Logic Programming
Query Optimization

Michael Genesereth
Computer Science Department
Stanford University

Semantic Equivalence

Two queries are semantically equivalent if and only if they produce identical results for every dataset.

Query 1:

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
```

Query 2:

```
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

Computational Disparity

Different semantically equivalent queries may have dramatically different computational properties.

Query 1: O(n^4)

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
```

Query 2: O(n^3)

```
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

Optimization

Types of Reformulation

Logical - Deleting and/or rearranging subgoals and rules

Conceptual - changing vocabulary

Examples of Logical Reformulation

Dropping Rules

Dropping Subgoals

Subgoal Ordering

Reformulation

Reification

Relationalization

Relational Reformulation

Rule Removal

Rule Removal

Original Rules

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(X) :- p(X,Y) & q(Y)
```

Reformulation

```
goal(X) :- p(X,Y) & q(Y)
```

Subsumption

A rule *r1 subsumes* a rule *r2* if and only if it is possible to replace some or all of the variables of *r1* in such a way that the heads are the same and all of subgoals of *r1* are members of the body of *r2*.

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(X) :- p(X,Y) & q(Y)
```

Here, the second rule subsumes the first. We just replace `X` in the second rule by itself and replace `Y` by `b`, with the following result.

```
goal(X) :- p(X,b) & q(b)
```

Subsumption Technique

Accept rule 1 and rule 2 as inputs where (a) the heads are identical and (b) neither rule contains any negations.

(1) Create a substitution in which each variable in *rule 2* is bound to a distinct, new symbol.

(2) Create a dataset consisting of the subgoals of *rule 2* where all variables are replaced by these bindings.

(3) Substitute the bindings for the head variables in rule 1.

(4) Evaluate this modified rule on the dataset created in (2). If there are answers, then rule 1 subsumes rule 2. If not, then rule 1 does *not* subsume rule 2.

Example

Inputs

```
goal(X) :- p(X,Y) & q(Y)
goal(X) :- p(X,b) & q(b) & r(Z)
```

Substitution: {`X←c1`, `Z←c3`}

Dataset: {`p(c1,b)`, `q(b)`, `r(c3)`}

Evaluate: `p(c1,Y) & q(Y)`
Result: {`X←c1`, `Y←b`}

The first rule *does* subsume the second.

Example

Inputs

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(X) :- p(X,Y) & q(Y)
```

Substitution: {X←c1, Y←c2}

Dataset: {p(c1,c2),q(c2)}

Evaluate: `p(c1,b) & q(b) & r(Z)`
Result: failure

The first rule *does not* subsume the second.

Rule Removal Technique

Compare every rule to every other rule (quadratic). If one rule subsumes another, okay to dropped the subsumed rule.

The technique is *sound* in that it is guaranteed to produce equivalent query.

In *the absence of any constraints* on the datasets to which the rules are applied, it is also guaranteed to be *complete* in that all surviving rules are needed for some dataset.

In *the face of constraints*, it may be possible to drop rules that are not detected by this method, i.e. *not complete*.

Extensions

If heads are not identical, they can sometimes be made identical by consistently replacing variables while avoiding clashes.

Original rules:

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(U) :- p(U,V) & q(V)
```

Equivalent rules:

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(X) :- p(X,V) & q(V)
```

There are extensions for dealing with rules involving negations and for dealing with constraints.

Subgoal Removal

Subgoal Removal

Original Rule:

```
goal(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

Equivalent Reformulation:

```
goal(X,Y) :- p(X,Y) & q(Y)
```

Subgoal Removal Technique

Accept query rule as input.

(1) Create new query with rule head and empty body.

(2) Create a substitution with an entry for each head variable and a distinct constant.

(3) For *each* subgoal, (a) augment the substitution with bindings for the remaining variables in the *other* subgoals. (b) Create a dataset by replacing variables in those subgoals with their bindings. (c) Replace variables in chosen subgoal with bindings, leaving unbound variables as is. (d) Evaluate query. If fail, add to body of new query. Otherwise, drop.

Output the new query.

Sketch of Subgoal Removal Technique

Query: `goal(X,Y) :- p(X,Y) & q(Y) & q(Z)`
Initial binding list - {`X`←`c1`, `Y`←`c2`}

Subgoal: `p(X,Y)`. Binding list: {`X`←`c1`, `Y`←`c2`, `Z`←`c3`}.
Create dataset {`q(c2)`, `q(c3)`}.
Evaluate `p(c1,c2)`. Fail. So add `p(X,Y)` to new query.

Subgoal: `q(Y)`. Binding list: {`X`←`c1`, `Y`←`c2`, `Z`←`c3`}.
Create dataset {`p(c1,c2)`, `q(c3)`}.
Evaluate `q(c2)`. Fail. So add `q(Y)` to new query.

Subgoal: `q(Z)`. Binding list: {`X`←`c1`, `Y`←`c2`}.
Create dataset {`p(c1,c2)`, `q(c2)`}.
Evaluate `q(Z)`. Succeed. So drop.

Output: `goal(X,Y) :- p(X,Y) & q(Y)`

Soundness

Original Rule

```
goal(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

Reformulation

```
goal(X,Y) :- p(X,Y) & q(Y)
```

Incompleteness

Original Rule

```
goal(X,Y) :- p(X,Y) & q(X,Y) & p(X,Z) & q(X,Z)
```

Reformulation

```
goal(X,Y) :- p(X,Y) & q(X,Y)
```

Subgoal Ordering

Subgoal Ordering

Original Rule

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
```

Reformulation

```
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

Analysis

Original Rule

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
```

$$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + n*(n\text{^}2 + 2n)) =$$
$$n\text{^}4 + 3n\text{^}3 + 3n\text{^}2 + 2n$$

Reformulation

```
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

$$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + 1*(n\text{^}2 + 2n)) =$$
$$2n\text{^}3 + 5n\text{^}2 + 2n$$

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound, add to new query and remove from original query. If none found, remove first subgoal, add to new query, and repeat.

Output the new query.

Example:

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

Example

Example

```
  SEND
+MORE
-----
MONEY
```

One Solution

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)

goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) &
  S!=0 & E!=S & N!=S & N!=E & D!=S & D!=E & D!=N &
  M!=0 & M!=S & M!=E & M!=N & M!=D &
  O!=S & O!=E & O!=N & O!=D & O!=M &
  R!=S & R!=E & R!=N & R!=D & R!=M & R!=O &
  Y!=S & Y!=E & Y!=N & Y!=D & Y!=M & Y!=O & Y!=R
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
```

Computational Analysis

Data

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) & ...
```

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 cases

111,111,110 unifications

Running time ~ minutes

Another Solution

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R &
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
```

Computational Analysis

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R & ...
```

Analysis

10x9x8x7x6x5x4x3 = 1,814,400 cases

5,989,558 / 7,921,010 unifications

Interpreted ~ 40 seconds Compiled ~ 4 seconds

Computational Analysis

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(D) &
  digit(E) & mutex(E,D) &
  digit(Y) & mutex(Y,D,E) &
  digit(S) & distinct(S,0) & mutex(S,D,E,Y) &
  digit(N) & mutex(N,D,E,Y,S) &
  digit(M) & distinct(M,0) & mutex(M,D,E,Y,S,N) &
  digit(O) & mutex(O,D,E,Y,S,N,M) &
  digit(R) & mutex(R,D,E,Y,S,N,M,O) & ...
```

Analysis

4,002,398 / 7,921,010 unifications

Interpreted < 30 seconds

Computational Analysis

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(D) &
  digit(E) & mutex(E,D) &
  digit(Y) & mutex(Y,D,E) &
             evaluate(remainder(plus(D,E),10),Y) &
  digit(S) & distinct(S,0) & mutex(S,D,E,Y) &
  digit(N) & mutex(N,D,E,Y,S) &
  digit(M) & distinct(M,0) & mutex(M,D,E,Y,S,N) &
  digit(O) & mutex(O,D,E,Y,S,N,M) &
  digit(R) & mutex(R,D,E,Y,S,N,M,O) & ...
```

Analysis

438,728 / 7,921,010 unifications

Interpreted ~ 4 seconds

Computational Analysis

Data

```
digit(9)
digit(5)
digit(6)
digit(7)
digit(1)
digit(0)
digit(8)
digit(2)
digit(3)
digit(4)
```

Analysis

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = **36** / 7,921,010 unifications

Interpreted ~ 0 seconds

Computational Analysis

Data

```
digit(9)
digit(5)
digit(6)
digit(7)
digit(1)
digit(0)
digit(8)
digit(2)
digit(3)
digit(4)
```

Analysis

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = **36** / 7,921,010 unifications

Interpreted ~ 0 seconds

Narrow and Wide Relations

Triples

Represent wide relations as collections of binary relations.

Wide Relation:

```
student(Student,Department,Advisor,Year)
```

Binary Relations:

```
student.major(Student,Department)
student.advisor(Student,Faculty)
student.year(Student,Year)
```

Always works when there is a field of the wide relation (called the **key**) that uniquely specifies the values of the other elements. If none exists, possible to create one.

Abstract Example

Wide Relation:

```
p(a,d,e)
p(b,d,e)
p(c,d,e)
```

Triples:

```
p1(a,d)     p2(a,e)
p1(b,d)     p2(b,e)
p1(c,d)     p2(c,e)
```

Analysis

Wide Relation:

```
p(a,d,e)
p(b,d,e)
p(c,d,e)
```

Query: `goal(X) :- p(X,d,e)`
Cost without indexing: 3 **Cost with indexing:** 3

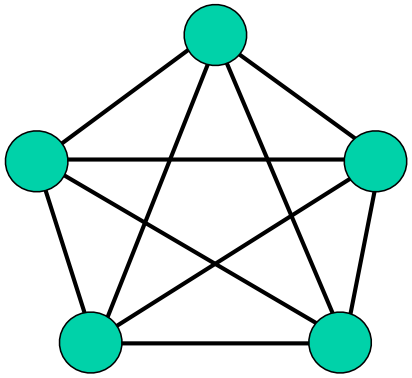
Triples:

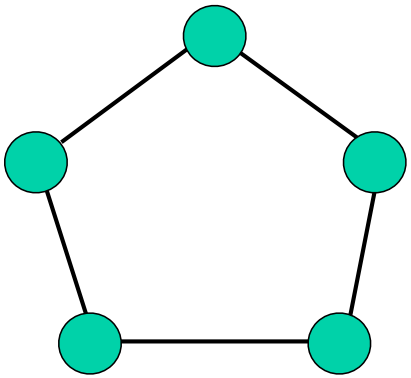
```
p1(a,d)    p2(a,e)
p1(b,d)    p2(b,e)
p1(c,d)    p2(c,e)
```

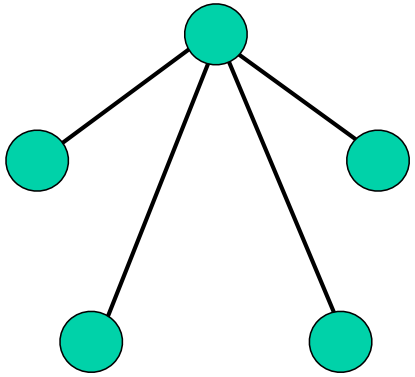
Query: `goal(X) :- p1(X,d) & p2(X,e)`
Cost without indexing: 24 **Cost with indexing:** 9

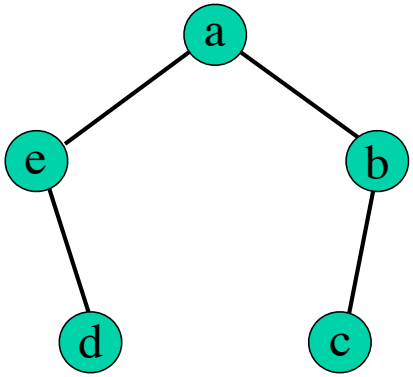
Minimal Spanning Trees

Social Isolation Cells









Representation

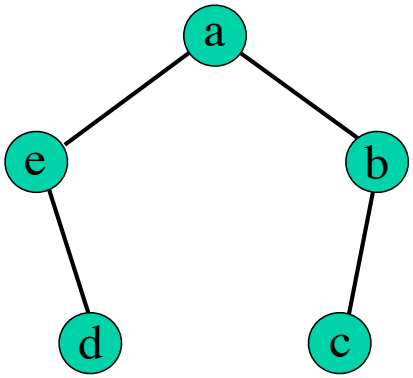
Vocaulary:

People - `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `j`, ...

Interaction - `r`/2

Example:

```
r(a,b)
r(a,e)
r(b,a)
r(b,c)
r(c,b)
r(d,e)
r(e,a)
r(e,d)
```



NB: Possible to represent undirected with only one factoid per arc rather than two, but we will ignore that for now.

Computational Analysis

Are two people are in the same cell?

```
goal(a,e) :- r(a,e)
goal(a,e) :- r(a,Y) & r(Y,e)
goal(a,e) :- r(a,Y1) & r(Y1,Y2) & r(Y2,e)
goal(a,e) :- r(a,Y1) & r(Y1,Y2) & r(Y2,Y3) & r(Y3,e)
```

Number of unifications for `goal(a,e)` (with indexing):

$$8$$

$$8 + 4*8 = 40$$

$$8 + 4*(8 + 4*8)) = 168$$

$$8 + 4*(8 + 4*(8 + 4*8)) = 680$$

Total: 896

Alternative Representation

Precompute and store the transitive closure of `r`

```
r(a,b)    r(b,a)    r(c,a)    r(d,a)    r(e,a)
r(a,c)    r(b,c)    r(c,b)    r(d,b)    r(e,b)
r(a,d)    r(b,d)    r(c,d)    r(d,c)    r(e,c)
r(a,e)    r(b,e)    r(c,e)    r(d,e)    r(e,d)
```

Are two people are in the same cell?

```
goal(a,e) :- r(a,e)
```

Number of unifications for `goal(a,e)` (with indexing):

8

Number of factoids for n objects:

$8*n$

MST Representation

Assign a number for each group and store with people

```
r(a,1)    r(f,2)    ...
r(b,1)    r(g,2)    ...
r(c,1)    r(h,2)    ...
r(d,1)    r(i,2)    ...
r(e,1)    r(j,2)    ...
```

Are two people are in the same cell?

```
goal(a,e) :- r(a,N) & r(e,N)
```

Number of unifications for `goal(a,e)` (with indexing):

2

Number of factoids for n objects:

n

Not Secure — logicprogramming.stanford.edu

File Dataset Channel Ruleset Operation Settings

Lambda

Save Revert Sort

```
p(a,b)
p(a,c)
p(a,d)
p(a,e)
p(b,a)
p(b,c)
p(b,d)
p(b,e)
p(c,a)
p(c,b)
p(c,d)
p(c,e)
p(d,a)
p(d,b)
p(d,c)
p(d,e)
p(e,a)
p(e,b)
p(e,c)
p(e,d)
```

Query

Pattern `goal(a,e)`

Query `p(a,Y1) & p(Y1,Y2) & p(Y2,Y3) & p(Y3,e) & false`

Show Next 100 result(s) Autorefresh

680 unification(s)

Not Secure — logicprogramming.stanford.edu

File Dataset Channel Ruleset Operation Settings

Lambda

Save Revert Sort

```
p(a,1)
p(b,1)
p(c,1)
p(d,1)
p(e,1)
```

Query

Pattern `goal(a,e)`

Query `p(a,N) & p(e,N)`

Show Next 100 result(s) Autorefresh

2 unification(s)

```
goal(a,e)
```