Improvement to the PF Algorithm

Ashish Gupta *
Department of Computer Science
Stanford University
Stanford, CA 94305-2140
agupta@cs.stanford.edu

Inderpal Singh Mumick
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA
mumick@research.att.com

Abstract

The PF algorithm, discussed in [HD92], computes changes to a materialized view when the underlying base relations change. The algorithm considers each changed base relation in turn and propagates the changes to the derived relations. We propose ways of making this propagation process efficient by exploiting stratification [Ull89] of programs and by storing and reusing some of the information derived by the PF algorithm in [HD92].

1 Introduction

The view maintenance strategy outlined by Harrison and Dietrich operates as follows (page 60 [HD92]). Each base relation is considered in turn and all changes to it are propagated to derived relations. $\mathcal{E}_u$ represents the set of updated base relations, and $\Delta\mathcal{E}_u$ is the corresponding delta set. If E is a base relation that is updated, then $\Delta E \in \Delta\mathcal{E}_u$.

Procedure *Update*
begin
  For each $\Delta E \in \Delta\mathcal{E}_u$ do begin
    p =pred_sym(ΔE);
    Rem_tuples = removals(ΔE);
    Add_tuples = additions(ΔE);
    If Rem_tuples $\neq \phi$ then PF(p,Rem_tuples,rem)
    If Add_tuples $\neq \phi$ then PF(p,Rem_tuples,add)
  end;
end $\{update\}$

That is, for each updated base relation, all changes to derived relations are computed; first all deletions are computed and then all the insertions. The procedures "removals" ("additions") identifies the tuples inserted into (deleted from) relation E. For the deleted base tuples in Rem_tuples, the PF algorithm first computes an overestimate of the set of deleted derived tuples. Each potentially deleted tuple in this overestimate, that has not been inferred as deleted before, is rederived using the new base relations. Those potentially deleted tuples that have alternative derivations are not actually deleted. In case a potentially deleted tuple f cannot be rederived, then tuple f is indeed regarded as deleted and the deletion is propagated to derived tuples that depend on

*This work was supported by NSF grants IRI-91-16646 and IRI-90-16358, and ARO DAAL-03-G-0177.

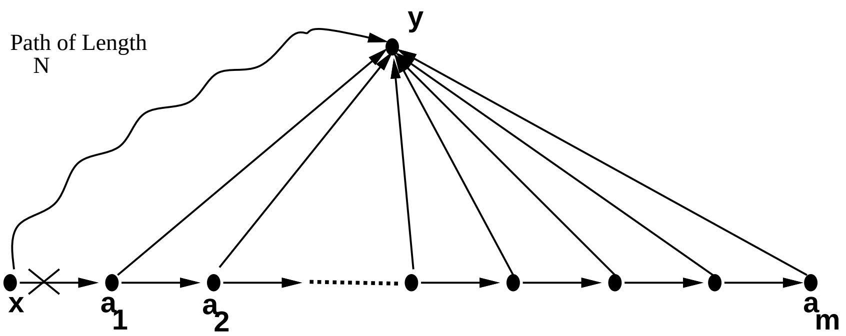

Figure 1: When DRed beats PF

f. If f can be rederived using the new base relations, then the deletion of f is not propagated. The advantage of doing this every time new potentially deleted tuples are derived is that spurious deletions are not propagated.

We make a few observations and describe a few optimizations to the PF algorithm as described in [HD92].

1.1 Use of Memoing/Caching

Memoing [Die87, BR86] can be used to make PF more efficient in the following ways.

Reduce Number of Tuples that are Rederived The following example illustrates that a tuple can be inferred to be potentially deleted multiple times:

EXAMPLE 1.1 Consider the derived view `tc` to be the transitive closure of a given edge predicate `e`. Consider the graph shown in Figure 1.

Say that tuple $\mathtt{e}(x, a_1)$ is deleted (the crossed out edge). The tuples $\mathtt{tc}(x, a_i), 1 \leq i \leq m$ should also be deleted. Whenever PF concludes that a tuple $\mathtt{tc}(x, a_i)$ is deleted, it also concludes that tuple $\mathtt{tc}(x, y)$ is deleted and then derives $\mathtt{tc}(x, y)$ using the alternative derivation of $\mathtt{tc}(x, y)$ of length N. Therefore PF rederives the tuple m times. □

This problem could be overcome by the following observation:

1. Every potentially deleted tuple f that is not deleted by the PF algorithm because of an alternative derivation, has a valid derivation in the new database. Therefore, tuple f need not be rederived more than once.

2. Every potentially deleted tuple f that is deleted by the PF algorithm because of the lack of an alternative derivation, has no valid derivation in the new database. Therefore, tuple f need not be rederived more than once. PF as discussed in [HD92] does this optimization.

Thus every tuple needs to be rederived atmost once. Using a memoing based strategy for execution provides this optimization. A bottom-up magic-sets like approach would need to explicitly store the potentially deleted tuples that have alternative derivations and those potentially deleted tuples that do not. In Example 1.1, tuple $\mathtt{tc}(x, y)$ would be cached as part of the new `tc` relation the first time $\mathtt{tc}(x, y)$ is rederived. In every subsequent iteration that $\mathtt{tc}(x, y)$ is potentially deleted, the cached tuples are checked and $\mathtt{tc}(x, y)$ is not rederived.

Reducing Length of Rederivations The PF algorithm rederives potentially delted tuples starting from base tuples. This feature is useful as it eliminates the need for storing intermediate derived relations. However, the feature also leads to inefficiency because all rederivations start from base relations. Using memoed rederived intermediate tuples in subsequent rederivations, makes the rederivation process more efficient. The optimization retains the useful property of not having to materialize intermediate derived relations.

1.2 Using Stratification to Rederive Fewer Tuples

The PF algorithm is defined for stratified programs [Ull89] (including negation). Stratification can be exploited to make PF more efficient as follows:

Avoid rederiving inserted tuples Currently, the PF algorithm computes an overestimate of the inserted tuples and rederives each potentially inserted tuple in the new database state. However, the rederivation process is avoidable for potentially inserted tuples (unlike potentially deleted tuples), because reinserting an existing tuple into a set is a "safe" operation (assuming that duplicates are not an issue). The following examples illustrate why PF cannot avoid rederiving potentially inserted tuples.

EXAMPLE 1.2 Consider the single rule program that defines predicate p:

$p(X)$:- $f(X)$ & $\neg e(X)$.

Predicates e and f are in stratum 1 and predicate p is in stratum 2. Let the initial database contain the tuples $e(1), f(1)$. Say we then delete both tuples $e(1)$ and $f(1)$. If we first consider the changes to predicate e and propagate them to the top, we infer that tuple $p(1)$ is potentially inserted (because $f(1)$ is true in the old database state). However, $f(1)$ is subsequently deleted and therefore rederiving $p(1)$ in the new database state, becomes necessary. Note that when the deletion of $f(1)$ is propagated, $p(1)$ would not be inferred as a potentially deleted tuple because $p(1)$ was not even true in the old database state; rederivation is the only way of avoiding the insertion. □

EXAMPLE 1.3 Consider a graph that consists of two kinds of edges: f, g. Let the `fg` relation be defined as follows:

`fg`(X, Y) :- $f(X, Z)$ & $g(Z, Y)$.

Given the relations $F = \{13\}, G = \{34\}$, relation `fg` has the tuple $\{14\}$. Let the tuple $f(1,3)$ be deleted and let tuples $g(3,5), \ldots, g(3,9)$ be added. If the relation g is considered before f, we add the spurious `fg` tuples $\{15, \ldots, 19\}$. As in Example 1.2, if the potentially inserted tuples are not rederived in the new database state, we will end up with an incorrect materialized view. □

The problem illustrated above would not arise if updates are propagated stratum by stratum, and for each stratum, insertions are propagated after all deletions are propagated. That is, the changes made to a predicate in stratum i are propagated only to stratum $i+1$ (and no higher). Once all the changes made to predicates in stratum i are propagated to stratum $i+1$, then the changes made to predicates in stratum $i+1$ are propagated further. Within each stratum, all the deletions are propagated before any of the insertions are propagated. Such a strategy would ensure that the status of all tuples in stratum $i+1$ is resolved before changes to stratum $i+1$ are propagated (proof of correctness is similar to the proof of the DRed algorithm in [GMS93]). Consequently, potentially inserted tuples will never be rederived.

Note that we do not need to change the way PF rederives tuples (rederivation can still start from base relations or use memoed tuples). Thus all advantages of PF and the other optimizations are still retained; in addition potentially inserted tuples need not be rederived.

1.3 Summary

Besides providing improvements for the PF algorithm described in [HD92], the above observations are a basis for integrating the PF algorithm and the DRed algorithm described in [GMS93] to obtain an algorithm that does better than either of the two original algorithms.

References

[BR86] Francois Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proceedings of ACM SIGMOD 1986 International Conference on Management of Data*, pages 16–52, Washington D. C., may 1986.

[Die87] S. W. Dietrich. Extension tables: Memo relations in logic programming. In *Proceedings of the Fourth IEEE Symposium on Logic Programming*, pages 264–272, 1987.

[GMS93] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *To appear in SIGMOD*, 1993.

[HD92] John V. Harrison and Suzanne Dietrich. Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach. In *Workshop on Deductive Databases, JICSLP 1992*, pages 56–65, 1992.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, New York, 1989.