Appears in
the Workshop on Deductive Databases and Logic Programming,
held in conjunction with
the Joint International Conference and Symposium on Logic Programming,
Bonn, Germany, September 1996. (ISBN 3-88457-295-4)

A Framework for Intensional Query Optimization

P. Godfrey[1,2]
godfrey@arl.mil

J. Gryz[1]
jarek@cs.umd.edu

[1] Department of Computer Science at the
University of Maryland at College Park
College Park, Maryland, USA

and

[2] U.S. Army Research Laboratory
Adelphi, Maryland, USA

1 Introduction

Advanced database technologies and applications, such as federation and mediation over heterogeneous database sources, object oriented databases and query languages, and data warehousing for decision support, lead to complex, expensive queries. Such systems tend also to have rich semantic information—often in the form of integrity constraints—available. Thus, semantic query optimization techniques may well augment existing syntactic optimizers to reduce greatly the evaluation costs for queries in these systems.

Most semantic optimization techniques which have been proposed in the past apply only to conjunctive queries; they do not consider queries over views which can employ unions. (In Datalog, this is an *intensional* predicate defined via more than one rule.) This limits their utility and applicability.

We propose a framework, to be called *intensional query optimization*, which allows for semantic optimization over queries that employ views (and over views themselves), and which extends the existing semantic optimization techniques to be useful over views.

Section 2 provides background on semantic optimization and query evaluation. Section 3 shows a problem of compatability between preferred query evaluation strategies and existing semantic optimization approaches, and Section 4 provides an example. This motivates our general approach. Section 5 presents a formal framework for *discounted queries*, Section 6 shows how the framework can be used for optimization, and Section 7 considers issues involved in evaluating discounted queries. Section 8 concludes with issues and future work.

This research was supported by grant NSF IRI-9300691.

2 Background

2.1 Optimization via Semantics

Techniques which use semantic information about the database and query have been developed to improve the efficiency of query evaluation. These methods include *semantic query optimization* (SQO) [1, 2, 13, 15, 23], *query folding* (QF) [14, 18] and *semantic data caching* (SDC) [4]. They have applications primarily in distributed and federated databases, although they (in particular, SQO) can also be applied in centralized databases.

SQO uses semantic knowledge in the form of *integrity constraints* to reformulate a database query into an equivalent form that can be evaluated more efficiently. The ultimate win in SQO is the discovery that a query is a *misconception* [9, 7, 11]; that is, it is subsumed by an integrity constraint, hence cannot return any answers, and so does not need be evaluated.

QF refers to the activity of determining whether (and how) a query can be answered using a given set of resources, in most cases, materialized views. Rewriting a query using this technique does not lead, in general, to an equivalent query, but to a query which is contained in the original query [24].

SDC was designed for use in client-server systems. It is assumed that the client maintains a semantic description of the data in cache, instead of maintaining a list of physical pages or tuple identifiers. Query processing makes use of the semantic description to determine what data are locally available in a cache (also called a *probe*), and what data are needed from the server (a *remainder query*). Cached data has the form of a table, which is a result of applying one or more select operations on a single database table.

2.2 Evaluation Strategies

Database queries containing intensional predicates can be evaluated in two different ways, often called *top-down* and *bottom-up* [24]. In the top-down approach, all *extensional unfoldings*[1] of the query are generated, each is evaluated and the answer sets are unioned together to produce the final answer set. This is the type of query processing implemented by Prolog, and tends to be very efficient when only one answer (but not all) to the query is required.

Bottom-up query evaluation does not require evaluating separately all extensional unfoldings of a query. Rather, the intensional atoms (views) in the query are substituted with their definitions and evaluated as specified by these definitions. The relational algebra representation of a query, in which all intensional predicates have been substituted with their definitions, implies through the order of the operations an evaluation plan for the query. Union and join operations are intermixed in the query formula.

[1] Informally, an *unfolding* is a result of applying resolution steps to a query using rules in the intensional part of the database. An *extensional unfolding* does not contain any intensional predicates.

3 Problem

Previous techniques, which we shall call *extensional*,[2] can only be applied to the *conjunctive* query. They do not operate over the *query tree*. If a top-down evaluation is used, all unfoldings of the query are manifested. In this case, these extensional optimization techniques can be applied to each unfolding (a conjunctive query), and thus be interleaved with the evaluation process.

The problem is that the top-down query evaluation is considered impractical. It may introduce redundancies both in the join evaluations done and between retrieved answer sets for different unfoldings.[3] More importantly, it may require evaluating an exponential number (in the number of rules plus query) of unfoldings. For relational databases, the use of nested views can render top-down evaluation strategies intractable.

Bottom-up evaluation strategies do not suffer these problems. For this reason, bottom-up strategies are what most systems use. In deductive database systems, well researched systems such as ADITI [20], CORAL [19], and LDL [3, 17] have committed to bottom-up strategies.

In bottom-up evaluation, the unfoldings of the query are *not* manifested. (Indeed, this is why bottom-up evaluation is usually more efficient.) Unfortunately, this means that extensional optimization techniques are not applicable, except when a technique fortuitously applies directly to the query itself.

The intensional query optimization framework allows the exploitation of information such as misconceptions (*null* unfoldings of a query), cached queries, and materialized views, and can incorporate previous optimization techniques, enabling them to be used in conjunction with bottom-up optimization strategies. The key idea of the approach is to rewrite the query so that certain unfoldings of it (to be defined formally later) are "removed". Such an unfolding may be a misconception, or be easily derived from a materialized view or a previously cached query. These unfoldings and the modified query may then be evaluated in a more efficient way than the original query itself.

4 Motivating Example

Let DB contain five base relations:

- *faculty*(*Name, Department, Rank*),
- *staff*(*Name, Department, Years_of_Employment*),
- *ta*(*Name, Department*),
- *life_ins*(*Name, Provider, Monthly_premium*), and
- *health_plan*(*Name, Provider, Monthly_premium*).

[2] We call these *extensional* since they treat the query as if it were extensional. Any intensional atoms in the query are considered to be materialized beforehand.

[3] The intersection of two answer sets may be substantial, representing redundant evaluation work.

Let there be two rules (views) in the **DB**: the first one defines an *employee* relation (via the union of the *ta* relation and projections from the *faculty* and *staff* relations); the second defines a *benefits* relation (via the union of projections from the *life_ins* and *health_plan* relations).

$$employee(X,Y) \leftarrow faculty(X,Y,Z).$$
$$employee(X,Y) \leftarrow staff(X,Y,Z).$$
$$employee(X,Y) \leftarrow ta(X,Y).$$
$$benefits(X,Z) \leftarrow life\_ins(X,Z,W).$$
$$benefits(X,Z) \leftarrow health\_plan(X,Z,W).$$

Let a query ask for the names of all employees of the physical plant, *p_p*, whose benefits are provided by *hmo*:

$$\mathcal{Q}: \leftarrow employee(X,p\_p),\ benefits(X,hmo).$$

A relational algebra (RA) representation[4] of this query is:

$$\mathcal{Q}: \quad (\pi_X faculty(X,\ p\_p,\ Y) \cup \pi_X staff(X,\ p\_p,\ Z) \cup \pi_X ta(X,\ p\_p)) \bowtie (\pi_X life\_ins(X,\ hmo,\ W) \cup \pi_X health\_plan(X,\ hmo,\ V))$$

Query $\mathcal{Q}$ can also be represented as a parse tree of its RA representation, which is an AND/OR tree, as shown in Figure 1.

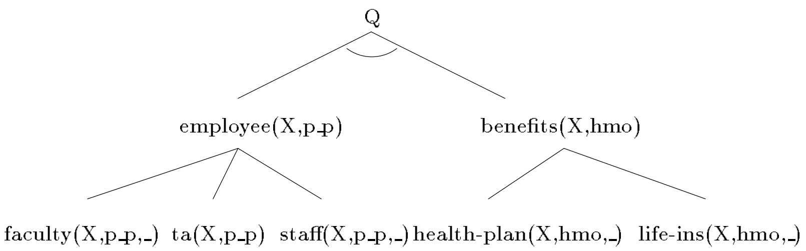

Figure 1: The query tree representation of the original query.

Evaluating the query as specified—by the order of operations in its RA representation—is equivalent to materializing all nodes of the query tree (the AND/OR tree expansion of the query). This requires evaluating the leaves first, then executing the unions to materialize the intermediate nodes *employee*$(X,\ p\_p)$ and *benefits*(X,hmo), and then joining them to materialize the root node, which represents the answer set of the query $\mathcal{Q}$. This type of evaluation is what we call *bottom-up*.

Now assume that the the following formula is an integrity constraint, a cached query, or a materialized view.

$$\mathcal{F}: \leftarrow ta(X,\ Y),\ life\_ins(X,\ Z,\ W).$$

If $\mathcal{F}$ is an integrity constraint, then the join:

$$\mathcal{U}: \quad ta(X,\ p\_p) \bowtie life\_ins(X,\ hmo,\_)$$

[4] We ignore for clarity explicit representation of select operations.

which is computed as part of the query cannot possibly return any answers. Thus this can be eliminated from the query without changing the result. On the other hand, if $\mathcal{F}$ is a cached query or a materialized view, it may still be advantageous to evaluate the expression $ta(X,\ p\_p) \bowtie life\_ins(X,\ hmo,\ \_)$ separately, as it would require executing only a select operation over $\mathcal{F}$. In each of these cases, it is still necessary to "remove" $\mathcal{U}$ from the rest of the query. This leads to the following RA representation of the query:

$$
\begin{array}{ll}
\mathcal{Q}: & ((faculty(X,\ p\_p,\ \_) \cup staff(X,\ p\_p,\ \_)) \bowtie life\_ins(X,\ hmo,\ \_)) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \bigcup \\
 & ((faculty(X,\ p\_p,\ \_) \cup ta(X,\ p\_p) \cup staff(X,\ p\_p,\ \_)) \bowtie health\_plan(X,\ hmo,\ \_)) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \bigcup \\
 & (ta(X,\ p\_p) \bowtie life\_ins(X,\ hmo,\ \_))
\end{array}
$$

The AND/OR tree for the modified query is in fact a query *forest*; it contains three trees as presented in Figure 2.

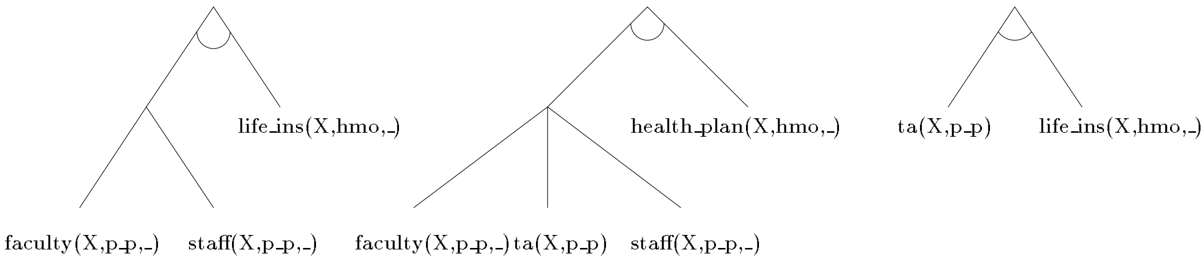

Figure 2: The query tree representation of the optimized query.

The new query does not require evaluating the join $ta(X,\ p\_p) \bowtie life\_ins(X,\ hmo,\ \_)$, although it is equivalent to the original query (that is, returns the same set of answers). The cost of evaluating the join is saved, so it can be considered the gross savings due to the optimization.

5 Discounted Queries

We make a second pass over the concepts presented above, and formalize the notion of *discounting* unfoldings from a query. First, we must give a definition for *query*. We restrict our interest to conjunctive Datalog queries. We shall define a *query* to be a set of atoms—to be interpreted as a conjunction—and the set of variables to be considered as *free* in the query (that is, the *distinguished* variables of the query).

Definition 1. Define a *query set* to be any set of atoms, composed over the predicates, variables, and constants of the database language. A *query* is a pair $\langle \mathcal{Q}, \mathcal{V} \rangle$ in which $\mathcal{Q}$ is a query set, and $\mathcal{V}$ is a set of variables—each of which must appear in $\mathcal{Q}$—to be called the *distinguished variables* of the query.

Next, we define what an *answer* is of a query, and, hence, its *answer set*.

Definition 2. The *answer set* for a query with respect to a database is the set of all its answers[5] with respect to that database. Denote the answer set for query $\langle \mathcal{Q}, \mathcal{V} \rangle$ as $[\![\mathcal{Q}]\!]^{\mathcal{V}}$. (We assume it is always known by context with respect to which database a query's answer set is intended.)

We give a formal definition for an *unfolding* of a query, introduced earlier.

Definition 3. Given query sets $\mathcal{Q}$ and $\mathcal{U}$, call $\mathcal{U}$ a 1-*step unfolding* of query set $\mathcal{Q}$ with respect to database **DB** *iff*, given some $Q_i \in \mathcal{Q}$ and a rule $\langle A \leftarrow B_1, \ldots, B_n . \rangle$ in **DB** such that $Q_i\theta \equiv A\theta$ where θ is a *most general unifier* [16], then

$$\mathcal{U} = \mathcal{Q} - \{Q_i\} \cup \{B_1\theta, \ldots, B_n\theta\}$$

Denote this by $\mathcal{U} \leq^1 \mathcal{Q}$. Call $\mathcal{U}_1$ simply an *unfolding* of $\mathcal{Q}$, written as $\mathcal{U}_1 \leq \mathcal{Q}$, *iff* there is some finite collection of query sets $\mathcal{U}_1, \ldots, \mathcal{U}_k$ such that $\mathcal{U}_1 \leq^1 \ldots \leq^1 \mathcal{U}_k \leq^1 \mathcal{Q}$.

A query set (unfolding) $\mathcal{Q}$ is called *extensional iff*, for every $Q_i \in \mathcal{Q}$, atom Q_i is written with an extensional predicate. Call the query set *intensional* otherwise.

A query can be considered to be equivalent to the union of all its extensional unfoldings.

Definition 4. Given any query set $\mathcal{Q}$, let **unfolds**/*1* denote the set of all extensional unfoldings of $\mathcal{Q}$.

$$\mathbf{unfolds}\,(\mathcal{Q}) \quad = \quad \{\mathcal{U} \mid \mathcal{U} \text{ is an extensional query set and } \mathcal{U} \leq \mathcal{Q}\}$$

We assume of our semantics that

$$[\![\mathcal{Q}]\!]^{\mathcal{V}} \quad = \bigcup_{\mathcal{U} \in \mathbf{unfolds}\,(\mathcal{Q})} [\![\mathcal{U}]\!]^{\mathcal{V}}$$

and that this is another way to define a query's answer set.

We introduce a new query structure to be called a *discounted query*, which can be considered to be the query with some of its unfoldings "removed" (discounted).

Definition 5. Define a new type of query structure to be called a *discounted query set*. Given a query set $\mathcal{Q}$ and unfoldings $\mathcal{U}_1, \ldots, \mathcal{U}_k$ of $\mathcal{Q}$, then $\mathcal{Q} \backslash \{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$ is a discounted query set, to be read as "query set $\mathcal{Q}$ discounting unfoldings $\mathcal{U}_1, \ldots, \mathcal{U}_k$." Define a *discounted query* to be a query with a discounted query set.

Definition 6. Let us extend the domain of **unfolds**/*1* to include discounted query sets, and its definition as follows. Given discounted query set $\mathcal{Q} \backslash \{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$,

$$\mathbf{unfolds}\,(\mathcal{Q} \backslash \{\mathcal{U}_1, \ldots, \mathcal{U}_k\}) \quad = \quad \{\mathcal{R} \mid \mathcal{R} \text{ is extensional, } \mathcal{R} \leq \mathcal{Q}, \text{ but } \mathcal{R} \not\leq \mathcal{U}_i, \text{ for } i \in \{1, \ldots, k\}\}$$

This definition—and the one for *unfoldings*—is semantic, not syntactic, since an unfolding $\mathcal{W}$ of any $\mathcal{U}_i$ does not appear in the collection even when $\mathcal{W}$ could be produced through a chain of resolution steps from $\mathcal{Q}$ which does not pass through any $\mathcal{U}_i$. (Thus there are cases in which our simplistic semantic definition and a syntactic definition for unfoldings would differ.) A

[5] An *answer* is any ground substitution θ over $\mathcal{V}$ such that $\mathbf{DB} \models \exists. \wedge \mathcal{Q}\theta$ [16].

syntactic definition may be more appropriate, but is much more cumbersome. However, the distinction is unimportant for our purposes here.

Let us define the answer set semantics of discounted queries as follows:

$$[\![\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\}]\!]^{\mathcal{V}} \quad = \bigcup_{\mathcal{R}\in\textbf{unfolds}\,(\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\})} [\![\mathcal{R}]\!]^{\mathcal{V}}$$

(Note that this is not the same as relational set minus.)

Clearly, given a query $\langle\mathcal{Q},\mathcal{V}\rangle$, and a collection of unfoldings $\mathcal{U}_1,\ldots,\mathcal{U}_k$ of $\mathcal{Q}$, then

$$[\![\mathcal{Q}]\!]^{\mathcal{V}} \quad = \quad [\![\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\}]\!]^{\mathcal{V}} \cup [\![\mathcal{U}_1]\!]^{\mathcal{V}} \cup \ldots \cup [\![\mathcal{U}_k]\!]^{\mathcal{V}}$$

This provides us a means to treat various unfoldings of the query independent of the query itself.

6 Intensional Optimization

6.1 Costs

For discounted queries to be useful for optimization, clearly the following cost equation must hold:

$$\textbf{cost}\,(\textbf{dis\_eval}\,(\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\},\mathcal{V})) + \sum_{i=1}^{k}\textbf{cost}\,(\textbf{eval}\,(\mathcal{U}_i,\mathcal{V})) + \mathbf{C} < \textbf{cost}\,(\textbf{eval}\,(\mathcal{Q},\mathcal{V})) \quad \mathbf{1)}$$

where **eval**/*2* is a procedure which determines an evaluation strategy for its query (and executes the chosen strategy), **cost**/*1* measures the evaluation time of the query (to manifest a strategy and execute it), and **C** is the cost involved in somehow choosing $\mathcal{U}_1,\ldots,\mathcal{U}_k$. There must be a specialized evaluation strategy for discounted queries, denoted as **dis_eval**/*2*, which is a focus of our research. It must be that discounted queries can be evaluated less expensively than the corresponding query, else this approach will not work. We briefly discuss strategies for **dis_eval**/*2* in the next section.

If we have *multiple query optimization* strategies [12, 22] available—that is, strategies which exploit similarities between queries to be evaluated in batch in order to optimize evaluation—we can weaken this cost equation. Let **mult_eval**/*2* represent such a strategy, which takes a set of queries (and discounted queries) to be evaluated in batch.

$$\textbf{cost}\,(\textbf{mult\_eval}\,(\{\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\},\mathcal{U}_1,\ldots,\mathcal{U}_k\},\mathcal{V})) + \mathbf{C} < \textbf{cost}\,(\textbf{eval}\,(\mathcal{Q},\mathcal{V})) \qquad \mathbf{2)}$$

We shall not consider multiple query optimization here.

6.2 Optimization by Discounting

Earlier we referred to a query as a *misconception* if the semantics of the database guarantees that the query must evaluate to the empty set. We shall call, in general, queries that evaluate empty *null queries*.

If it can be determined that a query is null, this is the ideal optimization: the query does not need to be evaluated because the answer set is already known (to be empty). Likewise, if a collection of null unfoldings can be found for a query, these may be quite useful to optimize evaluation. Let $\mathcal{N}_1, \ldots \mathcal{N}_k$ be null unfoldings of query $\langle \mathcal{Q}, \mathcal{V} \rangle$. Then one can optimize simply if

$$\mathbf{cost}\,(\mathbf{dis\_eval}\,(\mathcal{Q} \backslash \{\mathcal{N}_1, \ldots, \mathcal{N}_k\}, \mathcal{V})) + \mathbf{C} < \mathbf{cost}\,(\mathbf{eval}\,(\mathcal{Q}, \mathcal{V}))$$

The null unfoldings of course cost nothing to evaluate.[6]

Another case when it might be advantageous to separate out a certain unfolding from the query is when the unfolding can be easily answered from cached queries or materialized views. This can be particularly pertinent in a client/server environment, in which network communication costs can dominate the query processing. If the cached queries or materialized views are local, the unfolding can be evaluated locally, and so not involve any network communication overhead. When the unfolding can be completely evaluated from a single cached query or materialized view—that is, it can be computed simply via projections and selects on the cache—the cost is negligible, and could be counted as zero. Thus, this is similar to the null unfolding case. Otherwise, it may be that the unfolding can be computed by the joins of several caches; this is the method that QF explores. In this case, the cost of evaluating the unfolding via caches (and QF) must be weighed against re-evaluating the unfolding, or, perhaps, not discounting it from the query at all.

Existing SQO techniques can provide good optimizations. In the extreme, they can determine when a query is null. Otherwise, they can be used to rewrite a (conjunctive) query by eliminating some of the atoms (because they can be determined to be redundant), or by adding new atoms (when the addition is known to not change the semantics, but may simplify the evaluation by adding an index to a join or such).

The intensional optimization framework offers a way to incorporate these techniques more generally. We can simply replace the **eval**/*2* operations in cost equation **1)** with, say, **sem_eval**/*2*, an evaluation strategy which incorporates semantic optimization techniques. An important question is how to decide *which* unfoldings ought to be discounted from the query so that they may be semantically optimized.

6.3 Identifying Useful Unfoldings

So far we have ignored the first stage of intensional optimization: determining which unfoldings of a query are advantageous to discount; that is, finding the collection $\mathcal{U}_1, \ldots, \mathcal{U}_k$. This is

[6] Of course they do cost something to discover, and this cost, so far, is hidden in **C**.

the cost **C** in all the above cost equations. We do not consider or discuss this first stage of planning, albeit essential, in this paper.

These costs can be non-trivial. However, efficient techniques for such—in particular, identifying the null unfoldings of a query—have been explored [10], and developed within the context of the Carmin project, a cooperative database system [7, 8, 11]. These techniques could be extended to identify when cached queries and materialized views can be used to evaluate certain unfoldings of queries. It is a difficult question how to identify candidate unfoldings for SQO or QF. This is a research question we are currently exploring.

7 Evaluating Discounted Queries

Whether intensional optimization is feasible hinges on whether there exist good evaluation strategies for discounted queries, **dis_eval**/*2*, which are less expensive than evaluating the query itself. For top-down evaluations, it is clear that discounted queries can be evaluated less expensively than the corresponding query: because a top-down evaluation manifests all the unfoldings, those which match discounted unfoldings can be eliminated during the evaluation process.

Unfortunately, the unfoldings of the query are not manifested in a bottom-up evaluation. Thus there is a question of how to "discount" unfoldings during the evaluation, and whether this can be exploited for optimization. We believe that discounted queries can be, in general, evaluated bottom-up less expensively than their corresponding queries [10]. We briefly outline three approaches for a bottom-up **dis_eval**.

One approach is to *rewrite* a discounted query into a collection of conjunctive queries, for which evaluation strategies already exist [10]. Thus, given unfoldings $\mathcal{U}_1,\ldots,\mathcal{U}_k$ to discount from $\mathcal{Q}$, one should find unfoldings $\mathcal{W}_1,\ldots,\mathcal{W}_l$ of $\mathcal{Q}$ such that

$$\textbf{unfolds}\,(\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\}) \quad = \quad \bigcup_{i=1}^{l}\textbf{unfolds}\,(\mathcal{W}_i)$$

Ideally, the collection of $\mathcal{W}_1,\ldots,\mathcal{W}_l$ is *minimal* in the sense that there is no $\mathcal{R} > \mathcal{W}_i$ such that $\mathcal{W}_1,\ldots,\mathcal{W}_{i-1}$, $\mathcal{R}$, $\mathcal{W}_{i+1},\ldots,\mathcal{W}_l$ has the above property, and there are no distinct $\mathcal{W}_i$ and $\mathcal{W}_j$ such that $\mathcal{W}_i > \mathcal{W}_j$. Then we have a natural evaluation strategy for $\mathcal{Q}\backslash\{\mathcal{U}_1,\ldots,\mathcal{U}_k\}$:

$$\bigcup_{i=1}^{l}\textbf{eval}\,(\mathcal{W}_i,\mathcal{V}) \qquad \text{or} \qquad \textbf{mult\_eval}\,(\{\mathcal{W}_1,\ldots,\mathcal{W}_l\},\mathcal{V})$$

The advantage of this approach is that it accounts for all the unfoldings to be discounted "in parallel". The disadvantage is that the collection of $\mathcal{W}_i$'s may be large. In essence, this rewrite approach mixes top-down and bottom-up representation. The query is partially evaluated top-down, just enough so that the unfoldings to discount are explicitly represented. Therefore, this approach is equivalent in the limit to top-down evaluation.

A second type of approach is to modify a bottom-up evaluation strategy to account for the unfoldings to be discounted *during* the evaluation process. The advantage here is that the approach is dynamic: it interleaves evaluation and optimization planning. Thus, cost measures can be judged during the evaluation to decide whether to discount a given unfolding, depending on whether there would be any actual cost savings.

In [10], we describe one such dynamic bottom-up approach which rewrites the query tree dynamically during evaluation. We are exploring a second dynamic approach which would store label information about the unfoldings to discount in the materialized tables during bottom-up evaluation. These labels are then used during joins to ensure that the join expressions corresponding to the unfoldings to discount are never performed. Again, whether to employ a particular discount can be judged at evaluation time.

8 Conclusions

We believe intensional query optimization provides an adequate framework to apply effective semantic optimization such as *semantic query optimization* (SQO [1, 2, 13, 15, 23]), *query folding* (QF [14, 18]), and *semantic data caching* (SDC [4]) to queries in realistic domains. The concept of a *discounted* query—which is the query with some of its possible proof paths "removed"—is introduced for this purpose. We plan to continue to develop these techniques.

Our current and future research efforts include the following.

- *Evaluation of discounted queries.*
  Initial results for the case of null unfoldings appear in [10].
- *Consideration of optimization trade-offs.*
  Discounting cannot uniformly improve the efficiency of evaluation. In general, evaluating a discounted query mixes top-down and bottom-up evaluation. We need to better understand the cost trade-offs, and how best to balance them.
- *Interaction with the syntactic optimizer.*
  Currently, semantic optimization occurs before, and independent of, syntactic optimization. Better performance may be achieved with some degree of coupling of the two.
- *Extend for richer databases.*
  The ramifications of recursion and negation should be considered.
- *Extend to other applications.*
  - Discounting can be used to remove view expressions marked as *secure*, to which the given user should not have access. The query can still be answered, but with the secure parts excised.
  - Parts of a query may be excised because a user profile indicates the user is not interested in certain types of answers.
  - Discounting can optimize queries that use negation: unfoldings of the positive part of the query which imply the negated part need not be evaluated.

We also intend to implement the intensional query optimization framework and to test these optimization techniques over real-world queries and databases.

Acknowledgements

We would much like to thank Jack Minker for his support of this work and for his valuable discussions, input, and insights. We would like to thank the researchers at the U.S. Army Research Laboratory for their avid interest in this work and their support of these efforts. We also would like to thank the reviewers for their comments and suggestions.

References

[1] U. S. Chakravarthy. *Semantic Query Optimization in Deductive Databases.* PhD thesis, University of Maryland, College Park, Maryland 20742, 1985.

[2] U. S. Chakravarthy, J. Grant, and J. Minker. Logic based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[3] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL system prototype. *Transactions on Knowledge and Data Engineering*, 2(1):76–90, Mar. 1990.

[4] S. Dar, M. Franklin, B. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, Bombay, India, Sept. 1996. To appear.
URL: http://www.cs.umd.edu/projects/dimsum/papers/semantic_caching.ps.gz

[5] R. Demolombe and T. Imielinski, editors. *Nonstandard Queries and Nonstandard Answers.* Studies in Logic and Computation 3. Clarendon Press, Oxford, 1994.

[6] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992. Invited paper.

[7] T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. In Demolombe and Imielinski [5], chapter 1, pages 1–40. Appears orginally as [6].
URL: http://karna.cs.umd.edu:3264/papers/GGM92:survey/GGM92:survey.html

[8] T. Gaasterland, P. Godfrey, J. Minker, and L. Novik. A cooperative answering system. In A. Voronkov, editor, *Proceedings of the Logic Programming and Automated Reasoning Conference*, Lecture Notes in Artificial Intelligence 624, pages 478–480. Springer-Verlag, St. Petersburg, Russia, July 1992.
URL: http://karna.cs.umd.edu:3264/papers/GGMN92:lpar/GGMN92:lpar.html

[9] A. Gal and J. Minker. Informative and cooperative answers in databases using integrity constraints. pages 277–300. North Holland, 1988.

[10] P. Godfrey, J. Gryz, and J. Minker. Semantic query optimization for bottom-up evaluation. In Raś and Michalewicz [21], pages 561–571.
URL: http://karna.cs.umd.edu:3264/papers/GGM95:optimization/paper.html

[11] P. Godfrey, J. Minker, and L. Novik. An architecture for a cooperative database system. In W. Litwin and T. Risch, editors, *Proceedings of the First International Conference on Applications of Databases*, Lecture Notes in Computer Science 819, pages 3–24. Springer Verlag, Vadstena, Sweden, June 1994.
URL: http://karna.cs.umd.edu:3264/papers/GMN94:architecture/gmn94.html

[12] J. Grant and J. Minker. On optimizing the evaluation of a set of expressions. *International Journal of Computer and Information Sciences*, 11:179–191, 1982.

[13] L. V. S. Lakshmanan and R. Missaoui. On semantic query optimization in deductive databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 368–375, 1992.

[14] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Data Systems*. ACM, May 1995. To appear.
URL: ftp://twist.db.toronto.edu/pub/papers/pods95LMSS.ps.Z

[15] A. Y. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *Proceedings of Principles of Database Systems*, 1995.

[16] J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation—Artificial Intelligence. Springer-Verlag, Berlin, second edition, 1987.

[17] S. Nagvi and S. Tsur. *A logical language for data and knowledge bases*. Computer Science Press, ACM CR 9004-0267, 1989.

[18] X. Qian. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, pages 48–55, 1996.
URL: http://www.csl.sri.com/~qian/icde96.ps.Z

[19] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL—control, relations and logic. In L.-Y. Yuan, editor, *Proceedings of the 18*th *International Conference on Very Large Databases*, pages 238–250, Vancouver, Canada, Aug. 1992.

[20] K. Ramamohanarao. An implementation overview of the Aditi deductive database system. In *Proc. 3rd International Conference DOOD*, Arizona, 1993. Invited Talk.

[21] Z. W. Raś and M. Michalewicz, editors. *Foundations of Intelligent Systems: Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence 1079, Berlin, June 1996. Springer.

[22] T. Sellis and S. Shosh. On the multiple-query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, 2(2), June 1990.

[23] S. T. Shenoy and Z. M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):344–361, Sept. 1989.

[24] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes I & II*. Principles of Computer Science Series. Computer Science Press, Incorporated, Rockville, Maryland, 1988.