View Disassembly:
Evaluating Queries Piecemeal

Parke Godfrey

Information Science & Technology
U.S. Army Research Laboratory
Adelphi, Maryland, U.S.A.

godfrey@arl.mil

& University of Maryland

October 1998

Work done in conjunction with

Jarek Gryz

Department of Computer Science
York University
Toronto, Ontario, Canada

jarek@cs.yorku.ca

Work based on the paper

P. Godfrey & J. Gryz
View Disassembly

to be presented at
ICDT'99 in January 1999 in Jerusalem.

I. Motivation

Query Folding

- **Academic_units** (did, address)
- **Employees** (eid, did)
- **Benefits** (eid, premium, provider)

$$\begin{aligned} &\mathit{query}(A) \leftarrow \\ &\qquad \mathit{academic\_units}(D,\ A), \\ &\qquad \mathit{employees}(E,\ D), \\ &\qquad \mathit{benefits}(E,\ \_,\ \mathit{\ae tna}). \end{aligned}$$

$$\begin{aligned} &\mathit{cache\_one}(E,\ A) \leftarrow \\ &\qquad \mathit{academic\_units}(D,\ A), \\ &\qquad \mathit{employees}(E,\ D). \end{aligned}$$

$$\begin{aligned} &\mathit{cache\_two}(E,\ P) \leftarrow \\ &\qquad \mathit{employees}(E,\ D) \\ &\qquad \mathit{benefits}(E,\ \_,\ P). \end{aligned}$$

$$\mathit{query}(A) \Leftarrow \mathit{cache\_one}(E,\ A),\ \mathit{cache\_two}(E,\ \mathit{\ae tna}).$$

Query Unfolding

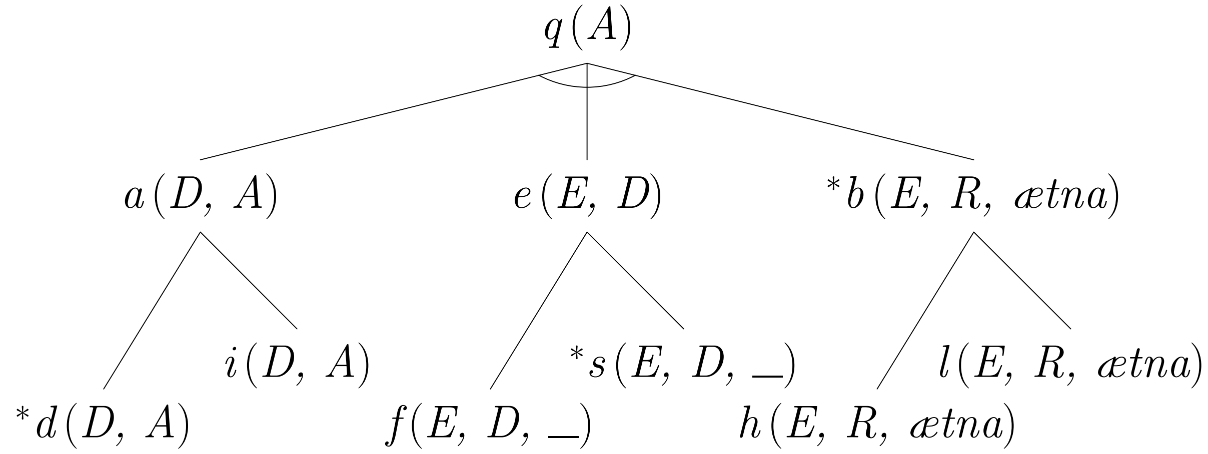
- **Departments** (did, address)
- **Institutes** (did, address)
- **Faculty** (eid, did, rank)
- **Staff** (eid, did, position)
- **Health_Ins** (eid, premium, provider)
- **Life_ins** (eid, premium, provider)

$$\begin{aligned}
\mathit{academic\_units}(X,\ Y) &\leftarrow \mathit{departments}(X,\ Y).\\
\mathit{academic\_units}(X,\ Y) &\leftarrow \mathit{institutes}(X,\ Y).\\
\mathit{employees}(X,\ Y) &\leftarrow \mathit{faculty}(X,\ Y,\ Z).\\
\mathit{employees}(X,\ Y) &\leftarrow \mathit{staff}(X,\ Y,\ Z).\\
\mathit{benefits}(X,\ Y,\ Z) &\leftarrow \mathit{health\_ins}(X,\ Y,\ Z).\\
\mathit{benefits}(X,\ Y,\ Z) &\leftarrow \mathit{life\_ins}(X,\ Y,\ Z).
\end{aligned}$$

$$\begin{aligned}
\mathit{query}(A) \leftarrow\ & \mathit{academic\_units}(D,\ A),\\
& \mathit{employees}(E,\ D),\\
& \mathit{benefits}(E,\ \_,\ \mathit{\ae tna}).
\end{aligned}$$

$$\begin{aligned}
\mathit{cache}(A,\ P) \leftarrow\ & \mathit{institutes}(D,\ A),\\
& \mathit{staff}(E,\ D,\ \_),\\
& \mathit{benefits}(E,\ \_,\ P).
\end{aligned}$$

AND/OR Trees



$$\begin{aligned} \mathit{query}(A) \leftarrow\ & \mathit{academic\_units}(D,\ A), \\ & \mathit{employees}(E,\ D), \\ & \mathit{benefits}(E,\ \_\,,\ \textit{ætna}). \end{aligned}$$

$$\begin{aligned} \mathit{cache}(A,\ P) \leftarrow\ & \mathit{institutes}(D,\ A), \\ & \mathit{staff}(E,\ D,\ \_), \\ & \mathit{benefits}(E,\ \_\,,\ P). \end{aligned}$$

II. Discounted Queries

Let $\mathcal{Q}$ be a query.

Let $\mathcal{U}_1$, …,$\mathcal{U}_k$ be unfoldings of $\mathcal{Q}$.

$\mathcal{Q}\backslash\{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$ denotes the *discounted query* of $\mathcal{Q}$ with *unfoldings-to-discount* $\mathcal{U}_1$, …,$\mathcal{U}_k$.

Define **unfolds**($\mathcal{Q}$) to be the set of *extensional unfoldings* of $\mathcal{Q}$.

We define $\mathcal{Q}\backslash\{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$ to mean

$$\mathbf{unfolds}\,(\mathcal{Q}) - (\bigcup_{i=1}^{k} \mathbf{unfolds}\,(\mathcal{U}_i))$$

A discounted query is a type of *remainder query.*

When $\mathcal{Q}$ is *covered* by $\mathcal{U}_1$, …,$\mathcal{U}_k$, then $\mathcal{Q}\backslash\{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$ evaluates empty.

- $\mathcal{Q}\backslash\{\mathcal{Q}\}$
- $\mathcal{Q}\backslash\mathbf{unfolds}\,(\mathcal{Q})$

Applications

Some unfoldings of the view may

- (*Cache use*) be effectively cached from previous queries, or may be materialized views,
- (*Void sub-queries*) be known to evaluate empty (by reasoning over the integrity constraints),
- (*Security*) match protected queries, which cannot be evaluated for all users, and
- (*Old answers*) be subsumed by previously asked queries, so are not of interest to the user.

Complex queries and views that employ interleaved unions and joins arise

- in mediation over heterogeneous databases,
- in data warehousing, and
- even in current database systems.

The general goal is optimization in all these tasks.

III. View Disassembly

- How can one find a query is equivalent answer set-wise to a discounted query?
- We treat queries involving views as AND/OR trees.
- We rewrite the the AND/OR tree of the original query to an AND/OR tree that is equivalent to the discounted query.

We call this rewrite procedure *view disassembly*.

Outline

1. Removing Simple Unfoldings
2. Deciding Coverage
3. Algebraically Optimal Rewrites
4. Approximation Rewrites
    - **a.** Naive View Disassembly
    - **b.** The Unfold/Refold Algorithm

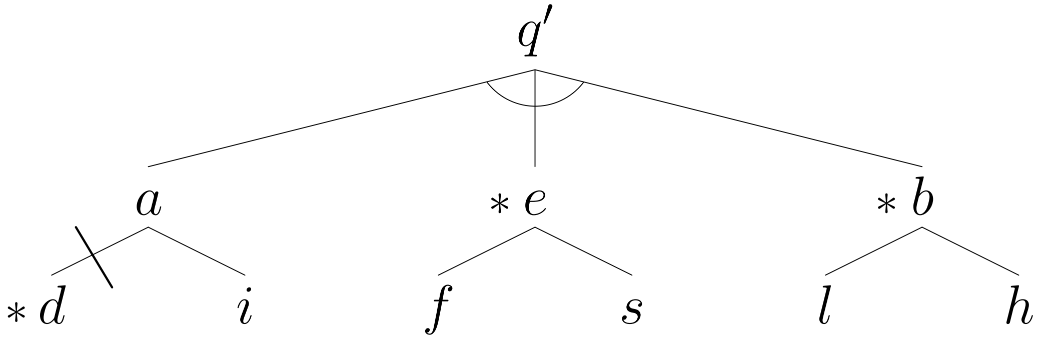
Removing Simple Unfoldings

Intuition: unfolding removal is always simple. The AND/OR tree can always be *pruned* somehow to "remove" the unfolding.

This intuition is wrong. However, for an important class of unfoldings we call *simple unfoldings*, this is true.

$$q \leftarrow a,\ e,\ b.$$

$$c_1 \leftarrow d,\ e,\ b.$$



$$c_2 \leftarrow d,\ f,\ l.$$

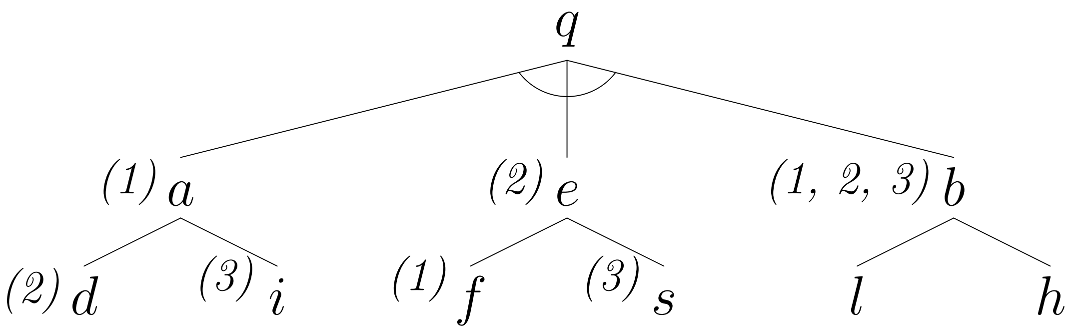
Note that c_2, however, is not *simple*.

Deciding Coverage

Example

$$\mathcal{Q}: \quad q \leftarrow a,\ e,\ b.$$

$$\mathcal{U}_1: \quad u_1 \leftarrow a,\ f,\ b.$$
$$\mathcal{U}_2: \quad u_2 \leftarrow d,\ e,\ b.$$
$$\mathcal{U}_3: \quad u_3 \leftarrow i,\ s,\ b.$$



$$\mathcal{Q} \backslash \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3\} \quad = \quad \emptyset$$

Deciding Coverage

Complexity

Query $\mathcal{Q}$ is covered by $\mathcal{U}_1, \ldots, \mathcal{U}_k$ *iff*

$$\mathbf{unfolds}\,(\mathcal{Q}) - (\bigcup_{i=1}^{k} \mathbf{unfolds}\,(\mathcal{U}_i)) \quad = \quad \emptyset$$

A *discounted view instance* $\mathcal{V}$ is a pair:

- AND/OR tree (the query); and
- a list of AND/OR trees (the unfoldings-to-discount).

Define **COV** as the set of all discounted view instances that are covered.

Theorem. **COV** is *co***NP***-complete.*

Proof. By reduction from 3-**SAT**.

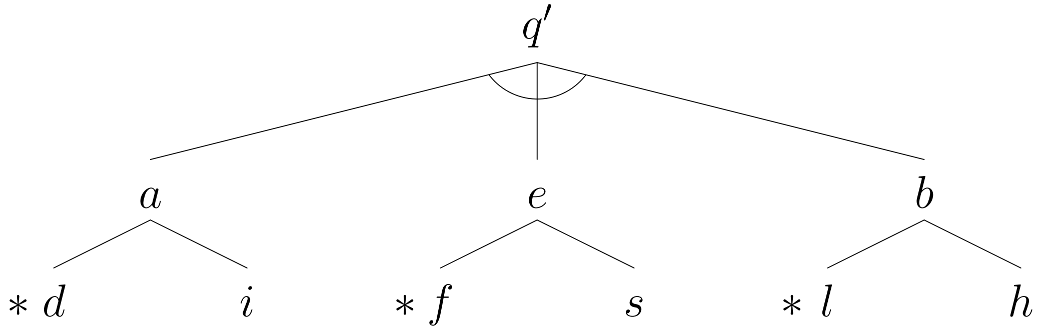
The complexity of deciding coverage is dictated by k, the number of unfoldings-to-discount, but *not* by the size or complexity of the AND/OR tree for $\mathcal{Q}$.

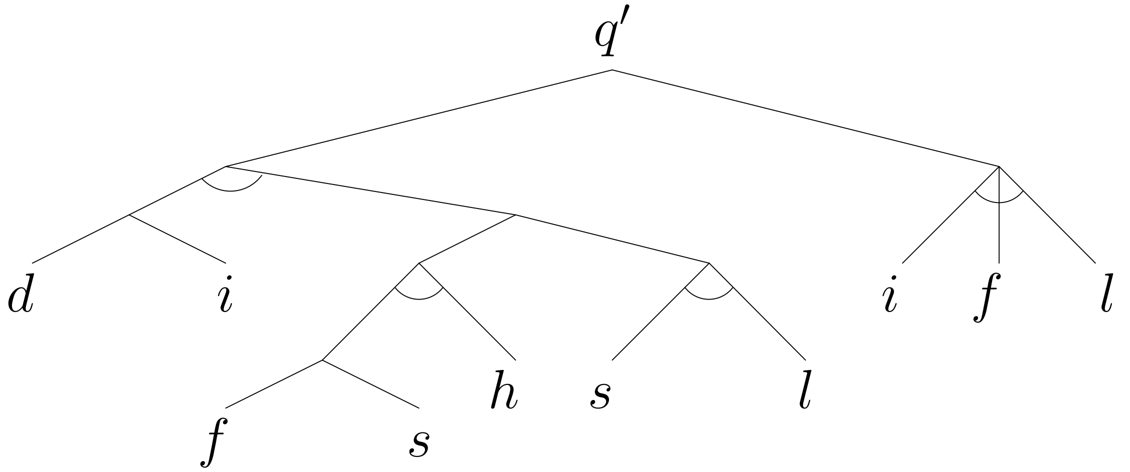
Algebraically Optimal Rewrites

Example

$$q \leftarrow a,\ e,\ b.$$

$$c_2 \leftarrow d,\ f,\ l.$$





Algebraically Optimal Rewrites

Complexity

Define the class *Minimization of Discounted Query* (**MDQ**) as follows. An instance is the triplet of

- a query $\mathcal{Q}$ with a two-level AND/OR tree (a join of unions),
- a collection of unfoldings-to-discount $\mathcal{U}_1$, …, $\mathcal{U}_n$ marked in $\mathcal{Q}$'s AND/OR tree, and
- a positive integer K.

An instance belongs to **MDQ** *iff* there is an AND/OR tree of K or fewer nodes that evaluates $Q \backslash \{\mathcal{U}_1, \ldots, \mathcal{U}_n\}$.

Theorem. Minimization of Discounted Query (**MDQ**) is **NP**-*complete.*

Proof. By reduction from a known **NP**-*hard* problem, minimum order partition into bipartite cliques.

The general problem is in the class Π_2^p. We conjecture it is Π_2^p-*complete.*

Approximation Rewrites

Naive View Disassembly

Strategy.

- Unfold the query in all possible ways.
- Unfold the unfoldings-to-discount in all possible ways.
- Subtract from the query's unfoldings the unfoldings of the unfoldings-to-discount.
- ○ *Refold* the remaining collection somehow.

$$\{d, i\} \times \{f, s\} \times \{h, l\}$$

$$\{a_1, b_1\} \times \ldots \times \{a_{10}, b_{10}\}$$

Problems

- Rewritten query can be exponentially larger than the query.
- Can take exponential time to compute.
- Resulting rewrite contains many redundancies.

Approximation Rewrites

Desired Properties

Let $\mathcal{Q}$ be the query, and $\mathcal{N}$ be the set of unfoldings-to-discount. Find a collection $\mathcal{C}$ of unfoldings of $\mathcal{Q}$ that represents $\mathcal{Q} \backslash \mathcal{N}$.

Collection $\mathcal{C}$ should have the following properties:

1. (*Coverage*) $\mathcal{N} \cup \mathcal{C}$ is a *cover* of $\mathcal{Q}$;
2. (*No overlap*) no two unfoldings in $\mathcal{C}$ should *overlap*;
3. (*Most general*) no unfolding in $\mathcal{C}$ can be refolded (and still preserve the above properties); and
4. (*Parsimonious*) for any $U \in \mathcal{C}$, $(\mathcal{N} \cup \mathcal{C}) - \{U\}$ is not a cover of the view.

Of course, we would like to find $\mathcal{C}$ efficiently.

Approximation Rewrites

The Unfold/Refold Algorithm

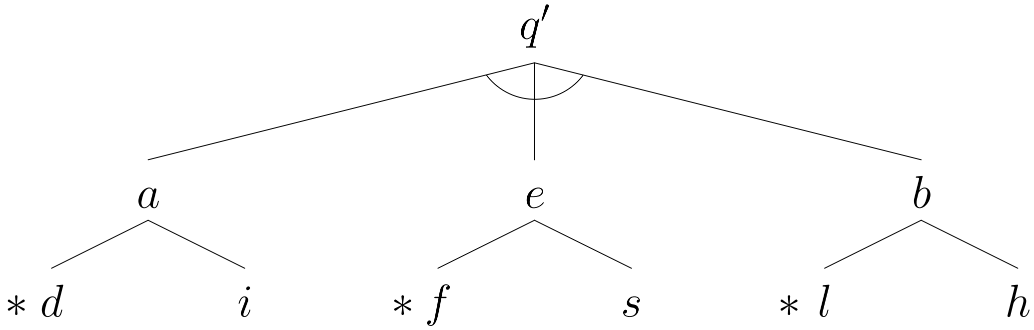
```
$\mathcal{C} := \{\}$
while $\mathit{new\_unfolding}(\mathcal{Q}, \mathcal{N} \cup \mathcal{C}, U)$
    $V := \mathit{refolding}(U, \mathcal{N}, \mathcal{C})$
    $\mathcal{C} := \mathcal{C} \cup \{V\}$
return $\mathit{parsimonious}(\mathcal{C})$
```

Properties

- The *refold* step is computationally inexpensive.
- The *parsimonious* step is also computationally inexpensive.
- The *new_unfolding* step is the computational bottleneck.
  - This is the co-problem of the *coverage* decision.
  - The complexity is dictated by $\mathcal{N} \cup \mathcal{C}$ each loop, *not* by the complexity of $\mathcal{Q}$'s AND/OR tree.

Approximation Rewrites

Example



Unfold/Refold Trace.

1. Start with $\mathcal{C} := \{\}$.
2. New unfolding: $\mathcal{V} = \{i, f, l\}$.
3. Refold $\mathcal{V}$ to $\{i, e, b\}$.

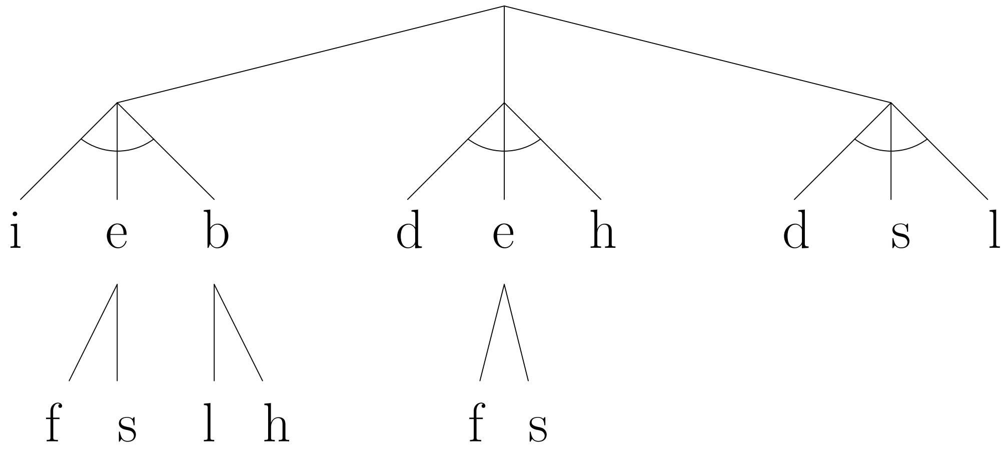
   $\mathcal{C} := \{\{i, e, b\}\}$
4. New unfolding: $\mathcal{V} = \{d, f, h\}$.
5. Refold $\mathcal{V}$ to $\{d, e, h\}$.

   $\mathcal{C} := \{\{i, e, b\}, \{d, e, h\}\}$
6. New unfolding: $\mathcal{V} = \{d, s, l\}$.
7. Cannot refold $\mathcal{V}$.

   $\mathcal{C} := \{\{i, e, b\}, \{d, e, h\}, \{d, s, l\}\}$
8. No new unfoldings possible.

Approximation Rewrites

Example p. 2



IV. Related Work

- **multiple query optimization**
  - View disassembly has the advantage that all unfoldings are from the *same* AND/OR tree.

    MQO must handle any collection of queries.
  - MQO could be used to optimize within view disassembly.
- **query folding**
  - View disassembly extends query folding possibilities.
  - Offers a technique for *remainder queries*.
- **query and algebraic rewrite techniques**
  - Rewrite techniques focus on preserving the query's evaluation.

    View disassembly is for evaluating discounted queries.
  - Rewrite techniques could be used in conjunction with view disassembly.
- **intensional query optimization**
  - IQO looks to apply semantic query optimization to queries with views.
  - This is one possible application for view disassembly.

V. Conclusions and Open Issues

View disassembly offers a new technology to evaluate partially queries that employ views, by "removing" certain unfoldings.

While devising algebraically optimal rewrites for view disassembly is intractable, other sub-optimal strategies are generally tractable.

- **Use of view disassembly for query optimization.**
  - How can it be coupled with existing optimization techniques?
  - With cost-based estimates?
- **When is the use of view disassembly effective?**
  - Which applications benefit from explicit rewrite technology?
  - Could we evaluate discounted queries directly?
- **What are better algorithms for view disassembly?**
  - Can find all *simple unfoldings* covered by the unfoldings-to-discount, and only remove those.
  - How can the unfold/refold algorithm be coupled with other rewrite techniques? With MQO?
- **A yet better understanding of the computational complexity of view disassembly tasks.**
  - What is average case performance of the unfold/refold algorithm?
  - How close to optimal can we achieve, on average?
  - How often do bad cases occur?