Planning and Learning: Explanation-Based Learning

Manuela Veloso

Carnegie Mellon University

Planning, Execution, and Learning
Fall 2016

Thanks to Daniel Borrajo

Learning in Planning

Opportunities and improvements along several dimensions:

- **Search Efficiency**: Learn control knowledge to guide the planner through its search space.
- **Domain Specification**: Learn the preconditions and effects of the planning actions.
- **Quality**: Learn control knowledge for high quality plans.

Choices... The Need for Learning!

- Inductive methods
  - Data-intensive
  - Extract a general description of a *concept* from many examples
- Deductive methods
  - Knowledge-intensive
  - Explain and analyze an example
  - Identify the explanation as the sufficient conditions for describing the concept
  - Generalize instantiated explanation to apply to other instances

Explanation-Based Generalization – EBG, (Mitchell ' 80s)

Inputs:

- Target concept definition
- Training example
- Domain theory
- Operationality criterion

Output:
Generalization of the training example that is

- sufficient to describe the target concept, and
- satisfies the operationality criterion.

The SAFE-TO-STACK Example

Input:

- **target concept:** SAFE-TO-STACK(x,y)

- **training example:**
  ON(OBJ1,OBJ2)
  ISA(OBJ1, BOX) ISA(OBJ2, ENDTABLE)
  COLOR(OBJ1, RED) COLOR(OBJ2, BLUE)
  VOLUME(OBJ1,1) DENSITY(OBJ1,0.1) ...

The SAFE-TO-STACK Example

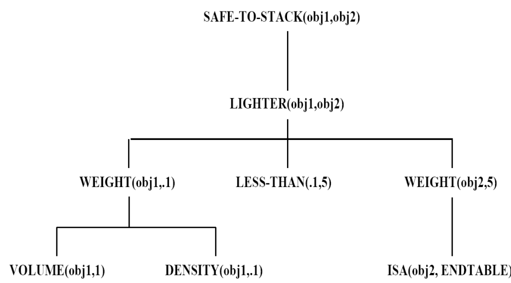
Input:

- **domain theory:**
  1. NOT(FRAGILE(y)) or LIGHTER(x,y) → SAFE-TO-STACK(x,y)
  2. VOLUME(x,v) and DENSITY(x,d) → WEIGHT(x,v*d)
  3. WEIGHT(x1,w1) and WEIGHT(x2,w2) and LESS(w1,w2)
     → LIGHTER(x1,x2)
  4. ISA(x,ENDTABLE) → WEIGHT(x,5)
  5. LESS(0.1,5) ...

- **operationality criterion:**
  learned description should be built of *terms* used to describe examples directly, or other “easily” evaluated, such as LESS.

The SAFE-TO-STACK Example

- Explain why obj1 is SAFE-TO-STACK on obj2.
  - Construct a proof.
  - Do **Goal regression**: regress target concept through proof structure.
  - Proof isolates *relevant* features.



Generating Operational Knowledge

- Generalize proof:
  - Sometimes simply replace constants by variables.
  - Prove that all identified relevant features are necessary in general (hard! -- may need a lot of "extra" knowledge, *domain axioms*).

Output:

VOLUME(x,v1) and DENSITY(x,d1) and ISA(y,ENDTABLE)
and
and LESS(v1*d1,5) → SAFE-TO-STACK(x,y)

EBL: A Deductive Learning Method

Why are examples needed?

- Domain theory contains all the information: simply operationalize target concept.
- Examples focus on the relevant operationalizations: characterize only examples that actually occur.

Actual purpose of EBL:

- not to "learn" more about target concept,
- but to "re-express" target concept in a more operational manner (=efficiency).
- control learning.

EBL in PRODIGY (Minton 87)

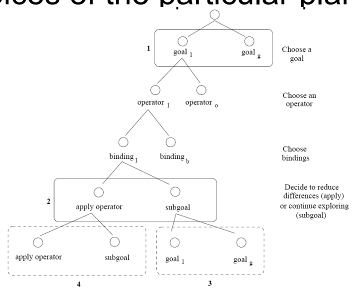
Goal: -- improve the efficiency of the planner
-- learn *control rules*.

Control rules:

- Apply at individual decisions.
- Antecedent matches the state of the planner at decision making time.
- Antecedent is operational -- planner can match its state using control rule language.
- Consequent *selects*, *rejects* or *prefers* particular alternatives.

Target Concepts

Identify the choices of the particular planner:



- Select goal *goal*
- Select operator *op* for achieving *goal*
- Select bindings for operator *op* and goal *goal*
- Decide subgoal if *op* is applicable
- Decide apply *op*

Examples of Control Rules in PRODIGY

```
(CONTROL-RULE SELECT-OP-UNSTACK-FOR-HOLDING
   (if (and  (current-goal (holding <x>))
             (true-in-state (on <x> <y>))))
   (then select operator UNSTACK))

(CONTROL-RULE SELECT-BINDINGS-UNSTACK-HOLDING
   (if (and  (current-goal (holding <x>))
             (current-ops (UNSTACK))
             (true-in-state (on <x> <y>))))
   (then select bindings ((<ob> . <x>) (<underob> . <y>))))

(CONTROL-RULE SELECT-OP-PUTDOWN-FOR-ARMEMPTY
   (if (and  (current-goal (arm-empty))
             (true-in-state (holding <ob>))))
   (then select operator PUT-DOWN))

(CONTROL-RULE SELECT-BINDINGS-PUTDOWN
   (if (and (current-ops (PUT-DOWN))
             (true-in-state (holding <x>))))
   (then select bindings ((<ob> . <x>))))
```

Discussion

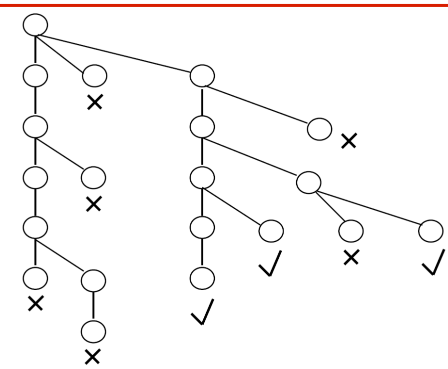
- Very successful in a variety of domains.
- Learned rules are applied as other rules, i.e. if their antecedent *totally* matches planning situation.
- Utility problem: The more rules learned, the slower the deliberation.
  - Matching cost (cost of utilization)
  - Frequency of application
  - Savings every time it is applied
  - Organization of learned rules!
- If EBL system is eager to learn provably correct, the explanation effort is really large, requiring a *complete* domain theory for generalization.
  - Incremental refinement of learned rules

HAMLET: Deduction and Induction

(Borrajo & Veloso 94)

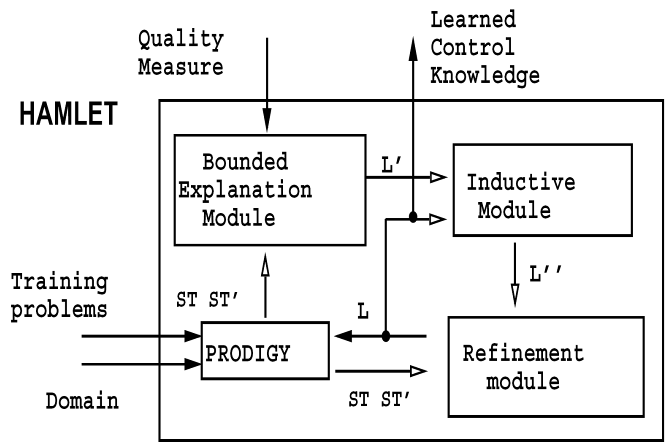
- Extend the basic EBL approach developed for linear problem solving
  - Define new learning opportunities
  - Consider solution quality
- Reduce the explanation effort
  - No need to acquire extra domain knowledge
- Incrementally refine control knowledge
  - Converges towards an experience-supported correct set of rules

A Typical Search Tree



What are the learning opportunities?

HAMLET's Architecture



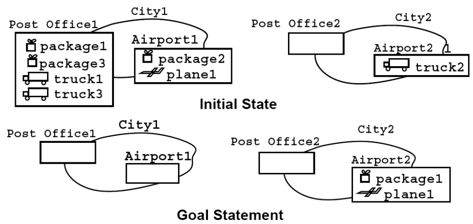
HAMLET's Algorithm

Let L refer to the set of learned control rules.
Let ST, ST′ refer to search trees.
Let P be a problem to be solved.
Let Q be a quality measure.
Initially L is empty.
For all P in training problems
  ST = Result of solving P without any rules.
  ST′ = Result of solving P with current set of rules L.
  If positive-examples-p(ST, ST′,Q)
  Then L′ = Bounded-Explanation(ST, ST′,Q)
    L′′ = Induce(L,L′)
  If negative-examples-p(ST, ST′,Q)
  Then L=Refine(ST, ST′,L′′)

Induction Module

- Why induction?
  - Bounded explanation generates possibly over-specific rules
- Inductive operators
  - Deletion of rules that subsume others
  - Intersection of preconditions. *state*
  - Refinement of subgoaling dependencies. *prior goal*
  - Relaxing the subgoaling dependencies. *prior goal*
  - Refinement of the set of interacting goals. *other goals*
  - Find common superclass. *type of object*

Rule Learned by HAMLET



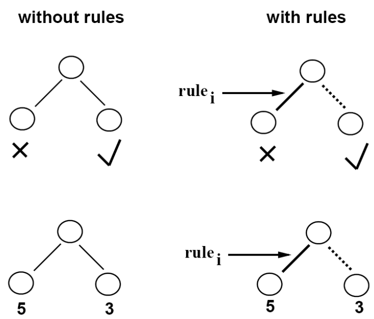
(control-rule select-bind-fly-airplane-1
  (if (current-operator fly-airplane)
       (current-goal (at-airplane <plane1> <airport3>))
       (true-in-state (at-airplane <plane1> <airport2>))
       (true-in-state (at-object <package4> <airport1>))
       (other-goals ((at-object <package4> <airport3>))))
  (then select bindings ((<plane> . <plane1>)
                         (<loc-from> . <airport1>)
                         (<loc-to> . <airport3>))))

Inducing Over Two Rules

- Old rule:
  (control-rule select-unload-airplane-1
    (if (current-goal (at-object <object1> <airport2>))
      (true-in-state (at-airplane <plane4> <airport3>))
      (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
- New rule:
  (control-rule select-unload-airplane-2
    (if (current-goal (at-object <object1> <airport2>))
      (true-in-state (at-airplane <plane4> <airport5>))
      (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
- Induced rule:
  (control-rule induced-select-unload-airplane-3
    (if (current-goal (at-object <object1> <airport2>))
      (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))

Refining

- Why refinement?
  - HAMLET may produce over-general rules
- Negative examples: occasions in which control rules have been applied and should have not



Overgeneralization

- Induced rule
  (control-rule induced-select-unload-airplane-3
  (if (current-goal (at-object <object1> <airport2>))
  (true-in-state (at-object <object1> <airport3>)))
  (then select operators unload-airplane))
- New rule
  (control-rule induced-select-unload-airplane-4
  (if (current-goal (at-object <object1> <airport2>))
  (true-in-state (inside-airplane <object1> <plane4>)))
  (then select operators unload-airplane))
- Overgeneral rule
  (control-rule induced-select-unload-airplane-5
  (if (current-goal (at-object <object1> <airport2>)))
  (then select operators unload-airplane))

Empirical Results

Test sets		Unsolved problems		Solved by both (279 problems, 53.14%)					
				Better solutions		Solution length		Nodes explored	
Goals	Problems	without rules	with rules	without rules	with rules	without rules	with rules	without rules	with rules
1	100	5	0	0	11	327	307	2097	1569
2	100	15	6	0	25	528	479	3401	2308
5	100	44	18	1	33	865	777	5170	3463
10	100	68	32	1	24	770	668	3482	2941
20	75	62	36	0	10	505	455	2216	1924
50	50	49	40	0	0	34	34	143	141
Totals	525	243	132	2	103	3029	2720	16509	12346
%		46.3%	25.1%	0.7%	36.9%			Ratio	1.3

Training problems	Unsolved problems		Solved by both				
			Better solutions		Ratio Solution Length	Ratio Time	Ratio Nodes
	without rules	with rules	without rules	with rules	without/ with rules	without/ with rules	without/ with rules
75	46.29 %	36.38 %	0.35 %	25.89 %	1.11	0.49	1
150	46.29 %	34.29 %	0.72 %	31.9 %	1.06	0.33	1.25
400	46.29 %	25.14 %	0.72 %	36.92 %	1.08	0.32	1.34

Summary – EBL in Planning

- Long-term goal of automating planning efficiency.
- Knowledge in domain theory is not usually effective.
- Explain examples to produce operational control knowledge for decisions.
- Provably correct explanations that generalize to new situations are hard to learn.
- Difficult goal and operator choice interactions can be learned through a combined deductive and inductive approach.
- User's quality metrics can be cast in the learned knowledge.