Query Flocks: A Generalization of Association-Rule Mining

CS 33510: Data Mining

Svetlozar Nestorov

Outline

- So far, we studied how to mine association rules from market-basket data.
  - AIS, Apriori, AprioriTID
  - Different implementations
- Can we generalize these techniques to arbitrary relations?
- Query Flocks!
- Market baskets as a query flock.
- Query flock plans.
- Searching for the optimal plan.

Problem Motivation

- Large amounts of data
  - stored in relational DBMS (data marts, data warehouses)
- Need to perform complex data analysis: ad-hoc, on-line data mining
- Currently, specialized, efficient algorithms for a small class of problems
  - at best, loosely coupled with RDBMS

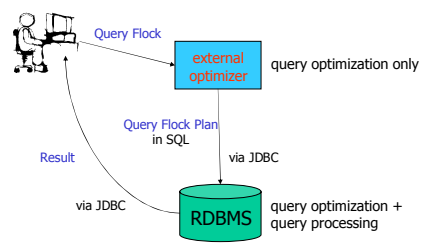
Query Flocks

- Programming tool that enables *efficient,* ad-hoc, on-line data mining
- With conventional RDBMS
  - transform complex query into a sequence of simpler, *optimizable* queries
- As a part of next-generation optimizers
  - new query optimization technique, e. g., generalization of the 'a priori' technique

Query Flocks Features

- Tightly-coupled integration
  - all query processing performed at DBMS
  - external query optimization
  - full use of DBMS features
    - recovery, concurrency control
- Main challenge: performance

Tightly-Coupled Architecture



Query Flocks Definition

- Parameterized query with implicit aggregation and a filter condition
  - nonrecursive datalog program with parameters

```
answer(B) :- baskets(B,$1)
```

  - arithmetic condition with aggregate functions

```
COUNT(answer.B) >= 20
```


Query Flock Example

- Relation `exhibits(Patient, Symptom)`
- Query Flock (about $1 and $2):

Query:

```
answer(P):- exhibits(P,$1) AND
      exhibits(P,$2) AND $1<$2
```

Filter:

```
COUNT(answer.P) >= c
```

(support)

Query Flocks Explained

- A query flock is about its parameters
- Generate-and-test paradigm:
  - pick parameters: cough and fever
  - evaluate query part: answer relation
  - if filter condition is satisfied add (cough, fever) to query flock result
- Why the name "flocks"?

Query Flock Result

- *Relation* over its *parameters* that meet the filter condition

$1	$2
cough	fever
fever	headache
headache	insomnia
…	…

Market Basket Problem

- Supermarket checkout data
- Find all pairs of items *frequently* bought together (in the same basket)
- Success based on
  - appropriateness of purpose
  - new optimization tricks: 'a priori'

Market Baskets as a Query Flock

- Relation `baskets(BID,Item)`
- Query Flock:

Query:

```
answer(B) :- baskets(B,$1)
          AND baskets(B,$2)
```

Filter:

```
COUNT(answer.B) >= c
```

Market Baskets in SQL

- Not optimized effectively in RDBMS

```
SELECT B1.Item, B2.Item
FROM   baskets B1, baskets B2
WHERE  B1.Item < B2.Item AND
       B1.BID = B2.BID
GROUP BY B1.Item, B2.Item
HAVING c <= COUNT(DISTINCT B1.BID)
```


The 'A Priori' Technique

- A *pair* of items is frequent only if *each* item is frequent
- Reduce the number of potentially frequent pairs by first finding all frequent items

```
INSERT INTO ok
SELECT     Item
FROM       baskets
GROUP BY   Item
HAVING c <= COUNT(DISTINCT BID)
```


Market Baskets with 'A Priori'

```
SELECT B1.item, B2.item
FROM Baskets B1,Baskets B2,
       ok T1,ok T2
WHERE  B1.item < B2.item
  AND  B1.item = T1.item
  AND  B2.item = T2.item
  AND  B1.BID = B2.BID
GROUP BY B1.item, B2.item
HAVING 20 <= COUNT(DISTICT B1.BID)
```


'A Priori' for Query Flocks

- Create *auxiliary* relations, as results of query flocks, that limit the values for some subsets of the parameters
  - *safe* subqueries of the original query; same filter

Query:
```
answer(B) :- baskets(B,$1)
             AND baskets(B,$2)
```

Filter:
```
COUNT(answer.B) >= c
```


Larger Example: Side Effects

- Relations

```
diagnoses(Patient, Disease)
exhibits(Patient, Symptom)
treatments(Patient, Medicine)
causes(Disease, Symptoms)
```

- Find possible side effects of medicines

Side-Effect Query Flock

Query:
```
answer(P) :- diagnoses(P,D)
       AND exhibits(P,$s)
       AND treatments(P,$m)
       AND NOT causes(D,$s)
```

Filter:
```
COUNT(answer.P) >= 20
```

Some Safe Subqueries

- `answer(P) :- treatments(P,$m)`
- `answer(P) :- exhibits(P,$s)`
- `answer(P) :- diagnoses(P,D)`
  `AND exhibits(P,$s)`
  `AND NOT causes(D,$s)`
- `answer(P) :- exhibits(P,$s)`
  `AND treatments(P,$m)`

Side Effects in SQL

```
select E.Symptom, T.Medicine
from diagnoses D, exhibits E, treatments T
where D.Patient = E.Patient
and D.Patient = T.Patient
and E.Symptom not in
   (select C.Symptom
    from causes C
    where C.Disease = D.Disease)
having count (distinct P) >= 20

group by E.Symptom, T.Medicine
```


Processing Flocks Efficiently

- Direct translation is too slow.
- Solution: Query Flock Plans
  - serve as an external optimizer.
  - transform complex flock into an *equivalent sequence* of simpler steps.
  - each step can be processed efficiently at the underlying DBMS.
  - all data processing done at DBMS.

Query Flock Plan Definition

- A sequence of query flocks
- Each flock defines an *auxiliary* relation
- Each flock has the same filter
- Each flock is derived from the original by
  - adding zero or more auxiliary relations
  - choosing *safe* subquery
- Final step: original query + auxiliary relations

Query Flock Plan: Limit parameters

- Step 1: Create auxiliary relation okM
  Query: `answer(P) :- treatments(P,$m)`
  Filter: `COUNT(answer.P) >= 20`
- Step 2: Create auxiliary relation okS1
  Query: `answer(P) :- exhibits(P,$s)`
  Filter: `COUNT(answer.P) >= 20`

Query Flock Plan: Limit parameters

- Step 3: Create auxiliary relation okS2
  Query: `answer(P):- okS1($s)`
  `AND diagnoses(P,D)`
  `AND exhibits(P,$s)`
  `AND NOT causes(D,$s)`
  Filter: `COUNT(answer.P) >= 20`

Query Flock Plan: Final Step

- Step 4: Final query appears to be harder but okS2 and okM can reduce the size of the intermediate results during the join

Query:
```
answer(P):- diagnoses(P,D)
    AND okM($m) AND okS2($s)
    AND exhibits(P,$s)
    AND treatments(P,$m)
    AND NOT causes(D,$s)
```

Filter: `COUNT(answer.P) >= 20`

In Reality…

- Current DB optimizers not nearly smart enough.
- The shapes of the query plans are limited.
- Solution: do it yourself!
- Break up the queries even further.

Query Flock Plans Improved

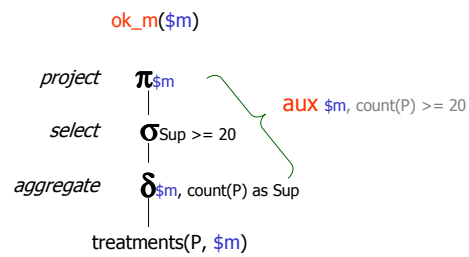
- Two types of steps:
  - limit parameters (auxiliary relations)

```
ok_m($m):
answer(P) :- treatments(P,$m)
COUNT(answer.P) >= 20
```

  - reduce base relations

```
t_1(P,$m) :- treatments(P,$m)
             AND ok_m($m)
```

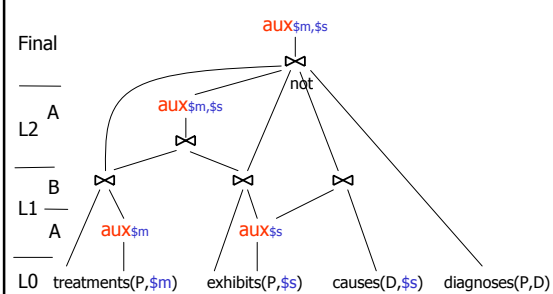

Auxiliary Relations



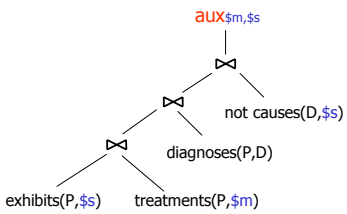
Generating Flock Plans

- Levelwise, rule-based algorithm
  - at each level k, two phases
    - A: materialize auxiliary relations (sets of k params)
    - B: reduce base relations
- Heuristics employed
  - take advantage of symmetry
  - smallest safe subqueries

Example Query Flock Plan



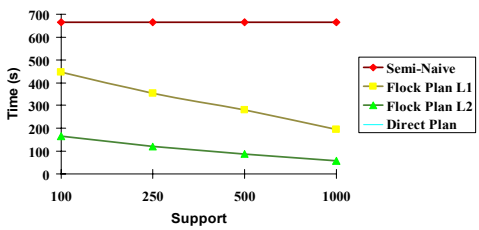
Direct Plan (in Oracle)



Why Is It Worthwhile?

- Flock plan appears more complex: 7 queries, final join of 5 relations, but:
  - first 6 queries are simple
  - final join is faster
    - smaller relations (base relation reductions)
    - smaller intermediate results (auxiliary relations)

Performance: Medical Data



Association-Rule Flavors

- Quantitative association rules
- Generalized association rules
- Multi-level association rules
- Qualified association rules
- Generalized qualified association rules

Flocks and Stars

- Most data warehouses are built using star schemas (dimensional modeling.)
- Qualified association rules take advantage of all dimensions (not just products)
- Can be expressed as query flocks!
- Example

Conclusions

- Tightly-coupled integration of data mining and DBMS is possible
  - external query optimization
- Leverages database technology
- Enables ad-hoc, on-line data mining