of MPF queries in current systems. We also extend existing probabilistic inference techniques such as Variable Elimination to develop novel optimization techniques for MPF queries. To the best of our knowledge, this paper presents the first approaches to probabilistic inference that provide scalability and cost-based query evaluation. Section 5 gives an empirical evaluation of these optimization techniques in a modified Postgres system.

In the rest of this introduction, we outline the probabilistic inference problem and explain the connection to MPF query evaluation, describe our approach to optimizing such queries, and illustrate the value of MPF queries for decision support.

1.1 Probabilistic Inference as Query Evaluation

Consider a joint probability distribution P over discrete random variables A, B, C and D (see Section 3 for an example). The probabilistic inference problem is to compute values of the joint distribution, say $P(A = a, B = b, C, D)$,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'07, June 11–14, 2007, Beijing, China.
Copyright 2007 ACM 978-1-59593-686-8/07/0006 ...$5.00.

or values from conditional distributions, $P(A|B = b, C = c, D = d)$ for example, or values from marginal distributions, for example $P(A, B)$. All of these computations are derived from the joint distribution $P(A, B, C, D)$. For example, computing the marginal distribution $P(A, B)$ requires summing out variables C and D from the joint.

Since our variables are discrete we can use a relation to store the joint distribution with a tuple for each combination of values of A, B, C and D. The summing out operation required to compute marginal $P(A, B)$ can then be done using an aggregate query on this relation. However, the size of the joint relation is exponential in the number of variables, making the probabilistic inference problem potentially expensive.

If the distribution was "factored" (see Section 3 for specifics) the exponential size requirement could be alleviated by using multiple smaller relations. Existing work addresses how to derive suitable factorizations [12], but that is not the focus of this paper; we concentrate on the inference task.

Given factorized storage of the probability distribution, probabilistic inference still requires, in principle, computing the complete joint before computing marginal distributions, where reconstruction is done by multiplying distributions together. In relational terms, inference requires reconstructing the full joint relation using joins and then computing an aggregate query. This paper addresses how to circumvent this requirement by casting probabilistic inference in the MPF setting, that is, as aggregate query evaluation over views. We will see conditions under which queries can be answered without complete reconstruction of the joint relation, thus making probabilistic inference more efficient. By optimizing query evaluation in a relational setting capable of expressing probabilistic inference, we provide direct scalable support to large-scale probabilistic systems. For a more complete discussion of Bayesian Networks and inference using MPF queries, see Section 3.2.

1.2 Optimization of MPF Queries

Like usual aggregate queries over views, there are two options for evaluating an MPF query: 1) the relation defined by view V is materialized, and queries are evaluated directly on the materialized view; or, 2) each query is rewritten using V's definition and then evaluated, so that constructing the relation defined by V is an intermediate step. The first approach requires that the materialized view is updated as base relations change. In the latter, the problem of view maintenance is avoided, but this approach is prohibitive if computing V's relation is too expensive. The rewriting 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 query rewrite approach.

Chaudhuri and Shim [4, 5] define an algorithm for optimizing aggregate query evaluation based on pushing aggregate operations inside join trees. We present and evaluate an extension of their algorithm and show that it yields significant gains over evaluation of MPF queries in existing systems (see Section 5). We also present and evaluate the Variable Elimination (VE) technique [27] from the literature on optimizing probabilistic inference and show similar gains over existing systems. Additionally, we present extensions to VE based on ideas in the Chaudhuri and Shim algorithm that yield better plans than traditional VE.

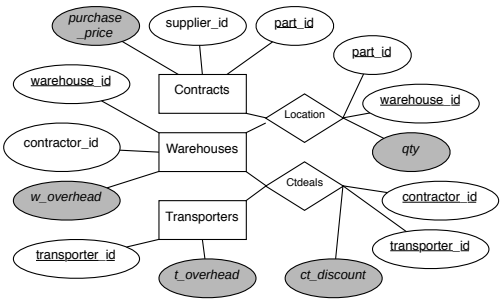


Figure 1: A supply chain decision support schema. Measure attributes are shaded.

1.3 MPF Queries and Decision Support

So far, we have emphasized the relationship between the MPF setting and probabilistic inference. However, MPF queries can be used in a broader class of applications. Consider the enterprise schema shown in Figure 1:

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 when 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.

Total investment on each supply chain is given by the product of these base relations for a particular combination of dimension values. This can be computed by the following view:

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

Now consider querying this view, not for a complete supply chain, but rather, only for each part. For example, we may answer the question *What is the minimum supply chain investment on each part?* by posing the MPF query:

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

Several additional types of queries over this schema are natural: *What is the cost of taking warehouse w1 offline? What is the cost of taking warehouse w1 offline if, hypothetically, part p1 had a 10% lower price?* See Section 2.2.

1.4 Contributions and Organization

The contributions of this paper are as follows:

1. We introduce MPF queries, which significantly generalize the relational framework introduced by Wong [23] for probabilistic models. With this generalized class of queries, probabilistic inference can be expressed as a query evaluation problem in a relational setting. MPF queries are also motivated by decision support applications.

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 significant performance improvement.

The paper is organized as follows: Section 2 formally defines the MPF query setting; in Section 3 we show how probabilistic inference can be cast in terms of MPF queries; Section 4 describes and analyzes optimization schemes for single MPF queries; experimental results are shown in Section 5.

2. MPF SETTING DEFINITION

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

DEFINITION 1. *Let s be a relation with schema $\{A_1, \ldots, A_m, f\}$ where $f \in \mathbb{R}$. Relation s is a* ***functional relation*** *(FR) if the Functional Dependency $A_1 A_2 \cdots A_m \rightarrow f$ 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 $A_i \rightarrow f$ 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 the domains of $A_1, \ldots, A_m$, although this is required in principle for probability measures. We refer to such relations as *complete*. Finally, any relation can be considered an FR where f is implicit and assumed to take the value 1.

Functional relations can be combined using a stylized join to create functions with larger domains. This join is defined with respect to a product operation on measure attributes:

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

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

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

Implicit in the Relational Algebra expression for product join are the assumptions that tables define a unique measure, and that measure attributes are never included in the set of join conditions. Note that the domain of the resulting joined function is the union of the domains of the operands, and that the product join of two FRs is itself an FR.

We propose the following SQL extension for defining views based on the product join:

```
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. This simplifies syntax and makes explicit that a single product operation is used in the product join.

For example, our decision support schema can be defined as:

```
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) as inv
  from contracts c, warehouses w, transporters t,
      location l,ctdeals ct
  where c.pid = l.pid and l.wid = w.wid ...
```

2.1 MPF Queries

We are now in position to define MPF queries.

DEFINITION 3. ***MPF Queries.*** *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*

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

where $X \subseteq \bigcup_{i=1}^{n} \mathrm{Var}(s_i)$, and AGG *is an 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.

To clarify the definition, we have not specified the MPF setting at its full generality. FRs may contain more than a single measure attribute as long as the required functional dependency holds for each measure attribute. For simplicity of presentation, all examples of FRs we use will contain a single measure attribute. Also, the requirement that the measure attribute f is real-valued ($f \in \mathbb{R}$) is not strictly necessary. However, f must take values from a set where a multiplicative and an additive operation are defined in order to specify the product operation in product join and the aggregate operation in the MPF query. For the real numbers we may, obviously, take $\times$ as the multiplicative operation and $+$, min or max as the additive operation. Another example is the set $\{0, 1\}$ with logical $\wedge$ and $\vee$ as the multiplicative and additive operations.

For the purposes of query evaluation, significant optimization is possible if operations are chosen so that the multiplicative operation distributes with respect to the additive operation. This corresponds to the condition that the set from which f takes values is a commutative semi-ring [1, 15]. Both the real numbers and $\{0, 1\}$ with their corresponding operations given in the previous paragraph possess this property.

2.2 MPF Query Forms

We can identify a number of useful MPF query variants that arise frequently. Using the schema in Figure 1, we present templates and examples for variants 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 MPF queries 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 function's measure 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 measure for the query variables conditioned on given values for other variables. We add a `where Y=c` clause to the Basic query with $Y \notin X$. *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 Section 4 are for the three query types above. Of course, there are other useful types of MPF queries. Future work might consider optimizing the following types:

Constrained range: Here function values in the result are restricted. This is useful 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 contractor c1's deal with transporter t1 to transporter t2?

3. MPF QUERIES AND PROBABILISTIC INFERENCE

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.

3.1 Probabilistic Databases

Dalvi and Suciu [7, 18], and Ré et al. [17, 18] define a representation for probabilistic databases [11], and present an approximate procedure to compute the probability of query answers. They represent probabilistic relations as what we have called functional relations, where each tuple is associated with a probability value. Queries are posed over these functional relations, with the probability of each answer tuple given by the probability of a boolean formula. Ré et al. [17] define a middleware solution to approximate the probability of the corresponding boolean formula.

A significant optimization in their framework pushes evaluation of suitable subqueries to the relational database engine. These subqueries are identical to MPF queries, that is, aggregate queries over the product join of functional relations. Thus, their optimization is constrained by the engine's ability to process MPF queries. Our optimization algorithms in Section 4 allow for significantly more efficient processing of these subqueries than existing systems, thus improving the efficiency of their middleware approximation method.

They specify two aggregates used in these subqueries: SUM, and PROD, where $\text{PROD}(\alpha, \beta) = 1 - (1 - \alpha)(1 - \beta)$. Optimization of the SUM case is handled directly by the algorithms we present, but the distributivity assumptions we require for optimization (see Section 4) are violated by the PROD aggregate, since $\text{PROD}(\alpha\beta, \alpha\gamma) \neq \alpha\text{PROD}(\beta, \gamma)$. However, we may bound the non-distributive PROD aggregate as follows:

$$\alpha\text{PROD}(\beta, \gamma) \leq \text{PROD}(\alpha\beta, \alpha\gamma) \leq 2\alpha \max(\beta, \gamma).$$

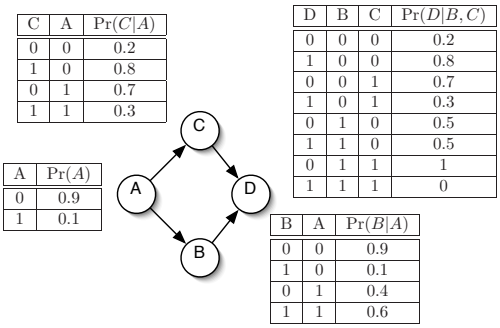
We can compute each of the two bounds in the MPF setting, so optimization is possible. In cases where this loss of precision is allowable, ranking applications for example, the gains of using the MPF setting is significant due to its optimized evaluation.

3.2 Bayesian Networks

In general, we can use the MPF setting to represent discrete multivariate probability distributions that satisfy certain constraints. In this section, we show how MPF queries can be used to query Bayesian Network (BN) models of uncertain data. BNs [16, 14, 6] are widely-used probabilistic models that satisfy some conditional independence properties that allow the distribution to be factored into local distributions over subsets of random variables.

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 was 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.

Consider binary random variables A, B, C, D. A func-



C	A	Pr($C\|A$)
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($D\|B,C$)
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($B\|A$)
0	0	0.9
1	0	0.1
0	1	0.4
1	1	0.6

Figure 2: A simple Bayesian Network

tional relation of size 2^4 can be used to 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, $\Pr(C|A = 0)$.

3.3 Discussion and Related Work

Wong et al. [23, 24, 25] address the probabilistic inference task in relational terms and propose an extended relational model and algebra that expresses exactly 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 MPF query optimization in Section 4 is a first step in this direction.

Finally, we remark that this paper applies to the problem of scaling *exact* probabilistic inference. This is required in settings where results are composed with other functions that are not monotonic with respect to likelihood, including systems that compute expected risk or utility. In these settings approximate probability values are not sufficient. However, for other systems where only relative likelihood suffices, e.g., ranking in information extraction, approximate inference procedures [21, 22, 26] are sufficient and may be more efficient.

4. MPF QUERY OPTIMIZATION

Section 2 hinted at the optimization benefit possible when MPF views and queries are defined over domains with operations chosen such that the multiplicative operation distributes with respect to the additive operation. We develop this observation in this section. A generic algorithm has been proposed for efficiently solving MPF problems [1, 15] in non-relational settings. It makes use of this key distributive property to reduce the size of function operands, thus making evaluation more efficient. We may cast this in relational terms as follows: the Group By ('additive') operation distributes with the product join ('multiplicative') operation so that Group By operator nodes can be pushed down into the join tree thus reducing the size of join operands.

We study two algorithms and their variants that use the distributivity property to optimize MPF query evaluation by pushing down Group By nodes into join trees: *(CS)* Chaudhuri and Shim's algorithm for optimizing aggregate queries [4, 5]; *(CS+)* our simple extension of CS that yields significant gains over the original; *(VE)* the greedy heuristic Variable Elimination algorithm [27] proposed for probabilistic inference; and *(VE+)* our extension to VE based on Chaudhuri and Shim's algorithm that finds significantly better plans than VE by being robust to heuristic choice. These algorithms optimize basic, restricted answer and constrained domain MPF query types. To the best of our knowledge, this is the first paper to cast VE as a join tree transformation operation.

In this central section of the paper, we will define and describe each of the optimization algorithms; present conditions under which evaluation plans can be restricted to the linear class, thus avoiding the extra overhead of searching over nonlinear plans[1]; we will characterize and compare the plan spaces explored by each of the algorithms given and show that the plan space explored by CS+ contains the space explored by VE; we will analyze the optimization time complexity of the algorithms, and also give conditions based on schema characteristics where VE will have significantly lower optimization time complexity than CS+; we will extend VE so that its plan space is closer to the space of CS+ plans without adding much optimization overhead; and finally, we will propose a cost-based ordering heuristic for Variable Elimination.

4.1 MPF Query Evaluation Algorithms

In this section, we will define the CS and VE algorithms along with our extensions. We make use of the example schema in Figure 1 again, with 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.

[1]We define linear and nonlinear plans in Section 4.1.

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

We need to define linear and nonlinear plans. In linear plans, every interior node in a join tree has at least one leaf node as a child. Conversely, in nonlinear plans both children of interior nodes may be interior nodes as well. Leaf nodes are base relations that appear in the query, whereas interior nodes are intermediate relations that result from performing join or Group By operations.

The CS Algorithm. Chaudhuri and Shim [4, 5] define an optimization scheme for aggregate queries that pushes Group By nodes into join trees. The CS algorithm explores the space of linear plans using an extension of the dynamic programming optimization algorithm of Selinger et al. [19]. They also define a condition that ensures the semantic correctness of the plan transformation.

Algorithm 1 illustrates the CS procedure. As in Selinger's dynamic programming algorithm, joinplan() in line 2 finds the best linear plan that joins base relation r_j to the optimal plan for relation set S_j (optPlan(S_j)). However, the usual algorithm is modified so that line 3 finds the best linear plan that joins r_j to the optimal plan for relation set S_j, this time modified to include a Group By node as its topmost node. Grouping in this added node is done on query variables and variables appearing in a join condition on any relation not yet joined into S_j. This ensures the semantic correctness of the plan transformation. The cheapest of these two candidate plans 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 naïve plan with a single Group By node at the root of the join tree.

Algorithm 1 The CS optimization algorithm

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

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

The CS+ Algorithm. We make a simple extension to the CS algorithm, denoted CS+, that produces much better plans. In the CS+ algorithm, joins are annotated as product joins and the distributive property of the aggregate and product join is verified. As in the CS algorithm, Group By interior nodes must have as grouping variables both query variables and variables appearing in any join condition on any relation not yet joined into the current subplan. This again ensures the semantic correctness of the resulting plan. Figure 4 shows the CS+ plan for Q1. A Group By node is added after the join of *Location* and *Contracts* since the subplan joining *Warehouses* is cheaper.

The Nonlinear CS+ Algorithm. We extend the CS+ procedure to consider nonlinear plans as follows: (a) for relation set S_j we consider joining every relation set of size $< j$; (b) we change joinplan() so that it returns the best nonlinear plan joining two relations; (c) instead of comparing two plans we now compare four: one without any Group By nodes (corresponding to line 2); another with a Group By on S_j (corresponding to line 3); another with a Group By on the operand (say, s') being joined to S_j; and finally, a plan with Group By nodes on both S_j and s'. The cheapest of these four plans is selected. From this point forward, will refer to this nonlinear extension as CS+.

The VE Algorithm. Variable Elimination [27] 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 that have not been eliminated so far. Algorithm 2 lists the VE algorithm. We denote the set of relations in S where variable v_j appears as $\text{rels}(v_j, S)$. So $\text{optPlan}(\text{rels}(v_j, S))$ is the optimal plan found by the optimizer for joining the set of relations where variable v_j appears. We abuse notation slightly in line 9 where p denotes the relation resulting from executing plan p of line 6.

Algorithm 2 The Variable Elimination Algorithm

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

Figure 5 shows the VE plan for Q1 with elimination order *tid,pid,cid*. The efficiency of VE for query evaluation is determined by the variable elimination order (see Section 4.6). We again require that grouping in interior nodes contain query variables and variables required for any subsequent joins as grouping variables to ensure semantic correctness of the resulting plans. In VE this is satisfied by definition since query variables are not candidates for elimination and variables are candidates for elimination as long as there is a relation in the current set that includes it in a join condition.

4.2 Nonlinear MPF Query Evaluation

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

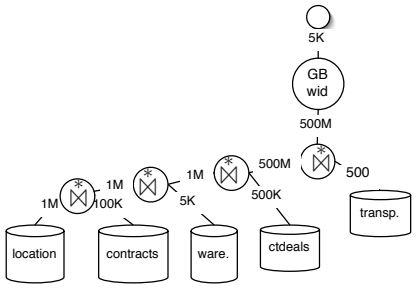


Figure 3: A CS plan for Q1

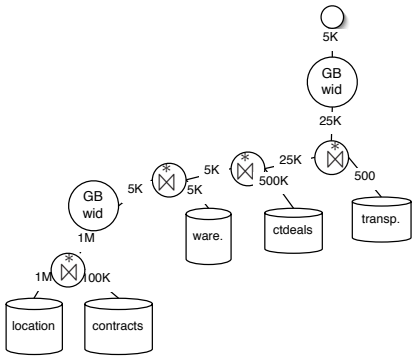


Figure 4: A CS+ plan for Q1

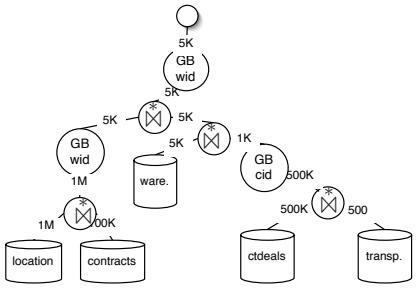


Figure 5: A VE plan for Q1

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 nonlinear join order. In fact, an advantage of VE is that it produces nonlinear plans with, usually, 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, $\hat{\sigma}_X = \min_{s \in \text{rels}(X)} |s|$. 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 Group By 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:

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

4.3 Plan Spaces

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

Definition 4 (Evaluation Plan Space $\mathscr{P}$). *Denote as $\mathscr{P}$ the space of all nonlinear semantically correct evaluation plans where either Group By or join nodes are interior nodes, and are equivalent to a plan with only join interior nodes and a single Group By 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 interior Group By nodes.

Definition 5 (CS+ Plan Space $\mathscr{P}(CS+)$). *Let $p \in \mathscr{P}$ have the following property: if a single interior Group By node is removed, the cost of the subplan rooted at its parent node is greater. We define $\mathscr{P}(CS+)$ to be the set of all plans in $\mathscr{P}$ that satisfy this property.*

As we saw before, CS+ yields a plan that is no worse than the plan with a single Group By at the root.

Definition 6 (VE Plan Space $\mathscr{P}(VE)$). *Let $p \in \mathscr{P}$ have the following properties for every non-query variable v: 1) a Group By node immediately follows the join node closest to the root where v appears as a join condition, and 2) all joins where v appears as a join condition are contiguous. We define $\mathscr{P}(VE)$ as the set of all plans in $\mathscr{P}$ that satisfy these properties.*

VE does not guarantee optimality due to its greedy heuristic search, and it is known that finding the variable ordering that yields the minimum cost plan is NP-complete in the number of variables.

Theorem 1 characterizes these plan spaces. We say that $p \in \mathscr{P}(A)$ if optimization algorithm A either computes its cost, or can *guarantee* that there exists a plan $p' \in \mathscr{P}(A)$ that is cheaper than p. Although CS+ uses dynamic programming, its greedy heuristic for adding Group By nodes makes its search through $\mathscr{P}$ incomplete. Not surprisingly, the plan space searched by VE is also incomplete. However, we see that the plan space searched by CS+ includes the plan space searched by VE. That is, CS+ will consider the the minimum cost plan returned by VE for a given ordering.

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

$$\mathscr{P} \supset \mathscr{P}(CS+) \supset \mathscr{P}(VE).$$

To prove this theorem, we need the following Lemma:

Lemma 1. *Consider relations $S_n = \{r_1, \ldots, r_n\}$ where variable v only appears in r_k. Let*

$$S'_n = \{r_1, \ldots, \text{GroupBy}(r_k), \ldots, r_n\}.$$

For the CS+ algorithm, the following holds: for each output tuple ordering, $\text{Cost}(\text{optPlan}(S_n)) \leq \text{Cost}(\text{optPlan}(S'_n))$.

Proof. By induction on n. If $n = 2$, the Lemma follows since the plans are compared directly in line 4 of Algorithm 1. Now assume Lemma is 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 of Algorithm 1. Otherwise, if $r_k \neq r_n$ then $r_k \in S_{n-1}$ we have by the inductive hypothesis $\text{Cost}(\text{optPlan}(S_{n-1})) \leq \text{Cost}(\text{optPlan}(S'_{n-1}))$ for each tuple ordering of S_{n-1} so the Lemma follows. □

PROOF. (Theorem 1)

($\mathscr{P}(CS+) \subseteq \mathscr{P}$) This follows by definition of CS+ and the semantic correctness of its plan transformation.

($\mathscr{P}(CS+) \neq \mathscr{P}$) By the greedy heuristic, any plan $p' \in \mathscr{P}$ extending the plan not chosen in line 4 of Algorithm 1 is not included in $\mathscr{P}(CS+)$. However, no guarantee is given that p' is more expensive than the plans extending the least expensive plan of line 4.

($\mathscr{P}(VE) \subseteq \mathscr{P}(CS+)$) Let p be the best VE plan for elimination order $v_1, \ldots, v_n$. We prove this statement by induction on n. If $n = 1$, the statement holds trivially. Now assume the statement is true for $m \leq n-1$ and consider variables v_m and v_n and $S_m = \text{rels}(v_m, S)$. By the inductive hypothesis we have that the subplan in p that eliminates v_m is in $\mathscr{P}(CS+)$. But, since v_m only appears in the relation resulting from optPlan(S_m), by Lemma 1 we have that the subplan in p eliminating v_n is in $\mathscr{P}(CS+)$ as well. Thus $p \in \mathscr{P}(CS+)$.

($\mathscr{P}(VE) \neq \mathscr{P}(CS+)$) Consider a plan $p \in \mathscr{P}(VE)$ for a variable ordering where v_1 is preceded by v_2 but $\text{rels}(v_1) \subseteq \text{rels}(v_2)$. In this case, VE does not consider adding Group By nodes to eliminate v_1 in the subplan that eliminates v_2, but there exists a plan $p' \in \mathscr{P}(CS+)$ that attempts to add a Group By node to 'eliminate' v_1 once $\text{rels}(v_1)$ are joined in p. Thus $p' \notin \mathscr{P}(VE)$. □

4.4 Extending the Variable Elimination Plan Space

We saw in the previous Section that the plan space considered by VE is a subset of the plan space considered by CS+. In this section, we extend VE to narrow this gap by delaying the elimination of variables if that results in cheaper plans and by pushing Group By nodes into elimination subplans. We use Functional Dependency information to implement the delay strategy, and also use cost-based local decisions similar to those used by the CS+ algorithm to implement both the delay and pushing strategies.

As defined, VE considers all variables as candidates for elimination; however, the elimination of some variables might have no effect, that is, the result of Group By is the same as projection. In other words there is exactly one tuple for each group in the Group By clause. The following property captures this:

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

PROOF. First, we note that for any functional relation s with $XY = \text{Var}(s)$ where the FD $X \rightarrow s[f]$ holds, then $\text{GroupBy}_{X'}(s) = \pi_{X'}(s)$ 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 can partition $\text{Var}(r)$ into $\cup_i X_i$ and Z with $Y \in Z$ and the Proposition follows. □

A sufficient condition for Proposition 1 to apply is that primary keys are given for each base relation where Y is not part of any key. Furthermore, this Proposition holds for any set of relations, so in any iteration of 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. Applying this Proposition has the effect of avoiding the addition of unnecessary Group By nodes.

In the absence of FD information, we present an extension to Variable Elimination that uses cost-estimation to both delay variable elimination and push Group By nodes into elimination subplan join trees.

The VE+ Algorithm. Algorithm 2 requires two changes: 1) in line 6 we set $p = \text{optPlan}(\text{rels}(v_j, S))$ to potentially delay elimination to later iterations of the algorithm, and 2) we assume that the function optPlan() uses the local greedy conservative heuristic of CS+ to push Group By nodes into elimination subplan join trees. The first modification removes the Group By node in line 6 which eliminates the variable chosen at the current iteration. This is done so that the greedy heuristic of the second modification (from the CS algorithm) is used to decide on the addition of this Group By node if it yields a locally better plan.

These additions have the effect of extending $\mathscr{P}(VE)$ as follows:

DEFINITION 7 (VE+ PLAN SPACE $\mathscr{P}(VE+)$). *Let $p \in \mathscr{P}$ satisfy the following conditions: 1) if a single interior Group By node is removed, the cost of the subplan rooted at its parent node is greater; and 2) for every non-query variable v all join nodes where v appears as a join condition are either contiguous or separated by only Group By nodes; that is, no join node where v does not appear as a join condition separates them. We define $\mathscr{P}(VE+)$ as the set of all plans that satisfy these properties.*

Now we may update our inclusion relationship:

THEOREM 2 (EXTENDED VE SPACE). *Using the notation above, we have:*

$$\mathscr{P}(VE) \subset \mathscr{P}(VE+) \subset \mathscr{P}(CS+).$$

PROOF. The proof is similar to that of Theorem 1.

($\mathscr{P}(VE) \subseteq \mathscr{P}(VE+)$) Given an elimination order, the same proof for $CS+$ and VE shows this case.

($\mathscr{P}(VE) \neq \mathscr{P}(VE+)$) Consider an elimination order where v_i follows v_j but $\text{rels}(v_i) \subset \text{rels}(v_j)$, $VE+$ considers adding Group By nodes to eliminate v_i while creating the plan for $\text{rels}(v_j)$, whereas VE does not. This is the same argument given above for VE and $CS+$.

($\mathscr{P}(VE+) \subseteq \mathscr{P}(CS+)$) The proof for this is the same as the proof of $\mathscr{P}(VE) \subseteq \mathscr{P}(CS+)$.

($\mathscr{P}(VE+) \neq \mathscr{P}(CS+)$) 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 $\mathscr{P}(VE+)$ and $\mathscr{P}(CS+)$ corresponding to plans where join nodes for a variable are not necessarily contiguous, our experimental results in Section 5 show that $CS+$ rarely produces plans that are not reachable by $VE+$.

4.5 Optimization Complexity

Another dimension of comparison between these procedures is time required to find optimum plans. Since search for optimal sub-plans in VE only occurs in line 6, for views where variables exhibit low connectivity, that is, variables

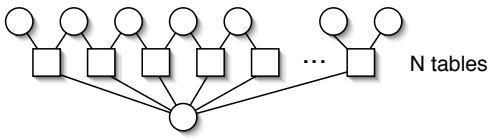


Figure 6: An example star MPF view.

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 heuristic, only two relations have to be joined at a time for each variable, yielding optimization time complexity of $O(M)$ for M variables.

Theorem 3 summarizes these findings. We refer to an ordering heuristic for VE as proper if it orders variables by connectivity. Of course, while this guarantees good performance in terms of optimization time, it does not guarantee good performance in terms of query evaluation time since the resulting plan with a 'proper' heuristic might be suboptimum.

THEOREM 3 (OPTIMIZATION TIME COMPLEXITY). *Let S be average variable connectivity, let 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 VE, a proper heuristic chooses a variable v_j in line 5 of Algorithm 2 where, on average, $|\text{rels}(v_j)| = S$. Finding a plan for these tables in line 6 takes $O(S2^S)$. At worst, this is done M times, once for each variable. □

4.6 Elimination Heuristics

We now define statistics to decide heuristic variable elimination orderings.

DEFINITION 8. *Define the degree and width statistics for variable v as:*

1. $degree(v) = |\text{GroupBy}(\text{optPlan}(\text{rels}(v, S)))|$;
2. $width(v) = |\text{optPlan}(\text{rels}(v, S))|$.

The degree heuristic orders variables increasingly according to estimates of the size of relation p in line 6 of Algorithm 2, while the width heuristic orders variables increasingly according to estimates of the size of p without its topmost Group By node.

In the VE literature [9] these statistics are estimated 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. This is an effect of the fact that the cost metric minimized in VE, as defined in the MPF literature [1, 15], is the number of addition and multiplication operations used in evaluating the query. This is a valid cost metric in that setting since operands are assumed to be memory-resident, and more significantly, single algorithms are assumed to implement each of the multiplication and summation operations. These are not valid assumptions in the relational case where there 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. Thus, relational cardinality estimates are used in our implementation to compute these statistics.

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:

DEFINITION 9. *Define the elimination cost statistic for variable v as $elimcost(v) = \text{Cost}(\text{optPlan}(\text{rels}(v, S))$.*

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 trade-off between greedily minimizing the cost of the current elimination subplan vs. minimizing the cost of subsequent elimination subplans. To address this trade-off we combine the *degree* and either *width* or *elimination cost* heuristics by computing the mean of their normalized values. We study the effect of these heuristics and their combinations in Section 5.3.

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

5. EXPERIMENTAL RESULTS

We now present experimental results illustrating the discussion in Section 4. We modified the PostgreSQL 8.1 optimizer to implement each algorithm at the server (not middleware) level. The extensions in Section 2 were added to the PostgreSQL language. 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 5.4 make this comparison and illustrate the significant difference in performance.

We use two testbeds for our experiments. The first is the decision support schema of Figure 1 for which we create a number of instances at random. The *Contracts, Warehouses* and *Transporters* relations were populated according to a *Scale* parameter, whereas *Location* and *CTdeals* were populated according to *Density* parameters. The cardinalities and domain sizes in Table 1 correspond to *Scale* = 100, *Density(CTDeals)* = 100% and *Density(Location)* = 20%. These are default settings unless specified otherwise. Non-key attributes in *Contracts* and *Warehouses*, compound keys in *Location* and *CTdeals* and all measure attributes are populated uniformly at random.

The second testbed consists of three variants of the Schema 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 multiple common variables each connecting to a distinct set of three tables in the linear part. The number of tables $N = 5$, all variables have domain size 10 and all functional relations are complete. Measure attributes are populated uniformly at random from the interval $[0, 1]$.

This section is organized as follows. First, in Section 5.1 we test the benefit of nonlinear evaluation of MPF queries and the linearity condition of Section 4.2. We will see that nonlinear evaluation performs better than linear evaluation except when linear plans are admissible as given by the linearity condition. In Section 5.2 shows how the extension of the Variable Elimination algorithm given in Section 4.4 benefits evaluation. We will see that VE+ with the degree heuristic finds the optimal CS+ plan, while never finding a plan that is worse than VE. Section 5.3 illustrates the effect of elimination heuristics for Variable Elimination. We will see that schema characteristics are the main determinant of performance of each heuristic. However, we will also see that VE+ is robust to heuristic choice and is able to find near-optimal plans for all three heuristics we have defined. Finally, Section 5.4 tests the trade-off between optimization complexity and plan quality in each of the algorithms presented. We will see that all algorithms proposed produce better quality plans than existing systems while, in some cases, not adding significant optimization time. Furthermore, we will also see that schema characteristics are the main determinants of both quality and planning time for these algorithms.

5.1 Nonlinear Evaluation

Section 4.2 showed the benefit of nonlinear plans for MPF query evaluation. The experiment in Figure 7 illustrates how the plan linearity condition is applied. On our first testbed 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(CTdeals)* parameter 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 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,

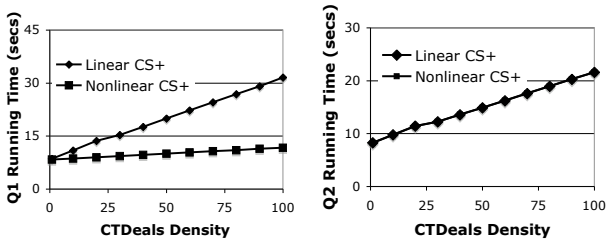


Figure 7: Plan Linearity Experiment

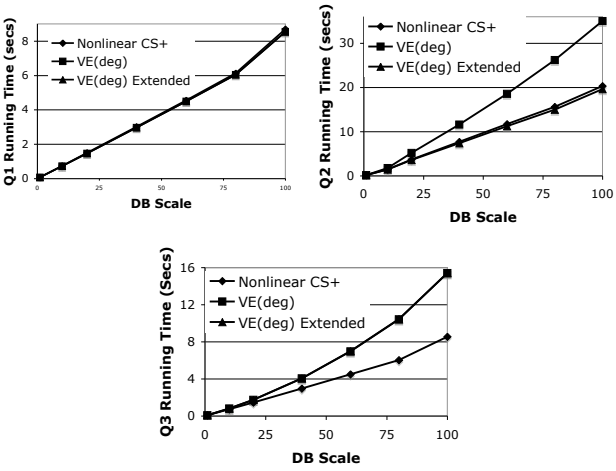


Figure 8: VE Extended Space Experiment

whereas for Q2, we have $\sigma_{tid} = \hat{\sigma}_{tid} = 500$ which makes the inequality hold showing the applicability of the linearity condition.

5.2 Extended Variable Elimination Space

Section 4.4 showed how to extend the VE plan space closer 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 *Scale* parameter 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. The VE+ extension to VE guarantees that we find a plan no worse than the plan obtained by VE without the extension; this is reflected in the results shown here.

5.3 Elimination Heuristics

We now show experimental results on the effect of ordering heuristic on plan quality for Variable Elimination. Using our first testbed, we run two queries and plot their running time as a function of the *Scale* parameter:

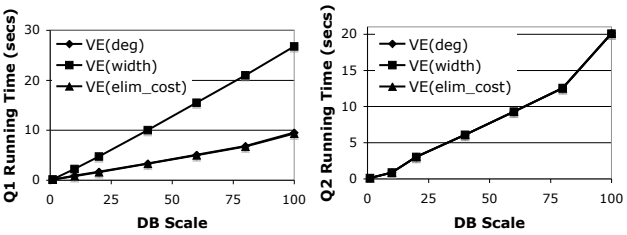


Figure 9: Ordering Heuristics Experiment

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**

```
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 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 using our second testbed. 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 4.6 we ran both the original VE algorithm and its extended space version described in Section 4.4. We implement the elimination cost heuristic using an overestimate: we fix a linear join ordering and allow choice of access paths and join operator algorithms. We also include results for combinations of the *degree and width* and *degree and elimination cost* heuristics[2]. 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 will select the common variable first since 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 no Group By optimization is done. 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.

[2]Combinations are implemented by normalizing each estimate and multiplying the normalized values

Table 3: Random Heuristic Experiment Result

Schema	VE(rand)	VE(rand) ext.
star	30830.42 ± 1470.78	770.78 ± 5.60
multistar	11730.35 ± 298.86	4559.58 ± 149.03
linear	72.04 ± 0.29	51.78 ± 0.36

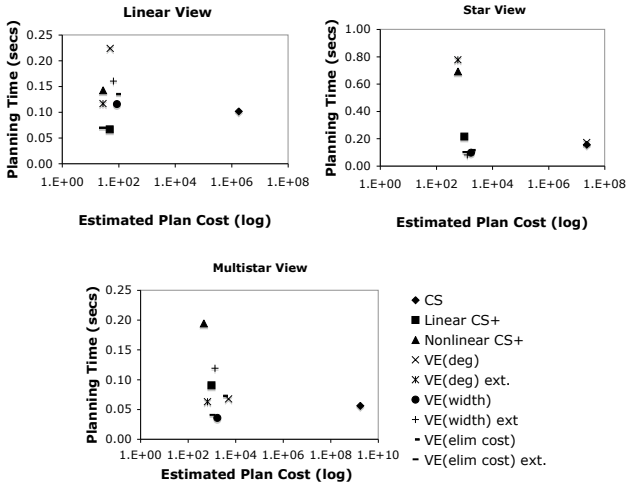


Figure 10: Optimization Time Tradeoff Experiment

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.

5.4 Optimization Cost

The following experiment illustrates the trade-off between plan quality and optimization time of the algorithms. For each view in our second testbed (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. Variable Elimination with 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 general, for VE compared to nonlinear CS+.

6. 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 them. We are currently working on optimization techniques for anticipated 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 directly 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.

Acknowledgements

This work was partially supported by an NSF Medium ITR grant ITR IIS-0326328, the NSF Cybertrust project, and a Ford Pre-Doctoral Fellowship from the National Academies.

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

[17] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. Technical Report 2006-06-05, University of Washington, 2006.

[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] M. Wainwright and M. Jordan. Graphical models, exponential families and variational inference. Technical Report 649, Department of Statistics, University of California, Berkeley, 2003.

[22] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

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

[24] 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.

[25] 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.

[26] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2002-35, Mitsubishi Electric Research Laboratories, 2002.

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