Planning to Gather Information

Chung T. Kwok & Daniel S. Weld
CSE, University of Washington

Occam:

- a query planning algorithm that determines the best way to integrate data from different sources.
- It seeks the *simplest plans* that gather all information requested by the user.

The Problem

- The exponential growth of Internet
  - documents
  - databases
  - services
- Almost any type of information is available *somewhere*, but most users can¡t find it!
- Even expert users waste copious time and effort searching for *appropriate information sources.*

A Simple Example

Problem:

- Find out the names of all people in an office.
- Assumption: no such database exists

Information sources:

> `finger  user@host`
  returns name of person with the specified email
> `userid-room 021`
  returns email addresses of all occupants in an office

A Solution

- Issue the `userid-room` command to get a list of email addresses
- Run `finger` on each of the email addresses returned

- Occam reasons about the capabilities of information sources.
- Occam generates multiple plans in order to gather as much information as possible.

Action Language

- Knowledge preconditions only
- No causal effects
- No sibling-subgoal interactions
- Model the *information* instead of world *state*
- A single unified world model, independent from the conceptualization used by the information sources.
- Highly specialized planning algorithm

World Model

- A single, unified relational database schema
  - email(F, L, E)
  - office(F, L, O)

- **Occam** is typed, for example
  - email(F, L, E)
    - ¡ F and L are of type *name*,
    - ¡ E is of type *email*

Information-Producing Sites

Represent information sources by modeling the type of queries they can handle.

query output <-> relations in the world model

A site may be described by multiple operators.

operator: head => body

- head: name of the operator + arguments
- body: conjunction of atomic formulae whose predicate symbols denote relations in the world model.
- op(X_1,¡ ,X_n) => r_1(¡ ,X_i,¡) ... r_m(¡ ,X_j,¡)

Binding Pattern

Each variable can be annotated with a binding pattern (denoted with $) to indicate that the variable much be bound, e.g.

The Unix `finger` command

finger(F, L, $E, O, Ph) => email(F, L, E) office(F, L, O) phone(O, Ph)

- bound variable: $E
- free arguments: F, L, O, Ph

Operator Representation

op(X_1, ..., X_n) =>

rel_1(..., X_i, ...) ... rel_m(..., X_j, ...)

when *op* is executed it will return some number of tuples of data

each tuple may be thought of as an assignment of values to the head¡s arguments X_1, ..., X_n

Operations are not guaranteed to return all tuples, since most information sources are incomplete.

Operator: Examples

finger(F, L, $E, O, Ph) and E is bound to ¡sam@cs¡

- (¡Sam¡, ¡Smith¡, ¡sam@cs¡, ¡501¡, ¡542-8907¡)
- (¡Sam¡, ¡Smith¡, ¡sam@cs¡, ¡501¡, ¡542-8908¡)

userid-room($O, E) =>

office(F, L, O) email(F, L, E)

- returns tuple (¡501¡, ¡sam@cs¡)
- Interpretation of *unbound* variables:
- exists F, L such that
- office(F, L, ¡501¡) email(F, L, ¡sam@cs¡)

Query Representation

Any tuple satisfying the body satisfied the query

For example:

query-for-first-names($O, F) <= office(F, L, O)

- Variable O must be bound.
- The query requests a set of values for F.

if Joe Researcher and Jane Goodhacker are the occupants of office 429, then the tuples (¡429¡, ¡Joe¡) and (¡429¡, ¡Jane) are possible answers for *query-for-first-name*(¡429¡, F)

Plans

query-for-first-name(¡429¡, F)

If some site stores the complete office relation office(F, L, O), it¡s easy.

Problem: such data repository may not exist

- data repository doesn¡t support relational queries
- data is distributed across multiple sites

Solution:

- Build a plan
- Execute the plan

Plan Representation

A plan has the same representation as an operator whose body is an ordered conjunction of *operator instances*.

Example: a two-step plan:

plan(¡429¡, F) =>

userid-room(¡429¡, E)   *finger*(F, L, E, ¡429¡, Ph)

Plan Interpretation

The body of a plan is a *logical conjunction*

- the order is unimportant

The body can be viewed *procedurally*

- the order is very important

A plan¡s head specifies what information is actually returned to the user. E.g.

plan(¡429¡, F) =>

userid-room(¡429¡, E)   *finger*(F, L, E, ¡429¡, Ph)

- last names are not returned to the user

Solutions to a Query

plan(X_1, ..., X_n) => O_1 ... O_k is a solution to

query(Y_1, ..., Y_n) <= rel_1(..., Y_i, ...) ... rel_m(..., Y_j, ...)

if

The binding patterns of the plan¡s operator instances are satisfied.

All tuples satisfying plan(X_1, ..., X_n) must satisfy query(X_1, ..., X_n)

Binding Pattern Satisfaction

The binding patterns of the plan¡s operator instances are satisfied.

I.e. if $V is a bound argument of Oj then

- V must be used as a free argument to some other operator instance Oi where i < j, or
- a value of V must be a bound argument in the query head.

Query Satisfaction

All tuples satisfying plan (X_1, ..., X_n) must satisfy query (X_1, ..., X_n)

In other words, for all c_1, ... , c_n

plan(c_1 , ..., c_n) => query(c_1 , ..., c_n)

where each c_i is a constant.

Solutions: Example

plan($O, F)

query-for-first-name($O, F)

userid-room($O, E)

finger(F, L, $E, O, Ph)

The plan

plan(¡429¡, F) =>

userid-room(¡429¡, E)   *finger*(F, L, E, ¡429¡, Ph)

is a solution to *query-for-first-name*(¡429¡, F)

The binding patterns are satisfied.

Solutions: Query Satisfaction

plan(c_1, c_2)
=> *userid-room*(c_1, E) *finger*(c_2, L, E, c_1, Ph)
=> office (F_0, L_0, c_1) *email*(F_0, L_0, E)
email(c_1, L, E) *office*(c_2, L, c_1) *phone*(O,Ph)
=> *office*(c_2, L, c_1)
=> *query-for-first-name*(c_1, c_2)

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Occam Planning Algorithm

input: a query and a set of operators
output: a set of plans, each of which is guaranteed to be a solution
Occam(Q,O): a forward-chaining algorithm for generating query plans
InstantiateOp(Op,B) : generate a set of operator instances given an operator Op and a set B of bound variables.
FindSolutions(Seq,Q): generate solution plans from given sequences

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Occam: Main Idea

Start from the empty sequence
Search the space of totally ordered sequences of operator instances
Proceed until all alternatives are exhausted, or a resource bound is exceeded
Each sequence is expanded by postpending an instance of each potential operator to produce several new sequences.

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Redundant Solutions

A solution is *redundant* if we can eliminate operator instances from the plan and still obtain a solution.

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Redundant Solutions: Example

> op_1(X) => rel_1(X)
> op_2($X, Y) => rel_2(X, Y)
> op_3($X, Y) => rel_2(X, Y) rel_1(Y)

query(X) => rel_1(X)

> plan1(A) => op_1(A)
> plan2(A) => op_1(A) op_2(A, B)
> plan3a(A) => op_1(A) op_3(A, B)
> plan3b(B) => op_1(A) op_3(A, B)

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Reducing Search (1/3)

Pruning Plans with Duplicate Operator Instances

O_1 and O_2 are equivalent if all bound arguments of O_1 are equal to the variables in O_2
userid-room(A, B), userid-room(C, B) [not euqal]
userid-room(A, B), userid-room(A, C) [euqal]

We reject any sequence that contain two equivalent operator instances.

Intelligent Mobile Robot Group
Jane Hsu
yjhsu@csie.ntu.edu.tw

Reducing Search (2/3)

Pruning Shuffled Sequences

> $op_1(X, Y) \Rightarrow rel_1(X, Y)$
> $op_2(\$X, Y) \Rightarrow rel_2(X, Y)$

> s_1: $op_1(A, B)$ $op_2(A, C)$ $op_2(B, D)$
> s_2: $op_1(A, B)$ $op_2(B, D)$ $op_2(A, C)$

Reducing Search (3/3)

We say operator instance O_i is independent on O_j, if neither

- O_i has a bound argument that appears as a free variable in O_j, nor
- There exists an instance O_k such that O_i is dependent on O_k and O_k is dependent on O_j

If two operator instances are independent, then Occam does not need to consider both ordering permutations.

Features of Occam

- Query planning algorithm
  - Domain-independent
  - Sound
  - Complete
  - Efficient
- Multiple information sources
  - legacy systems
  - relational databases
- Reasoning about capabilities of info sources
- Handling partial goal satisfaction

Occam's Razor

The simplest of two or more competing theories is preferable.

William of Occam (1285-1349):

¡It is vain to do with more what can be done with less.¡